

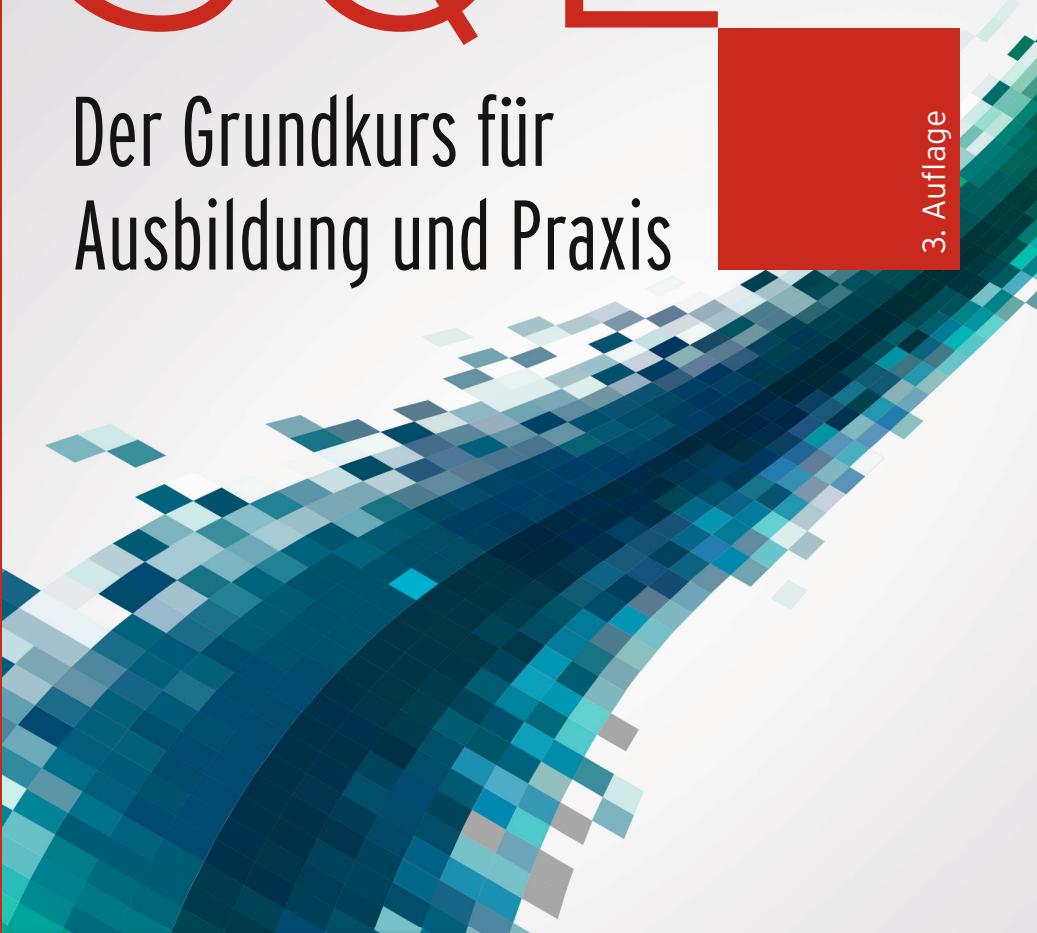
ralf ADAMS

Mit Beispielen in
**MySQL /
MariaDB,
PostgreSQL
und T-SQL**

SQL

Der Grundkurs für
Ausbildung und Praxis

3. Auflage



Mit Exkurs zu NoSQL

HANSER



Bleiben Sie auf dem Laufenden!

Unser **Computerbuch-Newsletter** informiert Sie monatlich über neue Bücher und Termine. Profitieren Sie auch von Gewinnspielen und exklusiven Leseproben. Gleich anmelden unter:
www.hanser-fachbuch.de/newsletter



Ralf Adams

SQL

Der Grundkurs für Ausbildung und Praxis

Mit Beispielen in MySQL/MariaDB,
PostgreSQL und T-SQL

3., aktualisierte Auflage

HANSER

Der Autor:

Ralf Adams, Bochum
Kontakt: sqlbuch@ralfadams.de

Alle in diesem Buch enthaltenen Informationen, Verfahren und Darstellungen wurden nach bestem Wissen zusammengestellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund sind die im vorliegenden Buch enthaltenen Informationen mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor und Verlag übernehmen infolgedessen keine juristische Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieser Informationen – oder Teilen davon – entsteht. Ebenso übernehmen Autor und Verlag keine Gewähr dafür, dass beschriebene Verfahren usw. frei von Schutzrechten Dritter sind. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Buch berechtigt deshalb auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Bibliografische Information der Deutschen Nationalbibliothek:

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Dieses Werk ist urheberrechtlich geschützt.

Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren) – auch nicht für Zwecke der Unterrichtsgestaltung – reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

© 2020 Carl Hanser Verlag München, www.hanser-fachbuch.de

Lektorat: Sylvia Hasselbach

Copy editing: Sandra Gottmann, Wasserburg

Layout: der Autor mit LaTeX

Umschlagdesign: Marc Müller-Bremer, www.rebranding.de, München

Umschlagrealisation: Max Kostopoulos, unter Verwendung von Grafiken von

© shutterstock.com/Viktorus

Datenbelichtung, Druck und Bindung: Kösel, Krugzell

Ausstattung patentrechtlich geschützt. Kösel FD 351, Patent-Nr. 0748702

Printed in Germany

Print-ISBN: 978-3-446-46110-9

E-Book-ISBN: 978-3-446-46274-8

E-Pub-ISBN: 978-3-446-46324-0

Dieses Buch möchte ich allen Lehrerinnen und Lehrern der ehemaligen Aufbau-realschule in Eslohe, Sauerland, widmen.

*Besonders denke ich dabei an meinen Klassenlehrer und späterem Schulleiter, Herrn Schmidt.
Durch Ihr stetes Bemühen um jeden Einzelnen sind Sie mir menschlich und heute als
Lehrer ein Vorbild.*

Inhalt

Vorwort zur 3. Auflage	XVII
------------------------------	------

Teil I Was man so wissen sollte	1
1 Datenbanksystem	3
1.1 Aufgaben und Komponenten	3
1.1.1 Datenbank	3
1.1.2 Datenbankmanagementsystem	5
1.2 Im Buch verwendete Server	7
1.2.1 MySQL und MariaDB	7
1.2.2 PostgreSQL	9
1.2.3 Microsoft SQL Server	10
2 Einführung in relationale Datenbanken	11
2.1 Was ist eine relationale Datenbank?	11
2.1.1 Abgrenzung zu anderen Datenbanken	11
2.1.2 Tabelle, Zeile und Spalte	13
2.1.3 Schlüssel, Primärschlüssel und Fremdschlüssel	16
2.2 Kardinalitäten und ER-Modell	22
2.2.1 Darstellung von Tabellen im ER-Modell	22
2.2.2 1:1-Verknüpfung	24
2.2.2.1 Wann liegt eine 1:1-Verknüpfung vor?	24
2.2.2.2 Wie kann ich eine 1:1-Verknüpfung darstellen?	25
2.2.2.3 Kann man die Kardinalität genauer beschreiben?	26
2.2.3 1:n-Verknüpfung	27
2.2.3.1 Wann liegt eine 1:n-Verknüpfung vor?	27
2.2.3.2 Wie kann ich eine 1:n-Verknüpfung darstellen?	28
2.2.3.3 Kann man die Kardinalität genauer beschreiben?	28

2.2.4	<i>n:m</i> -Verknüpfung	29
2.2.4.1	Wann liegt eine <i>n:m</i> -Verknüpfung vor?	29
2.2.4.2	Wie kann ich eine <i>n:m</i> -Verknüpfung darstellen?	30
2.2.4.3	Kann man die Kardinalität genauer beschreiben?	31
2.2.5	Aufgaben zum ER-Modell	31
2.3	Referenzielle Integrität	32
2.3.1	Verletzung der referenziellen Integrität durch Löschen	33
2.3.2	Verletzung der referenziellen Integrität durch Änderungen	34
2.4	Normalformen	34
2.4.1	Normalform 1	35
2.4.2	Normalform 2	37
2.4.3	Normalform 3	38
2.4.4	Normalform Rest	39
3	Unser Beispiel: Ein Online-Shop	41
3.1	Kundenverwaltung	41
3.2	Artikelverwaltung	42
3.3	Bestellwesen	43
Teil II	Datenbank aufbauen	45
4	Installation des Servers	47
4.1	MySQL unter Windows 10	47
4.2	MariaDB unter Windows 10	53
4.3	Andere Installationen mit Docker	57
4.3.1	MySQL	58
4.3.2	MariaDB	60
4.3.3	PostgreSQL	61
4.3.4	Microsoft SQL Server	62
5	Datenbank und Tabellen anlegen	63
5.1	Die Programmiersprache SQL	63
5.2	Anlegen der Datenbank	64
5.2.1	Wie ruft man den MySQL Client auf?	65
5.2.2	Wie legt man eine Datenbank an?	66
5.2.3	Wie löscht man eine Datenbank?	68
5.2.4	Wie wird ein Zeichensatz zugewiesen?	68
5.2.5	Wie wird eine Sortierung zugewiesen?	70
5.3	Anlegen der Tabellen	72

5.3.1	Welche Datentypen gibt es?.....	73
5.3.2	Wie legt man eine Tabelle an?	74
5.3.3	Wann eine Aufzählung (ENUM) und wann eine neue Tabelle?	77
5.3.4	Wann ein DECIMAL, wann ein DOUBLE?	78
5.3.5	Wann verwendet man NOT NULL?	80
5.3.6	Wie legt man einen Fremdschlüssel fest?	82
5.3.7	Wie kann man Tabellen aus anderen herleiten?	89
5.3.8	Ich brauche mal eben kurz 'ne Tabelle!	90
6	Indizes anlegen	93
6.1	Index für Anfänger	93
6.1.1	Wann wird ein Index automatisch erstellt?	95
6.1.2	Wie kann man einen Index manuell erstellen?.....	97
6.2	Und jetzt etwas genauer	99
6.2.1	Wie kann ich die Schlüsseleigenschaft erzwingen?.....	99
6.2.2	Wie kann ich Dubletten verhindern?	100
6.2.3	Was bedeutet Indexselektivität?	102
6.2.4	Wie kann man einen Index löschen?	104
7	Werte in Tabellen einfügen.....	105
7.1	Daten importieren	105
7.1.1	Das CSV-Format	106
7.1.2	LOAD DATA INFILE	107
7.1.3	Was ist, wenn ich geänderte Werte importieren will?	111
7.2	Daten anlegen	112
7.2.1	Wie legt man mehrere Zeilen mit einem Befehl an?	113
7.2.2	Wie kann man eine einzelne Zeile anlegen?	114
7.2.3	Vorsicht Constraints!.....	115
7.2.4	Einfügen von binären Daten über einen C#-Client.....	116
7.2.5	Einfügen von binären Daten LOAD FILE	119
7.3	Daten kopieren	120
Teil III	Datenbank ändern	123
8	Datenbank und Tabellen umbauen.....	125
8.1	Eine Datenbank ändern.....	125
8.2	Ein Schema löschen	127
8.3	Eine Tabelle ändern	129
8.3.1	Wie kann ich den Namen der Tabelle ändern?	129

8.3.2	Wie kann ich eine Spalte hinzufügen?	131
8.3.3	Wie kann ich die Spezifikation einer Spalte ändern?	132
8.3.4	Zeichenbasierte Spalten in der Länge verändern	133
8.3.5	Zeichensatz verändern	134
8.3.6	Zeichenbasierte Spalten in numerische Spalten verändern	134
8.3.7	Numerische Spalten im Wertebereich verändern	135
8.3.8	Datum- oder Zeitspalten verändern	135
8.3.9	Wie kann ich aus einer Tabelle Spalten entfernen?	137
8.4	Eine Tabelle löschen	138
8.4.1	Einfach löschen	139
8.4.2	Was bedeuten die Optionen CASCADE und RESTRICT?	140
9	Werte in Tabellen verändern	141
9.1	WHERE-Klausel	141
9.1.1	Wie formuliert man eine einfache Bedingung?	142
9.1.2	Wird zwischen Groß- und Kleinschreibung unterschieden?	143
9.1.3	Wie formuliert man eine zusammengesetzte Bedingung?	145
9.2	Tabelleninhalte verändern	146
9.2.1	Szenario 1: Einfache Wertzuweisung	148
9.2.2	Szenario 2: Berechnete Werte	148
9.2.3	Szenario 3: Gebastelte Zeichenketten	149
9.2.4	Was bedeutet die Option LOW_PRIORITY?	150
9.2.5	Was bedeutet die Option IGNORE?	150
9.3	Tabelleninhalte löschen	150
9.3.1	Und was passiert bei Constraints?	151
9.3.2	Was passiert mit dem AUTO_INCREMENT?	152
9.3.3	Was bedeutet LOW_PRIORITY?	153
9.3.4	Was bedeutet QUICK?	153
9.3.5	Was bedeutet IGNORE?	153
9.3.6	Wie kann man eine Tabelle komplett leeren?	154
Teil IV	Datenbank auswerten	155
10	Einfache Auswertungen	157
10.1	Ausdrücke	158
10.1.1	Konstanten	158
10.1.2	Wie kann man Berechnungen vornehmen?	159
10.1.3	Wie ermittelt man Zufallszahlen?	160

10.1.4 Wie steckt man das Berechnungsergebnis in eine Variable?	161
10.2 Zeilen- und Spaltenwahl.....	162
10.3 Sortierung.....	163
10.3.1 Was muss bei der Sortierung von Texten beachtet werden?.....	165
10.3.2 Wird zwischen Groß- und Kleinschreibung unterschieden?.....	167
10.3.3 Wie werden Datums- und Uhrzeitwerte sortiert?.....	169
10.3.4 Wie kann man das Sortieren beschleunigen?	170
10.4 Mehrfachausgaben unterbinden	173
10.4.1 Fallstudie: Datenimport von Bankdaten	174
10.4.2 Was ist beim DISTINCT bzgl. der Performance zu beachten?	176
10.5 Ergebnismenge ausschneiden	177
10.5.1 Wie kann man sich die ersten n Datensätze ausschneiden?	177
10.5.2 Wie kann man Teilmengen mittendrin ausschneiden?	178
10.6 Ergebnisse exportieren.....	179
10.6.1 Wie legt man eine Exportdatei auf dem Server an?	179
10.6.2 Wie legt man eine Exportdatei auf dem Client an?	180
10.6.3 Wie liest man mithilfe eines C#-Client binäre Daten aus?.....	181
11 Tabellen verbinden.....	183
11.1 Heiße Liebe: Primär-Fremdschlüsselpaare	184
11.2 INNER JOIN zwischen zwei Tabellen	187
11.2.1 Bauanleitung für einen INNER JOIN	188
11.2.2 Abkürzende Schreibweisen	192
11.2.3 Als Datenquelle für temporäre Tabellen	192
11.2.4 JOIN über Nichtschlüsselspalten	195
11.3 INNER JOIN über mehr als zwei Tabellen	197
11.4 Es muss nicht immer heiße Liebe sein: OUTER JOIN	200
11.5 Narzissmus pur: SELF JOIN	205
11.6 Eine Verknüpfung beschleunigen	208
12 Differenzierte Auswertungen.....	211
12.1 Statistisches mit Aggregatfunktionen	211
12.2 Tabelle in Gruppen zerlegen	214
12.3 Gruppenergebnisse filtern.....	218
12.4 Noch Fragen?	220
12.4.1 Kann ich nach Ausdrücken gruppieren?	220
12.4.2 Kann ich nach mehr als einer Spalte gruppieren?	220
12.4.3 Wie kann ich GROUP BY beschleunigen?.....	221
12.4.4 Parallele Bearbeitung – unterschiedliche Ergebnisse?	223
12.5 Aufgaben	223

13 Auswertungen mit Unterabfragen	225
13.1 Das Problem und die Lösung	225
13.2 Nicht korrelierende Unterabfrage	228
13.2.1 Skalarunterabfrage	228
13.2.1.1 Beispiel 1: Banken mit höchster BLZ	228
13.2.1.2 Beispiel 2: Überdurchschnittlich teure Artikel	229
13.2.1.3 Beispiel 3: Überdurchschnittlich wertvolle Bestellungen	230
13.2.2 Listenunterabfrage	232
13.2.2.1 Beispiel 1: IN0	232
13.2.2.2 Beispiel 2: ALL0	233
13.2.2.3 Beispiel 3: ALL0	234
13.2.2.4 Beispiel 4: ANY0	237
13.2.3 Unterschied zwischen IN0, ALL0 und ANY0	238
13.2.4 Unterschied zwischen NOT IN0 und <> ALL0	239
13.2.5 Tabellenunterabfrage	239
13.3 Korrelierende Unterabfrage	240
13.3.1 Beispiel 1: Rechnungen mit vielen Positionen	240
13.3.2 Beispiel 2: EXISTS	241
13.4 Fallstudie Datenimport	242
13.5 Wie ticken Unterabfragen intern?	245
13.6 Aufgaben	249
14 Mengenoperationen	251
14.1 Die Vereinigung mit UNION	251
14.2 Die Schnittmenge	254
14.2.1 Mit INTERSECT	254
14.2.2 Mit Unterabfragen	255
14.3 Die Differenzmenge	256
14.3.1 Mit EXCEPT	256
14.3.2 Mit Unterabfragen	257
14.4 UNION, INTERSECT und EXCEPT ... versteh' ich nicht!	258
15 Bedingungslogik	261
15.1 Warum ein CASE?	261
15.2 Einfacher CASE	263
15.3 SEARCHED CASE	265
15.4 Fallbeispiele	267
15.4.1 Lagerbestand überprüfen	267
15.4.2 Kundengruppen ermitteln	268
15.4.3 Aktive Lieferanten ermitteln	271
15.4.4 Aufgaben	272

16 Ansichtssache	273
16.1 Was ist eine Ansicht?	273
16.1.1 Wie wird eine Ansicht angelegt?.....	274
16.1.2 Wie wird eine Ansicht verarbeitet?.....	276
16.1.3 Wie wird eine Ansicht gelöscht?.....	279
16.1.4 Wie wird eine Ansicht geändert?	282
16.2 Anwendungsgebiet: Vereinfachung	282
16.3 Anwendungsgebiet: Datenschutz	285
16.4 Grenzen einer Ansicht	285
17 Exkurs NoSQL	289
17.1 Vorbereitung der MySQL-Shell	290
17.2 Datenmodellierung des Warenkorbs.....	291
17.2.1 JavaScript Object Notation (JSON)	291
17.2.2 Struktur unseres JSON-Dokuments	293
17.3 NoSQL: MySQL mit JavaScript-Client.....	294
17.3.1 Anlegen eines Warenkorbs	295
17.3.2 Inhalte des Warenkorbs anlegen	296
17.3.3 Inhalte des Warenkorbs auswerten	299
17.3.4 Inhalte des Warenkorbs verändern	302
17.4 NoSQL: klassisches SQL mit JSON-Funktionen	304
17.4.1 Anlegen eines Warenkorbs	304
17.4.2 Inhalte des Warenkorbs anlegen	305
17.4.3 Inhalte des Warenkorbs auswerten	307
17.4.4 Inhalte des Warenkorbs verändern	308
17.4.5 Inhalte des Warenkorbs löschen	311
Teil V Anweisungen kapseln	313
18 Locking	315
19 Transaktion	319
19.1 Das Problem	319
19.2 Was ist eine Transaktion?	321
19.3 Isolationsebenen.....	324
19.3.1 READ UNCOMMITTED	324
19.3.2 READ COMMITTED	326
19.3.3 REPEATABLE READ	327
19.3.4 SERIALIZABLE.....	328
19.4 Fallbeispiel in C#	329
19.5 Deadlock	331

20 STORED PROCEDURE	333
20.1 Einstieg und Variablen	334
20.2 Verzweigung	339
20.2.1 Einfache Verzweigung mit IF	339
20.2.2 Mehrfache Verzweigung mit CASE.....	342
20.3 Schleifen	345
20.3.1 LOOP-Schleife	346
20.3.2 WHILE-Schleife	348
20.3.3 REPEAT-Schleife.....	351
20.4 Transaktion innerhalb einer Prozedur	352
20.5 CURSOR.....	353
20.6 Aufgaben	359
21 Funktion.....	361
22 TRIGGER	363
22.1 Was ist das?	363
22.2 Ein Beispiel für einen INSERT-Trigger	365
22.3 Ein Beispiel für einen UPDATE-Trigger.....	366
22.4 Ein Beispiel für einen DELETE-Trigger	368
23 EVENT	371
23.1 Wie legt man ein Ereignis an?	371
23.2 Wie wird man ein Ereignis wieder los?.....	374
Teil VI Anhänge	375
24 Datenbank administrieren.....	377
24.1 Backup und Restore	377
24.1.1 Backup mit mysqldump.....	377
24.1.2 Restore mit mysqldump.....	379
24.2 Benutzerrechte	379
24.2.1 Benutzerrechte und Privilegien	379
24.2.2 Benutzer anlegen/Recht zuweisen	382
24.2.2.1 CREATE USER	382
24.2.2.2 GRANT	383
24.2.2.3 REVOKE	385
24.3 MySQL und MariaDB Engines	386

25 Rund um den MySQL Client	389
25.1 Aufruf(parameter)	389
25.2 Befehle	392
26 SQL-Referenz	397
26.1 Datentypen	397
26.1.1 Numerische Datentypen	397
26.1.1.1 Ganze Zahlen	397
26.1.1.2 Gebrochene Zahlen	398
26.1.2 Zeichen-Datentypen	399
26.1.3 Datums- und Zeit-Datentypen	400
26.1.4 Binäre Datentypen	403
26.1.5 JSON	404
26.1.6 Räumliche Datentypen	404
26.1.7 Standardwerte	405
26.1.8 Zusätze für Datentypen	406
26.2 Operatoren und Funktionen	408
26.2.1 Mathematische Operatoren	408
26.2.2 Mathematische Funktionen	408
26.2.3 Aggregatfunktionen	411
26.3 Bedingungen	414
26.3.1 Vergleichsoperatoren	414
26.3.2 Logikoperatoren	416
26.3.2.1 NOT, Negation, \neg	416
26.3.2.2 AND, Konjunktion, \wedge	417
26.3.2.3 OR, Disjunktion, \vee	418
26.3.2.4 XOR, Antivalenz, \otimes	418
26.4 Befehle	419
26.4.1 Data Definition Language	419
26.4.2 Data Manipulation Language	431
26.4.3 Benutzerverwaltung	435
27 Ausgewählte Quelltexte	439
27.1 DOUBLE versus DECIMAL	439
27.2 Rundungsfehler	443
27.3 NULL versus NOT NULL	444
27.4 Suchen mit und ohne Index	446
27.5 Messen der Performance der Einfügeoperation	449
27.6 Messen der Indexselektivität	452
27.7 Sortieren ohne und mit Index	454

28 Rund ums Zeichen	457
28.1 Für Deutsch relevante Zeichensätze	457
28.2 Für Deutsch relevante Sortierungen	458
29 Quelltexte	461
29.1 MySQL/MariaDB	461
29.1.1 Quelltexte zu Teil II	461
29.1.2 Quelltexte zu Teil III.....	473
29.1.3 Quelltexte zu Teil IV.....	477
29.1.4 Quelltexte zu Teil V.....	521
29.2 PostgreSQL.....	536
29.2.1 Quelltexte zu Teil II	536
29.2.2 Quelltexte zu Teil III.....	545
29.2.3 Quelltexte zu Teil IV.....	549
29.2.4 Quelltexte zu Teil V.....	580
29.3 Microsoft SQL Server	584
29.3.1 Quelltexte zu Teil II	584
29.3.2 Quelltexte zu Teil III.....	595
29.3.3 Quelltexte zu Teil IV.....	600
Literatur	635
Stichwortverzeichnis	639

Vorwort zur 3. Auflage

Und noch'n SQL-Buch. Es gibt so viele SQL-Bücher, dass man berechtigt die Frage stellen kann, warum man noch eines braucht. Ich kann die Frage nur indirekt beantworten. Als Lehrer für Anwendungsentwicklung an einem Berufskolleg habe ich über Jahre erlebt, dass die Auszubildenden sich sehr mit den üblichen Büchern abmühen.

Die fachliche Qualität dieser Bücher ist unbestritten. Aber die Sprache ist meist von *IT-Profi* zu *IT-Profi*, und genau damit sind Auszubildende und Berufsanfänger oft überfordert – zumindest wird der Einstieg erschwert.

Ich habe daher begonnen, leicht verständliche Skripte zu schreiben, aus denen sich dieses Buch speist. Dabei werden Befehle didaktisch reduziert und Beispiele möglichst lebensnah ausgesucht. Fachbegriffe werden nur verwendet, wenn sie IT-sprachlicher Umgang sind; akademische Begriffe werden vermieden, wobei ich ihre Berechtigung nicht in Abrede stellen möchte.

Primärziel ist ein möglichst umfangreicher Ersteinstieg (Grundkurs), der dann durch berufliche Praxis ausgebaut werden kann. Trotzdem vertiefe ich an vielen Stellen im Buch den Einblick in SQL oder den MySQL Server (Vertiefendes) – zum einen, um zu zeigen, dass ich auch ein bisschen was drauf habe, zum anderen, um Neugierde und Jagdtrieb beim Leser¹ zu wecken.

Ein weiterer Grund für dieses Buch ist, dass es mir großen Spaß gemacht hat, es zu schreiben. Ich hoffe, dass es Ihnen genau soviel Spaß macht, es zu lesen und damit zu arbeiten. Falls Sie mich fachlich korrigieren oder ergänzen möchten, senden Sie mir doch bitte eine E-Mail an sqlbuch@ralfadams.de.

Der Titel des Buches ist SQL und nicht MySQL. Ich habe deshalb an vielen Stellen den Unterschied zwischen SQL-Standard und seinen Dialekt(en) aufgezeigt. Trotzdem wird es schwer sein, die Beispiele *einfach so* auf andere DBMS zu übertragen. Auf jeden Fall werden Sie ein Verständnis für den allgemeinen Aufbau und die Funktionsweise der Befehle erwerben, sodass Sie leicht die verschiedenen SQL-Dialekte adaptieren können.

- Bitte beachten Sie, dass die Pfadangaben in den Skripten mit LOAD DATA INFILE angepasst werden müssen, je nachdem, wo Sie die Daten entpacken.

¹ Der besseren Lesbarkeit wegen verzichte ich auf weiblich/männlich-Konstruktionen. Bitte verstehen Sie dies nicht als stillschweigende Hinnahme des geringen Frauenanteils in den IT-Berufen.

- Ich habe angefangen, für die Aufgaben Musterlösungen bei YouTube (<http://www.youtube.com/channel/UCu4ZybNXw1y4Rs4Mgx-4HKw>) einzustellen. In diesen Videos kann ich einfach besser erklären, worauf es bei den Lösungen ankommt.
- Beim Test der Skripte unter MySQL 5.6.19 ist ein Fehler des Servers aufgetreten (siehe [Ada14]).
- Die im Internet unter <http://downloads.hanser.de> verfügbaren Skripte, Beispiele und Musterlösungen sind auf folgenden Servern getestet worden: *MySQL Community Server 8.0.17*, *MariaDB 10.4.6*, *PostgreSQL 11.4* und *MS SQL Server 14.0.3038.14*. Alle Server liefen in einer Dockerinstanz unter Ubuntu 18.10.
- Zwar haben sich MariaDB und MySQL auseinander entwickelt, aber bei den hier vorgestellten Befehlen konnte ich keine Inkompatibilitäten feststellen.
- Für alle Quelltexte, die bis einschließlich Kapitel 16 vorgestellt werden, gibt es Varianten in MySQL/MariaDB, PostgreSQL und T-SQL. Nur bei ganz wenigen Ausnahmen, die durch die jeweiligen Dialekte oder Eigenheiten begründet sind, musste ich auf eine Transkription verzichten.
- Neu beschrieben werden *Generierte Spalten*, die Option `WITH ROLLUP` bei Gruppierungen und *Common Table Expression* (einfach und rekursiv).
- Auch neu ist ein Kapitel über NoSQL, welches das Spektrum abrunden soll. Ursprünglich hatte ich vor, die NoSQL-Inhalte über das Buch auf die passenden Stellen zu verteilen. So sollte alles um Anlegen, Ändern und Löschen und alles über die Auswertungen in den entsprechenden Abschnitten behandelt werden. Das hat sich als unpraktisch erwiesen und deshalb habe ich alles zum Thema NoSQL in einem Kapitel zusammengefasst.

Danksagung

Als Erstes möchte ich mich bei Frau Sylvia Hasselbach vom Hanser Verlag dafür bedanken, dass sie diese Neuauflage – wie schon die Vorauflage – angestoßen und vorangetrieben hat. Frau Rothe und Frau Gottmann haben sprachliche Ausrutscher und flapsige Formulierungen glatt gebügelt. Das Layout wurde von Frau Irene Weilhart betreut.

Ich möchte meinen Kollegen Dr. Andreas Alef und Marco Bakera, mit denen ich das Vergnügen habe, an der Technischen Beruflichen Schule 1 in Bochum (<http://www.tbs1.de>) zu unterrichten, für ihre kritischen und aufmunternden Kommentare danken.

Besonders will ich meine Schülerinnen und Schüler erwähnen. Die hier vorgestellten Beispiele und Konzepte sind in großen Teilen durch ihre schonungslose Kritik an bestehenden Lehrmaterialien entstanden. Das penetrante *Kapier ich nicht!* hat mich immer weiter angespornt, es noch verständlicher zu versuchen. Falls dieses Buch SQL gut vermittelt, ist das auch deren Verdienst.

Dass nun die 3. Auflage dieses Buchs erscheinen kann, ist aber in erster Linie Ihnen, liebe Leserinnen und Leser, zu verdanken; dafür ein herzliches *Dankeschön!*

Ralf Adams, Oktober 2019

TEIL I

Was man so wissen sollte

1

Datenbanksystem

■ 1.1 Aufgaben und Komponenten



Es werden die wichtigsten Aufgaben und Komponenten eines Datenbanksystems vorgestellt. Die Begriffe werden lediglich eingeführt, weil sich ein detailliertes Verständnis erst in den nachfolgenden Kapiteln entwickeln kann.

- Grundkurs
 - Datenbanksystem
 - Datenbank
 - Datenbankmanagementsystem

Ein Datenbanksystem besteht aus einem Datenbankmanagementsystem (DBMS) und den Datenbanken (DB). Beide Komponenten sind in der Praxis eng miteinander verzahnt, sollten aber gedanklich unterschieden werden.

In [Bild 1.1 auf der nächsten Seite](#) ist der Aufbau eines Datenbanksystems schematisch dargestellt. Die Datenbanken enthalten die eigentlichen Daten und unmittelbar damit verknüpfte Datenobjekte wie z.B. eine Ansicht (siehe [Kapitel 16 auf Seite 273](#)). Über eine Kommunikationsschnittstelle werden diese Datenobjekte vom DBMS verwaltet. Das DBMS selbst besteht wiederum aus vielen kleinen Komponenten, die jeweils auf eine Aufgabe spezialisiert sind.

1.1.1 Datenbank

Die Aufgabe der Datenbank ist die logische und physische Verwaltung der Daten und damit eng verbundener Datenobjekte. Alle diese Datenobjekte können vom Programmierer angelegt, geändert und gelöscht werden. Die Änderungen beziehen sich sowohl auf die Struktur als auch auf den Inhalt. So können einer Tabelle neue Spalten (z.B. zweiter Vornname bei einer Adresse) als auch neue Zeilen (z.B. eine neue Adresse) hinzugefügt werden.

Datenbanksystem

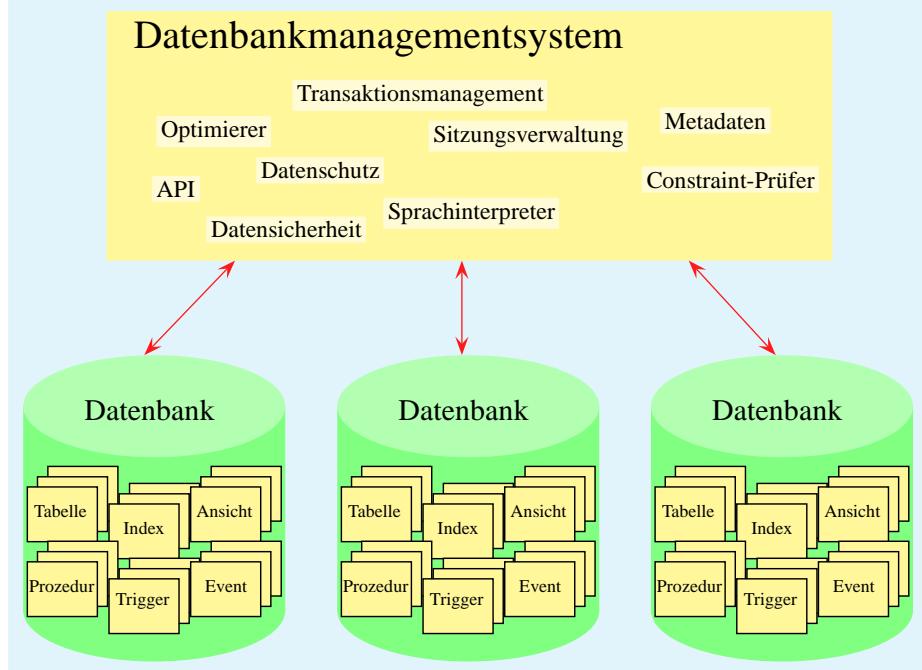


Bild 1.1 Aufbau eines Datenbanksystems

Üblicherweise werden in einer Datenbank folgende Datenobjekte vorkommen¹:

- **Tabellen:** Bei einer relationalen Datenbank werden die Daten in Tabellen organisiert (Kundentabelle, Artikeltabelle, Filmtabelle usw.). Deshalb sind die Tabellen das Herzstück einer Datenbank. Alle anderen Datenbankobjekte sind aus diesen Tabellen abgeleitet oder verwenden diese.
- **Temporäre Tabellen:** Sie werden explizit vom Programmierer oder implizit vom Optimierer angelegt, um Zwischenergebnisse wiederverwendbar zu machen. In der Regel werden diese automatisch nach Beendigung einer Sitzung gelöscht.
- **Indizes:** Diese erlauben eine erhebliche Beschleunigung bestimmter Auswertungen. Die Daten aus den Tabellen werden dabei in frei wählbaren, aber festgelegten Reihenfolgen sortiert.
- **Ansichten:** Auf Vorrat gebastelte Auswertungen, die wie Tabellen verwendet werden können, ohne dabei einen eigenen Datenbestand aufzubauen.
- **Prozeduren:** Kleine, selbst geschriebene SQL-Programme, die auf dem Server ausgeführt werden.

¹ Die Liste ist nicht vollständig. Ich werde mich aber auf diese beschränken.

- **Trigger:** Auch kleine, selbst geschriebene Programme, die aber automatisch aufgerufen werden, wenn Daten in den entsprechenden Tabellen verändert werden.
- **Ereignisse:** Schon wieder kleine, selbst geschriebene Programme, die zeitgesteuert aufgerufen werden.



Definition 1: Datenbank (eng)

Unter einer *Datenbank* im engeren Sinn versteht man eine Informationssammlung, die so strukturiert ist, dass sie zweckgebunden und effizient IT-gestützt verwaltet und ausgewertet werden kann.

Entscheidend ist, dass die Daten *strukturiert* sind! Jede Informationssammlung, die eine gewisse Struktur hat, kann letztlich von einem Computerprogramm verwaltet und ausgewertet werden. Fehlt die Struktur, sieht das Ganze schon anders aus.



Aufgabe 1.1: Überlegen Sie sich mindestens zwei Beispiele für strukturierte und unstrukturierte Datensammlungen.

In MySQL und MariaDB werden die Datenbanken durch die Storage Engines (z.B. InnoDB bzw. XtraDB) realisiert. Diese legen auf den Festspeicherplatten die Dateien an, in denen die Daten abgespeichert werden. Auch die Zugriffskontrolle erfolgt durch die Storage Engines.

1.1.2 Datenbankmanagementsystem

Oben habe ich erwähnt, dass ein Datenbanksystem aus der Datenbank und einem Datenbankmanagementsystem besteht.



Definition 2: Datenbankmanagementsystem (DBMS)

Eine Toolsammlung zur Verwaltung, Bearbeitung und Auswertung einer Datenbank nennt man *Datenbankmanagementsystem (DBMS)*.

Dies sind die Aufgaben eines DBMS²:

- **Sprachinterpreter:** Herzstück des DBMS ist der Interpreter³. Dieser übersetzt die Befehle in einen ausführbaren Code. Die Sprache, die wir hier im Buch verwenden werden, ist SQL. Es gibt und gab aber auch andere Datenbankabfragesprachen wie zum Beispiel dBBase, VB für MS-Access, OO-SQL, Sequel usw.
- **Optimierer:** Die Ausführung eines SQL-Befehls kann oft auf verschiedene Art und Weise passieren. Der Optimierer versucht, anhand von Schätzungen und Algorithmen einen Plan für die Ausführung anzulegen, der möglichst schnell abgearbeitet werden kann.

² Je nach Hersteller oder Lesart finden Sie andere Aufgabensammlungen, aber mit dieser kommen wir schon sehr weit.

³ Es ist müßig, darüber zu streiten, ob es sich um einen Interpreter oder einen Compiler oder einen Jitter handelt. Warum? Weil es keinen interessiert ;-)

- **Sitzungsverwaltung:** Wann immer ein Befehl an den Server gesendet werden soll, muss man sich in einer Sitzung (engl. *session*) befinden. Dazu muss zuerst eine Sitzung geöffnet werden. Jetzt können beliebig viele SQL-Befehle gesendet und Daten empfangen werden. Zum Schluss wird die Sitzung serverseitig – z.B. durch einen Timeout – oder clientseitig beendet.
- **Randbedingungsprüfer:** Für Tabellen können Randbedingungen (engl. *constraints*) formuliert werden, die immer gelten müssen. Würde die Ausführung eines Befehls dazu führen, dass diese Randbedingungen nicht erfüllt sind, wird die Ausführung des Befehls in der Regel verweigert.
- **Datenschutz:** Durch die Vergabe von Zugriffsrechten kann das Recht auf lesende und schreibende Zugriffe so wie auf das Ausführen von Prozeduren passgenau zugeschnitten werden.
- **Datensicherheit:** Der Verlust von Daten ist der GAU⁴ schlechthin. Das DBMS muss sicherstellen, dass nicht durch Serverabsturz oder Ähnliches Daten verloren gehen.
- **Transaktionsmanagement:** Transaktionen ermöglichen parallelen Zugriff und eine Art *undo* im Fehlerfall. Das zu gewährleisten, erfordert eine Menge Mühe. Die Qualität des Transaktionsmanagements ist oft ein entscheidendes Merkmal eines DMBS.
- **API⁵:** Die Daten werden in der Regel durch eine oder mehrere Anwendungen (Clients) bearbeitet. Damit die Anwendung auf das DBMS zugreifen kann, braucht es eine Schnittstelle, über die es zu den Daten gelangt. Der MySQL oder MariaDB Server bietet beispielsweise APIs für C, C++, C#/.NET, PHP, Perl, Python und Tcl. Auch stehen APIs für JDBS⁶ und ODBC⁷ zur Verfügung. Man nennt sie *Konnektoren*.
- **Metadaten:** Verwaltungsinformationen, Statistiken etc., eben der ganze Rest.

Der Begriff *Datenbankmanagementsystem* wird oft anstelle von *Datenbanksystem* verwendet. Gerade die schematischen Darstellungen in den Dokumentationen der Hersteller unterscheiden nicht zwischen diesen beiden Begriffen.

Sind die Datenbanken eines Datenbankmanagementsystems in Form von Tabellen organisiert, so handelt es sich um ein *relationales Datenbankmanagementsystem (RDBMS)*.

Und noch der Vollständigkeit halber:



Definition 3: Datenbanksystem

Ein System, welches die Datenbanken und das dazugehörige Datenbankmanagementsystem als Komplettpaket anbietet, nennt man *Datenbanksystem*.

⁴ Abkürzung für: Größter anzunehmender Unfall

⁵ Abkürzung für: Application Programming Interface; engl. für Programmierschnittstelle

⁶ Abkürzung für: Java Database Connectivity: Programmierschnittstelle für JAVA

⁷ Abkürzung für: Open Database Connectivity: Eine offene standardisierte Schnittstelle. Sie wird von fast allen Datenbanksystemherstellern angeboten. Wer ODBC-Programme schreibt, kann leicht zwischen verschiedenen Datenbanksystemherstellern wechseln.

■ 1.2 Im Buch verwendete Server



Kurzporträts der verwendeten SQL Server: Hersteller und Geschichte

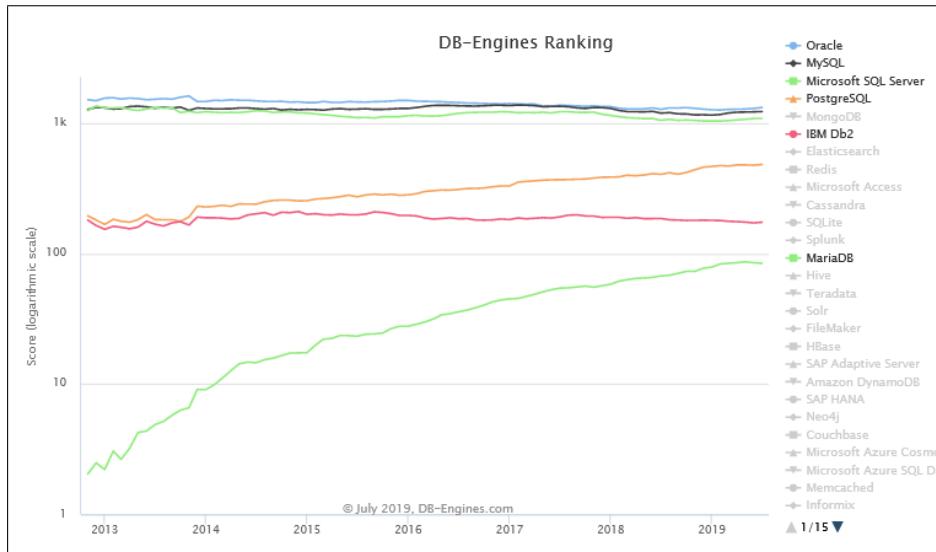


Bild 1.2 Ranking einiger RDBMS, Quelle [DE19]

In der ersten Auflage dieses Buchs habe ich fast ausschließlich MySQL/MariaDB als Plattform genutzt. Um SQL breiter vorstellen zu können, wurden in der zweiten Auflage die meisten Beispiele auch in PostgreSQL angeboten. Konsequenterweise wird in der dritten Auflage ein weiteres System bedient: der MS SQL Server. Wie Sie Bild 1.2 entnehmen können, decke ich damit eine Vielzahl von Installationen und SQL-Dialekt ab.

1.2.1 MySQL und MariaDB

Die schwedische Firma MySQL AB hat MySQL von 1994 bis 2008 entwickelt. Die ursprüngliche Intention war eine verbesserte und beschleunigte Verarbeitung eines selbst entwickelten Tabellensystems mit dem Namen ISAM. Dazu wurde mSQL⁸ genutzt. Von 2008 bis 2010 wurde MySQL AB von Sun Microsystems gepflegt, und seit Januar 2010 wird MySQL unter dem Schirm von Oracle weiterentwickelt.

Der Name *MySQL* kommt nicht vom englischen *my* (mein). Einer der Firmengründer, Michael Widenius, hat sympathischerweise den Vornamen *My* seiner Tochter verwendet. Der Name des Delphins im Logo ist Sakila. Er wurde in einem Wettbewerb ermittelt, den der

⁸ Kein Tippfehler! Siehe [Ltd10]

Open Source-Entwickler Ambrose Twebaze aus Uganda gewann. Sakila ist ein Mädchenname in der Sprache *siSwati* und auch der Name einer Stadt in Tansania.

Nach der Übernahme von MySQL durch Oracle haben die Spannungen zwischen den Entwicklern und Oracle ständig zugenommen, sodass der *Erfinder* von MySQL – Michael Widenius – sich mit der neu erstellen Engine Aria von MySQL abgespalten und 2009 das Projekt MariaDB ins Leben gerufen hat. Wie MySQL ist auch dieses Projekt nach einer Tochter von Widenius benannt. Wegen seiner offeneren Lizenzpolitik und der schnelleren Umsetzung von Neuerungen und Fehlerkorrekturen hat MariaDB an vielen Stellen – aber nicht, wie oft behauptet, an den meisten – MySQL abgelöst (siehe [Bild 1.2 auf der vorherigen Seite](#)).

MySQL und MariaDB sind Client-Server-Datenbanksysteme. Ein Server stellt alleine oder im Verbund mit anderen Servern den Anwendungen (Clients) die Datenbankdienste zur Verfügung. Der MySQL-/MariaDB-Server besteht – wie jedes DBMS – aus vielen Komponenten, die hier kurz angerissen werden.

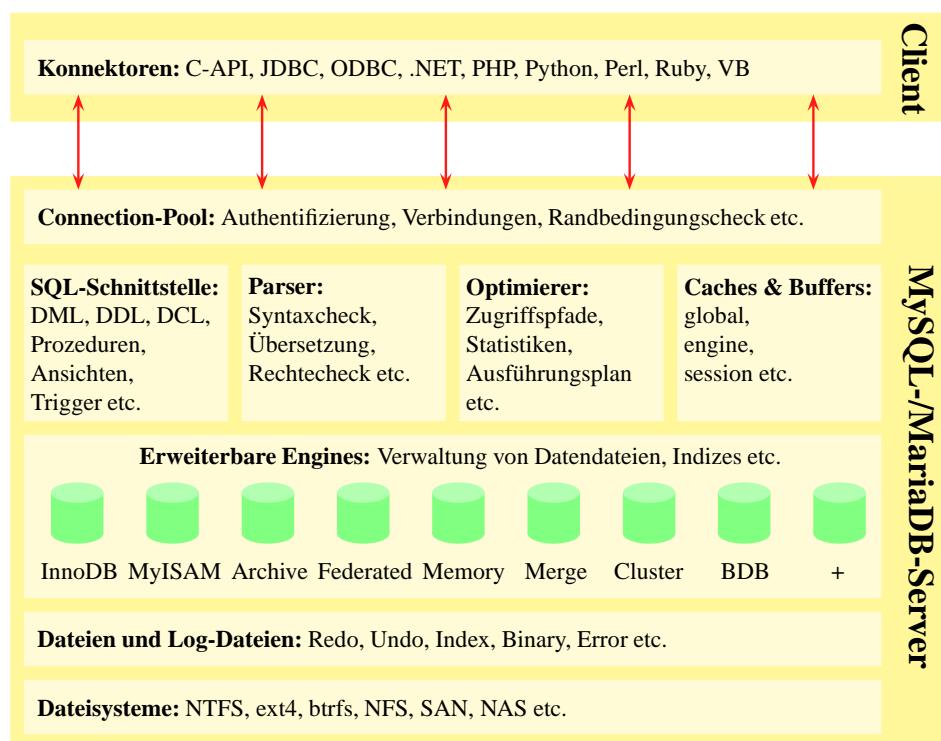


Bild 1.3 Komponenten von MySQL und MariaDB

- **Konnektoren** (Verbinder): Der Client baut über die Konnektoren eine Sitzung zum Server auf. Dies erfolgt über das TCP/IP-Protokoll und in der Regel über den Port 3306.
- **Connection-Pool:** Hier werden die Verbindungen zu den Clients verwaltet. Beim Verbindungsaufbau wird anhand des Benutzernamens und des Passworts die Verbindungsanfrage authentifiziert. Ist die Anzahl der maximal verfügbaren Verbindungen (Connections Limits) nicht überschritten, wird ein Verbindungsthread eingerichtet. Ebenso wer-

den die anderen Grenzwerte für die Verbindung überwacht: Datenübertragungsvolumen, Timeout etc.

- *SQL-Schnittstelle*: Hier werden die SQL-Befehle entgegengenommen. Sie werden dann zum Parser weitergereicht.
- *Parser*: Der Parser überprüft die Syntax eines Befehls und ob man die Ausführungsrechte für diesen Befehl hat.
- *Optimizer* (Optimierer): Anhand von Schätzungen, Statistiken und Algorithmen wird für nicht triviale Befehle ein Ausführungsplan erstellt. Dieser Ausführungsplan berücksichtigt ggf. im Cache vorhandene Ergebnisse.
- *Caches und Buffers* (Zwischenspeicher): Daten und Ergebnisse können in Zwischenspeichern aufgehoben werden. Diese sind nur in der Sitzung verfügbar, die diese erstellt hat (lokal) oder in allen Sitzungen (global).
- *Storage Engines*: Die Motoren des Servers. Hier werden die Daten tatsächlich verarbeitet. Jede Engine ist dabei für bestimmte Aufgabenstellungen besonders gut geeignet. Der große Vorteil von MySQL und MariaDB ist, dass jeder mit einem besonderen Bedarf eine Engine bauen kann. Er muss *nur* die Schnittstellen beachten (siehe [MyS19b]) und kann dann seine Speziallösungen anbieten.
- *File System* (Dateisystem): Je nach Betriebssystem werden hier Daten in unterschiedlichen Dateisystemen (NTFS, BTRFS, ETX4 etc.) abgelegt. Bis auf die Frage, ob das Dateisystem zwischen Groß- und Kleinschreibung unterscheidet, spielt dieses für die SQL-Programmierung keine Rolle.
- *Management Services & Utilities*: Parallel dazu gibt es die vielen kleinen Helferlein, ohne die nichts geht: für das Sichern und Wiederherstellen von Daten, Datenreplikation, Administration und Konfiguration, Datenmigration und Metadaten.

In den nachfolgenden Kapiteln werden wir gemeinsam eine Datenbank planen, installieren, einrichten, verändern und verwenden. Dabei werden die vielen Fragen beantwortet, die Sie nach dieser kurzen Einführung mit Sicherheit haben werden, also nur Geduld ...

1.2.2 PostgreSQL

Geboren wurde PostgreSQL 1986 als Universitätsprojekt an der Universität von Kalifornien, Berkeley. Professor Michael Stonebraker⁹ begann das Projekt als Nachfolgeprojekt der Ingres-Datenbank, welche ebenfalls noch heute verwendet wird. Daher leitet sich auch der Name her: *post* *ingres*. In den nächsten acht Jahren wurde Postgres von Prof. Stonebraker und seinen Studenten immer weiter entwickelt.

Bis 1995 verstand Postgres allerdings kein SQL, sondern nur eine eigene Sprache namens POSTQUEL. Die beiden Doktoranden Andrew Yu und Jolly Chen haben Postgres um SQL erweitert und als Postgres95 veröffentlicht.

1996 wurde Postgres95 als Open-Source-Projekt der Netzgemeinde zur Verfügung gestellt und wird seitdem sehr stark von Unterstützern weiterentwickelt und gefördert. Ebenso

⁹ Prof. Stonebraker ging später mit einem Fork von Postgres – Illustra – zu Informix, welche diesen in den Universal Server integrierte. Auch diese Datenbank besteht heute noch.

wurde das Erscheinungsjahr aus dem Produktnamen entfernt und die Datenbank heißt seitdem PostgreSQL.

Weitere Informationen über die Geschichte von PostgreSQL und die aktuellen Features des Servers können Sie auf der Homepage des Projekts (<https://www.postgresql.org>) nachlesen.

Von <http://www.postgresql.org/download/> können Sie die aktuelle PostgreSQL-Version für Linux-Distributionen und Windows herunterladen. Eine sehr ausführliche Online-Dokumentation steht unter <http://www.postgresql.org/docs/> zur Verfügung. Dort wird auch auf eine Reihe von weiteren Dokumentationen wie beispielsweise zum Thema *Sicherheit* verwiesen.

1.2.3 Microsoft SQL Server

Anders als bei den anderen Datenbankservern kann ich hier keine Geschichten über Töchter oder Universitätslaufbahnen erzählen. Die Entwicklung des MS SQL Servers ist da schon etwas nüchtern ([Gar16]).

In Kooperation mit Sybase brachte Microsoft 1989 die erste Version seines Servers auf den Markt. Zielplattform war das Betriebssystem OS/2. Die Kooperation sah so aus, dass beide gemeinsam am Sybase Server arbeiteten und Sybase das Produkt unter seinem Namen auf UNIX-basierenden Systemen anbot und Microsoft auf OS/2.

Spätestens seit 1992 speist sich der Quelltext beider Server-Produkte (Sybase 4.0 und MS SQL Server 4.2) gleichwertig aus beiden Firmen. Ab 1993 ist nicht mehr OS/2, sondern Windows NT die Zielplattform des MS SQL Servers. Mit der Portierung auf Windows NT wurden erhebliche Anpassungen im Quelltext notwendig, die nicht mehr für das Sybase-Produkt verwendet werden konnten; die beiden Systeme begannen sich voneinander zu entfernen. Besonders der Wunsch von Sybase, dass der Quelltext möglichst plattformneutral bleiben sollte, widersprach den strategischen Zielen Microsofts, die eine volle Unterstützung der damals wirklich bedeutsamen Neuerungen von Windows NT anstrebten.

Microsoft entschied, dass der SQL Server ein zentrales Produkt für die Windows NT Strategie werden sollte. Daher wurden von anderen Datenbankherstellern erfahrene Entwickler angeworben. Dieses geballte Know-how mündete 1998 in die Serverversion 7.0 mit dem Arbeitstitel *Sphinx*. Mit dieser völlig überarbeiteten Version wurde auch die Zusammenarbeit mit Sybase überflüssig und beide Produkte sind seitdem voneinander unabhängig.

Der MS SQL Server wurde in der Folge immer weiter ausgebaut: Es kamen Sprachelemente hinzu, die Sicherheit wurde verbessert, die Performance gesteigert, die Stabilität erhöht, andere Betriebssysteme werden unterstützt usw. Besonders das grafische Frontend und die hohe Integration in die Visual Studio Entwicklungsumgebung werden von vielen Entwicklern geschätzt.

Anders als MySQL/MariaDB oder PostgreSQL kommt der MS SQL Server aber nicht aus der Open-Source- oder Uni-Ecke. Er ist ein rein kommerzielles Produkt und hat daher in den entsprechenden Kreisen ein Imageproblem.

2

Einführung in relationale Datenbanken

■ 2.1 Was ist eine relationale Datenbank?



Worüber reden wir hier überhaupt? Relationale Datenbanken werden in Abgrenzung zu anderen Datenbanken eingeführt. Der Begriff einer Tabelle wird intensiv anhand eines Beispiels besprochen. Ebenso wird erklärt, wie Tabellen untereinander in Kontakt bleiben.

- Grundkurs
 - Abgrenzung relationale Datenbank zur hierarchischen Datenbank
 - Grundsätzlicher Aufbau einer Tabelle
 - Schlüssel und Primärschlüssel
 - Fremdschlüssel und Verknüpfungen
- Vertiefendes
 - COBOL-Daten als hierarchische Datenbank
 - XML als hierarchische Datenbank
 - Formale Definitionen zu den Grundbegriffen
 - Viele Worte für eine Sache: Begriffsübersicht
 - Starke und schwache Entitätentypen

2.1.1 Abgrenzung zu anderen Datenbanken

Als *Erfinder* relationaler Datenbanken gilt Dr. E. F. Codd. Dieser hatte in einem Paper¹ den Begriff eingeführt und die Vorteile seiner Lösung beschrieben.

Der Begriff *relationale Datenbank* selbst ist für uns ein bisschen missverständlich. Die Bedeutung wird meist fälschlicherweise wie folgt beschrieben: *Zwischen den Daten bestehen Beziehungen (Relationen); daher der Name*. Nun ist aber der englische Begriff für Beziehung *relationship* und nicht *relation*; somit kann das nicht die Bedeutung sein.

Relation ist ein wohldefinierter Fachbegriff und kann umgangssprachlich als *Tabelle* übersetzt werden. Relationale Datenbanken sind also Datenbanken, die in Tabellen organisiert

¹ Siehe [Cod70]

sind. Soll heißen: Die Daten werden in Spalten und Zeilen strukturiert. Das hört sich so selbstverständlich an, dass man sich klar machen muss, dass es auch Alternativen gibt.

Als Beispiel sei hier die Ablage von Bestelldaten eines Kunden in einem Format, wie es beispielsweise in einer COBOL Data Division² verwendet wird, angegeben:

```

01 12345 Gamdschie      Samweis   Beutelhaldenweg 5   67676   Hobbingen   Privatkunde
05 00001 20120512 12:30:00 bezahlt
10 7856 Silberzwiebel  15  6.15 €
10 7863 Tulpenzwiebel 10  32.90 €
10 9015 Spaten        1  19.90 €
05 00002 20120530 17:15:00 offen
10 9010 Schaufel      1  15.00 $
10 9015 Spaten        1  19.90 €
01 12346 Beutlin       Frodo    Beutelhaldenweg 1   67676   Hobbingen   Privatkunde
05 00001 20111110 11:15:00 bezahlt
10 3001 Papier (100) 10  23.00 €
10 3005 Tinte (gold) 1  55.70 €
10 3006 Tinte (rot)  1  6.20 €
10 3010 Feder         5  25.00 €

```



Aufgabe 2.1: Versuchen Sie, die Bedeutung der Informationen zu ermitteln. Welche Aufgabe hat die erste Zahl einer Zeile?

Die Informationen sind hier hierarchisch organisiert. Die Zugehörigkeit einer Information (beispielsweise eines bestellten Artikels) ergibt sich aus der Position, *wo* die Information steht, oder besser: *unter* welcher Information diese steht. In Bild 2.1 können Sie diesen Zusammenhang noch deutlicher erkennen.

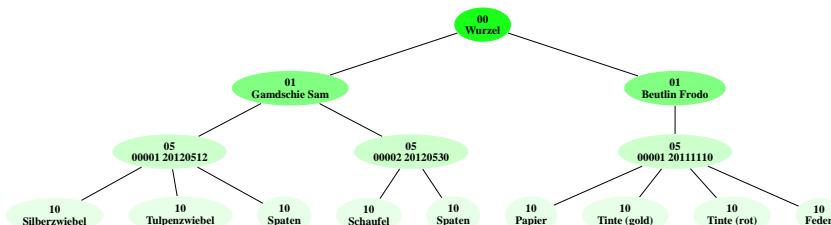


Bild 2.1 Hierarchische Darstellung der Bestelldaten

Die ersten beiden Ziffern sind das sogenannte *Satzkennzeichen*, welches dem verarbeitenden Programm mitteilt, was für eine Art von Information in der Zeile steht. Irgendwo im System müssen diese Satzkennzeichen z.B. in Copybooks³ spezifiziert sein.

Die Bestellposition *Silberzwiebel* gehört zur Bestellung vom 12.05.2012, weil sie *unterhalb* der Bestelldaten steht. Die hierarchische Struktur ist also selbst eine Information.

In vielen Büchern finden Sie Aussagen wie: *Hierarchischen Datenbanken sind veraltet*. Diese Aussage ist aber selbst schon veraltet. Betrachten Sie doch mal die Darstellung der gleichen Daten in diesem Format:

² Das Format ist hier vereinfacht. Aber es lohnt sich, mal einen Blick in die Spezifikation zu werfen: [Wik19c].

³ Siehe [Wik19h]

```

1 <?xml version="1.0" encoding="iso-8859-15"?>
2 <Rechnung>
3   <Kunde>
4     <Kundennummer>12345</Kundennummer>
5     <Nachname>Gamdschie</Nachname>
6     <Vorname>Sam</Vorname>
7     <Strasse>Beutelhaldenweg</Strasse>
8     <Hausnummer>5</Hausnummer>
9     <Postleitzahl>67676</Postleitzahl>
10    <Ort>Hobbingen</Ort>
11    <Kundenart>Privatkunde</Kundenart>
12    <Bestellung>
13      <Bestellnummer>00001</Bestellnummer>
14      <Bestelldatum>20120512 12:30:00</Bestelldatum>
15      <Status>bezahlt</Status>
16      <Artikel>
17        <Artikelnummer>7856</Artikelnummer>
18        <Artikelname>Silberzwiebel</Artikelname>
19        <Menge>15</Menge>
20        <Preis>6.13</Preis>
21        <Währung>EUR</Währung>
22      </Artikel>
23      <Artikel>
24        <Artikelnummer>7863</Artikelnummer>
25        <Artikelname>Tulpenzwiebel</Artikelname>
26        <Menge>10</Menge>
27        <Preis>32.90</Preis>
28        <Währung>EUR</Währung>
29      </Artikel>
30      [...]
31  </Rechnung>
```

Man erkennt deutlich, dass die Verwaltung von Daten im XML-Format auch eine hierarchische ist. Und gerade diese Art der Datenverwaltung hat in den letzten Jahren eine enorme Entwicklung vollzogen.

Erinnern Sie sich noch an die Definition einer Datenbank (siehe [Definition 1 auf Seite 5](#))? Dort wird lediglich darauf abgestellt, dass die Daten eine Struktur haben. Haben Daten eine effiziente Struktur, können diese leicht von Programmen ausgelesen und editiert werden. Der Begriff *Datenbank* ist somit unabhängig von der Art, wie die Daten strukturiert sind.

Eine andere *moderne* Art der Datenhaltung sind objektorientierte Datenbanken. Diese verwalten alle Daten, die zu einem Objekt gehören. So richtig durchgesetzt haben sich diese Datenbanken aber noch nicht, auch wenn Produkte wie MongoDB mittlerweile weit verbreitet sind (siehe [\[DE19\]](#)).

2.1.2 Tabelle, Zeile und Spalte

Wenn wir nun sagen, dass relationale Datenbanken in Tabellen verwaltet werden, stellt sich die Frage: Was genau sind eigentlich Tabellen? Gehen wir naiv an die Sache heran

und bauen die obigen Daten in Tabellenform um. Jeder Informationstyp wird jetzt in eine eigene Tabelle gepackt.

Tabelle: kunde							
kunde_id	nachname	vorname	straße	hnr	plz	ort	art
12345	Gamdschie	Samweis	Beutelhaldenweg	5	67676	Hobbingen	Privatkunde
12346	Beutlin	Frodo	Beutelhaldenweg	1	67676	Hobbingen	Privatkunde

Bild 2.2 Kunde in Tabellenform

Tabelle: bestellung		
bestellung_id	zeitstempel	status
1	20120512 12:30:00	bezahlt
2	20120530 17:15:00	offen
3	20111110 11:15:00	bezahlt

Bild 2.3 Bestellung in Tabellenform

Tabelle: position				
artikel_id	bezeichnung	menge	preis	währung
7856	Silberzwiebel	15	6.15	EUR
7863	Tulpenzwiebel	10	32.90	EUR
9015	Spaten	1	19.90	EUR
9010	Schaufel	1	15.00	USD
9015	Spaten	1	19.90	EUR
3001	Papier (100)	10	23.00	EUR
3005	Tinte (gold)	1	55.70	EUR
3006	Tinte (rot)	1	6.20	EUR
3010	Feder	5	25.00	EUR

Bild 2.4 Bestellposition in Tabellenform

Tabelle: artikel				
artikel_id	bezeichnung	warengruppe	einzelpreis	währung
7856	Silberzwiebel	pflanzen	0.41	EUR
7863	Tulpenzwiebel	pflanzen	3.29	EUR
9010	Schaufel	garten	15.00	USD
9015	Spaten	garten	19.90	EUR
3001	Papier (100)	büro	2.30	EUR
3005	Tinte (gold)	büro	55.70	EUR
3006	Tinte (rot)	büro	6.20	EUR
3010	Feder	büro	5.00	EUR

Bild 2.5 Artikel in Tabellenform

Sie erkennen sicherlich einen entscheidenden Vorteil der tabellarischen Darstellung: Die Daten sind inhaltlich konsistent angeordnet. Sowohl Mensch als auch Maschine haben eine eindeutige Zuordnung: Informationstyp zu Tabelle. Bei anderen Darstellungen ist diese

Eindeutigkeit nicht gegeben. Dort werden beispielsweise in einer XML-Datei alle Informationen, die zu einem Geschäftsprozess gehören, zusammengefasst; auch das kann vorteilhaft sein.

Werden aber die Kundendaten in einem anderen Geschäftsprozess benötigt, muss bei der XML-Darstellung dieser Informationstyp erst einmal extrahiert werden. Bei einer tabellarischen Darstellung kann diese Information sofort in einem anderen Zusammenhang verwendet werden.

Man hat sozusagen die Daten in inhaltliche Bestandteile zerlegt und damit Bausteine geschaffen, die man dann immer wieder in neuen Varianten zusammensetzen kann.

Wir verwenden den Begriff der *Tabelle*, sollten diesen aber etwas formaler umschreiben, indem wir erst seine Bestandteile definieren:



Definition 4: Spalte

Eine *Spalte* einer Tabelle enthält immer Informationen desselben Typs. Damit meint man Daten

1. denselben technischen Datentyps und
2. die zur gleichen inhaltlichen Kategorie gehören.

Jede Spalte hat eine Überschrift, die die Inhalte festlegt und innerhalb der Tabelle eindeutig ist.

Der *technische Datentyp* ist hier beispielsweise ein Text oder eine ganze Zahl oder ein Datum. In einer Spalte dürfen entweder nur Text oder ganze Zahlen oder Datumsangaben vorkommen.

Verschärfend müssen diese Werte auch inhaltlich das Gleiche umschreiben, zur selben Kategorie gehören. In einer bestimmten Textspalte dürfen beispielsweise nur Nachnamen, aber keine Vornamen vorkommen. In einem bestimmten Zahlenfeld nur die Menge der bestellten Artikel, aber keine Artikelnummer. In einem bestimmten Datumsfeld nur das Bestelldatum und nicht das Bezahldatum.

Was sich hier so trivial anhört, wird in der Praxis gerne mal verschlampt. Auslöser ist meist ein missverständlicher Spaltenname. So ist hier beispielsweise die Spalte *preis* nicht so eindeutig; ist er brutto oder netto, Einzel- oder Gesamtpreis? Der Spaltenname muss also möglichst präzise festlegen, was die Inhalte dieser Spalte sind. Falls dies wegen der Kürze des Spaltennamens nicht möglich ist, sollte dies in einer entsprechenden Dokumentation festgehalten werden. Dabei wird häufig auch festgelegt, welche Werte in einer Spalte stehen dürfen und welche – meist fachlich motiviert – falsch sind.



Definition 5: Wertebereich

Als *Wertebereich* (Domäne) einer Spalte bezeichnet man die fachlich und formal gültigen Spaltenwerte.

Den Wertebereich festzulegen, ist eine Aufgabe, die nicht zwingend dem Datenbankdesign zugeordnet werden muss. Meist werden die Wertebereiche bei der Formulierung des Pflichtenhefts mit der Spezifikation der Eingabemasken festgelegt. Dies ist deshalb sinn-

voll, weil bei der Dateneingabe in der Regel die fachliche und formale Plausibilisierung der Daten erfolgt (siehe Drei-Schichten-Architektur auf [Seite 80](#)).



Definition 6: Zeile

Eine *Zeile* einer Tabelle enthält inhaltlich zusammenhängende Informationen, die immer in der gleichen Reihenfolge in den Spalten vorkommen.

Jede Zeile beschreibt einen inhaltlichen Zusammenhang (Entität): Bei unserem Beispiel sind alle Daten der ersten Zeile in der Tabelle *kunde* die Daten der Entität *Samweis Gamdschie*. Der inhaltliche Zusammenhang ist, dass dies die Daten eines Kunden sind. Auch die Reihenfolge der Spalten ist in jeder Zeile gleich; technisch wird dadurch das Ausstanzen von Einzelinformationen erheblich vereinfacht.



Definition 7: Tabelle

Eine *Tabelle* besteht aus mindestens einer Spalte und einer endlichen Anzahl von Zeilen. Die Anzahl der Spalten ist ebenfalls endlich. Der Name der Tabelle legt den Informationstyp fest und muss innerhalb der Datenbank eindeutig sein.

Ich werde in diesem Buch folgende Begriffe verwenden: *Tabelle*, *Zeile* und *Spalte*. In der [Tabelle 2.1](#) finden Sie andere Begriffe, die in der Literatur auch vorkommen.

Tabelle 2.1 Namensübersicht

Buch	Alternative Namen
Zeile	Tupel, Entität (<i>entity</i>), Objekt, Datensatz, Record
Spalte	Attribut, Feld, Datenfeld, Item, Eigenschaft (<i>property</i>)
Wertebereich	Domäne
Tabelle	Matrix, Entitätentyp (<i>entity type</i>), Relation, Klasse, Recordset
Datenbank	Schema

Ich will nicht in Abrede stellen, dass so tolle Namen wie *Tupel* oder *Entitätentyp* irgendwo notwendig sind, aber zum Verständnis und zur Programmierung von Datenbanken sind einfache intuitive Namen völlig ausreichend. Bei Wikipedia⁴ können weitere Begriffe nachgelesen werden.

2.1.3 Schlüssel, Primärschlüssel und Fremdschlüssel

Zwar sind die Daten nun auf Tabellen verteilt (siehe beispielsweise [Bild 2.2 auf Seite 14](#)), aber der Zusammenhang zwischen den Informationen ist verloren gegangen!

Die Bestellungen haben die Information verloren, zu welchem Kunden sie gehören. Die Positionen haben die Information verloren, zu welcher Bestellung sie gehören. Lediglich

⁴ Siehe [\[Wik19m\]](#)

die Artikeldaten in der Tabelle **position** sind erhalten geblieben, kommen aber doppelt vor.

Nun kommt der Trick: In der Tabelle **position** sind die Artikelnummer und die Artikelbezeichnung abgespeichert. Hat man nun eine separate Artikeltabelle, ist die Artikelbezeichnung in der Tabelle **position** überflüssig, da die Positionstabelle über die Artikelnummer auf die Artikelbezeichnung verweist. Dieser Verweis kann aber nur dann klappen, wenn jeder Artikel eine Artikelnummer, die sonst kein anderer Artikel verwendet, hat. Wäre dies nicht der Fall, wüsste man nicht, welcher Artikel bestellt wurde.

Haben wir die Idee verstanden, wird es Zeit für eine formale Beschreibung.



Definition 8: Schlüssel

Ein **Schlüssel** ist mindestens eine Spalte einer Tabelle, welche jede Zeile konzeptionell eindeutig macht, d.h., der Schlüsselwert kann per Definition in der Tabelle nur einmal vorkommen.

Ein Schlüssel kann aus mehreren Spalten zusammengesetzt werden, muss aber minimal sein. Ein Schlüssel ist dann *minimal*, wenn er nicht mehr reduziert werden kann, ohne seine Eindeutigkeit zu verlieren.

Was bedeutet das? Nehmen wir die Tabelle **position** in [Bild 2.4 auf Seite 14](#). Betrachten Sie jede Zeile in der Tabelle. Es fällt auf, dass die Position mit dem Spaten zweimal vorkommt. Warum auch nicht? Beide Positionen wurden in unterschiedlichen Bestellungen aufgegeben. Somit machen die Spalten **artikelnr**, **bezeichnung**, **menge**, **preis** und **währung** die Zeile nicht eindeutig, d.h. unterscheidbar von allen anderen. Wir brauchen eine Spalte, die unabhängig von den anderen Spalteninhalten jede Zeile einmalig, also eindeutig macht.

Üblicherweise werden die Positionen einer Rechnung mit 1 beginnend durchnummieriert. Die Tabelle wäre dann wie in [Bild 2.6](#) aufgebaut.

Tabelle: position					
positionsnr	artikel_id	bezeichnung	menge	preis	währung
1	7856	Silberzwiebel	15	6.15	EUR
2	7863	Tulpenzwiebel	10	32.90	EUR
3	9015	Spaten	1	19.90	EUR
1	9010	Schaufel	1	15.00	USD
2	9015	Spaten	1	19.90	EUR
1	3001	Papier (100)	10	23.00	EUR
2	3005	Tinte (gold)	1	55.70	EUR
3	3006	Tinte (rot)	1	6.20	EUR
4	3010	Feder	5	25.00	EUR

Bild 2.6 Tabelle **position**, 1. Stufe des Umbaus

Leider werden dabei die Zeilen nur zufällig eindeutig. Die Spaten-Zeilen unterscheiden sich nun, aber eben zufälligerweise. Wäre die erste Spaten-Zeile die zweite Position in der Bestellung, dann wären die Zeilen wieder gleich. Somit ist die Positionsnummer alleine keine Spalte, die die Zeilen konzeptionell eindeutig macht. Wir bräuchten noch die Bestellnummer, da die Position innerhalb der Bestellung sehr wohl eindeutig ist (siehe [Bild 2.7](#)).

Tabelle: position							
bestellung_id	positionsnr	artikel_id	bezeichnung	menge	preis	währung	
1	1	7856	Silberzwiebel	15	6.15	EUR	
1	2	7863	Tulpenzwiebel	10	32.90	EUR	
1	3	9015	Spaten	1	19.90	EUR	
2	1	9010	Schaufel	1	15.00	USD	
2	2	9015	Spaten	1	19.90	EUR	
3	1	3001	Papier (100)	10	23.00	EUR	
3	2	3005	Tinte (gold)	1	55.70	EUR	
3	3	3006	Tinte (rot)	1	6.20	EUR	
3	4	3010	Feder	5	25.00	EUR	

Bild 2.7 Tabelle position, 2. Stufe des Umbaus

Nun haben wir einen Schlüssel im Sinne der [Definition 8 auf der vorherigen Seite](#):

- Ein *Schlüssel* ist mindestens eine Spalte: `bestell_id + positionsnr`
- Jede Zeile ist konzeptionell eindeutig: Da die Bestellnummer in `bestell_id` pro Bestellung und die Positionsnummer in `positionsnr` innerhalb der Bestellung eindeutig sind, ist die Kombination von beiden innerhalb der Tabelle `position` eindeutig.
- Er ist minimal: Nimmt man die Bestellnummer weg, ist die Zeile nicht mehr konzeptiell eindeutig. Nimmt man die Positionsnummer weg, ist die Zeile nicht mehr konzeptiell eindeutig. Da der Schlüssel aus keinen weiteren Bestandteilen zusammengesetzt ist, ist er insgesamt minimal.

Es fällt auf, dass wir für die Tabelle `position` einen zusammengesetzten Schlüssel verwenden. Besonders erwähnenswert ist, dass ein Teil des Schlüssels der Tabelle `position` ein Schlüssel der Tabelle `bestellung` ist. Es gibt Tabellen, die können *aus eigener Kraft* einen Schlüssel liefern (z.B. `bestellung`), und Tabellen, deren Schlüssel andere Schlüssel enthalten müssen (z.B. `position`):



Definition 9: Starke Tabelle

Eine Tabelle wird als *stark* bezeichnet, wenn sie den Schlüssel ausschließlich aus eigenen Feldern bilden kann.

Definition 10: Schwache Tabelle

Eine Tabelle wird als *schwach* bezeichnet, wenn sie zur Schlüsselbildung Spalten anderer Tabellen benötigt.

Was auch immer man mit *stark* oder *schwach* ausdrücken wollte: Starke Entitätentypen (Tabellen) sind häufig solche, deren Existenz nicht von anderen abhängig ist. Schwache Entitätentypen sind hingegen oft von der anderen Tabelle existenzabhängig. An unserem Beispiel wird dies leicht deutlich: Ohne eine Bestellung gibt es keine Bestellpositionen. Löscht man eine Bestellung, werden zwangsläufig auch die damit verbundenen Positionen gelöscht⁵.

⁵ Eine analoge Unterscheidung findet Sie auch in den OO-Begriffen *Aggregation* und *Komposition*.



Definition 11: Primärschlüssel

In einer Tabelle können mehrere Schlüssel vorkommen. Einer der Schlüssel wird herausgehoben und als *Primärschlüssel* gekennzeichnet. Die anderen Schlüssel nennt man *Sekundärschlüssel* oder *Schlüsselkandidaten*.

Die **Definition 11** kommt ein bisschen seltsam daher, ist aber für die Praxis sehr wichtig. Schlüssel werden bei relationalen Datenbanken sehr oft und intensiv verwendet, wie wir noch sehen werden. Deshalb muss festgelegt werden, welcher der möglichen Schlüssel zur Weiterverarbeitung verwendet wird und welcher nicht.

Beispiel: Nehmen wir eine Tabelle, in der PKW-Daten für eine Autovermietung abgespeichert sind. Neben vielen anderen Spalten wird es sicherlich eine Spalte kennzeichnen⁶ geben. Schnell haben Sie überprüft, dass diese Spalte ein Schlüssel ist. Jetzt gibt es aber auch die Spalte *fahrgestellnummer*⁷. Auch diese Spalte ist für sich alleine schon ein Schlüssel der Tabelle. Beide Spalten würden jede Zeile der Tabelle unabhängig voneinander konzeptionell eindeutig machen. Es stellt sich nun die Frage, welcher der beiden Schlüssel für die Programmierung verwendet werden soll, welcher also meine Nummer 1 ist.

Oft werden die Begriffe *Schlüssel* und *Primärschlüssel* ohne Unterscheidung verwendet. Dies ist immer dann zulässig, wenn es in der Tabelle nur einen Schlüssel gibt.



Aufgabe 2.2: Ermitteln Sie für die Tabellen kunde, bestellung, position und artikel die Primärschlüssel.

Jetzt kommen wir zum Ursprungsproblem zurück: Zwar kennen nun die Positionen wieder ihre Bestellungen, aber was ist mit den anderen Zusammenhängen?

Fangen wir damit an, dass die Bestellungen wieder wissen sollen, wer sie aufgegeben hat. Das Problem ist nun einfach zu lösen. Wie bei der Position müssen wir den passenden Primärschlüsselwert der Tabelle *kunde* in die Tabelle *bestellung* aufnehmen (siehe **Bild 2.8**).

Tabelle: bestellung				
bestellung_id	kunde_id	zeitstempel	status	
1	12345	20120512 12:30:00	bezahlt	
2	12345	20120530 17:15:00	offen	
3	12346	20111110 11:15:00	bezahlt	

Bild 2.8 Tabelle bestellung

⁶ Siehe [Wik19k]

⁷ Siehe [Wik19g]

**Definition 12: Fremdschlüssel**

Ein *Fremdschlüssel* ist mindestens eine Spalte der Tabelle A, welche Primärschlüsselwerte der Tabelle B enthält/enthalten. Diese Definition gilt auch für $A = B$. Der Fremdschlüssel muss dieselbe Syntax und Semantik wie der dazugehörige Primärschlüssel haben.

Dadurch, dass die Kundennummer als Schlüssel eindeutig ist, ist auch durch die Verwendung des Fremdschlüssels exakt festgelegt, welcher Kunde die Bestellung aufgegeben hat.

Bitte beachten Sie, dass der Fremdschlüssel in der Regel selbst kein Schlüssel ist. Wie in [Bild 2.9 auf der nächsten Seite](#) zu sehen ist, kommt der Wert 12345 in der Spalte kunde_id in der Tabelle bestellung zweimal vor und ist somit nicht mehr eindeutig.

Der Nutzen des Fremdschlüssels wird deutlich, wenn man sich die Tabellen *position* und *artikel* anschaut. In der Positionstabelle kommen Werte vor, die offensichtlich schon in der Artikeltabelle erfasst sind. Diese Doppelung der Information nennt man Redundanz.

**Definition 13: Redundanz**

Kommt eine Information syntaktisch (formal) und semantisch (inhaltlich) mehrfach in einer Datenbank vor, spricht man von einer *Redundanz*.

Die Nachteile der Redundanz sind offensichtlich: Zum einen verschwendet man Speicherplatz, und zum anderen müssen Änderungen – hier beispielsweise an der Artikelbezeichnung – mehrfach vorgenommen werden (*Wartungsinstabilität*). Auch können unter bestimmten Umständen Performanceverluste auftreten, da mehr Datenmenge verarbeitet werden muss. Wir werden später aber noch sehen, dass Redundanzen auch gewollt und sinnvoll sein können⁸.



Aufgabe 2.3: Bauen Sie die Tabelle *position* so um, dass keine Redundanzen mehr vorkommen.

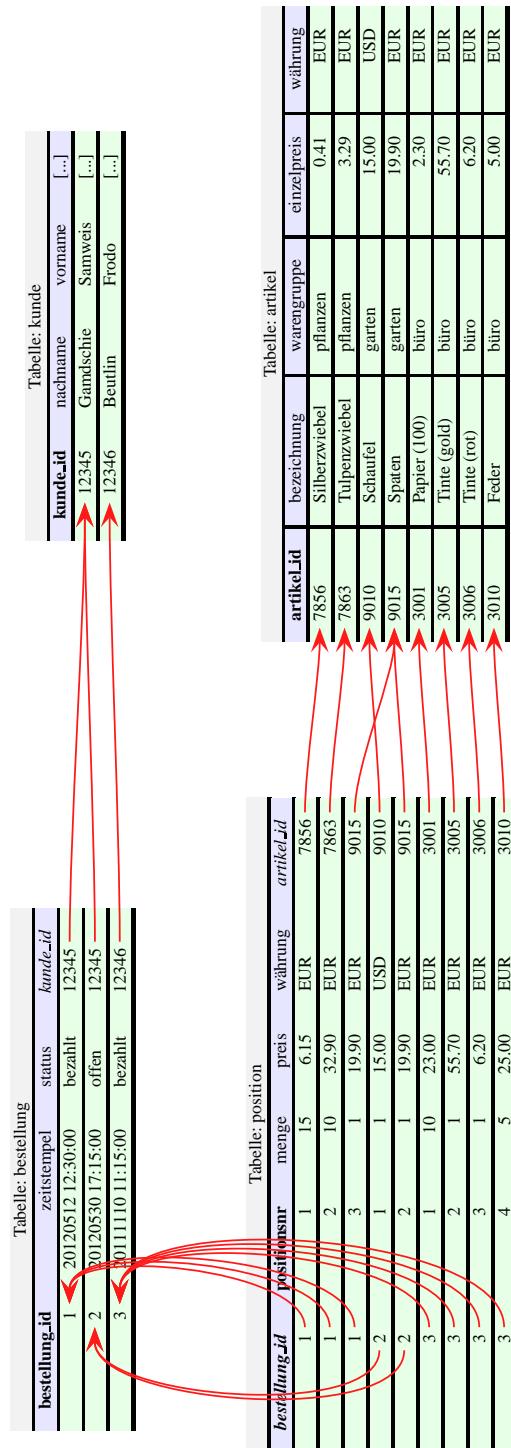
Wie Primär- und Fremdschlüssel zusammen Informationen in Tabellen zu einem System verknüpfen, können Sie in [Bild 2.9 auf der nächsten Seite](#) sehen.

Die Fremdschlüsselwerte in *kunde_id* zeigen auf die Primärschlüsselwerte in der Tabelle *kunde*. Das Gleiche tun die Fremdschlüsselwerte in den anderen Tabellen.

**Definition 14: Verknüpfung**

Eine *Verknüpfung* oder *Referenz* entsteht durch die Verwendung von Fremdschlüsseln.

⁸ Z.B. Vermeidung der Änderungsweitergabe (siehe [Seite 34](#)) oder NoSQL-Dokumente (siehe [Seite 293](#))

**Bild 2.9** Bestellung: Primär-Fremdschlüsselbeziehungen

■ 2.2 Kardinalitäten und ER-Modell



Ein Bild sagt mehr als 1000 Worte. Das ER-Modell als zentrale Planungs- und Darstellungsform wird eingeführt und geübt.

- Grundkurs
 - Darstellung einer Tabelle
 - Darstellung einer $1:1$ -Verknüpfung, $1:n$ -Verknüpfung und $n:m$ -Verknüpfung
 - Aufgaben zum ER-Modell
- Vertiefendes
 - Verschiedene Notationen
 - Varianten der $1:1$ -Verknüpfung, $1:n$ -Verknüpfung und $n:m$ -Verknüpfung
 - Identifizierende $1:n$ -Verknüpfung
 - Umsetzung der Kardinalitäten in den Tabellen
 - Die Hilfstabelle bei der $n:m$ -Verknüpfung
 - Selbstreferenzen bei verschiedenen Kardinalitäten

Die Zeilen der verknüpften Tabellen stehen in einem gewissen Mengenverhältnis zueinander. Im Prinzip stehen drei Arten von Mengenverhältnissen zur Verfügung, von denen es aber Varianten gibt.

Bevor wir uns diese genauer anschauen, möchte ich eine grafische Notation vorstellen, die uns bei der Darstellung der Tabellen helfen soll.

2.2.1 Darstellung von Tabellen im ER-Modell

In der Informatik ist es üblich, dass man EDV-Systeme als Modell darstellt. Vielleicht kennen Sie schon solche Modelle aus der Programmierung: Programmablaufplan nach [ISO85] (ISO/IEC 5807, DIN 66001), Struktogramm nach [Nor85] (DIN 66261), Datenflussdiagramm nach [ISO85] (ISO 5807), UML-Klassendiagramme usw. Die Modellierungstechnik für relationale Datenbanken ist das *Entity Relationship Model* oder auch ER-Modell.



Definition 15: Entity Relationship Model

Das Entity Relationship Model (ER-Modell oder ERM) ist eine grafische Darstellung von Tabellen und ihren Beziehungen untereinander.

Die Tabellen unseres Online-Shops sollen in einer einfachen und übersichtlichen Form dargestellt werden. Ein kurzer Blick in [Wik19f] liefert folgende Notationen (Auszug):

- **Chen (modifiziert):** Hier werden die Tabellen als Rechtecke dargestellt. Die Spalten der Tabellen werden als Blasen um die Tabelle herum notiert. In die Blase schreibt man den Spaltennamen. Der Name des Primärschlüssels wird dabei unterstrichen. Genaueres unter [Wik18] und [Che76].
- **IDEFIX:** Die Tabellen werden auch hier als Rechtecke (ggf. mit abgerundeten Ecken) dargestellt. Die Spalten werden aber innerhalb des Rechtecks notiert. Die Primärschlüs-

selbspalten werden dabei durch einen Linie von den anderen Spalten abgetrennt. Diese Notation ist der Quasistandard US-amerikanischer Behörden. Genaueres unter [Wik17].

- **UML-Klassendiagramm:** Die Tabellen werden dabei wie Klassen in der UML-Notation dargestellt. Die Verknüpfung ist dabei vom Typ *Assoziation*. Genaueres unter [Oes06] oder [JRH⁺04].
- **Krähenfuß (Martin):** Die Tabellen werden in Rechtecken dargestellt. Diese enthalten die Spaltennamen. Vor dem Spaltennamen ist Platz für eine weitere Spezifikation der Spalte (z.B. als Fremdschlüssel). Auch dazu gibt es einen Wikipedia-Eintrag: [Wik19].

Die Chen-Notation ist m.E. nach zu unübersichtlich bei Tabellen mit mehr als fünf Spalten. Da die Spalten als Blasen um die Tabelle herum gruppiert werden, erscheint die Darstellung schnell überladen und ist aufwendig zu editieren.

Die IDEF1X-Notation ist im europäischen Raum eher unüblich. Ich werde diese deshalb hier nicht verwenden.

Die UML-Notation wird eigentlich nur als Behelf in der objektorientierten Programmierung verwendet, da es in der UML keine eigene ER-Notation gibt.

Verbleibt die Krähenfuß-Notation. Sie ist einfach zu lesen, leicht zu erlernen und wird in vielen Tools – wie z.B. der MySQL Workbench – verwendet. Anders als bei der Chen-, IDEF1X- oder UML-Notation ist die Krähenfuß-Notation nicht im Sinne einer Norm festgeschrieben und kann daher ruhigen Gewissens auf die praktischen Bedürfnisse eines Projekts angepasst werden.

In Bild 2.10 wird beispielhaft die Tabelle `bestellung` dargestellt. Die erste Zeile des Rechtecks enthält den Tabellennamen. Dieser wird – wie auch die Spaltennamen – per Konvention immer klein geschrieben. Alle nachfolgenden Zeilen sind in zwei Spalten aufgeteilt. Die linke Spalte kann dazu verwendet werden, eine Spalte bzgl. ihrer Besonderheiten auszuweisen. Meist durch zwei Abkürzungen: PK⁹ für den Primärschlüssel und FK¹⁰ für den Fremdschlüssel. In der rechten Spalte stehen die Spaltennamen.

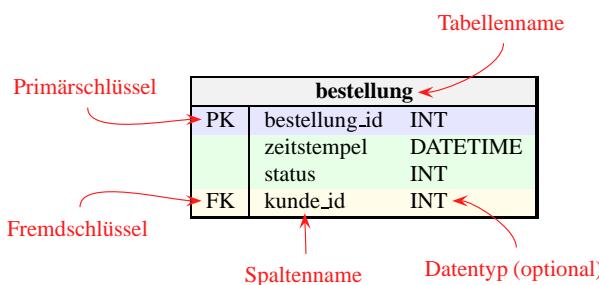


Bild 2.10 Darstellung einer Tabelle in der Krähenfuß-Notation

Wird die Darstellung in CASE-Tools¹¹ wie MySQL Workbench erstellt, kann man oft den Datentyp als dritte Spalte oder durch ein Trennzeichen hinter dem Namen angeben. In

⁹ Abkürzung für: primary key; engl. für Primärschlüssel

¹⁰ Abkürzung für: foreign key; engl. für Fremdschlüssel

¹¹ Abkürzung für: Computer Aided Software Engineering; engl. für rechnerunterstützte Softwareentwicklung

Anlehnung an das UML-Klassendiagramm geschieht dies immer häufiger in der Notation *name:datentyp*.



Aufgabe 2.4: Erstellen Sie den ersten Entwurf eines ER-Modells für den Online-Shop mit den Tabellen aus [Bild 2.9 auf Seite 21](#). Vergessen Sie nicht, die Primär- und Fremdschlüssel zu kennzeichnen.

2.2.2 1:1-Verknüpfung

2.2.2.1 Wann liegt eine 1:1-Verknüpfung vor?

Nehmen wir an, wir wollten bei den Kunden unseres Online-Shops eine Umfrage starten. Für *Produktqualität*, *Service* und *Internetauftritt* können die Kunden Schulnoten vergeben. Damit diese Umfrage aussagekräftig ist, müssen wir sicherstellen, dass jeder Kunde nur genau einen Satz Antworten liefern kann.

Im Prinzip stehen zwei Möglichkeiten zur Verfügung:

1. Die Tabelle *kunde* wird um die Spalten *produktqualität*, *service* und *internet-auftritt* erweitert.
2. Es wird die Tabelle *umfrage* eingeführt. Um sicherzustellen, dass die Antworten dem richtigen Kunden zugewiesen werden, haben die Antworten die Kundennummer *kunde_id* als Primärschlüssel.

Für Variante 1 spricht, dass dies einfach umzusetzen ist und man keine weitere Tabelle anzulegen braucht.

Für Variante 2 spricht, dass diese Lösung langfristig *wartungsstabiler* ist. Die Tabelle *kunde* ist sicherlich eine der wichtigsten Tabellen des Online-Shops. Viele Module des Programms greifen in unterschiedlichen Zusammenhängen darauf zu. Eine Änderung an der Struktur dieser Tabelle kann somit an vielen Stellen zu unangenehmen Nebenwirkungen führen. Vor allem, wenn man die Ergebnisse der Umfrage nach einer angemessenen Frist wieder löscht. Bei Variante 2 muss man nur die Tabelle *umfrage* entfernen, die Kundendaten bleiben unberührt. Bei Variante 1 muss die Kundentabelle schon wieder in ihrer Struktur verändert werden, was nochmalig zu unerwünschten Nebeneffekten führen kann. Ich entscheide mich daher für Variante 2 und richte die Tabelle *umfrage* ein.



Definition 16: Kardinalität

Unter der *Kardinalität* (Mengenverhältnis) versteht man die Angabe darüber, wie viele Zeilen aus der Tabelle *A* einer Zeile aus der Tabelle *B* zugeordnet sind und umgekehrt.

Die **Definition 16** schließt $A = B$ nicht aus, da es vorkommen kann, dass eine Tabelle mit sich selbst verknüpft ist. Denken Sie beispielsweise an eine Personentabelle, die für jede Person auch den Ehepartner festlegt¹².

¹² Mehr unter dem Stichwort SELF JOIN in [Kapitel 11.5 auf Seite 205](#).



Definition 17: 1:1-Verknüpfung

Zwei Tabellen A und B stehen in einer **1:1-Verknüpfung**, wenn es zu jeder Zeile aus der Tabelle A höchstens eine Zeile in der Tabelle B gibt und wenn es zu einer Zeile aus der Tabelle B genau eine Zeile in der Tabelle A gibt.

Diese Formulierung lässt zu, dass es zu einem Datensatz in der Tabelle A keinen in der Tabelle B gibt, aber *nicht* umgekehrt.

Überprüfen wir nun die [Definition 17](#) an unserem Beispiel: Ein Kunde kann nur eine Bewertung abgeben, und eine Bewertung kann nur von einem Kunden abgegeben werden sein. Somit liegt hier eine **1:1-Verknüpfung** vor.

Um sicherzustellen, dass ein Kunde nur einen Antwortsatz haben kann, müssen wir in die Datenbank einbauen, dass das Mengenverhältnis zwischen den Kunden und den Antworten immer passt. Haben zwei Tabellen den gleichen Primärschlüssel, muss eine **1:1-Verknüpfung** vorliegen. In unserem Beispiel würde man den Primärschlüssel der Tabelle Kunde auch zum Primärschlüssel der Tabelle **umfrage** machen. Es gibt zwar auch andere Wege, eine **1:1-Verknüpfung** herzustellen, aber dieser Weg ist der einfachste¹³.



Aufgabe 2.5: Machen Sie sich anhand einer Zeichnung wie in [Bild 2.9 auf Seite 21](#) am Beispiel **umfrage** klar, warum eine **1:1-Verknüpfung** vorliegen muss, wenn beide den gleichen Primärschlüssel haben.

Die **1:1-Verknüpfung** ist in der Praxis recht selten, aber nicht unüblich. Sie entsteht eigentlich immer dann, wenn man große Tabellen – wie z.B. eine Kundentabelle – in inhaltlich abgeschlossene Teiltabellen zerhackt. Die 4. Normalform bietet bzgl. der *mehrwertigen Abhängigkeiten* das theoretische Grundgerüst zu diesem Vorgang (siehe [Kapitel 2.4.4 auf Seite 39](#)).

Beispiel: Wenn Sie Wetterdaten einer Wetterstation z.B. vom Deutschen Wetterdienst bekommen, finden Sie pro Wetterstation in einer Zeile *zig* Angaben. Diese kann man inhaltlich aufteilen und in **1:1-verknüpfte** Tabellen auslagern. Eine Tabelle enthält alle Daten zur Luft (Temperatur, Windgeschwindigkeit und -richtung usw.) und eine andere die Daten zum Niederschlag (Menge, Art usw.). Hintergrund dieser Aufteilung ist, dass man für eine Auswertung meist nur die inhaltlich zusammenhängenden Informationen benötigt und die anderen Daten die Auswertung – selbst wenn man sie aktiv ausblendet – nur belasten würden.

2.2.2.2 Wie kann ich eine **1:1-Verknüpfung** darstellen?

In [Bild 2.11 auf der nächsten Seite](#) erkennt man eine Linie zwischen den beiden Tabellen. An beiden Enden ist ein einfacher senkrechter Strich zu erkennen, der eine idealisierte 1 darstellen soll. Mit dieser Notation wird ausgesagt, dass jeder Kunde eine Umfrage machen kann und es zu jeder Umfrage einen Kunden gibt.

¹³ In [Kapitel 6.2.1 auf Seite 99](#) wird erklärt, wie man einen Index so erstellt, dass er die Schlüsseleigenschaft (siehe [Definition 8 auf Seite 17](#)) erzwingt. Ist so ein Index ein Fremdschlüssel, kann dadurch auch eine **1:1-Verknüpfung** realisiert werden.

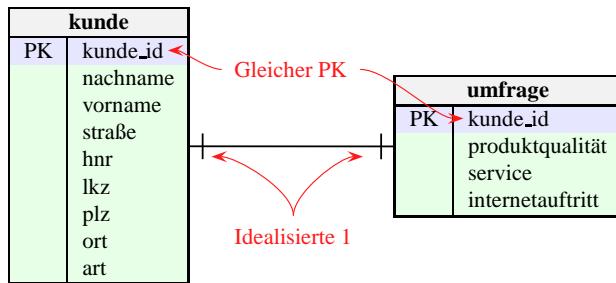


Bild 2.11 Einfache 1:1-Darstellung in Krähenfuß-Notation

2.2.2.3 Kann man die Kardinalität genauer beschreiben?

Diese Information ist zwar richtig, aber wir können es noch besser: Ein Kunde muss sich nicht an der Umfrage beteiligt haben, aber eine Umfrage muss zu einem Kunden gehören. Dazu kann man die Notation wie in Bild 2.12 erweitern.

- Ein Kunde hat keine oder maximal eine Umfrage beantwortet (0 oder 1). Der senkrechte Strich ist durch ein O erweitert worden, was eine idealisierte 0 darstellen soll. Es handelt sich somit um eine Art (min, max) -Notation: von 0 bis 1.
- Eine Umfrage ist von genau einem Kunden beantwortet worden. Es ist ein zweiter senkrechter Strich hinzugefügt worden. Auch hier kann man eine Art (min, max) -Notation erkennen: von 1 bis 1.

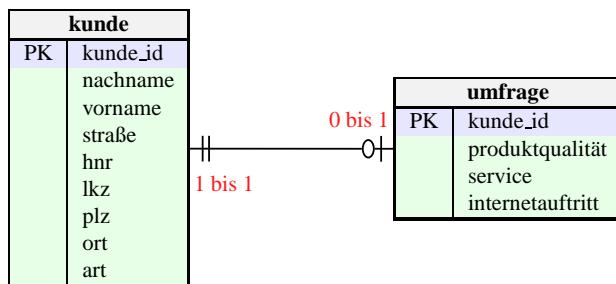


Bild 2.12 Erweiterte 1:1-Darstellung in Krähenfuß-Notation

Wann verwendet man die einfache und wann die erweiterte Notation?

Ob man die einfache oder die erweiterte Notation wählt, hängt davon ab, wie genau die Analyse der Daten sein muss. Das ER-Modell wird gerne in Kundengesprächen verwendet, weil die Notation intuitiv ist und schnell handschriftlich dokumentiert werden kann. Je mehr Informationen Sie im Kundengespräch aus dem Kunden herausholen können, desto besser. In diesem Fall ist die verfeinerte Notation als Ergebnisdokumentation sinnvoll.

Wollen Sie nur einen groben inhaltlichen Zusammenhang darstellen, reicht die einfache Notation aus.

2.2.3 1:n-Verknüpfung

2.2.3.1 Wann liegt eine 1:n-Verknüpfung vor?

Betrachten wir eine Zeile in der Tabelle `bestellung` in [Bild 2.9 auf Seite 21](#), so ist diese mit einem Kunden verknüpft; eine Bestellung kann nur von einem Kunden aufgegeben werden. Eine Zeile aus der Tabelle `kunde` hingegen kann zu mehreren Zeilen in `bestellung` gehören, da ein Kunde mehrere Bestellungen aufgeben kann. Somit haben wir eine *1:n-Verknüpfung* mit dem *n* auf der Seite von `bestellung`.



Definition 18: 1:n-Verknüpfung

Zwei Tabellen A und B stehen in einer *1:n-Verknüpfung*, wenn es zu jeder Zeile aus der Tabelle A beliebig viele Zeilen in der Tabelle B gibt und wenn es zu jeder Zeile aus der Tabelle B genau eine Zeile in der Tabelle A gibt.



Hinweis: Diese Formulierung lässt zu, dass es zu einem Datensatz in der Tabelle A keinen in der Tabelle B gibt, aber *nicht* umgekehrt.



Aufgabe 2.6: Kann $A = B$ sein? Mit anderen Worten: Kann eine Zeile einer Tabelle mit mehreren Zeilen der eigenen Tabelle verknüpft sein?

Eine *1:n-Verknüpfung* identifiziert man leicht anhand der Daten. Findet man den Primärschlüssel einer Tabelle als Fremdschlüssel in der anderen wieder und ist er dort nicht selbst ein Schlüssel, muss es eine *1:n-Verknüpfung* sein. Das *n* steht dann immer auf der Seite, wo der Fremdschlüssel ist.

Umgekehrt lassen sich die Tabellen genauso konstruieren. *Beispiel:* Sie haben eine Tabelle `gebäude` und eine Tabelle `raum`. Ein Gebäude hat mehrere Räume, aber ein Raum kann nur zu einem Gebäude gehören. Wir haben somit eine *1:n-Verknüpfung* mit dem *n* bei der Tabelle `raum`. Will ich nun die Tabellen konkret im System anlegen, weiß ich, dass der Primärschlüssel `gebäude_id` als Fremdschlüssel nach `raum` muss und nicht die `raum_id` als Fremdschlüssel nach `gebäude`.



Aufgabe 2.7: Stellen Sie fest, ob die Verknüpfungen in [Bild 2.9 auf Seite 21](#) die Kardinalität *1:n* haben. Notieren Sie sich, wo das *n* steht.

Wir haben in der [Definition 10 auf Seite 18](#) festgehalten, dass ein Primärschlüssel einer Tabelle Teil des Primärschlüssels einer anderen sein kann:



Definition 19: Identifizierende 1:n-Verknüpfung

Besteht der Primärschlüssel der Tabelle B unter anderem aus einem Fremdschlüssel für Tabelle A, so spricht man von einer *identifizierenden 1:n-Verknüpfung*. Ist das nicht der Fall, von einer *nicht identifizierenden 1:n-Verknüpfung*.

Bei vielen Programmen zur Rechnungsstellung wird beispielsweise die Kundennummer oft in die Rechnungsnummer mit aufgenommen. Ein anderes Beispiel ist die ISBN. Diese enthält eine weltweit einmalige Verlagsnummer, welche dann ein Fremdschlüssel auf eine Verlagstabelle wäre.

2.2.3.2 Wie kann ich eine 1:n-Verknüpfung darstellen?

Die Kardinalität zwischen kunde und bestellung ist nach [Definition 18 auf der vorherigen Seite](#) eine 1:n-Verknüpfung.

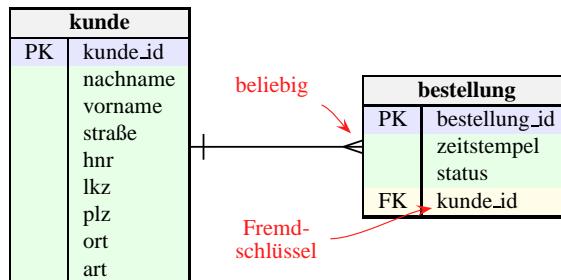


Bild 2.13 Einfache Darstellung der 1:n-Verknüpfung in Krähenfuß-Notation

In [Bild 2.13](#) ist auf der linken Seite der Tabelle bestellung ein komischer Dreizack zu sehen. Dies ist der Namensgeber der Notation, soll er doch an einen *Krähenfuß* erinnern. Er symbolisiert das n in der 1:n-Verknüpfung.

2.2.3.3 Kann man die Kardinalität genauer beschreiben?

Wie schon bei der 1:1-Darstellung kann es auch hier notwendig sein, die 1:n-Verknüpfung genauer zu bestimmen. Ein Kunde kann, muss aber nicht eine Bestellung aufgegeben haben. Daher ist die Kardinalität der Bestellung minimal 0 und maximal n . Eine Bestellung aber muss immer genau einem Kunden zugeordnet werden (siehe [Bild 2.14](#)).

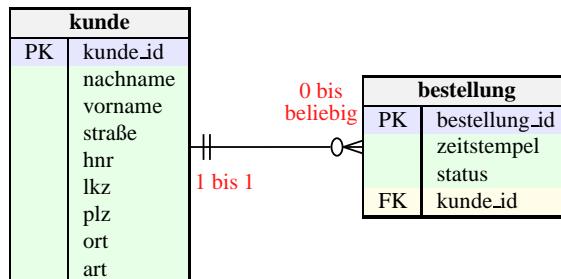


Bild 2.14 Erweiterte Darstellung der 1:n-Verknüpfung in Krähenfuß-Notation

Würde man verlangen, dass ein Kunde immer mindestens auch eine Bestellung aufgegeben haben muss¹⁴, würde anstelle der idealisierten 0 wieder ein senkrechter Strich stehen.

¹⁴ Vom Ansatz her sehr bedenklich. Plausibilitätsprüfungen sind Aufgabe der Funktionsschicht: siehe [Seite 80](#)

In manchen CASE-Tools wie der MySQL-Workbench wird zwischen identifizierender und nicht identifizierender $1:n$ -Verknüpfung (siehe [Definition 19 auf Seite 27](#)) im ER-Modell dadurch unterschieden, dass die Linie zwischen den Tabellen durchgezogen oder gestrichelt wird. Ich werde auf diese Unterscheidung verzichten, da sich der Charakter der Kardinalität aus den Angaben der Primär- und Fremdschlüssel der Tabellen ergibt.

2.2.4 $n:m$ -Verknüpfung

2.2.4.1 Wann liegt eine $n:m$ -Verknüpfung vor?

Wir beziehen die Artikel unserer Tabelle `artikel` von Lieferanten, deren Stammdaten wir in der Tabelle `lieferant` ablegen. Ein Artikel kann mehrere Lieferanten haben und ein Lieferant mehrere Artikel liefern.



Definition 20: $n:m$ -Verknüpfung

Zwei Tabellen A und B stehen in einer $n:m$ -Verknüpfung, wenn es zu jeder Zeile aus der Tabelle A beliebig viele Zeilen in der Tabelle B gibt und wenn es zu jeder Zeile aus der Tabelle B beliebig viele Zeilen in der Tabelle A gibt.



Hinweis: Diese Formulierung lässt zu, dass es zu einem Datensatz in der Tabelle A keinen in der Tabelle B gibt und umgekehrt.



Aufgabe 2.8: Kann $A = B$ sein? Mit anderen Worten: Kann eine Zeile einer Tabelle mit mehreren Zeilen der eigenen Tabelle verknüpft sein und umgekehrt? Finden Sie ein Beispiel.

Wie baut man eine solche Verknüpfung? In [Bild 2.15](#) ist die Implementierung einer $n:m$ -Verknüpfung zu sehen. Auf den ersten Blick sicherlich verwirrend, aber eben nur auf den ersten.

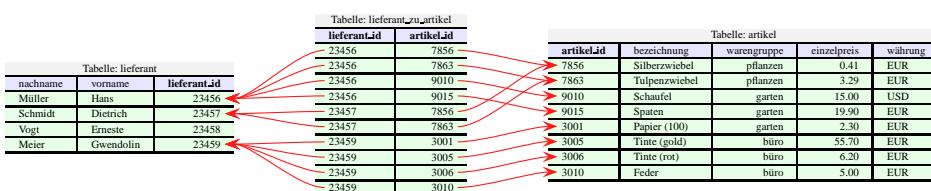


Bild 2.15 Aufbau einer $n:m$ -Beziehung

Fangen wir mit der linken Seite an. *Ein Lieferant kann beliebig viele Artikel liefern*: Dies muss nun anhand der Daten überprüft werden. Der Lieferant Müller beispielsweise kann alle möglichen Gartenartikel liefern, Schmidt nur Zwiebeln und Meier das Büromaterial. Der Vogt hingegen liefert nichts, was nicht weiter schlimm ist; vielleicht hat er mal was geliefert und wird es in Zukunft wieder tun.

Nun die rechte Seite. Ein Artikel kann von beliebig vielen Lieferanten geliefert werden: Die Zwiebeln werden von zwei Lieferanten angeboten; Schaufel und Spaten nur von einem, ebenso das Büromaterial.

Beide Seiten der [Definition 20 auf der vorherigen Seite](#) treffen somit zu, und deshalb ist dies eine Implementierung der *n:m*-Verknüpfung.



Definition 21: Hilfstabelle

Eine *n:m*-Verknüpfung zwischen der Tabelle A und der Tabelle B wird durch eine *Hilfstabelle* realisiert. Die Hilfstabelle enthält die beiden Primärschlüssel der Tabellen A und B als Fremdschlüssel und hat somit immer mindestens zwei Spalten.

Es stellt sich aber die Frage, ob das mit der Hilfstabelle `lieferant_zu_artikel` nicht irgendwie umständlich ist, ob es nicht einen einfacheren Weg gibt. Die Antwort ist ein beherztes *Nein*.

Man könnte die *n:m*-Verknüpfung auch dadurch erreichen, indem in den Spalten nicht nur *einzelne* Werte abgespeichert werden können. Besonders die Darstellung von Listen (Wiederholungsgruppen) bietet sich hier an. In der Tabelle `artikel` gäbe es eine Spalte `lieferant_id`, und diese würde alle Primärschlüsselwerte der entsprechenden Lieferanten enthalten. Analoges würde in der Tabelle `lieferant` passieren. Die damit verbundenen Nachteile sind aber gravierend.

Nimmt beispielsweise ein Lieferant einen Artikel aus seinem Angebot, so müsste man komplizierte Dinge tun: Die Liste in `lieferant_id` in `artikel` müsste in seine einzelnen Werte aufgeteilt werden. Anschließend muss dieser Lieferant aus der Liste gelöscht und die Liste wieder zusammenmontiert zurückgeschrieben werden. Analoges müsste mit der Liste `artikel_id` in `lieferant` geschehen¹⁵.

Bei unserer Lösung mit der Hilfstabelle muss man nur die entsprechende Zeile mit den passenden `lieferant_id` und `artikel_id` löschen. Eine solche Operation ist hinsichtlich der Performance ungleich billiger¹⁶.



Hinweis: Man kann auch mehr als zwei Tabellen auf diese Art und Weise miteinander verbinden. Man spricht dann beispielsweise bei drei Tabellen von einer *n:m:k-Verknüpfung*. Die Hilfstabelle besteht ebenfalls aus den entsprechenden drei Fremdschlüsseln.

2.2.4.2 Wie kann ich eine *n:m*-Verknüpfung darstellen?

Wir haben eigentlich schon alle Elemente der Darstellung beisammen, um die *n:m*-Verknüpfung zu zeichnen. [Bild 2.16 auf der nächsten Seite](#) sollte daher nichts Überraschendes enthalten.

¹⁵ Objektorientierte Datenbanken oder auch schon objektrelationale Datenbanken gehen genauso vor. Dies muss technisch erheblich unterstützt werden, damit die Zugriffe einigermaßen performant sind.

¹⁶ Um ehrlich zu sein, nur bei den Operationen `INSERT` und `DELETE`. Auswertungen durch `SELECT` können sehr wohl durch die Hilfstabelle verlangsamt werden.

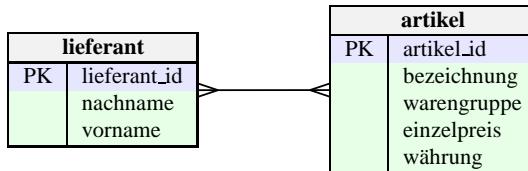


Bild 2.16 Einfache Darstellung der $n:m$ -Verknüpfung in Krähenfuß-Notation

Tatsächlich überraschend ist aber, dass die Hilfstabelle nicht mit eingezeichnet wird. Dies ist aus Sicht des Datenbankdesigners auch nicht erforderlich. Jedem Datenbankprogrammierer ist klar, dass er eine solche Kardinalität mit einer Hilfstabelle realisieren muss.



Aufgabe 2.9: Wie muss das ER-Modell aussehen, wenn man die Hilfstabelle mit einzeichnet? Probieren Sie es aus und überlegen Sie gut, bevor Sie die Kardinalitäten zur Hilfstabelle festlegen.

2.2.4.3 Kann man die Kardinalität genauer beschreiben?

Aber auch hier kann das Mengenverhältnis genauer beschrieben werden. Ein Artikel muss von mindestens einem Lieferanten bezogen werden, und ein Lieferant kann aber auch keinen Artikel liefern. Wie das? Nun, vielleicht habe ich mal bei ihm bestellt, aber im Moment ist er zu teuer.

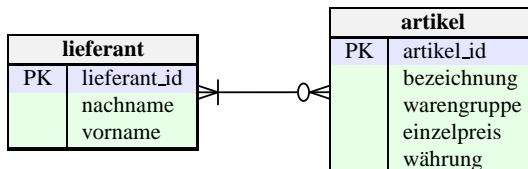


Bild 2.17 Erweiterte Darstellung der $n:m$ -Verknüpfung in Krähenfuß-Notation

2.2.5 Aufgaben zum ER-Modell



Aufgabe 2.10: Erstellen Sie ein ER-Modell für folgende Anforderungen: Für eine Bücherei sollen Kunden, Bücher und ein Verleihvorgang in einer Datenbank abgelegt werden. Zum Kunden sollen die üblichen Stammdaten wie Name etc. erfasst werden. Der Verleihvorgang besteht aus den Informationen: Wer hat was von wann bis wann ausgeliehen?

Aufgabe 2.11: Eine Clown-Agentur beauftragt Sie mit der Erstellung einer Clown-Datenbank. Diese soll Informationen über Clowns, ihre Verträge etc. enthalten. Erstellen Sie ein ER-Modell für die Tabellen *clown*, *vertrag*, *veranstalter* und *programm*. In *clown* sind die Basisdaten des Künstlers erfasst, *veranstalter* analog. In *vertrag* werden Ort und Termin eines Auftritts festgelegt. Jeder Clown bietet mehrere Programme

an. Bei einigen Programmen arbeiten mehrere Clowns zusammen. Es kann im Vertrag festgelegt werden, welche der angebotenen Programme bei der Veranstaltung vorgeführt werden sollen!

Aufgabe 2.12: Für den deutschen Brauereiverband soll eine Datenbank entwickelt werden, die über alle Biere Auskunft geben kann. Erstellen Sie ein ER-Modell für die Tabellen *bier*, *brauerei*, *großhändler* und *brauart*. Großhändler führen mehrere Biere im Sortiment!

Aufgabe 2.13: Sie sollen Auswertungen in einer CD-Sammlung programmieren. Gehören Sie von folgendem Sachverhalt aus: Es gibt die Tabellen: *cd*, *titel* und *interpret*. Erstellen Sie ein sinnvolles ER-Modell.

Aufgabe 2.14: Für eine Stundenplan-Software sollen folgende Informationen erfasst werden: *lehrer*, *klasse*, *fach* und *raum*. Für eine einzelne Unterrichtsstunde sollen die Daten *schuljahr*, *halbjahr*, *wochentag*, *uhrzeit* und *dauer* verwaltet werden. Zusätzlich soll die *Klassenlehrerschaft* (Klassenlehrer und Stellvertreter) abgebildet werden.

■ 2.3 Referenzielle Integrität



Der Daten-GAU kommt ganz harmlos daher. Verletzte referenzielle Integrität wird in den Symptomen und Ursachen dargestellt. Strategien zur Vermeidung werden aufgezeigt und diskutiert.

- Grundkurs
 - Wie Sie eine verletzte referenzielle Integrität erkennen können
 - Wie die Verletzung entstehen kann
 - Vorteil einer laufenden Nummer als Primärschlüssel
- Vertiefendes
 - Löschweitergabe
 - Löschkennzeichen
 - Änderungsweitergabe
 - Duplizieren

Tabellen werden über Primär-Fremdschlüsselpaare miteinander verknüpft. Wenn man eine Datenbank als ein fein verknüpftes Netzwerk von Tabellenzeilen versteht, sind die Primär-Fremdschlüsselpaare die Knoten. Und was passiert, wenn diese Knoten sich lösen oder die Enden falsch verknüpft werden? Betrachten Sie das Beispiel in [Bild 2.18 auf der nächsten Seite](#) und tun wir mal so, als ob dies alle Zeilen der beiden Tabellen wären.



Aufgabe 2.15: Verbinden Sie alle Primär-Fremdschlüsselpaare mit einer Linie. Was fällt Ihnen auf?

Tabelle: bestellung			
bestellung.id	[...]	kunde.id	
1	[...]	12345	
2	[...]	12345	
3	[...]	12347	
4	[...]	12346	

Tabelle: kunde			
kunde.id	nachname	vorname	[...]
12345	Gamdschie	Samweis	[...]
12346	Beutlin	Frodo	[...]

Bild 2.18 Vollständiger Datenauszug von kunde und bestellung**Definition 22: Referenzielle Integrität**

Wenn es zu jedem Fremdschlüsselwert einen passenden Primärschlüsselwert gibt, ist die *Verknüpfung/Referenz integer*. Ist diese Bedingung nicht erfüllt, spricht man von einer *verletzten referenziellen Integrität*.

Ist die Integrität einmal kaputt, hat man ein echtes Problem :-(. In der Regel ist es nur mit sehr kostenintensiven manuellen Analysen möglich, die verbogenen Referenzen wieder zurechtzubiegen. Also: Vorsicht bei Daten verändernden Operationen, die direkt oder indirekt den Primärschlüssel betreffen.

2.3.1 Verletzung der referenziellen Integrität durch Löschen

Der Kunde mit der Kundennummer 12347 ist in der Tabelle kunde gelöscht worden. Die Referenz war also mal vorhanden und fehlt nun.

In vielen Systemen kann man das DBMS anweisen, alle Zeilen, die mit einer zu löschen Zeile verknüpft sind, ebenfalls zu löschen (siehe [Tabelle 5.5 auf Seite 88](#)). Man spricht hier von *kaskadierendem Löschen* oder *Löschweitergabe*.

Dieses Feature ist mit größter Vorsicht zu genießen: Sie löschen den Kunden. Dadurch werden alle Bestellungen des Kunden gelöscht. Dadurch werden alle Rechnungen zu den Bestellungen gelöscht. Dadurch werden alle Buchungen zu den Rechnungen gelöscht :-().

Das Löschen von Datensätzen ist bei nicht trivialen Systemen mit das Schwierigste, was man sich vorstellen kann. Aus diesen Gründen wird in vielen Programmierrichtlinien das Verwenden der Löschweitergabe untersagt.

Oftmals wird überhaupt nicht gelöscht! Man fügt den Tabellen standardmäßig eine Spalte mit dem Namen *aktiv* oder *deleted* hinzu. Diese Spalte dient als Löschkennzeichen. Will man einen Datensatz löschen, wird dieses Kennzeichen beispielsweise auf 1 gesetzt. Bei allen weiteren Aktivitäten wird immer darauf geachtet, dass man nur Zeilen mit einem Löschkennzeichen 0 verarbeitet¹⁷.

¹⁷ Ob dabei datenschutzrechtliche Bestimmungen wie das informelle Selbstbestimmungsrecht oder die DSGVO berührt werden, muss von Ihnen überprüft werden! Eine nähere Betrachtung der Löschprobleme, die durch die DSGVO ausgelöst werden, kann hier nicht erfolgen.

2.3.2 Verletzung der referenziellen Integrität durch Änderungen

Der Kunde mit der Kundennummer 12347 hatte vorher die Nummer 12346. Der Primärschlüsselwert hat sich also geändert.

Wie bei der Löschweitergabe kann man bei den meisten DBMSen (siehe [Tabelle 5.5 auf Seite 88](#)) eine *Änderungswertewtergabe* oder *kaskadierende Änderung* verwenden.

Die Folgen sind zwar in der Regel nicht ganz so katastrophal wie bei der Löschweitergabe, können aber auch schon gehörigen Schaden anrichten: Die Kundennummer hat sich geändert. In allen archivierten Schriftverkehren steht aber die alte Kundennummer. Ein Anruf seitens des Kunden könnten dann echte Verwirrung auslösen, weil auf seiner Rechnung eine Kundennummer steht, die es im System gar nicht mehr gibt, oder noch viel schlimmer: nun einem anderen Kunden gehört. Gleiches gilt für elektronisch signierte Archive (z.B. für die Steuer).

Man vermeidet das Problem, indem man *nichtinformationstragende* Primärschlüssel verwendet. Laufende Nummern sind deshalb sehr gute Primärschlüssel. Schlüssel, die sich aus informationstragenden Spalten zusammensetzen, laufen immer Gefahr, dass sie sich ändern müssen (z.B. geändertes Kennzeichen bei einem PKW).

Lässt sich das nicht vermeiden, dupliziert man die Zeile und ändert den Primärschlüsselwert in der neuen Zeile; die alte bleibt unverändert und wird *versiegelt*. Für alle zeitlich nachfolgenden Ereignisse wird die neue Zeile mit dem geänderten Primärschlüsselwert verwendet. Die alten Referenzen bleiben dabei erhalten und zeigen auf die nun veraltete, aber intakte Zeile¹⁸.

■ 2.4 Normalformen



Ein schöner Garten entsteht nicht durch Wildwuchs: Designregeln für Datenbanken.

- Grundkurs
 - Wiederholungsgruppen und Atomarität
 - Die ersten drei Normalformen
- Vertiefendes
 - Unterschied zwischen teilfunktionalen und transitiven Informationen
 - Erkennen, wann Normalformen sinnvoll sind und wann nicht

Der inhaltliche Aufbau von Tabellen erfolgt oft sehr intuitiv. Dies hat zur Folge, dass viele Datenbanken im Laufe der Zeit sehr langsam im Zugriff werden und die Konsistenz der Daten abnimmt. Ebenso kommt es oft vor, dass Daten redundant sind und somit an allen Stellen gepflegt werden müssen. Man sucht Wege, Datenbanken grundsätzlich so zu gestalten, dass diese Probleme erst gar nicht entstehen. Das Ergebnis ist die Formulierung

¹⁸ Hier haben wir ein gutes Beispiel dafür, dass Redundanzen sinnvoll sein können.

von Normalformen. Die Anpassung einer bestehenden Datenbank an die Normalformen nennt man Normalisierung.

2.4.1 Normalform 1

In einer Tabelle soll der Warenkorb eines Shop-Besuchs abgelegt werden. Jeder Warenkorb wird durch seinen Primärschlüssel identifiziert. Wir gehen hier davon aus, dass der Warenkorb nur von angemeldeten Kunden gefüllt wird. Im Wesentlichen wird hier die Information gespeichert, welcher Artikel wie oft im Warenkorb abgelegt wurde. Das Ergebnis erster Überlegungen kann in [Bild 2.19](#) bewundert werden.

Tabelle: warenkorb				Tabelle: artikel			
warenkorb.id	artikel	kunde_id		artikel_id	bezeichnung	warengruppe	[...]
1	7856 30;7863 50;9015 1	12345		7856	Silberzwiebel	pflanzen	[...]
2	3006 1;3010 4	12346		7863	Tulpenzwiebel	pflanzen	[...]
				9010	Schaufel	garten	[...]
				9015	Spaten	garten	[...]
				3001	Papier (100)	büro	[...]
				3005	Tinte (gold)	büro	[...]
				3006	Tinte (rot)	büro	[...]
				3010	Feder	büro	[...]

Bild 2.19 Tabelle warenkorb vor der Normalisierung



Aufgabe 2.16: Betrachten Sie die Inhalte der Tabelle warenkorb und diskutieren Sie die möglichen Nachteile. Gibt es auch Vorteile?

Die Werte der Spalte `artikel` in der Tabelle `warenkorb` sind im Grunde Listen, und die Listenelemente bestehen aus zwei Informationen: Artikelnummer und Anzahl. Beides will man vermeiden; die Listen sollen aufgelöst und die Teilinformationen in jeweils eigene Spalten überführt werden.

Fangen wir mit den Listen an. Der Fachbegriff für solche Listen ist *Wiederholungsgruppe*.



Definition 23: Wiederholungsgruppe

Eine *Wiederholungsgruppe* ist eine Liste von Informationen desselben inhaltlichen Typs in einer Spalte.

Wir haben eine Wiederholungsgruppe mit drei Elementen; sie werden durch ein Semikolon getrennt. Die Wiederholungsgruppe enthält die Informationen zu einer Bestellposition: Artikelnummer und Anzahl.



Definition 24: Wiederholungsgruppenfreiheit

Eine Tabelle ist dann *wiederholungsgruppenfrei*, wenn alle ihre Spaltenwerte keine Wiederholungsgruppen enthalten.

Um die Wiederholungsgruppenfreiheit herzustellen, brauchen wir eine neue Tabelle, die ähnlich der Tabelle `position` die Artikel aufnimmt (siehe [Bild 2.20 auf der nächsten Seite](#)).

Tabelle: warenkorb

warenkorb_id	kunde_id
1	12345
2	12346

Tabelle: korbposition

warenkorb_id	positionsnr	artikel
1	1	7856 30
1	2	7863 50
1	3	9015 1
2	1	3006 1
2	2	3010 4

Tabelle: artikel

artikel_id	bezeichnung	warengruppe	[...]
7856	Silberzwiebel	pflanzen	[...]
7863	Tulpenzwiebel	pflanzen	[...]
9010	Schaufel	garten	[...]
9015	Spaten	garten	[...]
3001	Papier (100)	büro	[...]
3005	Tinte (gold)	büro	[...]
3006	Tinte (rot)	büro	[...]
3010	Feder	büro	[...]

Bild 2.20 Tabelle warenkorb ohne Wiederholungsgruppen

Was sofort störend ins Auge fällt, ist die Spalte **artikel** in der Tabelle **korbposition**. Die Artikelnummer und die Anzahl sind in der gleichen Spalte abgelegt, was eine Auswertung und Veränderung der Daten erheblich erschwert.

**Definition 25: Atomar**

Ein Spaltenwert ist **atomar**, wenn er nicht mehr in Teileinformationen zerlegt werden kann, ohne seinen Sinn zu verlieren. Eine Tabelle ist **atomar**, wenn alle ihre Spaltenwerte atomar sind. Eine Datenbank ist **atomar**, wenn alle ihre Tabellen atomar sind.

Wir erreichen die Eigenschaft **atomar**, indem wir die Spalte **artikel** in die Spalten **artikel_id** und **menge** aufteilen (siehe Bild 2.21). Jetzt können SQL-Operationen problemlos jede Information einzeln auswerten und verändern.

Tabelle: warenkorb

warenkorb_id	kunde_id
1	12345
2	12346

Tabelle: korbposition

warenkorb_id	positionsnr	menge	artikel
1	1	30	7856
1	2	50	7863
1	3	1	9015
2	1	1	3006
2	2	4	3010

Tabelle: artikel

artikel_id	bezeichnung	warengruppe	[...]
7856	Silberzwiebel	pflanzen	[...]
7863	Tulpenzwiebel	pflanzen	[...]
9010	Schaufel	garten	[...]
9015	Spaten	garten	[...]
3001	Papier (100)	büro	[...]
3005	Tinte (gold)	büro	[...]
3006	Tinte (rot)	büro	[...]
3010	Feder	büro	[...]

Bild 2.21 Tabelle warenkorb in der 1. Normalform**Definition 26: Erste Normalform**

Eine Tabelle ist dann in der **1. Normalform**, wenn sie atomar und wiederholungsgruppenfrei ist. Eine Datenbank ist dann in der **1. Normalform**, wenn alle Tabellen in der 1. Normalform sind.

Die Werte dürfen nicht zufällig atomar oder wiederholungsgruppenfrei sein. Wären beispielsweise alle Warenkörbe mit nur einem einzigen Artikel gefüllt, liegt trotzdem keine Wiederholungsgruppenfreiheit vor!



Aufgabe 2.17: Überführen Sie [Bild 2.21 auf der vorherigen Seite](#) in ein ER-Modell in Krähenfuß-Notation.

Tipp: Es kommen nur $1:n$ -Verknüpfungen vor.

Das Auflösen der Wiederholungsgruppen und das Aufteilen der nicht atomaren Spalten nennt man *Normalisierung*:

1. Legen Sie für *jede* Spalte mit einer Wiederholungsgruppe in Tabelle A eine neue Tabelle an. Diese enthält als Fremdschlüssel den Primärschlüssel von Tabelle A.
2. Legen Sie für *jede* nicht atomare Spalte so viele neue Spalten an, dass Sie jede Teilinformation dort ablegen können.

Warum sollte man überhaupt auf die Idee kommen, die Daten wie in [Bild 2.19 auf Seite 35](#) abzuspeichern? Nun, so abwegig ist das auch nicht. Die Daten des Warenkorbs werden selbst noch nicht weiter verarbeitet. Erst, wenn der Kunde den Inhalt des Warenkorbs bestellt, werden diese Daten in eine Bestellung umgewandelt. Vorher sind dies vorläufige Daten, die jederzeit verworfen werden können.

Die Aufbereitung und Darstellung der Daten wird durch Skriptsprachen wie PHP oder JavaScript geleistet. Diese können die Daten des Warenkorbs leicht dekodieren und ggf. wieder zusammengepackt an den Datenbankserver zurückschicken, da für den einzelnen Vorgang nur geringe Datenmengen verarbeitet werden müssen. Meist werden die Daten dazu im JSON-Format bereitgestellt. Mehr dazu in [Abschnitt 17 auf Seite 289](#).

2.4.2 Normalform 2

Betrachten Sie die Inhalte der Tabelle *artikel* in [Bild 2.21 auf der vorherigen Seite](#).



Aufgabe 2.18: Welche Bedeutung hat vermutlich die vorangestellte Ziffer des Primärschlüssels der Tabelle *artikel*? Entspricht diese Spalte der 1. Normalform (siehe [Definition 26 auf der vorherigen Seite](#))?



Definition 27: Voll- und teilfunktional

Wenn die Werte einer Spalte nur von einem Teil des Primärschlüssels abhängen, ist diese Spalte *teilfunktional* vom Primärschlüssel abhängig. Ist dies nicht der Fall, ist sie *vollfunktional* vom Primärschlüssel abhängig.

Teilfunktionalität tritt nur bei zusammengesetzten Schlüsseln auf (siehe [Definition 8 auf Seite 17](#)). Laufende Zähler sind – außer bei *bösartig* konstruierten Gegenbeispielen – davon nicht betroffen. Vermeiden Sie daher informationstragende Schlüssel und verwenden Sie laufende Zähler.

**Definition 28: Zweite Normalform**

Eine Tabelle ist dann in der *2. Normalform*, wenn sie den Bedingungen der 1. Normalform entspricht und alle Nichtschlüsselpalten vollfunktional vom Primärschlüssel abhängig sind.

Eine Datenbank ist dann in der *2. Normalform*, wenn alle Tabellen in der 2. Normalform sind. ■

Weitere bekannte Beispiele für einen zusammengesetzten Primärschlüssel und eine damit ggf. verbundene teilfunktionale Abhängigkeit:

- Eine Tabelle *buch* mit dem Primärschlüssel *isbn* und einer Spalte *verlagsnamen*. Der Name des Verlags ergibt sich aus einer der Zifferngruppen der ISBN (siehe [Wik19j]).
- Die Telefonnummer setzt sich aus mehreren unterschiedlichen Teilnummern zusammen, wobei zwischen Festnetz- und Mobilfunknummern unterschieden wird. Kommt jetzt in der Tabelle noch eine Spalte *ort* bzw. *anbieter* vor, wäre diese Spalte nicht vom gesamten Schlüssel, sondern nur von einem Teil abhängig (siehe [Wik19n]).
- Die IBAN ist ebenfalls eine Komposition aus verschiedenen Teilinformationen. Sie enthält neben anderen Informationen auch die Bankleitzahl. Kommt nun in der Tabelle mit dem Primärschlüssel *iban* die Spalte *bankname* vor, wäre diese nicht vom gesamten Primärschlüssel, sondern nur von einem Teil der IBAN abhängig (siehe [Wik19i]).
- Die Kennzeichnung von Eiern setzt sich aus den Informationen *Haltungsform*, *Herkunftsland* und *Betriebsnummer* zusammen. In einer Tabelle mit dem Primärschlüssel *erzeugercode* wäre die Spalte *herkunftsland* somit teilfunktional (siehe [Wik19e]).



Aufgabe 2.19: Überlegen Sie anhand der oben genannten Beispiele, was der Unterschied zwischen informationstragenden und nicht informationstragenden Primärschlüsseln ist. Überlegen Sie sich auch, welche Vor- und Nachteile solche zusammengesetzten PKs im Gegensatz zu laufenden Nummern als PKs haben. ■

2.4.3 Normalform 3

Wir wollen es ermöglichen, dass zu jedem Kunden mehrere Bankverbindungen existieren können. In Bild 2.22 sehen Sie einen ersten Entwurf der Tabelle *bankverbindung*.

Tabelle: bankverbindung					
kunde_id	bankverbindung_nr	blz	kontonr	bankname	iban
12345	1	50041597	1234506789	Sparkasse Aulenland	DEXX500415971234506789
12345	2	50287667	5432109876	Volksbank Eriador	DEXX502876675432109876
12346	1	50287667	5432109880	Volksbank Eriador	DEXX502876675432109890

Bild 2.22 Tabelle *bankverbindung* mit transitiven Informationen



Aufgabe 2.20: Ermitteln Sie den Zusammenhang zwischen Bankleitzahl (blz) und dem Banknamen sowie ebenso zwischen Bankleitzahl, Kontonummer und IBAN. ■

Wenn Nichtschlüssel Spalten aus anderen Nichtschlüssel Spalten herleitbar sind, bedeutet dies in der Regel, dass Informationen redundant in der Tabelle gehalten werden. Hier werden beispielsweise die Banknamen mehrfach genannt. Dies verbraucht nicht nur Speicherplatz, sondern macht eine Änderung der Banknamen teuer, da diese in vielen Zeilen durchgeführt werden müssen.



Definition 29: Transitiv

Eine Nichtschlüssel Spalte ist *transitiv*, wenn sie sich aus anderen Nichtschlüssel Spalten herleiten lässt.

Transitive Informationen begegnen Ihnen relativ oft. Der Kontoinhaber ergibt sich aus der Kontonummer, der Ortsname aus der Postleitzahl, der Rabatt aus der Kundenart etc.



Definition 30: Dritte Normalform

Eine Tabelle entspricht der 3. *Normalform*, wenn sie den Bedingungen der 2. Normalform entspricht und die Werte in den Spalten nicht transitiv sind.

Eine Datenbank ist dann in der 3. *Normalform*, wenn alle Tabellen der 3. Normalform entsprechen und auch zwischen den Tabellen keine Information transitiv ist.



Aufgabe 2.21: Erstellen Sie zu Bild 2.23 ein ER-Modell in Krähenfuß-Notation.

Tabelle: bankverbindung					Tabelle: bank		
kunde_id	bankverbindung_nr	kontonr	iban	blz	blz	bankname	lkz
12345	1	1234506789	[...]	50041597	50041597	Sparkasse Aulenland	DE
12345	2	5432109876	[...]	50287667	50287667	Volksbank Eriador	DE
12346	1	5432109880	[...]	50287667			

Bild 2.23 Tabelle bankverbindung und bank ohne transitive Informationen



Hinweis: Nicht nur der Bankname ist eine transitive Information. Auch die Spalte *iban* setzt sich größtenteils aus anderen Spalteninhalten zusammen. Neu sind nur das Länderkennzeichen (DE) und die zweistellige Prüfziffer. Wird man nun diese Spalte ebenfalls aufteilen? In der Praxis wohl kaum. Zum einen, weil die IBAN im Bereich der *Kontoidentifikation* in jedem Land anders zusammengesetzt wird, und zum anderen, weil die IBAN immer als Informationseinheit verwendet wird. Eine Konsistenzprüfung – zumindest für deutsche Banken – sollte allerdings implementiert sein.

2.4.4 Normalform Rest

Es gibt noch die Boyce-Codd¹⁹-Normalform (BCNF), die 4. und die 5. Normalform, auf denen genaue Darstellung ich hier verzichten möchte. Die BCNF ist in diesem Buch in die

¹⁹ Derselbe Codd, der die relationale Datenbank *erfunden* hat.

3. Normalform eingeflossen. Die Unterscheidung zwischen der 3. Normalform laut Literatur²⁰ und der BCNF wird in der Praxis als akademisch empfunden. Schließlich sind beide einschränkende Vorschriften zum Thema transitive Informationen. Nichts spricht dagegen, gleich die schärfere BCNF als 3. Normalform zu verwenden.

Die 4. Normalform verlangt, dass für jede inhaltlich abgeschlossene Einheit eine neue Tabelle erstellt wird. Also beispielsweise nicht Kundendaten und Adresse in einer Tabelle, sondern in zwei. Das Ziel ist hier, die Wartungsstabilität herzustellen. Wenn sich Sachverhalte ändern oder überflüssig werden, muss nur die betroffene Tabelle geändert werden, andere bleiben davon unberührt.

Die 5. Normalform ist so was wie "*Jetzt hören wir aber auf mit dem Normalisieren*". Diese verlangt, dass Tabellen, deren Inhalte sich durch Verbundoperationen (JOIN, UNION, INTERSECT etc.) herstellen lassen, aufgelöst werden. Hier geht es wieder darum, Redundanzen (siehe [Definition 13 auf Seite 20](#)) zu vermeiden und wartungsstabiler zu werden.

Die letzten beiden Normalformen sind in der Praxis recht umstritten. Ich sollte genauer sein: Es wird nicht bestritten, dass es sinnvoll ist, sein Design bzgl. dieser Normalformen zu überprüfen. Aber gerade die Performance einer Anwendung leidet enorm, wenn diese beiden Normalformen *immer* angewendet werden.

Themen wie *temporäre Tabellen* (siehe [Kapitel 11.2.3 auf Seite 192](#)) und *Ansichten* (siehe [Kapitel 16 auf Seite 273](#)) sind ein einziger Verstoß gegen diese Normalformen.

²⁰ Siehe [KE01]

3

Unser Beispiel: Ein Online-Shop



Falls Sie auf Basis dieser Modellierung später einen erfolgreichen Shop betreiben, will ich am Umsatz beteiligt sein ;-)

Jeder Online-Shop muss die Vorgänge Kundenverwaltung, Artikelverwaltung und Bestellwesen abbilden können. Dazu wollen wir die essenziellen Tabellen ermitteln und die Spalten angeben.



Hinweis: Alle Tabellen enthalten die Spalte deleted. Diese Spalte markiert, ob eine Zeile als gelöscht markiert ist (=1) oder nicht (=0). Hintergrund ist die Erhaltung der referentiellen Integrität bei Löschoperationen (siehe Hinweis zum Löschkennzeichen auf [Seite 33](#)).

■ 3.1 Kundenverwaltung

Tragen wir zusammen, welche Tabellen wir für die Kundenverwaltung schon angesprochen haben: kunde, bankverbindung und bank.

- **kunde:** Die bisherige Tabelle kunde enthält neben dem Primärschlüssel und dem Namen auch die Adresse. Es ist aber üblich, dass man in Online-Shops neben der Rechnungsadresse auch eine Lieferadresse angeben kann. Es ist somit sinnvoll, eine Adress-tabelle anzulegen und auf diese zweimal zuzugreifen.
- **bankverbindung:** Die Bankverbindung ist nur eine mögliche Zahlungsart. Diese wird beim Einzugsverfahren verwendet. Andere Möglichkeiten sind Kreditkarte, PayPal etc. Da eine vollständige Modellierung dieser Möglichkeiten den Rahmen dieses Buchs sprengen würde, lassen wir erst mal nur den *Bankeinzug* und *per Rechnung* zu. Die bevorzugte Zahlungsart soll aber für den Kunden festgelegt werden können.
- **bank:** Diese Tabelle erfüllt ihren Zweck ganz gut und muss weiter nicht verändert werden.

Diese Anforderungen werden nun in einem ER-Modell zusammengefasst (siehe [Bild 3.1 auf der nächsten Seite](#)). Wir haben hier den interessanten Fall, dass zwei Tabellen (kunde und

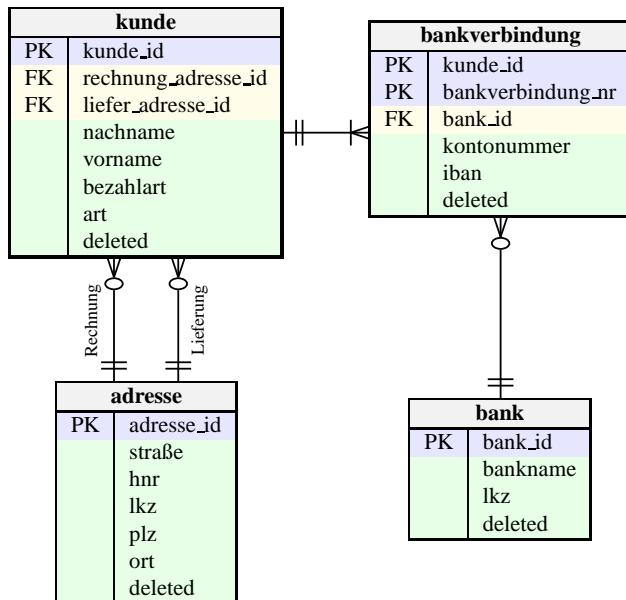


Bild 3.1 ER-Modell: Kundenverwaltung

adresse) zweimal *1:n* verknüpft sind und keine *n:m*-Verknüpfung darstellen. Um deutlich zu machen, welche Rolle die jeweilige Verknüpfung darstellt, wird auf der Verbindung der Name der Rolle notiert.

Diese Situation kommt öfter vor, als man glaubt: Nehmen Sie beispielsweise die beiden Tabellen **verein** und **spielpaarung** bei einem Fußballturnier. In **spielpaarung** kommt der Fremdschlüssel auf die Tabelle **verein** zweimal vor: als Heim- und als Gastmannschaft.

3.2 Artikelverwaltung

Auch hier tragen wir erst mal zusammen, was bisher schon bezüglich der Artikel gesagt wurde: Wir haben die Tabellen **artikel**, **warengruppe** und **lieferant**.

- **artikel**: Diese Tabelle enthält die Artikelstammdaten, d.h. einen Primärschlüssel, die Bezeichnung und den Preis.
- **warengruppe**: In der Warengruppe wird die Kategorie oder der Suchbegriff festgelegt. Somit werden lediglich eine Bezeichnung und ein Primärschlüssel gebraucht.
- **lieferant**: Der Lieferant eines Artikels besteht aus einem Firmennamen und seiner Adresse.

Da schon eine Tabelle **adresse** vorhanden ist, sollten wir für den Lieferant die Adressdaten nicht in der Tabelle **lieferant** speichern. Da ein Lieferant mindestens eine Adresse

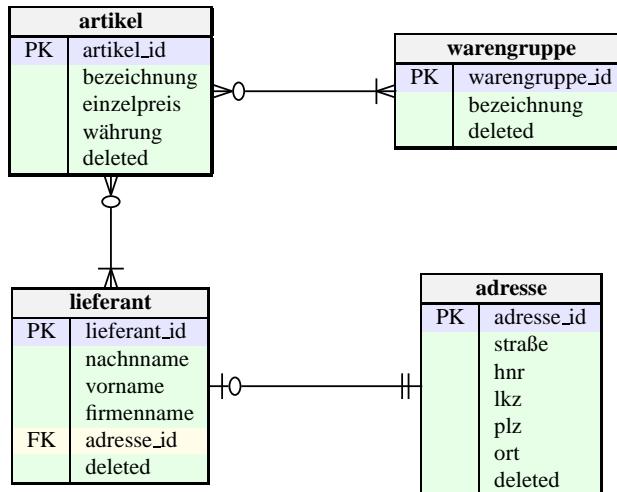


Bild 3.2 ER-Modell: Artikelverwaltung

haben muss und eine Adresse zu keinem oder einem Lieferanten gehört, handelt es sich hier um eine 1:1-Verknüpfung, die nicht wie oben auf Seite 25 beschrieben, durch einen gemeinsamen Primärschlüssel gelöst wird.

Da ein Artikel zu mindestens einer Warengruppe gehören muss, aber eine Warengruppe beliebig viele Artikel haben kann, liegt hier eine n:m-Verknüpfung vor. Analoges gilt für das Mengenverhältnis zwischen **lieferant** und **artikel** (siehe Bild 2.17 auf Seite 31).

3.3 Bestellwesen

Das Bestellwesen geht von folgendem vereinfachten Geschäftsprozess aus: Der Kunde bestätigt den Inhalt des Warenkorbs. Dadurch wird der Warenkorb in eine Bestellung umgebaut. Zu der Bestellung wird gleichzeitig eine Rechnung angelegt. Bestellung und Rechnung haben jeweils einen Bearbeitungsstatus, der darüber Auskunft gibt, ob die Bestellung versendet bzw. die Rechnung bezahlt oder storniert wurde.

- **bestellung**: Zu einer Bestellung gehört der Verweis auf den bestellenden Kunden, das Bestelldatum und den Bearbeitungsstatus der Bestellung.
- **bestellung_position**: Die Bestellung setzt sich aus den Positionen zusammen. Die Position muss angeben, welcher Artikel in welcher Menge bestellt wurde.
- **rechnung**: Rechnungen sind im Prinzip ähnlich wie Bestellungen aufgebaut. Zusätzlich wird aber die Möglichkeit eines Gesamtrabatts eingeräumt. Auch wird ein Skontofeld Auskunft darüber geben, ob Skonto gewährt wird oder nicht.
- **rechnung_position**: Die Rechnung setzt sich aus den Positionen zusammen. Die Position muss angeben, welcher Artikel in welcher Menge bestellt wurde. Ebenso soll pro Position ein Rabatt möglich sein.

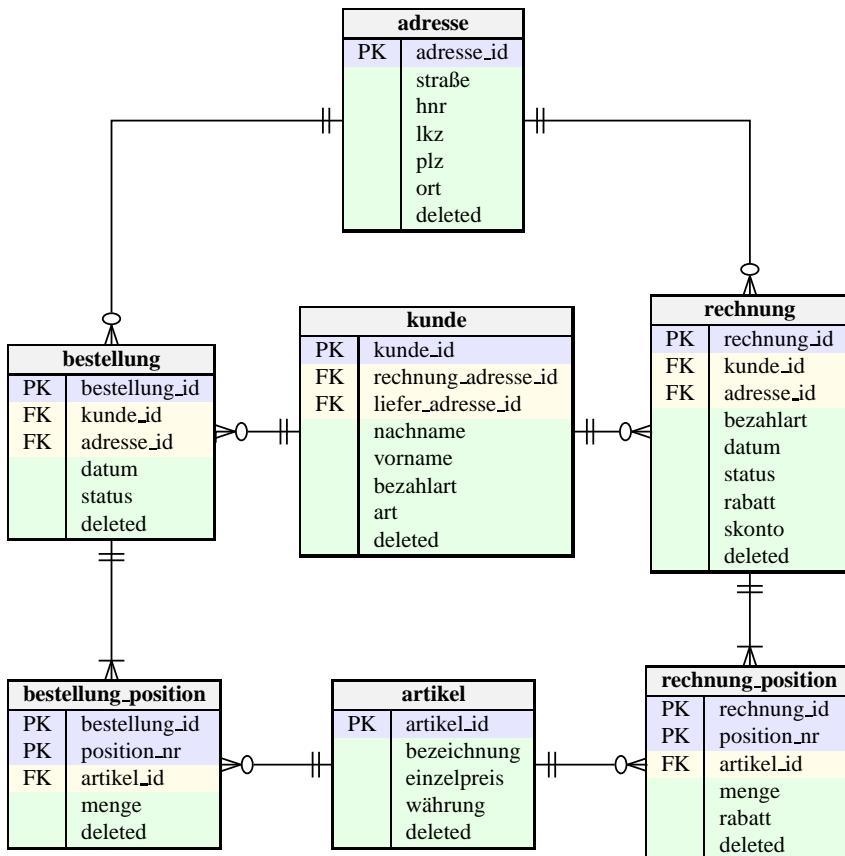


Bild 3.3 ER-Modell: Bestellwesen

Der Kunde hat in Bild 3.3 zu Rechnung und Position jeweils die Kardinalität 1:n. Dabei ist zu beachten, dass der Kunde nicht zwingend schon eine Bestellung aufgegeben oder eine Rechnung erhalten haben muss. Umgekehrt muss eine Rechnung oder Bestellung aber auf einen Kunden verweisen.

Die Tabellen rechnung und bestellung sind jeweils mit der Tabelle adresse 1:n verknüpft. Die Frage ist, ob diese Information nicht aus der Tabelle kunde ermittelt werden kann. Dort gibt es doch eine Liefer- und eine Rechnungsadresse. Dies hängt von der Implementierung ab. Der Kunde kann, muss aber nicht unterschiedliche Adressen angeben; dies soll auf der Oberfläche vom Kunden ausgewählt werden. Diese getroffene Auswahl wird als Fremdschlüsselwert in die jeweilige Tabelle geschrieben.

Jede Bestellung/Rechnung muss mindestens eine Position enthalten, und jede Position muss in einer Bestellung/Rechnung eingebettet sein. Daher die strikte 1:n-Verknüpfung.

Ein Artikel kann beliebig oft in einer Position auftauchen, aber eine Position muss genau einen Artikel enthalten.

TEIL II

Datenbank aufbauen

4

Installation des Servers

■ 4.1 MySQL unter Windows 10



Installation des MySQL Servers als Entwicklerserver

- Grundkurs
 - Installationsquelle
 - Installation unter Windows 10
- Vertiefendes
 - Verzeichnisstruktur
 - Verschiedene Aufrufparameter des SQL Clients
 - Windows-Suchpfad anpassen



Hinweis: Sie müssen Administratorenrechte haben, um den Installer ausführen zu können.

Durch einen Doppelklick auf die heruntergeladene `msi`-Datei¹ wird die Installation begonnen. Akzeptieren Sie die Lizenzbedingungen und wählen Sie den Installationsmodus (siehe [Bild 4.1 auf der nächsten Seite](#)). Da ich keine vertiefende Einführung in unterschiedliche Installationsszenarien schreiben möchte, wählen wir hier CUSTOM. Dabei werden wir folgende Komponenten auswählen:

- **MySQL Server:** Community Server

¹ Stand 22.07.2019: `mysql-installer-web-community-8.0.17.0.msi`

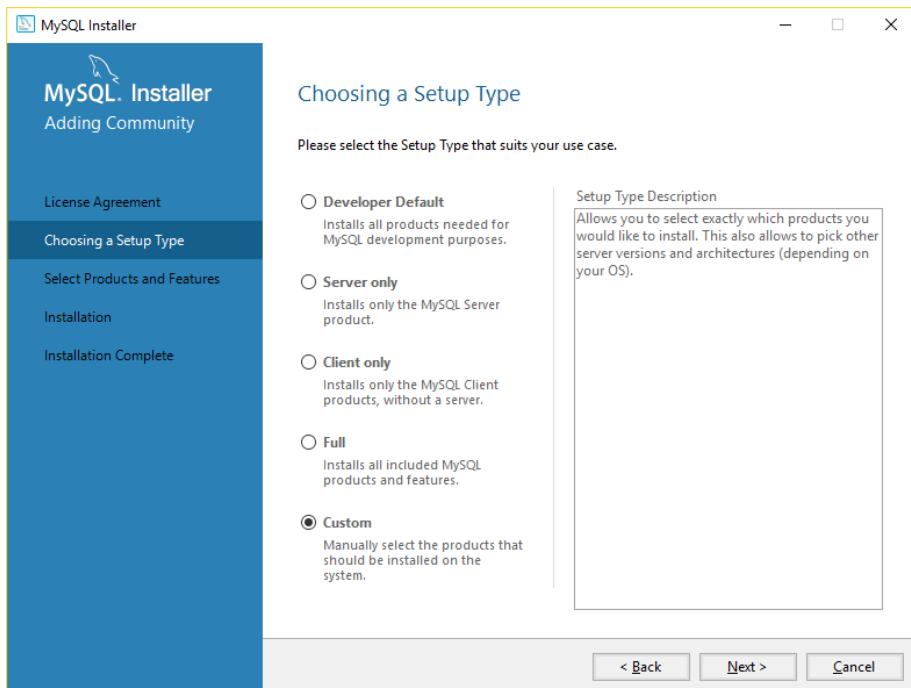


Bild 4.1 Schritt 1: Auswahl Installationsmodus

- **MySQL Workbench:** Sie ist in jedem Fall ein nützliches Werkzeug und daher zu empfehlen. Sie können damit über eine grafische Oberfläche Datenbanken planen, anlegen und bearbeiten.
- **MySQL Notifier:** Ein Monitor für den Server (in der Workbench integriert)
- **MySQL Shell:** Konsolenbasiertes Kommandofenster
- **MySQL Utilities:** Kleinere Hilfsprogramme zur Administration des Servers
- **Connectoren:** Es werden die APIs für ODBC, Java und .NET installiert. Falls Sie eine Anwendung in einer der Sprachen entwickeln und auf den MySQL Server zugreifen wollen, ist es genau das, was Sie brauchen.
- **Beispiele:** Beispieldatenbanken und Skripte
- **Dokumentation:** Handbuch und HowTos



Hinweis: Ist schon ein MySQL Server oder ein MySQL-Hilfsmittel installiert, müssen diese vorher nicht deinstalliert werden. Sie erhalten die Möglichkeit, ein Update durchzuführen.

Klicken Sie nun auf **NEXT >**. Es kann sein, dass der Installer merkt, dass bestimmte Komponenten fehlen, z.B. .NET Framework 4 Client Profile oder MS Visual C++ 2013 Redistributable (X64). Klicken Sie auf **EXECUTE** und folgen den Anweisungen. Irgendwann klicken Sie dann wieder auf **NEXT >**.

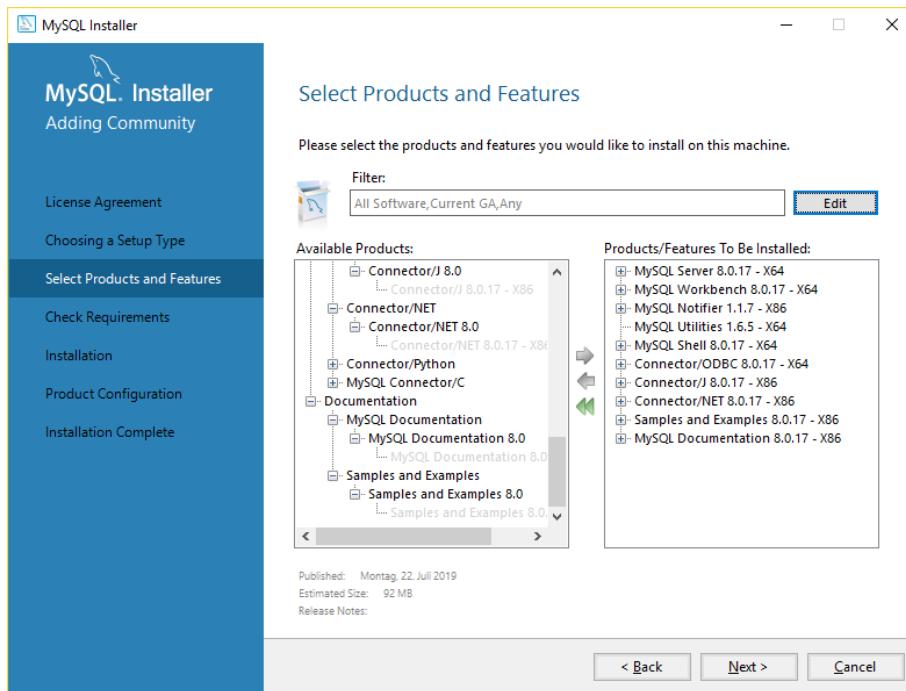


Bild 4.2 Schritt 2: Komponenten auswählen

Falls Sie jetzt merken, dass Sie diese benötigten Komponenten nicht installieren möchten, gehen Sie mit < BACK zurück und wählen Sie CUSTOM. Deselektieren Sie dann die nicht erwünschte Komponente (z.B. ein Excel Plugin). Klicken Sie auf EXECUTE und beobachten Sie dabei die Fortschrittsanzeige. Ich empfehle Ihnen, vor Ort zu bleiben, um Fehlermeldungen gleich bearbeiten zu können.

Zuerst werden die Komponenten mit TO BE DOWNLOADED heruntergeladen. Lief was schief, klicken Sie auf TRY AGAIN. Falls das nichts fruchtet (wie bei mir mit den J/Connector): einfach die Rückfrage, ob man schon mal die erfolgreich heruntergeladenen Produkte installieren möchte, mit JA quittieren.

Der Installationsassistent bietet Ihnen zwei Grundarchitekturen an (siehe [Bild 4.3 auf der nächsten Seite](#)):

- **Standalone MySQL Server:** Es wird ein Server installiert, der sowohl für eine Lernumgebung als auch für die Produktion geeignet ist.
- **InnoDB Cluster:** Hier wird eine Architektur installiert, die von Beginn an darauf eingestellt ist, mehrere (mindestens drei) Rechner in einem Replikations-Cluster zu verwalten.

Da Replikations-Cluster auf dem Level des InnoDB Clusters in diesem Buch keine Rolle spielen (leider), kann diese Auswahl ignoriert werden, also wählen wir *Standalone MySQL Server*. Jetzt können wir die ersten eigenen Angaben machen (siehe [Bild 4.4 auf Seite 51](#)).

Unter *Type and Networking* kann aus drei Verwendungsmöglichkeiten des Servers ausgewählt werden:

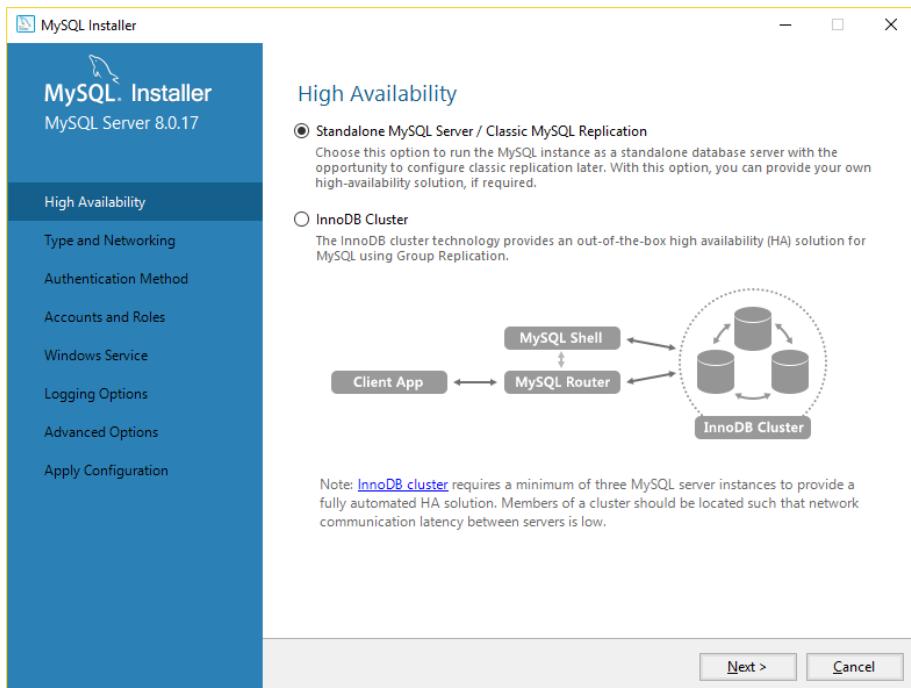


Bild 4.3 Schritt 3: Architekturauswahl

- **Development Computer:** Einstellungen wie Puffergrößen etc. werden so gewählt, dass auf dem Rechner andere Anwendungen wie z.B. eine Entwicklungsumgebung problemlos weiterlaufen können. Auch einige Sicherheitseinstellungen sind etwas *weicher* vorgenommen worden, damit Sie als Entwickler leichter auf die Komponente des Servers zugreifen können.
- **Server Computer:** Die Einstellungen werden so vorgenommen, dass auch andere Server – wie beispielsweise Apache, LDAP etc. – bequem auf dem gleichen Rechner laufen können. Allerdings sollte die Hardware tatsächlich schon auf den Serverbetrieb abgestimmt sein. Nehmen Sie diese Einstellung, wenn Sie einen kompletten Web- oder Application-Server betreiben wollen.
- **Dedicated Computer:** MySQL reißt sich alle Ressourcen unter den Nagel, welches es kriegen kann. Besonders der Arbeitsspeicher wird maximal verwendet und lässt damit keinen Raum für andere Services – außer denen des Betriebssystems.

Da wir SQL-Entwickler sind, wählen wir den *Development Computer*. Von den anderen Optionen möchte ich hier nur zwei besprechen:

- **TCP/IP:** Wenn Sie diese Option auswählen (Default), kann der Server außerhalb Ihres Rechners über eine IP-Adresse angesprochen werden. Falls Sie die Option deselektrieren, kann der Server nur über 127.0.0.1 oder localhost erreicht werden. Die vorgeschlagene Portnummer 3306 sollte nur verändert werden, wenn Sie genau wissen warum. Der Client – eine eigene Anwendung oder der MySQL Client – muss dann über die andere Portnummer informiert werden, da er sonst versucht, den Standardport zu verwenden.

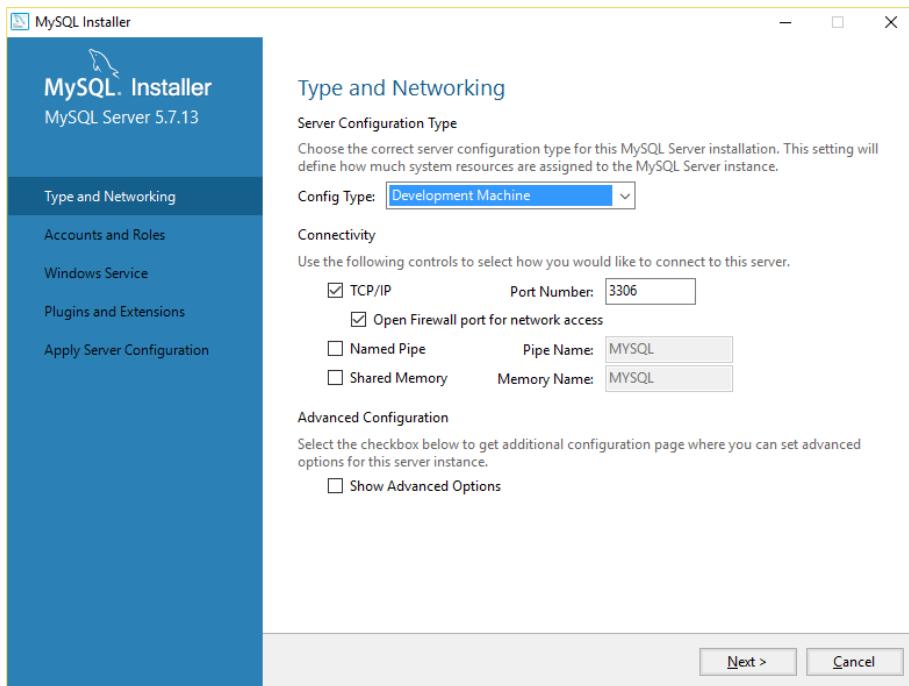


Bild 4.4 Schritt 4: Type and Networking

- **Open Firewall port for network access:** Die Firewall hält normalerweise die Ports geschlossen, um unerwünschten Besuchern das Leben etwas zu erschweren. Damit aber der Server für einen Client erreichbar ist, muss die Portnummer in der Firewall freigeschaltet werden. Lassen Sie diese Checkbox also selektiert.

Bei Schritt 5 kann die Methode ausgewählt werden, mit der das Passwort verschlüsselt wird. Hier sollten Sie die Voreinstellung nicht verändern und die neue, sicherere Methode wählen.

In Schritt 6 legen Sie das root-Passwort fest. Merken Sie sich dieses unbedingt. Mir ist kein Verfahren bekannt, ein einmal vergessenes root-Passwort wieder zurückzusetzen. Es ist daher sinnvoll, einen zweiten Administrator einzurichten, der dann das root-Passwort neu setzen kann. Das Passwort selbst sollte mindestens acht Stellen haben sowie Groß- und Kleinbuchstaben, Zahlen und Sonderzeichen enthalten. Bewährt haben sich Eselsbrücken. Beispielsweise wird *Wer 1mal lügt, dem glaubt man nicht, selbst wenn er nun die Wahrheit spricht!* zu **W1l , dgmn , swendWs ! ;-**

Danach legen Sie fest, ob der MySQL Server als Window Service gestartet werden soll. Wenn Sie diese Option auswählen (siehe [Bild 4.5 auf der nächsten Seite](#)), steht das Programm MySQL als Dienst zur Verfügung und wird beim Start von Windows automatisch angeworfen. Jeder MySQL-Service muss einen eigenen Namen haben.

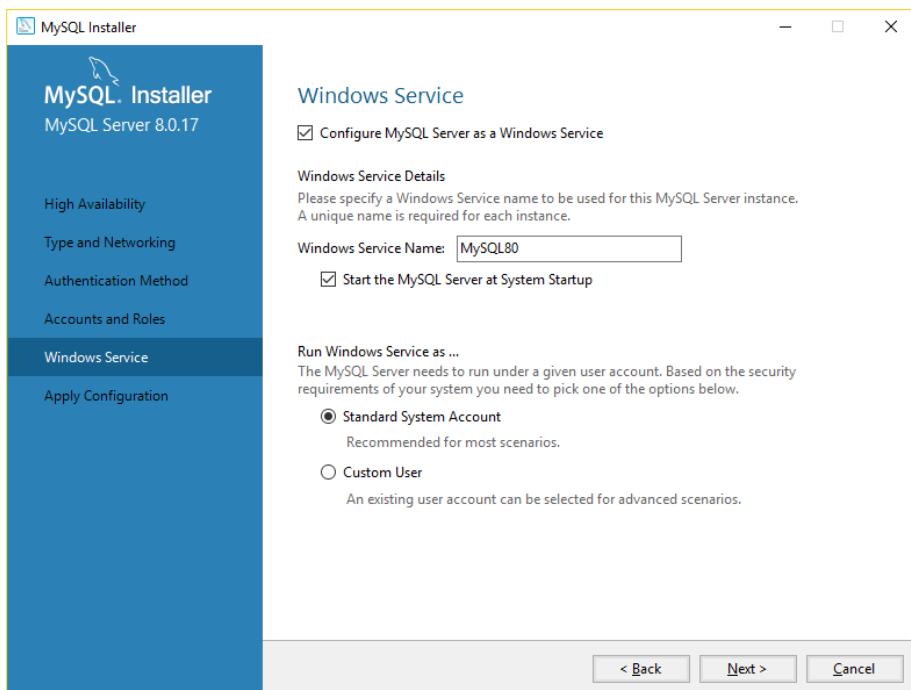


Bild 4.5 Schritt 7: Windows Service

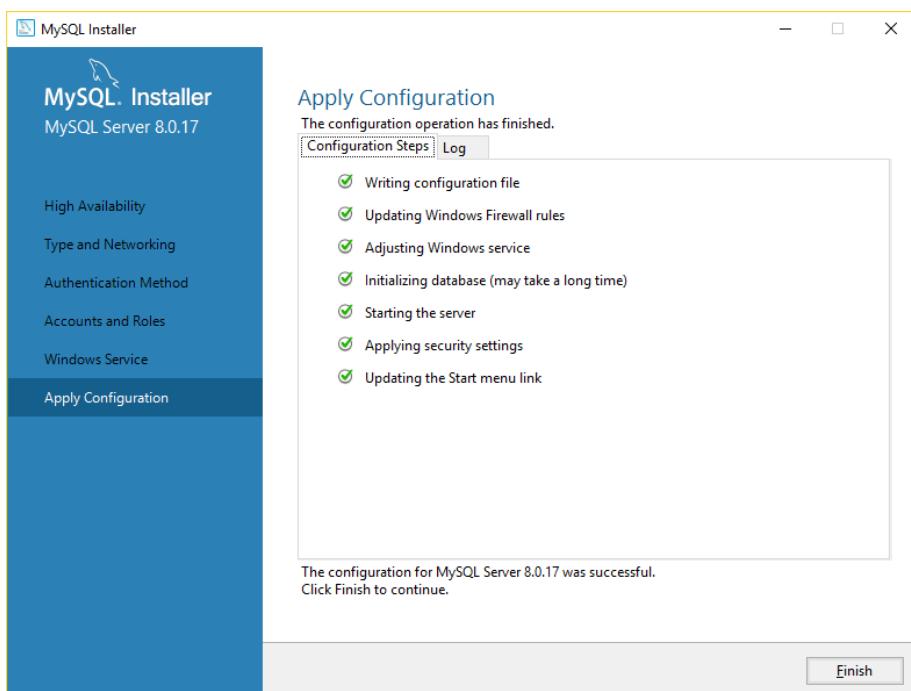


Bild 4.6 Schritt 8: Fertig

Wenn Sie wollen, können Sie auch den Service manuell starten und nicht automatisch beim Systemstart. Dann sollten Sie die Checkbox *Start the MySQL Server at System Start-up* deselektrieren.

Klicken Sie auf EXECUTE, und es sollte der Server (siehe [Bild 4.6 auf der vorherigen Seite](#)) gestartet werden. Mithilfe der Workbench oder des MySQL Clients kann nun überprüft werden, ob der Server läuft.

```
1 C:\>"c:\Program Files\MySQL\MySQL Server 8.0\bin\mysql" -uroot -p
2 Enter password: *****
3 Welcome to the MySQL monitor. Commands end with ; or \g.
4 Your MySQL connection id is 14
5 Server version: 8.0.17 MySQL Community Server - GPL
6
7 Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.
8
9 Oracle is a registered trademark of Oracle Corporation and/or its
10 affiliates. Other names may be trademarks of their respective
11 owners.
12
13 Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
14
15 mysql> SHOW DATABASES;
16 +-----+
17 | Database      |
18 +-----+
19 | information_schema |
20 | mysql          |
21 | performance_schema |
22 | sys            |
23 +-----+
24 4 rows in set (0.00 sec)
25
26 mysql> exit
27 Bye
28
29 C:\>
```

Prima, der Client findet den Server.

■ 4.2 MariaDB unter Windows 10



Die Installation einer Entwicklungsumgebung für Datenbankprogrammierer mithilfe von ApacheFriends ist so einfach, da würde meine Erstgeborene sagen: *Das ist ja babyeinfa*ch!

- Grundkurs
 - Installationsquelle
 - Installation unter Windows 10
- Vertiefendes
 - Verschiedene Aufrufparameter des SQL Clients
 - Windows-Suchpfad anpassen

MariaDB kann als Server von <https://downloads.mariadb.org/> heruntergeladen werden. Eine Dokumentation steht in Form einer *knowledge base* auf <https://mariadb.com/kb/en/> zur Verfügung. Für dieses Buch empfehle ich allerdings die Installation mittels XAMPP.

Für Linux und Windows ist XAMPP² eine weit verbreitete Entwicklungs- und Testumgebung für Datenbank- und Webentwickler. Seit dem Release v5.5.30 bzw. v5.6.14 wird anstelle von MySQL MariaDB verwendet. Die Gründe dafür können Sie unter <https://www.quora.com/Why-did-XAMPP-switch-to-MariaDB-from-MySQL> nachverfolgen. Die Installationspakete sowohl für Windows als auch für verschiedene Linux-Distributionen können unter <https://www.apachefriends.org/de/download.html> heruntergeladen werden. Installationsanweisungen und Dokus zu XAMPP sind in Form von FAQs vorhanden:

- Linux: https://www.apachefriends.org/faq_linux.html
- Windows: https://www.apachefriends.org/faq_windows.html
- OS X: https://www.apachefriends.org/faq_osx.html

Auch für interessierte Privatleute ist XAMPP wegen seiner Einfachheit ein guter Einstieg. XAMPP ist **nicht** für den Einsatz in der Produktion gedacht. Um für die Entwicklung flexibel verwendbar zu sein, ist die Standardinstallation offen bezüglich vieler Angriffszenarien.

Die hier verwendete Konfiguration ist XAMPP 7.3.7 mit MariaDB 10.3.18. Falls Sie eine ältere Version von XAMPP installiert haben, sollten Sie diese zuerst deinstallieren: START → SYSTEMSTEUERUNG → PROGRAMME UND FUNKTIONEN → XAMPP DEINSTALLIEREN. Laden Sie anschließend von [Sei19] die Windows-Version mit Installer herunter.

Überprüfen Sie den Download mit einem md5-Programm Ihrer Wahl³ und starten Sie durch Doppelklick auf den Download die Installation. Gegebenenfalls werden Sie darüber informiert, dass die Installation wegen Schutzprogrammen auf Ihrem Rechner – meist Antivirensoftware – langsamer ablaufen oder gestört wird. Falls Sie deshalb Probleme bekommen, folgen Sie dem angegebenen Link.

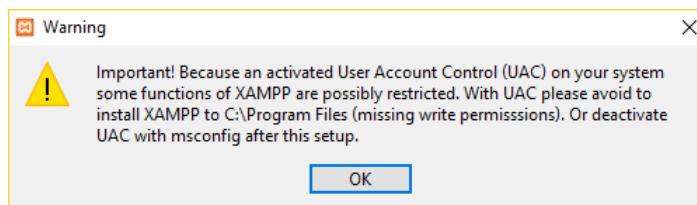


Bild 4.7 Warnung wegen der Zugriffsrechte

Vielelleicht bekommen Sie eine Warnung, wie in Bild 4.7 zu sehen. Wir wollen uns den Ratsschlag zu Herzen nehmen und später nicht ins Programmverzeichnis installieren.

Anschließend können Sie auswählen (siehe Bild 4.8 auf der nächsten Seite), welche Komponenten installiert werden sollen. Sie können bedenkenlos alles ausgewählt lassen; ich habe hier die Komponenten ausgewählt, die Sie für dieses Buch brauchen werden.

² X = {W|L} und steht für {Windows|Linux}, Apache, MySQL, PHP und Perl.

³ Auf diesen Schritt kann man verzichten, wenn man keine Probleme erwartet. Es wird lediglich überprüft, ob der Download fehlerfrei war. Die Installation kann aber auch ohne den md5-Vergleich fortgesetzt werden.

Im nächsten Schritt wählen Sie das Zielverzeichnis aus. Wenn Sie die obige Warnung erhalten haben, sollten Sie tatsächlich die Daten unter c:\xampp oder einem vergleichbaren Pfad ablegen.

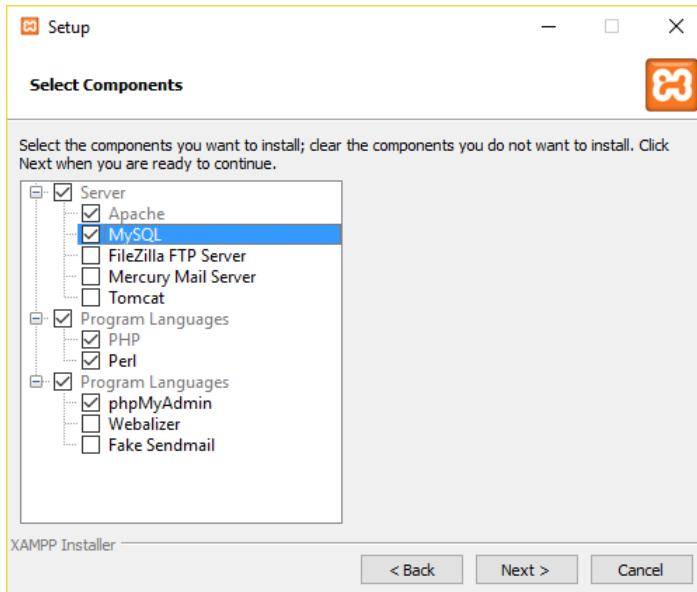


Bild 4.8 Auswahl der Komponenten

Nach zwei weiteren Klicks startet die Installation. Die dauert einige Zeit, aber bei Weitem nicht mehr so lange, wie ich es noch für die erste Auflage dieses Buches berichten musste :-). Wenn Sie nun auf FINISH klicken, achten Sie darauf, dass auch der Start des Control Panel aktiviert ist.

Sie können nun die Sprache auswählen (**Bild 4.9**), und es erscheint das Control Panel ([Bild 4.10 auf der nächsten Seite](#)). Starten Sie nun Apache und MySQL – wobei tatsächlich MariaDB gestartet wird. Wenn Sie in der MySQL-Zeile auf ADMIN klicken, wird *phpMyAdmin* geöffnet. Dieses sehr bequeme Tool dient der Administration des Servers und der Datenabfrage.

Der MariaDB Server liegt nun in einem XAMPP-Pfad: c:\xampp\mysql\bin. Hier sind die Programme und – *sehr wichtig!* – die my.ini (siehe [Zeile 12](#)) abgelegt. Der MariaDB Client ist in [Zeile 16](#) zu finden, der Server selbst in [Zeile 20](#).

```

1 c:\xampp\mysql\bin>dir
2 Verzeichnis von c:\xampp\mysql\bin
3
4
5 22.07.2019 18:41    <DIR>          .
6 22.07.2019 18:41    <DIR>          ..
7 15.06.2019  03:19        4.011.944 aria_chk.exe
8 15.06.2019  03:19        3.569.064 aria_dump_log.exe

```



Bild 4.9 Sprache

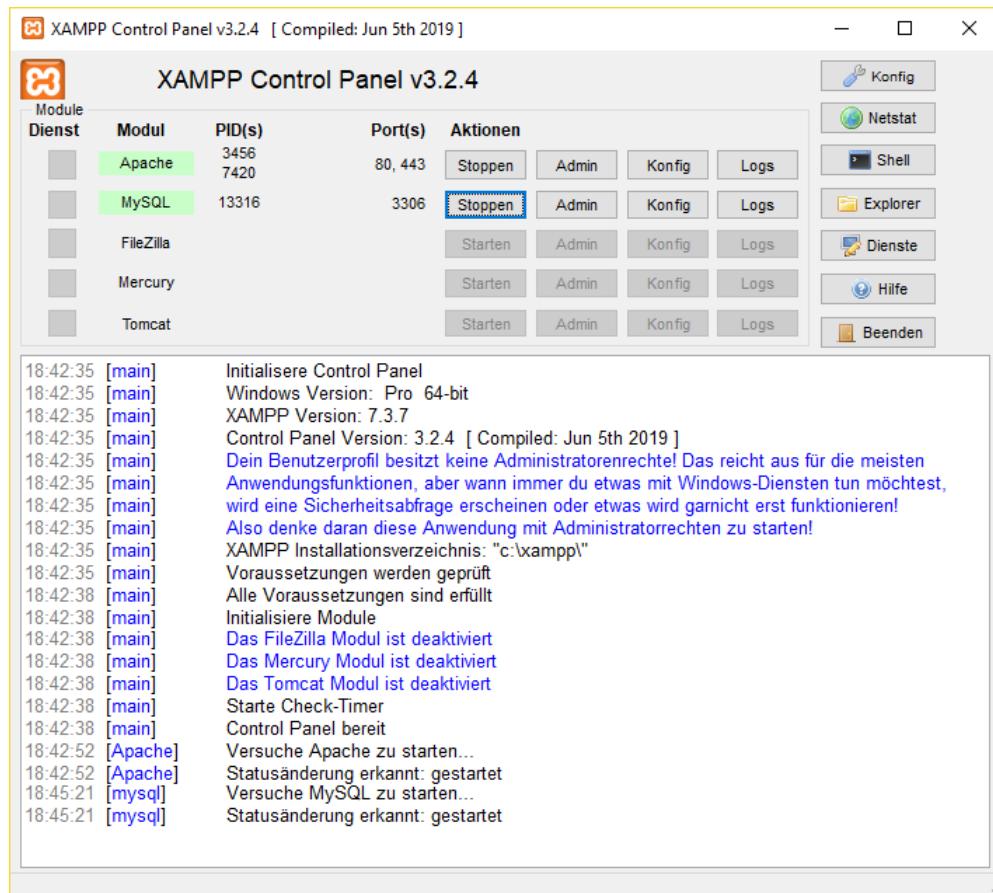


Bild 4.10 Das XAMPP-Panel

```

9 15.06.2019 03:19      3.800.488 aria_ftdump.exe
10 [...]
11 15.06.2019 03:18      3.460.520 mbstream.exe
12 22.07.2019 18:41      5.817 my.ini
13 15.06.2019 03:19      3.778.984 myisamchk.exe
14 [...]
15 15.06.2019 03:02      25.373 myrocks_hotbackup
16 15.06.2019 03:18      3.856.808 mysql.exe
17 15.06.2019 03:18      3.755.432 mysqladmin.exe
18 15.06.2019 03:18      3.903.400 mysqlbinlog.exe
19 15.06.2019 03:18      3.766.184 mysqlcheck.exe
20 15.06.2019 03:18      15.841.704 mysqld.exe
21 15.06.2019 03:18      3.828.136 mysqldump.exe
22 [...]
23 15.06.2019 03:19      3.245.992 sst_dump.exe
24 40 Datei(en), 141.378.794 Bytes
25 2 Verzeichnis(se), 224.103.747.584 Bytes frei

```

Das Datenverzeichnis ist: c:\xampp\mysql\data. Für jede Datenbank wird ein Verzeichnis und für jede Tabelle eine oder mehrere Dateien angelegt. Dieses Verzeichnis sollte re-

gelmäßig gesichert werden. Man kann hier gut die schon installierten Datenbanken erkennen: `mysql`, `performance_schema`, `test` und `phpmyadmin`.

```

1 c:\xampp\mysql\data>dir
2 Verzeichnis von c:\xampp\mysql\data
3
4 22.07.2019  18:45    <DIR>          .
5 22.07.2019  18:45    <DIR>          ..
6 31.05.2019   15:52      16.384 aria_log.00000001
7 31.05.2019   15:52          52 aria_log_control
8 31.05.2019   16:08      12.582.912 ibdata1
9 22.07.2019  18:45      12.582.912 ibtmp1
10 31.05.2019   15:52      1.147 ib_buffer_pool
11 22.07.2019  18:45      5.242.880 ib_logfile0
12 31.05.2019   15:52      5.242.880 ib_logfile1
13 31.05.2019   15:52          0 multi-master.info
14 22.07.2019  18:40    <DIR>          mysql
15 22.07.2019  18:45          6 mysql.pid
16 22.07.2019  18:45          1.518 mysql_error.log
17 22.07.2019  18:40    <DIR>          performance_schema
18 22.07.2019  18:40    <DIR>          phpmyadmin
19 22.07.2019  18:40    <DIR>          test
20 10 Datei(en),     35.670.691 Bytes
21 6 Verzeichnis(se), 223.542.648.832 Bytes frei

```

Leider ist der Suchpfad zu den MariaDB-Binaries nicht durch XAMPP in die Pfadvariable von Windows eingefügt worden. Wollen Sie nun den MariaDB Client aufrufen, gibt es zwei Varianten:

1. Sie geben bei jedem Aufruf des MariaDB Clients den Pfad mit an:

`c:\xampp\mysql\bin\mysql irgendwelche aufrufparameter`

2. Sie fügen den Pfad dem Windows-Suchpfad an: WINDOWS + PAUSE → BASISINFORMATIONEN ÜBER DEN COMPUTER ANZEIGEN → EINSTELLUNGEN ÄNDERN → REGISTERKARTE ERWEITERT → UMGEBUNGSVARIABLEN → unter Systemvariablen PATH markieren → BEARBEITEN → NEU → Pfadangabe auswählen und ganz oft OK drücken. Nun lässt sich der Client von jeder Stelle aus aufrufen. Sie müssen aber die aktuelle COMMAND-Box erst schließen und eine neue öffnen, damit die Änderungen übernommen werden.

Der MariaDB Server ist mit einem `root` installiert worden, der kein Passwort hat. Dies sollten Sie sehr schnell nach der Installation ändern. Selbst in Entwicklungsumgebungen sollten minimale Sicherheitsregeln angewendet werden – obwohl ich gestehen muss, dass ich es für diese *Buchumgebung* auch nicht gesetzt habe.

Das `root`-Passwort lässt sich über die COMMAND-Box leicht ändern:

`c:\>mysqladmin -uroot -pPasswortAlt password PasswortNeu.`

Beachten Sie bitte, dass zwischen `-p` und `PasswortAlt` kein Leerzeichen steht.

■ 4.3 Andere Installationen mit Docker

Einen SQL Server in einem Docker laufen zu lassen, ist eine wirklich ressourcenschonende und elegante Art, verschiedene SQL Server parallel auf einem Rechner bzw. unter einem

Betriebssystem laufen zu lassen. Ich möchte hier kurz vorstellen, wie ich unter Ubuntu⁴ im Docker die verschiedenen Server installiere und aufrufe⁵. Als Docker Repository habe ich folgende Internetquelle eingetragen: <https://hub.docker.com/>.

4.3.1 MySQL

Zunächst wird das Image aus [Doc17] heruntergeladen und installiert. Die dabei aktuellste Linux-MySQL-Version ist dabei 8.0.17.

```

1 adams:~$ sudo docker pull mysql
2 Using default tag: latest
3 latest: Pulling from library/mysql
4 0a4690c5d889: Pull complete
5 [...]
6 c80d654aa77e: Pull complete
7 Digest: sha256:
     b1b2c176a45f4ff6875d0d7461fe1fdd8606deb015b38b69545f72de97714efd
8 Status: Downloaded newer image for mysql:latest
9 docker.io/library/mysql:latest

```

Damit wir später (MySQL-)Client-Programme mit dem Server verbinden können, müssen wir noch ein künstliches Netzwerk erstellen. Die dann laufenden Dockerinstanzen können über dieses Netzwerk so kommunizieren, als wären sie miteinander verbundene Rechner.

```

1 adams:~$ sudo docker network create mydockernet
2 08f593153972f5ba3d8e1784003ea67da80110cb6bee107fdb86676924310ffc

```

Und schon kann der MySQL Server gestartet werden. Dabei werden ihm einige MySQL- und einige Docker-spezifische Parameter mitgegeben:

```

1 adams:~$ sudo docker run --name mysql_server01 -v /media/sf_Version_3:/home/
   Buch --network mydockernet -e MYSQL_ROOT_PASSWORD=weinschlauchX10 -d
   mysql:latest
2 3f834e65f9fc803312a3ea07181b353568d326ad2dfd31033a35a46fc11238b0

```

Die Parameter im Einzelnen:

- **--name mysql_server01**
Die Dockerinstanz bekommt einen Namen, der später das Auffinden und Zuordnen erleichtert. Bitte machen Sie sich klar, dass Sie ja auch mehrere Instanzen mit `run` erzeugen können.
- **-v /media/sf_Version_3:/home/buch**
Mit diesem Parameter richte ich ein gemeinsames Verzeichnis für die Dockerinstanz und das Ubuntu-Host-System ein.
- **--network mydockernet**
Diese Dockerinstanz ist nun Teil des Netzwerks mit dem Namen *mydockernet*. Das Netzwerk muss vorher angelegt worden sein (s.o.).
- **-e MYSQL_ROOT_PASSWORD=weinschlauchX10**
Dem MySQL Server wird ein root-Passwort über eine Umgebungsvariable mitgegeben.

⁴ Ubuntu-Version: 18.10

⁵ Eine Einführung in die Docker-Technologie finden Sie beispielsweise unter [Arb17].

- **-d**

Die Ausführung des Containers erfolgt im Hintergrund, d.h., die Konsole meines Fens-ters wird nicht blockiert.

- **mysql:latest**

Es wird die aktuellste Version des Servers im Docker-Image – hier 8.0.17 – gestartet.

In einer weiteren Dockerinstanz wird nun die MySQL-Konsole gestartet. Diese liegt im gleichen Netzwerk wie der Server und verwendet das gleiche gemeinsame Verzeichnis. Der MySQL Server ist im Netzwerk `mydockernet` unter dem Namen `mysql_server01` erreichbar. Der Name ist automatisch beim Start der ersten Dockerinstanz vergeben worden.

```

1 adams:~$ sudo docker run -it --name mysql_client01 -v /media/sf_Version_3:/home/Buch --network mydockernet --rm mysql mysql -hmysql_server01 -uroot -pweinschlauchX10
2 mysql: [Warning] Using a password on the command line interface can be
   insecure.
3 Welcome to the MySQL monitor.  Commands end with ; or \g.
4 Your MySQL connection id is 8
5 Server version: 8.0.17 MySQL Community Server - GPL
6
7 [...]
8
9 mysql> quit
10 Bye

```

Wunderbar! Der MySQL Client ist gestartet worden, er hat den MySQL Server gefunden und konnte sich anmelden. So kann man arbeiten.

Diese Parameter sind neu:

- **-it**

Eine Kombination aus `-i` und `-t`. Mit `-i` wird die Standardeingabe (STDIN) auf die Dockerinstanz umgeleitet und mit `-t` wird eine Konsolenumgebung (tty) erstellt und angezeigt; man erkennt dies am veränderten Cursor in der Konsole.

- **--rm**

Beim Start einer Dockerinstanz wird ein Dateisystem erstellt, auf welchem die Programme arbeiten können. Dieser Parameter bereinigt dieses Dateisystem nach Beendigung. Will man die Dateien beispielsweise zwecks Analyse bestehen lassen, lässt man `--rm` weg.

- **Das erste mysql**

Die Dockerinstanz wird auf Basis des Docker-Image `mysql` erstellt.

- **Das zweite mysql**

Der MySQL Client ist eine ausführbare Datei mit dem Namen `mysql`. Eine hier leider verwirrende Namensgleichheit.

- **-hmysql_server01**

Das ist kein Docker-Parameter, sondern ein Parameter des MySQL Clients. Er weist den Client an, sich mit dem Server namens `mysql_server01` zu verbinden.

- **-uroot**

Auch ein Client-Parameter: Der Client meldet sich mit dem Benutzernamen `root` an.

- **-pweinschlauchX10**

Und noch ein Client-Parameter: das zum Benutzernamen passende Passwort.

Eine weitere Möglichkeit, mit dem Server zu arbeiten, besteht darin, in der Dockerinstanz eine Shell aufzumachen und die Befehle dort direkt einzugeben:

```

1 adams:~$ sudo docker exec -it mysql_server01 bash
2 root@3f834e65f9fc:/# mysql -uroot -pweinschlauchX10
3 mysql: [Warning] Using a password on the command line interface can be
   insecure.
4 Welcome to the MySQL monitor. Commands end with ; or \g.
5 Your MySQL connection id is 9
6 Server version: 8.0.17 MySQL Community Server - GPL
7
8 [...]
9
10 mysql> exit
11 Bye
12 root@3f834e65f9fc:/# ls /home/
13 Buch
14 root@3f834e65f9fc:/# exit

```

Bei Dockerinstanzen, die im Vordergrund laufen, so wie unser MySQL Client, wird die Instanz automatisch beendet/heruntergefahren, wenn die Vordergrundanwendung beendet wird. Bei den Serveranwendungen wird beim Stoppen der Instanz ein Signal an den MySQL Server gesendet, welches diesen im Normalfall sauber herunterfährt; wir müssen das also nicht noch manuell anstoßen.

```

1 adams:~$ sudo docker stop mysql_server01
2 mysql_server01

```

Will ich die Dockerinstanz wiederverwenden, müssen die ganzen Parameter nicht erneut eingegeben werden. Sie können die Instanz einfach wieder starten:

```
1 adams:~$ sudo docker start mysql_server01
```



Hinweis: In der Datei /etc/mysql/my.cnf müssen folgende Einträge hinzugefügt werden:

```

1 # secure-file-priv= NULL
2 secure-file-priv= /var/lib/mysql/interchange/
3 sql_mode      = STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,
                  ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION

```

4.3.2 MariaDB

Das Vorgehen entspricht völlig dem von MySQL (siehe [Abschnitt 4.3.1](#)) und wird deshalb hier kürzer abgehandelt. Auch hier wird das Image aus [[Doc17](#)] heruntergeladen und installiert.

```

1 adams:~$ sudo docker pull mariadb
2 adams:~$ sudo docker network create mydockernet
3 adams:~$ sudo docker run --name mariadb_server01 -v /media/sf_Version_3/:/home
           /Buch --network mydockernet -e MYSQL_ROOT_PASSWORD=weinschlauchX10 -d
           mysql:latest

```

Lassen Sie sich nicht durch den Namen der Umgebungsvariablen verwirren. An vielen Stellen wird in MariaDB noch das Element mit *MySQL* benannt, obwohl es eine MariaDB-Komponente ist.

```

1 adams:~$ sudo docker run -it --name mariadb_client01 -v /media/sf_Version_3/:/
   home/Buch --network mydockernet --rm mariadb mysql -hmariadb_server01 -
   uroot -pweinschlauchX10
2 Welcome to the MariaDB monitor. Commands end with ; or \g.
3 Your MariaDB connection id is 10
4 Server version: 10.4.6-MariaDB-1:10.4.6+maria~bionic mariadb.org binary
   distribution
5
6 [...]
7
8 MariaDB [(none)]> quit
9 Bye

```

Mit `stop` und `start` kann die Dockerinstanz nun nach Bedarf ein- bzw. ausgeschaltet werden.

```

1 adams:~$ sudo docker start mariadb_server01
2 mariadb_server01
3 adams:~$ sudo docker stop mariadb_server01

```

4.3.3 PostgreSQL

Eine kostenfreie Version für Windows kann von <https://www.postgresql.org/download/windows/> heruntergeladen werden.

Wie bei der Einrichtung des MySQL-Containers⁶ (siehe [Abschnitt 4.3.1](#)) und des MariaDB-Containers (siehe [Abschnitt 4.3.2](#)), so gestaltet sich auch die Einrichtung des PostgreSQL-Containers denkbar einfach. Die Bedeutung der einzelnen Parameter lesen Sie bitte in den obigen Abschnitten nach.

```

1 adams:~$ sudo docker pull postgres
2 [...]
3 f39bb2a104c1: Pull complete
4 Digest: sha256:68
   b49a280d2fbe9330c0031970ebb72015e1272dfa25f0ed7557514f9e5ad7b7
5 Status: Downloaded newer image for postgres:latest
6 docker.io/library/postgres:latest
7
8 adams:~$ sudo docker network create mydockernet
9
10 adams:~$ sudo docker run --name postgresql_server01 -v /media/sf_Version_3/:/
   home/Buch --network mydockernet -e POSTGRES_PASSWORD=weinschlauchX10 -d
   postgres
11 aacd0b14157ee5ba23529cb88a030786b1f9d75d3f5a9529c41f50fe771d2f3e
12
13 adams:~$ sudo docker run -it --rm --network mydockernet postgres psql -h
   postgresql_server01 -U postgres
14 Password for user postgres:
15 psql (11.4 (Debian 11.4-1.pgdg90+1))

```

⁶ Container ist ein anderes Wort für eine Docker-Instanz.

```

16 Type "help" for help.
17
18 postgres=# SELECT 'Hello World';
19 ?column?
20 -----
21 Hello World
22 (1 row)
23
24 postgres=# quit
25 adams:~$ sudo docker stop postgresql_server01

```

4.3.4 Microsoft SQL Server

Eine kostenfreie Developer-Version für Windows kann unter <https://www.microsoft.com/de-de/sql-server/sql-server-downloads> heruntergeladen werden. Aber auch hier verwende ich die Docker-Version.

Anders als bei den drei anderen Docker-Images verwende ich eine Microsoft-Internetquelle als Docker-Repository. Auch verwende ich nicht explizit den Befehl `docker pull`, sondern Docker erkennt, wenn ein Image angesprochen ist, welches er noch nicht kennt, und versucht es dann aus der angegebenen Quelle herunterzuladen.

```

1 adams:~$ sudo docker network create mydockernet
2
3 adams:~$ sudo docker run --name mssql_server01 --network mydockernet -v /media
      /sf_Version_3:/home/Buch -e 'ACCEPT_EULA=Y' -e 'SA_PASSWORD=
      weinschlauchX10' -p 1433:1433 -d mcr.microsoft.com/mssql/server:latest
4 [...]
5 3e6f1aaa79f4: Pull complete
6 Digest: sha256:1
      bbf3b11687ce4d97eb5e6b6e61ccc500d0eff92f92e51893112a3fc665ce7b7
7 Status: Downloaded newer image for mcr.microsoft.com/mssql/server:latest
8 86e3151cc6656208df8b4d50ea733d9e9e330477b4b1633aad073081fefde7e5
9
10 adams:~$ sudo docker exec -it mssql_server01 bash
11 root@86e3151cc665:/# /opt/mssql-tools/bin/sqlcmd -U sa -P weinschlauchX10
12 1> SELECT 'Hallo';
13 2> GO
14
15 -----
16 Hallo
17
18 (1 rows affected)
19 1> quit
20 root@86e3151cc665:/# exit
21 exit
22 adams:~$ sudo docker stop mssql_server01

```

Zu allen Serversystemen existieren Unmengen an Wikis, Tutorials, YouTube-Videos und Foren. Zu fast jedem Aspekt wie *Installation, Sicherheit, Performance, Verlässlichkeit usw.* gibt es Spezialdokumentationen, die in der Regel aktueller und ausführlicher sind, als ich es hier in einem Einführungsbuch über SQL sein kann und will.

5

Datenbank und Tabellen anlegen

■ 5.1 Die Programmiersprache SQL



Was ist SQL?

- Grundkurs
 - Geschichte und Struktur von SQL
- Vertiefendes
 - SQL:2016 vs. MySQL vs. MariaDB vs. PostgreSQL vs. T-SQL

Jetzt geht es endlich mit SQL¹ los. Aber was bedeutet SQL genau? Eigentlich nichts, da die Abkürzung laut Standard ein Eigenname ist; oft wird sie aber mit *Structured Query Language*² aufgelöst. SQL ist eine Sprache zur Bearbeitung und Auswertung von relationalen Datenbanken. Sie umfasst drei Bereiche:

1. Data Definition Language (DDL): Befehlssatz zum Anlegen, Ändern und Löschen von Datenbanken, Tabellen usw. und ihren Strukturen.
2. Data Manipulation Language (DML): Befehlssatz zum Einfügen, Ändern, Löschen und Auslesen von Daten aus den Tabellen.
3. Data Control Language (DCL): Befehlssatz zur Administration von Datenbanken³.

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 Datenbankserver ermittelt das Ergebnis. Solche Arten von Programmiersprachen nennt man *deklarativ*.

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

¹ Aussprache: EsKjuEl; die Aussprache SiQwL ist nicht korrekt, da damit SEQUEL, der Vorläufer von SQL, gemeint ist.

² Die Auflösung als *Structured Query Language* ergibt sich dadurch, dass das SEQUEL die Abkürzung für *Structured English Query Language* ist.

³ Nicht zur Administration des Servers!

1986 wurde der erste SQL-Standard vom ANSI⁴ verabschiedet, der 1987 von der ISO⁵ ratifiziert wurde (SQL1). 1992 wurde der Standard überarbeitet und als SQL-92 (oder auch SQL2) veröffentlicht. Es folgte die Version SQL:1999 (ISO/IEC 9075:1999, auch SQL3 genannt) mit ihrer letzten Revision SQL:2011 (ISO/IEC 9075:2011). Derzeitiger Standard ist SQL:2016 (ISO/IEC 9075:2016), welcher 2019 um den Datentyp der mehrdimensionalen Arrays erweitert wurde (SQL/MDA:2019)⁶.

Abgrenzung zu SQL:2016⁷

Die hier vorgestellten Befehle sind alle mit dem MySQL Community Server 8.0.17, MariaDB 10.4.6-bionic, PostgreSQL 11.4 und MS SQL Server 14.0.3038.14 getestet worden.

Schwerpunkt der Darstellung ist der MySQL Server. Die Befehlssätze von MySQL/MariaDB, PostgreSQL und Transact-SQL⁸ enthalten Befehle und Datentypen, die es im Standard SQL:2016 nicht gibt. Ebenso gibt es in SQL:2016 welche, die in diesen Produkten nicht implementiert sind.

Wie oben erwähnt, werden die Beispiele in MySQL entwickelt und programmiert. Das Buch hat aber nicht MySQL, sondern SQL im Titel stehen, sodass ich auf Unterschiede aufmerksam machen werde. Der Textfluss soll dabei aber nicht unnötig gestört werden. Die Listings für MySQL/MariaDB und PostgreSQL habe ich online gestellt: Für PostgreSQL im Verzeichnis pg, für T-SQL in tsql und für MySQL/MariaDB in mysql.

■ 5.2 Anlegen der Datenbank



Jeder Hausbau fängt beim Fundament an.

- Grundkurs
 - Der Aufruf des MySQL Clients
 - Anzeigen aller Datenbanken des Servers mit `SHOW DATABASES`
 - Anlegen einer Datenbank mit `CREATE DATABASE`
 - Bedingtes Anlegen der Datenbank mit `IF EXISTS`
 - Löschen einer Datenbank mit `DROP DATABASE`
- Vertiefendes
 - Unterschied `DATABASE` und `SCHEMA`
 - Zuweisen einer Zeichenkodierung für die Datenbank mit `CHARACTER SET`
 - Zuweisen einer Sortierung für die ganze Datenbank mit `COLLATE`

⁴ American National Standards Institute; US-amerikanisches Institut zur Normierung, vergleichbar mit dem deutschen DIN

⁵ International Organization for Standardization

⁶ Quelle: [Wik19p]

⁷ Als Quelle für den SQL:2016 habe ich [Whe19] verwendet.

⁸ Transact-SQL (T-SQL) ist der Name des SQL-Dialekt des Microsoft SQL Servers.



Die Quelltexte dieses Kapitels stehen in der Datei `Listing01.sql` (siehe [Listing 29.1 auf Seite 461](#), [Listing 29.37 auf Seite 584](#) und [Listing 29.22 auf Seite 536](#)). ■

5.2.1 Wie ruft man den MySQL Client auf?

Bevor wir damit loslegen, eine Datenbank anzulegen, möchte ich kurz die Verwendung des MySQL Clients vorstellen. Der Zugriff auf den MySQL Server kann auf verschiedenen Wegen erfolgen. MySQL selbst bringt zwei Werkzeuge mit: den MySQL Query Browser, welcher in der MySQL Workbench integriert ist, und den MySQL/MariaDB Client⁹.

Der MySQL Query Browser bietet eine komfortable grafische Oberfläche, muss aber nachinstalliert werden. Der MySQL Client ist zwar sperriger in der Bedienung, aber in der Regel schon vorinstalliert. Tun wir uns den also mal an:

```
1 dockerroot:/# mysql -uroot -p
2 Enter password:
3
4 Welcome to the MySQL monitor. Commands end with ; or \g.
5 Your MySQL connection id is 9
6 Server version: 8.0.17 MySQL Community Server - GPL
7
8 Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.
9
10 Oracle is a registered trademark of Oracle Corporation and/or its
11 affiliates. Other names may be trademarks of their respective
12 owners.
13
14 Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
15
16 mysql>
```

In [Zeile 1](#) wird der Client aufgerufen – hier unter Linux im Docker-Container. Der Aufruf setzt voraus, dass die Datei `mysql.exe` im Suchpfad liegt. Ist dies nicht der Fall, muss der Pfad – wie beispielsweise für eine Windows-XAMPP-Installation – mit angegeben werden¹⁰:

```
1 c:\>c:\xampp\mysql\bin\mysql -uroot -p
```

Die Aufrufparameter des Clients können Sie auszugsweise dem [Abschnitt 25.1 auf Seite 389](#) entnehmen. Was der Client noch so drauf hat, entnehmen Sie bitte dem Handbuch. Mit dem Parameter `-u` wird ein Benutzer angemeldet, hier `root`. Wird dieser Parameter nicht verwendet, wird der Benutzer `guest` angemeldet, der darf aber fast nichts. `-p` hat zur Folge, dass Sie nach dem Passwort gefragt werden. Die entsprechende Eingabezeile lasse ich aus Platzgründen zukünftig weg.

In [Zeile 5](#) wird angegeben, welche Verbindungsnummer zwischen Client und Server hergestellt wurde. Über diese ID kann man mithilfe von Analysewerkzeugen die Verbindung näher betrachten.

⁹ MariaDB bringt ebenfalls einen bis auf Kleinigkeiten kompatiblen Client mit.

¹⁰ Lassen Sie sich nicht irritieren: Obwohl Sie die `mysql.exe` aufgerufen haben, wird bei XAMPP ein MariaDB Client gestartet.

Die Versionsnummer des Servers – nicht des Clients – wird in [Zeile 6](#) angezeigt. Diese Versionsnummer ist immer zu beachten, wenn man im MySQL-Handbuch oder im Internet nach Hilfen sucht.

Die [Zeile 16](#) ist der eigentliche Eingabeprompt. Hier werden die Kommandos/Anweisungen eingegeben. Der Prompt `mysql>`¹¹ zeigt an, dass er auf eine neue Anweisung wartet. Der Prompt `->` hingegen markiert, dass die Anweisung noch nicht abgeschlossen ist und weitere Eingaben zur laufenden Anweisung gemacht werden können.

Natürlich kann man den MySQL Client auch wieder verlassen:

```
1 mysql> EXIT
2 Bye
```

Sie sollten es sich angewöhnen, den Client immer über `EXIT` zu verlassen und nicht einfach das Konsolenfenster schließen. Die Verbindung zum Server bleibt nämlich noch einige Zeit erhalten und blockiert Ressourcen auf dem Server. Diese werden erst wieder freigegeben, wenn ein Timeout eintritt. Der aktuell eingestellte Timeout kann wie folgt ermittelt werden:

```
1 mysql> SHOW VARIABLES LIKE 'wait_timeout';
2 +-----+-----+
3 | Variable_name | Value |
4 +-----+-----+
5 | wait_timeout | 28800 |
6 +-----+-----+
```

Der Wert 28000 hat die Maßeinheit Sekunden, weshalb erst nach 8 Stunden (!) ohne Kommunikation zwischen dem Client und dem MySQL/MariaDB Server die Verbindung beendet wird.

5.2.2 Wie legt man eine Datenbank an?

	SQL:2016 <code>CREATE SCHEMA <i>datenbankname</i> [DEFAULT CHARACTER SET <i>zeichensatz</i>] ;</code> MySQL/MariaDB <code>CREATE {DATABASE SCHEMA} [IF NOT EXISTS] <i>datenbankname</i> [[DEFAULT] CHARACTER SET <i>zeichensatz</i>] [[DEFAULT] COLLATE <i>sortierung</i>] [DEFAULT ENCRYPTION {'Y' 'N'}] ;</code> PostgreSQL <code>CREATE DATABASE <i>datenbankname</i> [ENCODING <i>zeichenkodierung</i>]</code>
---	--

¹¹ Der MariaDB Client meldet sich mit dem Prompt `MariaDB [(none)]>`. In den eckigen Klammern steht die gerade verwendete Datenbank. Da wir noch keine ausgewählt haben, eben `(none)`. Später steht dort beispielsweise `oshop`. Der MySQL Client kennt dieses Feature nicht und ich persönlich bin mir auch nicht sicher, ob ich es mag.

```
[LC_COLLATE sortierung]
[LC_CTYPE ctype]
;

T-SQL
CREATE DATABASE datenbankname
    [COLLATE sortierung]
;
```

In MySQL und MariaDB sind **DATABASE** und **SCHEMA** nur zwei Wörter für das Gleiche (siehe [Tabelle 2.1 auf Seite 16](#)). In SQL:2016 gibt es den Begriff *Datenbank* nicht, sondern nur das *Schema*.

In anderen DBMSen wie MS SQL Server, PostgreSQL, Oracle und DB2 besteht zwischen diesen beiden Begriffen ein Unterschied. Ein Schema ist dort eine sinnvolle Gruppe von Datenbankobjekten wie Tabellen, Ansichten etc. innerhalb einer Datenbank; somit kann diese viele Schemata enthalten. Mit ihrer Hilfe können große oder komplexe Datenbanken in inhaltliche Gruppen unterteilt und mit verschiedenen Zugriffsrechten versehen werden.

Wie liest man eigentlich so eine Spezifikation? Die geschweiften Klammern ({}) stehen für eine *Liste von Alternativen*. Innerhalb der Klammern werden die Alternativen durch einen senkrechten Strich (|) voneinander getrennt. Eine davon muss verwendet werden.

Die eckigen Klammern ([]) umschließen eine Option, d.h. eine Angabe, die man machen kann, aber nicht zwingend machen muss. So ist IF NOT EXISTS eine Option und schränkt offensichtlich die Ausführung des Befehls auf den Fall ein, dass die Datenbank noch nicht vorhanden ist.

Anschließend folgt der Name der Datenbank, die in unserem Beispiel oshop heißen soll. Zum Zeichensatz und zur Sortierung kommen wir später¹².

```
1 mysql> CREATE DATABASE IF NOT EXISTS oshop;
2 Query OK, 1 row affected (0.12 sec)
```

Bitte beachten Sie, dass am Ende eines SQL-Befehls immer ein Semikolon steht!¹³ Ohne dieses geht der MySQL Client davon aus, dass der Befehl noch nicht vollständig ist:

```
1 mysql> CREATE DATABASE
2      -> IF NOT EXISTS
3      -> oshop;
4 Query OK, 1 row affected, 1 warning (0.00 sec)
```

Sie sehen, wie der Befehl auf mehrere Zeilen verteilt wurde. Sie sollten sich dies angewöhnen, da der SQL-Quelltext dadurch erheblich leichter zu lesen und zu verstehen ist. Das Gleiche gilt auch für die Einrückungen, die eine gewisse inhaltliche Gruppierung der Anweisung verdeutlichen.

Aber Achtung! In der letzten Zeile steht, dass es eine Warnung gegeben hat. Mit SHOW WARNINGS¹⁴ kann man sich die Warnungen bezüglich des letzten Befehls anschauen:

¹² Die Verschlüsselung überlasse ich dem interessierten Leser.

¹³ Dies ist nicht nötig, wenn Sie den SQL-Befehl aus einer Anwendung heraus auf den Server ausführen wollen.

Die MySQL-APIs von PHP, PERL, C# etc. hängen das Semikolon immer an.

¹⁴ Die ganzen SHOW-Befehle sind nicht Teil des SQL:2016-Standards. Sie werden deshalb hier syntaktisch nicht näher erläutert.

```

1 mysql> SHOW WARNINGS;
2 +-----+-----+
3 | Level | Code | Message
4 +-----+-----+
5 | Note | 1007 | Can't create database 'oshop'; database exists |
6 +-----+-----+

```

Die Warnung bedeutet, dass es diese Datenbank schon gibt und daher nicht neu angelegt wird.

5.2.3 Wie löscht man eine Datenbank?

 SQL:2016	MySQL/MariaDB	PostgreSQL, T-SQL
DROP SCHEMA datenbankname {CASCADE RESTRICT} ;	DROP {DATABASE SCHEMA} [IF EXISTS] datenbankname ; ;	DROP SCHEMA [IF EXISTS] datenbankname ;

Wollen Sie eine Datenbank auf jeden Fall neu anlegen, unabhängig davon, ob eine Datenbank mit gleichem Namen schon vorhanden ist, müssen Sie die ggf. vorhandene erst mal löschen.

In der SQL:2016-Version löscht die Angabe CASCADE alle in der Datenbank angesiedelten Objekte wie Tabellen, Ansichten etc. Die Angabe RESTRICT verweigert das Löschen der Datenbank, solange noch andere Objekte der Datenbank vorhanden sind.

Elementare Englischkenntnisse sagen uns, dass im MySQL-Dialekt die Option IF EXISTS die Datenbank nur dann *dropt*, wenn sie vorhanden ist. Da in MySQL und MariaDB keine Angabe wie CASCADE oder RESTRICT gemacht werden kann, stellt sich die Frage, wie diese sich verhalten. MySQL und MariaDB sind hier sehr brutal. Wenn der Anwender die Datenbank löschen will, soll er doch. Sie verhalten sich also wie ein CASCADE.

Probieren wir das mal aus:

```

1 mysql> DROP DATABASE IF EXISTS oshop;
2 Query OK, 0 rows affected (0.29 sec)
3
4 mysql> DROP DATABASE oshop;
5 ERROR 1008 (HY000): Can't drop database 'oshop'; database doesn't exist

```

In Zeile 4 wird noch mal versucht, die Datenbank zu löschen. Da wir diese aber schon in Zeile 1 gelöscht hatten, wird eine Fehlermeldung erzeugt.

5.2.4 Wie wird ein Zeichensatz zugewiesen?

Die Befehlsreferenz auf [Seite 66](#) gibt noch weitere Optionen an. Eine ist hier jetzt relevant: CHARACTER SET.

Was ist ein Zeichensatz? Auf dem Computer werden Buchstaben, Ziffern, Satz- und Sonderzeichen durch Zahlen kodiert. So ist beispielsweise der Buchstabe A im ASCII¹⁵ die Zahl 0x41 und a die Zahl 0x61. Da für die Kodierung des ASCII nur ein Byte (= 8 Bit) zur Verfügung steht, können nur $2^8 = 256$ 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 werdenen *utf8*, *utf16* und *utf32*.

Welche Zeichensätze von Ihrem Server unterstützt werden, können Sie leicht mit SHOW CHARACTER SET herausfinden. Bei mir waren es 41! Möchten Sie die Ausgabe auf bestimmte Zeichensätze einschränken, geht das natürlich auch:

```

1 mysql> SHOW CHARACTER SET LIKE 'latin%';
2 +-----+-----+-----+-----+
3 | Charset | Description          | Default collation | Maxlen |
4 +-----+-----+-----+-----+
5 | latin1  | cp1252 West European    | latin1_swedish_ci |   1 |
6 | latin2  | ISO 8859-2 Central European | latin2_general_ci |   1 |
7 | latin5  | ISO 8859-9 Turkish        | latin5_turkish_ci |   1 |
8 | latin7  | ISO 8859-13 Baltic         | latin7_general_ci |   1 |
9 +-----+-----+-----+-----+
10 4 rows in set (0.00 sec)
11
12 mysql> SHOW CHARACTER SET LIKE 'utf%';
13 mysql> SHOW CHARACTER SET LIKE 'utf%';
14 +-----+-----+-----+-----+
15 | Charset | Description          | Default collation | Maxlen |
16 +-----+-----+-----+-----+
17 | utf16   | UTF-16 Unicode        | utf16_general_ci  |   4 |
18 | utf16le | UTF-16LE Unicode      | utf16le_general_ci |   4 |
19 | utf32   | UTF-32 Unicode        | utf32_general_ci  |   4 |
20 | utf8    | UTF-8 Unicode          | utf8_general_ci   |   3 |
21 | utf8mb4 | UTF-8 Unicode        | utf8mb4_0900_ai_ci |   4 |
22 +-----+-----+-----+-----+
23 5 rows in set (0.00 sec)
```

¹⁵ Siehe [Ass76]

Für die meisten Anwendungen in Deutschland sind die Zeichensätze latin1, cp850 und utf8 ausreichend. Welchen Zeichensatz Sie tatsächlich brauchen, ist genau zu untersuchen und hängt in der Regel von der Datenquelle ab¹⁶.

Stammen Ihre Daten aus einer älteren Quelle und sind ggf. über ein COMMAND-Terminal eingegeben worden, ist cp850 vermutlich richtig¹⁷. Neue Anwendungen sollten von Anfang an utf8 verwenden. Diese Kodierung ist recht stabil bezüglich der Sonderzeichen und Umlaute und macht auch die Verwaltung von anderssprachlichen Zeichen einfach möglich. Ein weiterer Vorteil ist der im Verhältnis zu utf16 und utf32 geringere Speicherverbrauch.

Microsoft verwendet in seiner .NET-Umgebung utf16. Falls Sie also eine Anwendung bauen, die mit ADO/.NET arbeiten soll, ist die Verwendung von utf16 sinnvoll.

So sieht unser Anlegen einer Datenbank bisher aus:

```
1  DROP DATABASE IF EXISTS oshop;
2  CREATE DATABASE oshop
3  DEFAULT CHARACTER SET utf8;
```

In T-SQL des Microsoft SQL Servers kann der Zeichensatz nicht unabhängig von der Sortierung ausgewählt werden. Vielmehr wurden alle notwendigen Zeichensätze mit Sortierungen und einigen Zusätzen kombiniert, sodass man beim Anlegen der Datenbank mit Angabe der Sortierung auch den Zeichensatz festlegt:

```
1  CREATE DATABASE oshop
2  COLLATE German_PhoneBook_CI_AI
3  ;
```

Dies führt alleine schon für German_ zu 52 Varianten und insgesamt standen 3995 Collations zu Verfügung. Aber damit sind wir eigentlich schon beim nächsten Abschnitt angekommen.

5.2.5 Wie wird eine Sortierung zugewiesen?

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

Die verfügbaren Sortierungen lassen sich leicht mit SHOW COLLATION anzeigen. Bei meiner MySQL-Installation sind es 272 und bei MariaDB 322, was bei 41 Zeichensätzen schon deutlich macht, dass es mehrere Sortierreihenfolgen für einen Zeichensatz geben kann. Eine Liste der für die Sprache Deutsch relevanten Sortierungen finden Sie in [Abschnitt 28.2 auf Seite 458](#).

Betrachten wir die Sortierreihenfolgen für den Zeichensatz latin1. MySQL liefert mir acht und MariaDB zehn mögliche Sortierungen:

¹⁶ Eine Liste der Zeichensätze, die für die deutsche Sprache relevant sein könnten, finden Sie in [Abschnitt 28.1 auf Seite 457](#).

¹⁷ Denken Sie lieber darüber nach, die Daten mit einem Tool wie iconv in ein modernes Format zu konvertieren.

```

1 mysql> SHOW COLLATION LIKE 'latin1%';
2 +-----+-----+-----+-----+-----+-----+
3 | Collation      | Charset | Id | Default | Compiled | Sortlen | Pad[...] |
4 +-----+-----+-----+-----+-----+-----+
5 | latin1_bin     | latin1  | 47 |        | Yes      |       1 | PAD[...]  |
6 | latin1_danish_ci | latin1  | 15 |        | Yes      |       1 | PAD[...]  |
7 | latin1_general_ci | latin1  | 48 |        | Yes      |       1 | PAD[...]  |
8 | latin1_general_cs | latin1  | 49 |        | Yes      |       1 | PAD[...]  |
9 | latin1_german1_ci | latin1  |  5 |        | Yes      |       1 | PAD[...]  |
10 | latin1_german2_ci | latin1  | 31 |        | Yes      |       2 | PAD[...]  |
11 | latin1_spanish_ci | latin1  | 94 |        | Yes      |       1 | PAD[...]  |
12 | latin1_swedish_ci | latin1  |  8 | Yes    | Yes     |       1 | PAD[...]  |
13 +-----+-----+-----+-----+-----+-----+
14 8 rows in set (0.01 sec)

```

Es fällt auf, dass es zwei Sortierungen für Deutsch gibt: latin1_german1_ci und latin1_german2_ci. Dies hängt damit zusammen, dass in Wörterbüchern wie dem Duden die Umlaute anders sortiert werden als im Telefonbuch (siehe [Tabelle 5.1](#)).

Tabelle 5.1 Sortierungen für latin1_german

Zeichen	Wörterbuch (DIN-1)	Telefonbuch (DIN-2)
Ä	A	AE
Ö	O	OE
Ü	U	UE
ß	s	SS

Für utf8 gibt es auch viele verschiedene Sortierungen. Für Deutsch die *Wörterbuchssortierung* nach DIN 5007-1, die *Telefonbuchsortierung*, in utf8mb4_unicode_ci und nach DIN 5007-2 in utf8mb4_german2_ci¹⁸. Ebenfalls nach DIN 5007-2 sind utf8mb4_de_pb_0900_ai_ci und utf8mb4_de_pb_0900_as_cs sortiert, wobei pb für *phonebook* steht und 900 die entsprechende Codepage ist.

```

1 mysql> SHOW COLLATION LIKE 'utf8%';
2 +-----+-----+-----+-----+-----+-----+
3 | Collation      | Charset | Id | Default | Compiled | Sortlen | 
4 +-----+-----+-----+-----+-----+-----+
5 | utf8mb4_0900_ai_ci | utf8mb4 | 255 | Yes    | Yes     |       0 | 
6 [...] 
7 | utf8mb4_bin     | utf8mb4 |  46 |        | Yes     |       1 | 
8 [...] 
9 | utf8mb4_de_pb_0900_ai_ci | utf8mb4 | 256 |        | Yes     |       0 | 
10 | utf8mb4_de_pb_0900_as_cs | utf8mb4 | 279 |        | Yes     |       0 | 
11 [...] 
12 | utf8mb4_general_ci | utf8mb4 |  45 |        | Yes     |       1 | 
13 | utf8mb4_german2_ci | utf8mb4 | 244 |        | Yes     |       8 | 
14 [...] 
15 | utf8mb4_unicode_ci | utf8mb4 | 224 |        | Yes     |       8 | 
16 [...] 
17 | utf8_bin         | utf8   |  83 |        | Yes     |       1 | 
18 [...] 
19 | utf8_general_ci  | utf8   |  33 | Yes    | Yes     |       1 | 
20 | utf8_general_mysql500_ci | utf8   | 223 |        | Yes     |       1 | 

```

¹⁸ Die Spalte Pad_attribute habe ich aus Platzgründen weggelassen

```

21 | utf8_german2_ci          | utf8    | 212 |           | Yes   |     8 |
22 | [...]                   |
23 | utf8_vietnamese_ci      | utf8    | 215 |           | Yes   |     8 |
24 +-----+-----+-----+-----+-----+-----+
25 103 rows in set (0.01 sec)

```



Aufgabe 5.1: Wofür stehen die Anhänge _ci, _cs, _ai, _as, _ki, _ks und _bin bei den Namen der Sortierungen?

Aufgabe 5.2: Was ist der Unterschied zwischen utf8 und utf8mb4?

Das Anlegen der Datenbank kann jetzt vervollständigt werden:

```

1 DROP DATABASE IF EXISTS oshop;
2 CREATE DATABASE oshop
3 CHARACTER SET utf8
4 COLLATE utf8_unicode_ci;

```

Verbleibt nur noch nachzuschauen, ob die Datenbank wirklich angelegt wurde.

```

1 mysql> SHOW DATABASES;
2 +-----+
3 | Database      |
4 +-----+
5 | information_schema |
6 | mysql          |
7 | oshop          |
8 | test           |
9 +-----+
10 4 rows in set (0.09 sec)

```



Aufgabe 5.3: Finden Sie heraus, wo und wie in der Verzeichnisstruktur des SQL Servers die Datenbank angelegt wird. Versuchen Sie dabei, die Frage zu beantworten, warum bei Dateisystemen, die zwischen Groß- und Kleinschreibung unterscheiden, Datenbanknamen mit anderer Groß- und Kleinschreibung ebenfalls unterscheiden werden.

■ 5.3 Anlegen der Tabellen



Die Tabelle als universeller Datencontainer

- Grundkurs
 - Vermeiden von Tipparbeit mit USE
 - Datentypen für Tabellenspalten
 - Zusätze für Tabellenspalten
 - Anlegen einer Tabelle mit CREATE TABLE
 - Bedingtes Anlegen einer Tabelle mit IF EXISTS

- Festlegen des Primärschlüssels mit PRIMARY KEY
- Anzeigen aller Tabellen einer Datenbank mit SHOW TABLES
- Vertiefendes
 - Festlegen des Fremdschlüssels mit FOREIGN KEY
 - Festlegen der Constraints mit ON UPDATE und ON DELETE zur Wahrung der referenziellen Integrität
 - Entscheidung bzgl. ENUM
 - Performancebetrachtungen bzgl. DOUBLE und DECIMAL
 - Genauigkeitsbetrachtungen bzgl. DOUBLE und DECIMAL
 - Entscheidung bzgl. NOT NULL inkl. einer Performancebetrachtung
 - Die Drei-Schichten-Architektur und das SRP-Prinzip
 - Entscheidung bzgl. der zu verwendenden Engine (InnoDB und MyISAM)



Die Quelltexte dieses Kapitels stehen in der Datei `listing02.sql` (siehe [Listing 29.2 auf Seite 462](#), [Listing 29.38 auf Seite 585](#) und [Listing 29.23 auf Seite 537](#)).

Die Datenbank soll die Struktur wie in den ER-Modellen oben (siehe [Bild 3.1 auf Seite 42](#), [Bild 3.2 auf Seite 43](#) und [Bild 3.3 auf Seite 44](#)) beschrieben haben.

5.3.1 Welche Datentypen gibt es?

Aus der Programmierung wissen Sie sicherlich, dass man Daten in Variablen oder Objekten ablegt. Diese Variablen oder Objekte müssen einen Datentyp haben¹⁹. Der Datentyp legt fest, welcher Wertebereich zur Verfügung steht und wie die Werte kodiert werden.

In SQL gibt es auch Datentypen. Den Spalten wird beim Erstellen der Tabelle ein Datentyp zugewiesen. In [Abschnitt 26.1 auf Seite 397](#) sind auszugsweise die in MySQL möglichen Datentypen aufgelistet. Datentypen, die nicht SQL:2016-Standard sind, werden durch ein Sternchen (*) gekennzeichnet. Datentypen, die in MySQL nicht, aber in SQL:2016 vorkommen, werden hier nicht angegeben.

Jeder Datenbankhersteller liefert einen eigenen Satz von Datentypen wie hier die proprietären²⁰ in MySQL. Viele davon sind exotisch wie mehrdimensionale Arrays, andere sehr sinnvoll wie binäre Felder.



Hinweis: Die Austauschbarkeit von Daten nimmt mit der Verwendung proprietärer Datentypen ab. Überlegen Sie sich gut, ob die Daten mit anderen Systemen ausgetauscht werden müssen oder ob ein Wechsel des DBMS z.B. nach MS-SQLSERVER wahrscheinlich ist.

¹⁹ Selbst bei nicht typisierten Sprachen werden intern Datentypen vergeben.

²⁰ lat.: proprietas = Eigentümlichkeit

5.3.2 Wie legt man eine Tabelle an?



SQL:2016, MySQL/MariaDB, PostgreSQL, T-SQL

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



Hinweis: Dies ist nur eine Variante des Befehls; wir werden noch andere kennenlernen.

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

1. **Spaltenname:** Der 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 26.1 auf Seite 397](#).
3. **Zusätze:** Mit diesen kann man eine Spalte ausführlicher bestimmen. Eine Liste möglicher Zusätze finden Sie in [Abschnitt 26.1.8 auf Seite 406](#).

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

Versuchen wir nun, die Tabelle `adresse` (siehe [Bild 3.1 auf Seite 42](#)) zu erstellen:

```
1 mysql> CREATE TABLE adresse (
2     -> adresse_id      INT UNSIGNED AUTO_INCREMENT,
3     -> strasse          VARCHAR(255),
4     -> hnr              VARCHAR(255),
5     -> lkz              CHAR(2),
6     -> plz              CHAR(9),
7     -> ort              VARCHAR(255),
8     -> deleted          TINYINT UNSIGNED NOT NULL DEFAULT 0,
9     -> PRIMARY KEY(adresse_id)
10    -> );
11 ERROR 1046 (3D000): No database selected
```

Was ist hier schiefgelaufen? Die Fehlermeldung in [Zeile 11](#) sagt mir, dass keine Datenbank ausgewählt wurde. Dabei habe ich doch gerade die Datenbank `oshop` angelegt. Das bedeutet aber nicht, dass sich automatisch alle nachfolgenden Befehle auf diese Datenbank beziehen; schließlich gibt es mehrere Datenbanken auf dem Server, wie wir oben gesehen haben. Es gibt nun zwei Möglichkeiten:

- Man schreibt vor dem Tabellennamen durch einen Punkt getrennt den Datenbanknamen: `CREATE TABLE oshop.adresse (...)`. Dies entspricht eher der oben erwähnten Gleichstellung von Schema und Datenbank, ist auf Dauer aber umständlich zu schreiben.

- Man verwendet USE *datenbankname*, um die verwendete Datenbank festzulegen.



Hinweis: USE ist wie SHOW kein SQL-Befehl, sondern ein Befehl des MySQL-/MariaDB- und des MS SQL Clients. In anderen Systemen gibt es Entsprechungen wie beispielsweise in PostgreSQL \c für connect.

```
1 mysql> USE oshop;
2 Database changed
```

In [Zeile 2](#) erfolgt die Rückmeldung, dass nun die angegebene Datenbank verwendet wird. Führen Sie jetzt den CREATE TABLE noch mal aus.

Jetzt aber zur Tabelle selbst: Die Spalte `adresse_id` ist der Primärschlüssel. Dieser soll eine ganze Zahl sein. Da wir keine negativen Primärschlüsselwerte brauchen, verwenden wir hier den Zusatz `UNSIGNED`. Durch den Zusatz `AUTO_INCREMENT` nutzen wir einen Automatismus von MySQL/MariaDB. Jedes Mal, wenn ein neuer Datensatz hinzugefügt wird, wird ein interner Zähler um 1 erhöht und dieser Wert in das Feld `adresse_id` eingetragen. Bitte beachten Sie, dass es pro Tabelle nur einen `AUTO_INCREMENT` geben kann und dieser auch der Primärschlüssel sein muss.



Hinweis: `AUTO_INCREMENT` ist kein SQL-Standard, sondern ein MySQL-/MariaDB-Zusatz. SQL:2016 verwendet den Zusatz `GENERATED art AS IDENTITY` (siehe [Abschnitt 26.1.8 auf Seite 406](#)).

In den [Zeilen 3ff.](#) werden diverse `VARCHAR(255)`-Spalten deklariert. Grundsätzlich gilt, dass bei einem `VARCHAR` nur so viel Speicherplatz verbraucht wird²¹, wie man Text in die Spalte einfügt. Somit kann man ohne Speicherplatzverschwendug eine ausreichend großzügige obere Grenze angeben. Verwundern tut einen das schon, besonders bei Feldern wie der Hausnummer ([Zeile 4](#)). Aber aus der Perspektive des Platzverbrauchs ist es völlig egal, ob bei der Hausnummer `VARCHAR(255)` oder `VARCHAR(10)` steht, wenn Sie maximal zehnstellige Daten erwarten. Was aber, wenn aus irgendeinem Grund elf Stellen gebraucht werden? Bei `VARCHAR(10)` wird abgeschnitten, bei `VARCHAR(255)` nicht²².

Anders sieht das in den [Zeilen 5ff.](#) aus. Dort kann die Anzahl der benötigten Zeichen gut festgelegt werden. Das Länderkennzeichen ist per Definition immer zweistellig (DE, US etc.). Die Postleitzahl (*postal code*) ist mit 9 Zeichen groß genug, um auch ausländische Postleitzahlen aufnehmen zu können. Ich verwende hier einen zeichenbasierten Datentyp, da Postleitzahlen auch führende Nullen und Buchstaben enthalten können.

In [Zeile 8](#) wird die Löschmarkierung deklariert. Da sie nur wenige Zustände hat (in der Regel 0 oder 1), kann der kleinste ganzzahlige Datentyp `TINYINT` hier verwendet werden. Auch hier werden keine negativen Werte benötigt. Durch den Zusatz `NOT NULL` erzwingen wir eine Angabe. Dies ist hier evident, da wir immer klar wissen müssen, ob ein Datensatz

²¹ +2 Bytes zur Speicherung der Textlänge.

²² Ganz so egal ist es dann doch nicht. Für die Performance ist es wichtig, dass die durchschnittliche Zeilenlänge gut geschätzt werden kann. Sind in der Tabelle ausreichend Zeilen vorhanden, wird diese Schätzung anhand der tatsächlichen Zeilenlängen durchgeführt. Sind erst wenige Zeilen vorhanden, wird die Länge im `VARCHAR` in die Schätzung einbezogen. Weiterführendes finden Sie in [[SZT+09](#)].

noch aktiv ist oder nicht. Falls keine Angaben gemacht werden, soll der Wert 0 (= aktiv) verwendet werden.

Schauen wir nach, ob die Tabelle auch wirklich angelegt wurde:

```

1 mysql> SHOW TABLES;
2 +-----+
3 | Tables_in_oshop |
4 +-----+
5 | adresse         |
6 +-----+

```

Als zweite Tabelle wird die Tabelle kunde angelegt:

```

1 CREATE TABLE kunde (
2   kunde_id          INT UNSIGNED AUTO_INCREMENT,
3   nachname          VARCHAR(255),
4   vorname           VARCHAR(255),
5   rechnung_adresse_id INT UNSIGNED,
6   liefer_adresse_id INT UNSIGNED,
7   bezahlart          INT UNSIGNED NOT NULL DEFAULT 0,
8   art                INT UNSIGNED NOT NULL DEFAULT 0,
9   deleted            TINYINT UNSIGNED NOT NULL DEFAULT 0,
10  PRIMARY KEY(kunde_id)
11 );

```



Hinweis: Bitte achten Sie darauf, dass die Datentypen der Fremdschlüsseleinträge rechnung_adresse_id und liefer_adresse_id gleich denen des dazugehörigen Primärschlüssels sein müssen, da hier die gleichen Werte vorkommen.

In der [Zeile 7](#) wird die Art der Bezahlung durch einen INT kodiert. Die Bedeutung könnte beispielsweise so sein: 0 = unbekannt, 1 = Bankeinzug etc. Ähnliches gilt für die Kundenart: 0 = unbekannt, 1 = privat, 2 = geschäft etc.

Nun sind die anderen schnell gemacht:

```

1 CREATE TABLE bank (
2   bank_id           CHAR(12),
3   bankname          VARCHAR(255),
4   lkz               CHAR(2),
5   deleted           TINYINT UNSIGNED NOT NULL DEFAULT 0,
6   PRIMARY KEY(bank_id)
7 );
8
9 CREATE TABLE bankverbindung (
10  kunde_id          INT UNSIGNED,
11  bankverbindung_nr INT UNSIGNED,
12  bank_id           CHAR(12),
13  kontonummer       CHAR(25),
14  iban              CHAR(34),
15  deleted           TINYINT UNSIGNED NOT NULL DEFAULT 0,
16  PRIMARY KEY(kunde_id,bankverbindung_nr)
17 );

```

In der [Zeile 2](#) wird ein Primärschlüssel deklariert, der keine AUTO_INCREMENT, ja noch nicht einmal ein INT ist. Die bank_id ist das, was man landläufig als Bankleitzahl bezeichnet. Der Inhalt dieses Feldes ist also vorgegeben. In bestimmten Ländern kann die Bank-

leitzahl auch Bindestriche, Leerzeichen oder sogar Buchstaben²³ enthalten. Somit muss ein Datentyp für Zeichen verwendet werden. Analoges gilt für kontonummer und iban.

Auch die bankverbindung_nr ist keine AUTO_INCREMENT, da die Nummerierung für jeden Kunden (jede Kundennummer) neu begonnen wird. Man nennt solche Konstruktionen *Nummernkreis*. In der Zeile 16 wird ein zusammengesetzter Primärschlüssel deklariert (siehe [Definition 8 auf Seite 17](#)).



Aufgabe 5.4: Bauen Sie die drei CREATE TABLE auf den SQL:2016-Standard um. Beachten Sie dabei auch die Datentypen.

Aufgabe 5.5: Kontrollieren Sie, ob alle Tabellen auch angelegt sind. Finden Sie heraus, wie man die Tabellenstruktur durch einen Befehl herausbekommt.

5.3.3 Wann eine Aufzählung (ENUM) und wann eine neue Tabelle?

Die Kundenart (art) wird über einen ganzzahligen Datentyp kodiert. Ein solcher Nummerncode ist aber sehr unschön, da man sich ständig die Zahlenbedeutung merken muss. Die erste Idee wäre, eine neue Tabelle kundenart zu erstellen. Diese bestünde aus zwei Spalten: dem Primärschlüssel als laufende Zahl und der Art als Zeichenkette. Die Spalte art in kunde wäre dann ein Fremdschlüssel auf die neue Tabelle. Nun könnte ich beliebig viele Kundenarten erfassen und zuweisen.

Dagegen spricht aber der Aufwand, für vielleicht zwei bis fünf Kundenarten eine ganze Tabelle zu pflegen und später für die Aufbereitung auch wieder zusammenführen zu müssen. Zum Glück gibt es den Datentyp ENUM. Mit diesem kann man eine Liste möglicher Werte für eine Spalte angeben²⁴.

```

1 CREATE TABLE kunde (
2   kunde_id          INT UNSIGNED AUTO_INCREMENT,
3   [...]
4   art               ENUM('unb', 'prv', 'gsch') NOT NULL DEFAULT 'unb',
5   [...]
6 );
```

In der Zeile 4 wird die *Aufzählung* mit einem ENUM deklariert. Die Werte werden als Zeichenketten in einer Liste angegeben. Wie bei anderen Datentypen kann man eine Vorbelegung mit DEFAULT festlegen.

Nun kann der Spalte art nur einer der drei Werte aus der Aufzählung zugewiesen werden. Da die Werte der Aufzählung Zeichenketten sind, kann die Wertzuweisung ohne Dekodierungsfehler erfolgen, wenn man die Werte der Aufzählung selbsterklärend gestaltet, also nicht so wie ich hier in Zeile 4.

Ein weiterer Vorteil ist, dass der Datentyp ENUM intern wieder als INT abgespeichert wird. MySQL und MariaDB merken sich in den Tabelleninformationen, welche Zahl für welchen Aufzählungswert steht. Deshalb braucht in den Zeilen nur der entsprechende Zahlenwert abgespeichert werden.

²³ Z.B. das 'X' für die Prüfziffer 10

²⁴ In PostgreSQL wird dazu ein entsprechender Datentyp erzeugt (siehe Seite ??).

Eine Suche oder ein Vergleich auf Zahlen ist sehr viel schneller als eine auf Zeichenketten. Bitte machen Sie sich klar, dass der Vergleich zweier Zeichenketten im schlechtesten Fall (nämlich bei Gleichheit der Zeichenketten) verlangt, dass alle Zeichen einzeln verglichen werden. Schließlich könnte ja das letzte Zeichen doch noch unterschiedlich sein.

Bei zwei Zeichenketten der Länge n bedeutet das maximal n Vergleichsoperationen. Bei einem Datentyp ENUM wird der gesuchte Wert zuerst in eine Zahl kodiert. Mislingt dies, weiß ich sowieso, dass die beiden Werte ungleich sind. Habe ich nun die zum Suchwert passende Zahl, kann diese in den Zeilen verglichen werden. Ein Zahlenvergleich kann in der CPU in der Regel in einem Taktzyklus sehr schnell durchgeführt werden.

Ich habe die Kriterien in der [Tabelle 5.2](#) als Entscheidungshilfe zusammengefasst.

Tabelle 5.2 Entscheidungshilfe Datentyp ENUM

Kriterium	ENUM	Tabelle
Anzahl der Werte	wenige	viele
Änderungshäufigkeit	fast nie	damit ist zu rechnen
Neue Werte hinzufügen	nicht oft	oft
Verwendete Werte löschen	nie	möglich
Werte als Sortierkriterium	selten	oft

Das Sortierkriterium spielt hier deshalb eine Rolle, weil Spalten mit dem Datentyp ENUM nicht nach dem Aufzählungstext, sondern nach dem Zahlenwert sortiert werden. Am besten gibt man in der Aufzählungsliste die Aufzählungstexte gleich in der sortierten Reihenfolge an; dann stimmen Sortierung der Aufzählungstexte und der Zahlenwerte überein. Da die Sortierung nach Zahlen erheblich schneller ist als nach Texten, gewinnt man dadurch Performance.



Hinweis: Das Ändern, Hinzufügen oder Löschen von Elementen einer Aufzählung geschieht über ALTER TABLE ... MODIFY (siehe [Kapitel 8.3](#) ab [Seite 132](#)). ■

5.3.4 Wann ein DECIMAL, wann ein DOUBLE?

Sowohl DOUBLE [PRECISION] als auch DECIMAL sind dazu da, Zahlen mit Nachkommastellen zu verarbeiten (siehe [Tabelle 26.2 auf Seite 398](#)).

Der Datentyp DOUBLE kann einen großen Zahlenraum relativ ungenau abdecken. Abgespeichert werden die Daten binär nach der IEEE795-Norm²⁵. Die Verarbeitung dieser Zahlen erfolgt relativ schnell, da eigentlich alle in Frage kommenden Prozessoren die Verarbeitung von DOUBLE-Werten integriert haben.

Anders verhält es sich bei dem Datentyp DECIMAL. Dieser kann einen kleineren Zahlenraum relativ genau abdecken. Die Werte werden als Ziffernfolge – also wie ein CHAR() – abgespeichert. Dies erfordert, dass für die Verarbeitung die Ziffernfolge erst in eine Zahl

²⁵ Die IEEE795 ist meines Wissens nach aber nicht vollständig umgesetzt. So gibt es beispielsweise kein NaN.

umgewandelt werden muss. Die Verarbeitung selbst wird nicht durch die Prozessoren unterstützt und muss durch die SQL-Ausführungseinheit erfolgen. Diese softwarebasierte Berechnung ist naturgemäß langsamer als eine, die durch den Prozessor unterstützt wird.

Ich wollte mir diesen Effekt anhand einer einfachen Performancemessung anschaulich machen. Ich habe daher bei steigender Anzahl von Datensätzen einmal auf einem **DOUBLE**- und einmal auf einem **DECIMAL**-Feld eine Durchschnittsberechnung durchgeführt. Das Ergebnis können Sie in Bild 5.1 betrachten²⁶.

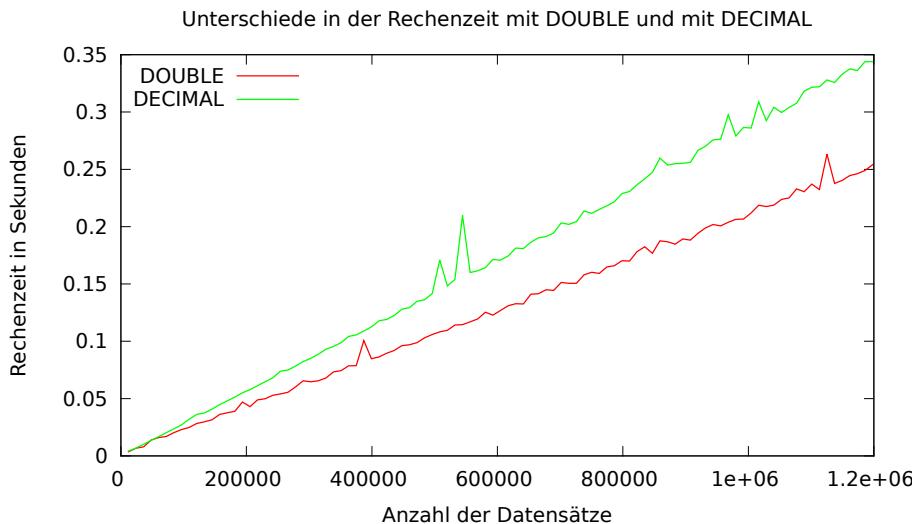


Bild 5.1 Unterschiedlicher Zeitverbrauch von Berechnungen bei **DOUBLE** und **DECIMAL**

Man erkennt, dass die Berechnung mit einem **DOUBLE** ab einer gewissen Anzahl zu verarbeitender Zahlen deutlich schneller ist als mit einem **DECIMAL**. Warum sollte man dann überhaupt **DECIMAL** verwenden?

Der wichtigste Vorteil von **DECIMAL** ist die Vermeidung von Rundungsfehlern. Obwohl es ein blödes Beispiel ist: Die Summe von 1000 einzelnen 0.001-Werten ergibt Folgendes²⁷:

```

1 +-----+
2 | FLOAT          | DECIMAL          |
3 +-----+-----+
4 | 1.0000000474974513 | 1.000000000000000 |
5 +-----+-----+

```

Zwar wäre hier der **DOUBLE** dem **FLOAT** vorzuziehen, aber ich wollte den Rundungsfehlerfehler aufzeigen, und der tritt bei einem **FLOAT** schneller auf als bei einem **DOUBLE**. Dasselbe Experiment mit **DOUBLE** liefert übrigens auch einen Rundungsfehler von $0.7 \cdot 10^{-15}$.

Die Argumente für die **DOUBLE/DECIMAL**-Entscheidung sind in Tabelle 5.3 zusammengefasst worden, wobei ich die Verwendung von **DECIMAL** bei kaufmännischen und fiskalischen Berechnungen besonders betonen möchte.

²⁶ Mehr zum Versuch: siehe Seite 439.

²⁷ Das Experiment finden Sie hier: Listing 27.2 auf Seite 443.

Tabelle 5.3 Entscheidungshilfe DOUBLE oder DECIMAL

Kriterium	DOUBLE	DECIMAL
Breite des Wertebereichs	groß	klein
Schnelligkeit der Berechnung	wichtig	nicht so wichtig
Rundungsfehler	tolerierbar	nicht tolerierbar
kaufmännische Berechnung	trifft nicht zu	trifft zu
fiskalische Berechnung	trifft nicht zu	trifft zu



Hinweis: Einige SQL Server bieten sogar eigene Datentypen für das Abspeichern von Währungswerten. So stehen in T-SQL die beiden Datentypen MONEY und SMALLMONEY zur Verfügung.



5.3.5 Wann verwendet man NOT NULL?

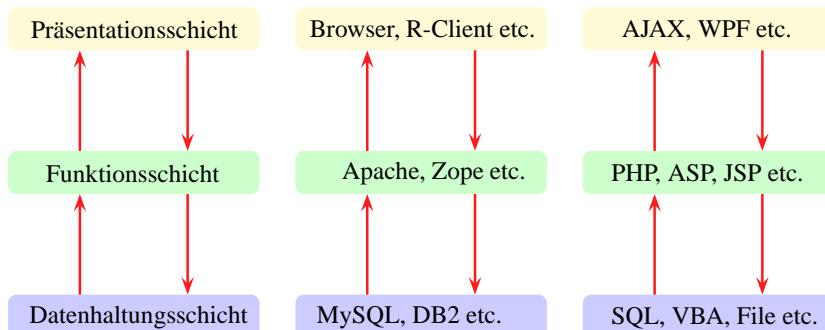
Der Wert NULL steht nicht für die Zahl 0, sondern für *nichts*. Steht in einer Zeile irgendwo NULL, bedeutet dies, dass hier keine Angaben über den Inhalt vorliegen.

Wird eine Spalte mit dem Zusatz NOT NULL deklariert (siehe [Abschnitt 26.1.8 auf Seite 406](#)), ist es verboten, zu einer Spalte keine inhaltlichen Angaben zu machen. Versuche, einen Datensatz mit fehlenden Angaben einzufügen oder eine Angabe durch NULL zu ersetzen, führen zu einer Fehlermeldung (hier mit der Spalte nachname):

```
1 ERROR 1048 (23000): Column 'nachname' cannot be null
```

In der Praxis stellt sich oft die Frage, ob man sogenannte Pflichtfelder wie beispielsweise die *E-Mail-Adresse* auch in der Tabellenspezifikation mit NOT NULL versehen sollte.

Die Antwort ist nicht so einfach: Grundsätzlich sind Datenbanken immer Teil einer oder mehrerer Anwendungen. Diese Anwendungen sollten nicht monolithisch, sondern zumindest nach der Drei-Schichten-Architektur (siehe [Bild 5.2](#)) entwickelt werden.

**Bild 5.2** Drei-Schichten-Architektur

Jede Schicht sollte danach nur die Aufgaben bearbeiten, für die sie zuständig ist (*Single Responsibility Principle*). Da stellt sich nun die Frage, wer die Plausibilitätsprüfungen – und

somit auch die Pflichtfeldprüfung – vornehmen muss. In trivialen Fällen sicherlich die Präsentationsschicht, ansonsten aber die Funktionsschicht. Aus der Perspektive der Datenhaltungsschicht ist die Eingabeplausibilisierung aber ein ALP²⁸.

Haben die oberen Schichten einen Datensatz geprüft und keine Fehler gefunden, steht es meines Erachtens nach der Datenhaltungsschicht nicht zu, diesen Datensatz abzulehnen. Schließlich würde dadurch eine fachliche Anforderung zweimal überprüft, was im Wartungsfall nicht so schön ist (Wartungsinstabilität).

Als Letztes sei angemerkt, dass eine Tabelle auch von mehreren Anwendungen verwendet werden kann, und nicht bei jeder muss eine bestimmte Spalte zwingend gefüllt sein.

Langer Rede kurzer Sinn: NOT NULL ist für Pflichtfelder keine gute Lösung.

Und jetzt kommt das große ABER. In [SJT⁺09] findet sich ein Hinweis darüber, dass NULL-Werte die Performance belasten. Das hat mir keine Ruhe gelassen, und ich habe mir dazu einen kleinen Versuch ausgedacht (Näheres siehe Seite 444). Das Ergebnis in Bild 5.3 ist für mich überraschend gewesen.

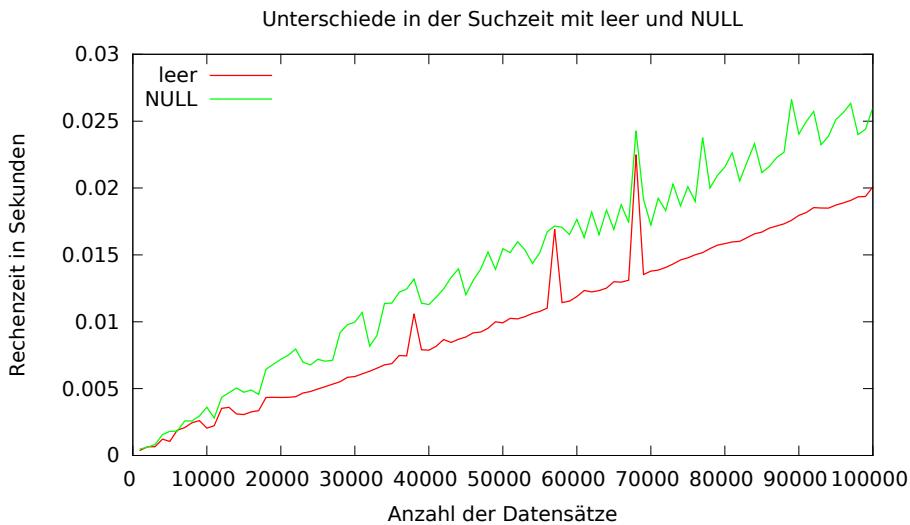


Bild 5.3 Performanceunterschied zwischen leer und NULL

Zwar lässt sich feststellen, dass die Suche nach bzw. der Vergleich mit leeren Inhalten wie dem Leer-String '' messbar schneller verläuft als die Suche nach bzw. der Vergleich mit NULL, aber *sooo* beeindruckend schneller auch wieder nicht²⁹.

Trotzdem, das Argument der Drei-Schichten-Architektur steht nun neben dem der Performance, und es muss eine Entscheidung gefällt werden. Hilfreich könnte dabei Tabelle 5.4 sein.

²⁸ ALP = Anderer Leute Problem

²⁹ Falls jemand einen signifikanten Fehler im Versuchsaufbau findet, lasse er es mich wissen. Für einen Versuchsaufbau, der den Effekt unter realistischen Bedingungen deutlicher ausfallen lässt, wäre ich sehr dankbar ;-).

Tabelle 5.4 Entscheidungshilfe NULL oder NOT NULL

Kriterium	NULL	NOT NULL
Größe der Tabelle	klein	groß
Häufigkeit der Suche auf der Spalte	selten	oft
Häufigkeit der NULL-Werte	selten	oft

Eine Möglichkeit, den beiden Anforderungen gerecht zu werden, ergibt sich durch den Zusatz DEFAULT (siehe [Abschnitt 26.1.8 auf Seite 406](#)), der seit SQL3 auch Standard ist. Werden die Spalten mit NOT NULL deklariert und mit einem DEFAULT versehen, wird dieser DEFAULT verwendet, wenn versucht wird, einen NULL-Wert einzufügen.

Das Anlegen der Tabelle adresse hätte dann folgendes Aussehen:

```

1 CREATE TABLE adresse (
2   adresse_id          INT UNSIGNED AUTO_INCREMENT,
3   strasse              VARCHAR(255) NOT NULL DEFAULT '',
4   hnr                  VARCHAR(255) NOT NULL DEFAULT '',
5   lkz                  CHAR(2) NOT NULL DEFAULT '',
6   plz                  CHAR(9) NOT NULL DEFAULT '',
7   ort                  VARCHAR(255) NOT NULL DEFAULT '',
8   deleted              TINYINT UNSIGNED NOT NULL DEFAULT 0,
9   PRIMARY KEY(adresse_id)
10 );

```

Jetzt können auch unvollständige Adressdaten aus der Funktionsschicht angenommen werden, und trotzdem werden keine NULL-Werte erzeugt.

5.3.6 Wie legt man einen Fremdschlüssel fest?

Zunächst sei erwähnt, dass man Fremdschlüssel nicht explizit durch einen SQL-Befehl markieren muss. Ein Fremdschlüssel wird zum Fremdschlüssel, wenn man dort Primärschlüsselwerte zwecks Referenz einträgt. Punkt.

Trotzdem hat man den Bedarf, den Fremdschlüssel besonders zu kennzeichnen. Dies erlaubt es beispielsweise Analysetools oder Optimieralgorithmen innerhalb des Servers, den Fremdschlüssel zu erkennen und zu berücksichtigen.

Am besten, wir schauen uns das am Beispiel der Tabelle kunde in MySQL/MariaDB an:

```

1 CREATE TABLE kunde (
2   kunde_id            INT UNSIGNED AUTO_INCREMENT,
3   nachname             VARCHAR(255) NOT NULL DEFAULT '',
4   vorname              VARCHAR(255) NOT NULL DEFAULT '',
5   rechnung_adresse_id  INT UNSIGNED,
6   liefer_adresse_id    INT UNSIGNED,
7   bezahlart             INT UNSIGNED NOT NULL DEFAULT 0,
8   art                  ENUM('unb', 'prv', 'gsch') NOT NULL DEFAULT 'unb',
9   deleted              TINYINT UNSIGNED NOT NULL DEFAULT 0,
10  PRIMARY KEY(kunde_id),
11  FOREIGN KEY (rechnung_adresse_id) REFERENCES adresse(adresse_id),
12  FOREIGN KEY (liefer_adresse_id)    REFERENCES adresse(adresse_id)
13 );

```

In [Zeile 11](#) wird mit dem Schlüsselwort FOREIGN KEY der Fremdschlüssel benannt. In Klammern steht der Spaltenname des Fremdschlüssels. Nach dem Schlüsselwort REFERENCES wird die Tabelle angegeben, auf die der Fremdschlüssel zeigt; in den runden Klammern steht der Name des Primärschlüssels, der die Werte für den Fremdschlüssel liefert. Bei diesem Beispiel werden sie als Constraints³⁰ der Tabellen deklariert.

Obwohl die [Zeilen 11ff](#) in SQL:2016 genauso funktionieren würden, hier die gleiche Tabelle nun in SQL:2016, aber durch Zusätze bei der Spaltendeklaration.

```

1 CREATE TABLE kunde (
2   kunde_id           INT GENERATED BY DEFAULT AS IDENTITY,
3   [...]
4   rechnung_adresse_id  INT REFERENCES adresse(adresse_id),
5   liefer_adresse_id    INT REFERENCES adresse(adresse_id),
6   [...]
7   deleted            INT CHECK(deleted IN (0, 1)),
8   PRIMARY KEY(kunde_id)
9 );

```

Bei den Deklarationen der Spalten `rechnung_adresse_id` und `liefer_adresse_id` wird durch den Zusatz REFERENCES ([Zeile 4ff](#)) der Fremdschlüssel festgelegt. In [Zeile 2](#) wird der Primärschlüssel deklariert. Sie sehen hier ein Beispiel für die Verwendung von GENERATED. In [Zeile 7](#) wird die Domäne (siehe [Definition 5 auf Seite 15](#)) von deleted auf die Menge (0, 1) eingeschränkt.

In MySQL/MariaDB wird es jetzt kompliziert. Wir müssen uns nun mit einer ihrer Besonderheiten beschäftigen: den *Engines*. Der MySQL oder MariaDB Server ist *eigentlich* nur so etwas wie eine Arbeitsumgebung. Er stellt Funktionen zum Verbindungsauflauf oder der Administration zur Verfügung. Er überprüft empfangene SQL-Befehle auf ihre Richtigkeit und baut die Befehle in ein standardisiertes Format um. Das Ergebnis eines Befehls wird auch vom Server verwaltet an den Client gesendet. Aber das Dazwischen, das Ausführen der Befehle, das machen die Engines³¹.

Die Idee hinter dieser Architektur ist, dass jeder einen solchen SQL-Verarbeiter programmieren kann, der auf seine Bedürfnisse angepasst ist. Diesen SQL-Verarbeiter kann man dann in den MySQL oder MariaDB Server als Engine integrieren.

In [Tabelle 24.3 auf Seite 386](#) finden Sie einen Auszug der Engines, die Teil der aktuellen MySQL- oder MariaDB-Distribution sind. Für eine genauere Beschreibung der Engines empfehle ich dringend, die entsprechenden Handbuchkapitel zu lesen. Weitere gute Übersichten sind für MySQL [[MyS19a](#)] und für MariaDB [[Mar19](#)].

Darüber hinaus gibt es viele Engines von kommerziellen Anbietern oder für spezielle Anforderungen. So ist beispielsweise mit der Engine *BrightHouse* das Verwalten von Datawarehouse-Objekten³² besonders gut möglich.

Da wir beim CREATE TABLE in MySQL oder MariaDB keine Engine angegeben haben, wird die InnoDB verwendet. Lassen wir uns das durch MySQL bzw. MariaDB mithilfe von SHOW CREATE TABLE bestätigen (siehe [Zeile 18](#)):

³⁰ engl.: Randbedingung

³¹ Die Abgrenzung ist nicht ganz so einfach, aber das soll uns hier und jetzt nicht stören.

³² Siehe [[Wik19d](#)]

```

1 mysql> SHOW CREATE TABLE kunde\G
2 **** 1. row ****
3     Table: kunde
4 Create Table: CREATE TABLE `kunde` (
5   `kunde_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
6   `nachname` varchar(255) COLLATE utf8_unicode_ci NOT NULL DEFAULT '',
7   `vorname` varchar(255) COLLATE utf8_unicode_ci NOT NULL DEFAULT '',
8   `rechnung_adresse_id` int(10) unsigned DEFAULT NULL,
9   `liefer_adresse_id` int(10) unsigned DEFAULT NULL,
10  `bezahlart` int(10) unsigned NOT NULL DEFAULT '0',
11  `art` enum('unb','prv','gsch') COLLATE utf8_unicode_ci NOT NULL DEFAULT 'unb
12  ',
13  `deleted` tinyint(3) unsigned NOT NULL DEFAULT '0',
14  PRIMARY KEY (`kunde_id`),
15  KEY `rechnung_adresse_id` (`rechnung_adresse_id`),
16  KEY `liefer_adresse_id` (`liefer_adresse_id`),
17  CONSTRAINT `kunde_ibfk_1` FOREIGN KEY (`rechnung_adresse_id`) REFERENCES `adresse`(`adresse_id`) ON UPDATE CASCADE,
18  CONSTRAINT `kunde_ibfk_2` FOREIGN KEY (`liefer_adresse_id`) REFERENCES `adresse`(`adresse_id`) ON DELETE SET NULL ON UPDATE CASCADE
19 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci

```

Warum war dieser Vortrag über Engines nötig? Weil die Unterstützung von Fremdschlüsseln sehr stark von der Engine abhängt. Bei MyISAM beispielsweise läuft es letztlich darauf hinaus, dass der Zusatz `FOREIGN KEY` zur Kenntnis genommen wird, aber keine praktische Konsequenz daraus folgt. Ganz anders bei der InnoDB.

Sie können die Engine auch beim Anlegen der Tabellen explizit angeben (siehe [Zeile 5](#)):

```

1 CREATE TABLE adresse (
2   adresse_id           INT UNSIGNED AUTO_INCREMENT,
3   [...]
4   PRIMARY KEY(adresse_id)
5 ) ENGINE=InnoDB;

```

In der Ausgabe des `SHOW CREATE TABLE` oben steht vor dem `FOREIGN KEY` in [Zeile 16](#) auf einmal das Schlüsselwort `CONSTRAINT`. Hier wird eine Randbedingung festgelegt, die von der InnoDB-Engine überwacht wird.

Die Überwachung bemerkt, ob sich der Wert oder das Vorhandensein des Primärschlüsselwerts, auf welchen der Fremdschlüssel verweist, verändert. Der Primärschlüsselwert kann durch `DELETE33` verschwinden oder durch ein `UPDATE34` verändert werden. Aber wie soll darauf reagiert werden? Ziel ist der Erhalt der referenziellen Integrität (siehe [Definition 22 auf Seite 33](#)).

Man kann einstellen, wie die Überwachung auf eine Verletzung der referenziellen Integrität reagieren soll (siehe [Tabelle 5.5 auf Seite 88](#)). Wird keine Angabe gemacht, gilt der Modus `RESTRICT`.



Hinweis: Das kaskadierende Löschen (CASCADE) ist mit großer Vorsicht anzuwenden und in vielen Unternehmen in den Programmierrichtlinien verboten. Bitte machen Sie sich klar, was da alles passieren kann.

³³ Siehe [Kapitel 9.3 auf Seite 150](#)

³⁴ Siehe [Kapitel 9.2 auf Seite 146](#)

Beispiel: Betrachten wir das Löschen eines Kunden (siehe Bild 5.4). Sie löschen einen Kunden. Dann werden alle Bestellungen, die sich auf den Kunden beziehen, gelöscht. Es werden alle Buchungen im Rechnungswesen zu dem Kunden gelöscht. Es wäre so, als hätte es diesen Kunden niemals gegeben! Das widerspricht natürlich nicht nur den gesetzlichen Aufbewahrungspflichten eines Unternehmers, sondern kann auch nicht im Sinne des Anwenders sein. Ausweg ist die schon angesprochene Spalte **deleted** (siehe Seite 33).

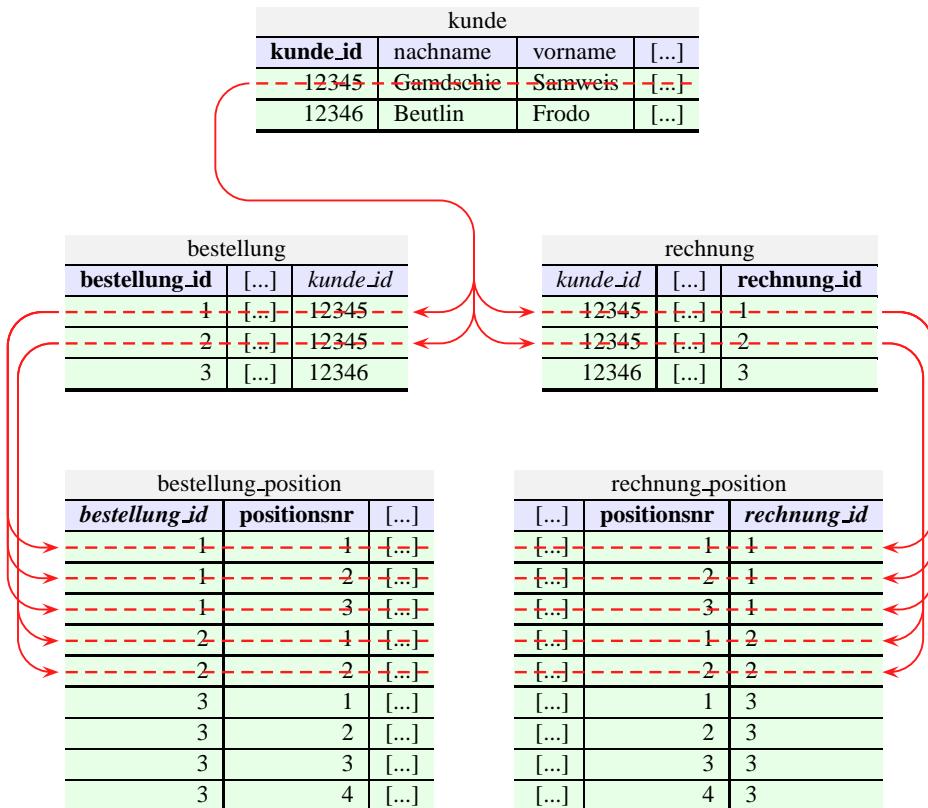


Bild 5.4 Löschweitergabe am Beispiel kunde



Hinweis: Das kaskadierende Ändern (CASCADE) ist mit großer Vorsicht anzuwenden und in vielen Unternehmen in den Programmierrichtlinien verboten.

Beispiel: Die Kundennummer ändert sich. Eine Änderungsweitergabe würde nun bedeuten, dass in allen Rechnungstabellen, Bestelltabellen, Buchungstabellen, Kontakttabellen usw. die Kundennummer angepasst wird. Nun ruft der Kunde an und meldet sich mit seiner ihm aus dem Rechnungsausdruck bekannten Kundennummer. Folge: Er wird im System nicht gefunden. Ausweg: Verwenden Sie nach Möglichkeit laufende Nummern als Primärschlüssel. Da diese keine Informationen enthalten, besteht eine verschwindend geringe Wahrscheinlichkeit, dass sich diese ändern muss.

Auf Seite 74 haben wir eine erste Version von CREATE TABLE kennengelernt. Diese muss nun erweitert werden:



SQL:2016, MySQL/MariaDB, PostgreSQL, T-SQL

```
CREATE TABLE tabellenname
(
    spaltenspezifikation
    [, spaltenspezifikation]*
    [, PRIMARY KEY(spaltenliste)]
    [, FOREIGN KEY(spaltenliste)
        REFERENCES zieltable ( primärschlüssel )
        [ON UPDATE constraintmodus]
        [ON DELETE constraintmodus]]*
) [tabellenoptionen]
;
```

Genug der Theorie: Die Tabelle kunde sieht in MySQL/MariaDB nun so aus:

```
1 CREATE TABLE kunde (
2     kunde_id          INT UNSIGNED AUTO_INCREMENT,
3     nachname          VARCHAR(255) NOT NULL DEFAULT '',
4     vorname           VARCHAR(255) NOT NULL DEFAULT '',
5     rechnung_adresse_id INT UNSIGNED NOT NULL,
6     liefer_adresse_id INT UNSIGNED,
7     bezahlart          INT UNSIGNED NOT NULL DEFAULT 0,
8     art                ENUM('unb', 'prv', 'gsch') NOT NULL DEFAULT 'unb',
9     deleted            TINYINT UNSIGNED NOT NULL DEFAULT 0,
10    PRIMARY KEY(kunde_id),
11    FOREIGN KEY (rechnung_adresse_id)
12        REFERENCES adresse(adresse_id)
13        ON UPDATE CASCADE
14        ON DELETE RESTRICT,
15    FOREIGN KEY (liefer_adresse_id)
16        REFERENCES adresse(adresse_id)
17        ON UPDATE CASCADE
18        ON DELETE SET NULL
19 ) ENGINE=InnoDB;
```

Die Rechnungsadresse kann eine Änderung des Primärschlüsselwerts problemlos akzeptieren, und deshalb wird hier der Modus CASCADE verwendet ([Zeile 13](#)). Analoges gilt für die Lieferadresse.

In der [Zeile 14](#) wird die Rechnungsadresse auf RESTRICT gesetzt. Hintergrund ist, dass es immer eine gültige Rechnungsadresse geben muss, ein Löschen der Rechnungsadresse somit nicht akzeptiert wird.

Anders bei der Lieferadresse ([Zeile 18](#)). Wenn die Lieferadresse gelöscht wird, wird von der Programmlogik die Rechnungsadresse verwendet. In Folge kann der Fremdschlüsselwert auf NULL gesetzt werden. Beachten Sie bitte dazu die [Zeile 6](#); es wird kein NOT NULL verwendet³⁵.

³⁵ Ich verletze hier bewusst die Drei-Schichten-Architektur (siehe [Bild 5.2 auf Seite 80](#)), da ich die Möglichkeiten der Sprache aufzeigen möchte.

In T-SQL funktioniert dieser Quelltext nicht! Neben Gründen, die sich aus dem T-SQL-Dialekt ergeben – beispielsweise kein UNSIGNED, IDENTITY anstelle von AUTO_INCREMENT etc. –, akzeptiert er die beiden Fremdschlüssel-Constraints nicht in der Form: *Introducing FOREIGN KEY constraint 'FK_kunde_liefer_ad_0F624AF8' on table 'kunde' may cause cycles or multiple cascade paths. Specify ON DELETE NO ACTION or ON UPDATE NO ACTION, or modify other FOREIGN KEY constraints.*

Hintergrund ist folgendes Problem: Durch kaskadierendes Löschen oder Ändern können zyklische oder mehrfache sich widersprechende Änderungen auf einem Datensatz ausgelöst werden. So könnte einmal eine Änderung und einmal eine Löschung auf demselben Datensatz innerhalb der gleichen Transaktion ausgelöst werden. Viele Datenbankprodukte ignorieren das Problem oder stecken viel Analyseaufwand hinein. Der MS SQL Server begegnet dem dadurch, dass er schlichterweise zählt, wie viele sich widersprechende Kaskaden auf einem Datensatz definiert werden. Bei mehr als einem wirft er die obige Fehlermeldung aus³⁶. Aufgrund dieser Einschränkung des Systems ist der T-SQL-Quelltext entsprechend abgeändert.



Aufgabe 5.6: Bauen Sie die anderen CREATE TABLE-Anweisungen so um, dass auch diese den Fremdschlüsselverweis und die Constraints enthalten.

Aufgabe 5.7: Erstellen Sie die CREATE TABLE für SQL:2016.

Tabelle 5.5 CONSTRAINT-Modi (* = MySQL-proprietär)

Modus	Bedeutung
RESTRICT	Es werden alle Lösch- und Änderungsoperationen auf den Primärschlüsselwert zurückgewiesen, wenn es einen dazugehörigen Fremdschlüsselwert gibt. Bei der InnoDB ist dies funktionsgleich mit NO ACTION.
CASCADE	Bei einer Löschoperation werden alle Zeilen in der Fremdschlüsseltabelle gelöscht, die auf den gelöschten Primärschlüsselwert verweisen. Bei einer Änderung des Primärschlüsselwerts werden die Fremdschlüsselwerte ebenfalls geändert.
SET NULL	Der Fremdschlüsselwert wird auf NULL gesetzt. Dies setzt natürlich voraus, dass der Fremdschlüssel nicht mit NOT NULL deklariert ist.
SET DEFAULT	Der Fremdschlüsselwert wird auf die serverseitig eingestellte Vorbelegung gesetzt. InnoDB akzeptiert, aber ignoriert diese Angabe.
NO ACTION	Es werden alle Lösch- und Änderungsoperationen auf den Primärschlüsselwert zurückgewiesen, wenn es einen dazugehörigen Fremdschlüsselwert gibt. Bei der InnoDB ist dies funktionsgleich mit RESTRICT.

³⁶ Eine schöne Diskussion darüber finden Sie unter [Sta18]

5.3.7 Wie kann man Tabellen aus anderen herleiten?

In einer anderen Anwendung möchte ich auch eine Tabelle `adresse` verwenden. Diese wird zwar andere Daten enthalten, soll aber genauso aufgebaut sein. Die Struktur meiner neuen Tabelle sollte sich daher aus der Tabelle `adresse` herleiten.

Für diesen Zweck gibt es in MySQL und MariaDB eine Variante des CREATE TABLE-Befehls:



MySQL/MariaDB

```
CREATE TABLE [datenbankname .]tabellename_neu
  [(]LIKE [datenbankname .]tabellename_alt[")]
;
```

```
CREATE TABLE verwaltung.adresse LIKE oshop.adresse;
```

In der Datenbank der neuen Anwendung `verwaltung` wird eine Tabelle mit dem Namen `adresse` angelegt, die die gleichen Spalten, Datentypen und Constraints wie die Tabelle `adresse` in der Datenbank `oshop` hat.



Hinweis: Der neue Tabellename muss nicht der gleiche wie der alte sein. Sie hätten der neuen Tabelle auch den Namen `wurstbrot` geben können. Bitte achten Sie dann darauf, dass der Name des Primärschlüssels nicht mehr passt (siehe [Zeile 4](#) unten). Entweder Sie benennen den Primärschlüssel um oder Sie nehmen das in Kauf.

```
1 mysql> SHOW CREATE TABLE wurstbrot\G
2 *************************** 1. row ****
3   Table: wurstbrot
4 Create Table: CREATE TABLE `wurstbrot` (
5   `adresse_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
6   `nachname` varchar(255) COLLATE utf8_unicode_ci NOT NULL DEFAULT '',
7   [...]
8   PRIMARY KEY (`adresse_id`)
9 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
```

Weitergehend und letztlich mächtiger ist das Konzept der Vererbung von Tabellen. In SQL:2016 kann man angeben, dass eine Tabelle die Spezifikation einer anderen Tabelle beinhaltet. SQL:2016 verwendet dazu das Schlüsselwort `UNDER`:



SQL:2016

```
CREATE TABLE tabellename
(
  tabellenspezifikation
) UNDER (elterntabelle[, elterntabelle]*)
;
```

Die Tabellenstruktur ist in SQL:2016 somit keine Kopie der `elterntabelle`, sondern eine Spezialisierung. Die Elterntabelle wird dabei um die eigenen Spalten und Constraints erweitert, also spezialisiert.

T-SQL kennt keine Vererbung, und in PostgreSQL steht dafür ein anderes Schlüsselwort zur Verfügung: `INHERITS`. Die Verwendung entspricht dem von `UNDER`:



PostgreSQL

```
CREATE TABLE tabellename
(
    tabellenspezifikation
) INHERITS (elterntabelle[, elterntabelle]*)
```

Die bei der Vererbung entstehenden Probleme wie Namensgleichheit in der Eltern- und Kindtabelle werden in den SQL:2016-Implementationen unterschiedlich gelöst. Bitte schauen Sie in das Handbuch.

5.3.8 Ich brauche mal eben kurz 'ne Tabelle!

Unser Kundenstamm ist mittlerweile sehr groß geworden. Für eine Marketingaktion möchte ich viele Auswertungen über unsere Bochumer Kunden machen. Natürlich kann ich die Auswertungen auf der Tabelle `kunde` direkt durchführen. Da die Bochumer aber nur 10 % der Kundschaft ausmachen, würden die Auswertungen immer sehr viele Daten besuchen, die gar nicht benötigt werden.

Eine naheliegende Idee ist es, nur die benötigten Daten in eine Tabelle auszulagern und diese Tabelle nach den Auswertungen wieder zu löschen. Auch hier steht eine Variante des `CREATE TABLE` zur Verfügung.



SQL:2016

```
CREATE [GLOBAL|LOCAL] TEMPORARY TABLE tabellename
tabellenspezifikation
;
```

MySQL/MariaDB, PostgreSQL

```
CREATE TEMPORARY TABLE tabellename
tabellenspezifikation
;
```

T-SQL

```
CREATE TABLE #tabellename
tabellenspezifikation
;
```

Die dabei angelegte Tabelle ist an eine bestimmte Existenzbedingung geknüpft. In MySQL und MariaDB ist diese Existenzbedingung die Verbindung³⁷. Wird die Verbindung geschlossen, wird automatisch die temporäre Tabelle entfernt. Sie ist auch nur in dieser

³⁷ Erinnern Sie sich noch an die Verbindungsnummer, die im MySQL/MariaDB Client angezeigt wird?

Verbindung sichtbar, sodass mehrere Verbindungen temporäre Tabellen gleichen Namens anlegen können.

In SQL:2016 wird ebenfalls eine temporäre Tabelle angelegt, aber man kann die Sichtbarkeit der Tabelle genauer festlegen. Wird die Option LOCAL verwendet, ist das Verhalten so, wie gerade bei MySQL/MariaDB beschrieben. Bei GLOBAL können auch andere Verbindungen (Sessions) die temporäre Tabelle verwenden.

**Aufgabe 5.8:** Was fällt hier auf?

```
mysql> CREATE TEMPORARY TABLE tmp_kunde LIKE kunde;
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> SHOW TABLES;
+-----+
| Tables_in_oshop |
+-----+
| adresse
| artikel
| artikel_nm_lieferant
| artikel_nm_warengruppe
| bank
| bankverbindung
| bestellung
| bestellung_position
| kunde
| lieferant
| rechnung
| rechnung_position
| warengruppe
+-----+
```

6

Indizes anlegen

■ 6.1 Index für Anfänger



Indizes vernünftig eingesetzt, beschleunigen viele der Datenbankzugriffe enorm.

- Grundkurs
 - Vorteil eines Index bei Suchen und Sortieren
 - Anzeigen von Indizes mit `SHOW INDEX FROM`
 - Erstellen eines Index mit `CREATE INDEX`



Die Quelltexte dieses Kapitels stehen in der Datei `listing01.sql` (siehe [Listing 29.3 auf Seite 467](#), [Listing 29.39 auf Seite 589](#) und [Listing 29.24 auf Seite 541](#)).

Sie brauchen ebenso die Bankleitzahldatei aus `blz_20120305.csv` im jeweiligen Verzeichnis.

Zunächst möchte ich hier eine kleine Performancemessung¹ vorstellen, die den Effekt von Indizes verdeutlichen soll. Die Darstellung in [Bild 6.1 auf der nächsten Seite](#) zeigt, dass die Such- und Sortierdauer mit Index nahezu konstant² bleibt. Die Such- und Sortierdauer ohne Index steigt hingegen linear an. Aus der Grafik ergibt sich unmittelbar, dass ein Suchen und Sortieren mithilfe von Indizes erheblich schneller abläuft als ohne.

Interessant ist, dass es für die ersten 100 Zeilen umgekehrt ist (siehe [Bild 6.2 auf der nächsten Seite](#)). Hier scheint der Verwaltungsaufwand für Indizes den Nutzen noch zu überlagern. Entscheidend ist aber der erhebliche Performancegewinn mit einem Index.

Wie kommt dieser Effekt zustande? Der *normale* Index organisiert die Daten einer Tabelle in Form von B-Bäumen. In [\[Wik19a\]](#) findet man eine übersichtliche Erklärung. Letztlich

¹ Der genaue Versuchsaufbau kann auf [Seite 446](#) nachgelesen werden.

² Was nur eine optische Täuschung ist. Suchoperationen lassen sich durch Indizes auf eine Größenordnung von $O(\log(n))$ reduzieren.

nutzt man aus, dass die Daten in einer Sortierung vorliegen und nur mit sehr wenigen Zugriffen garantiert, dass das zu suchende Element gefunden wird³.

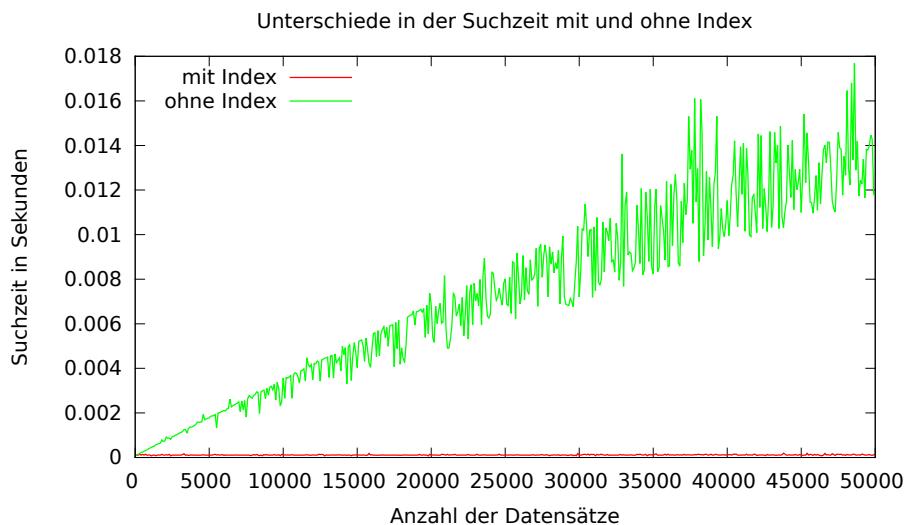


Bild 6.1 Zeitdauer von Such- und Sortieroperationen mit und ohne Index

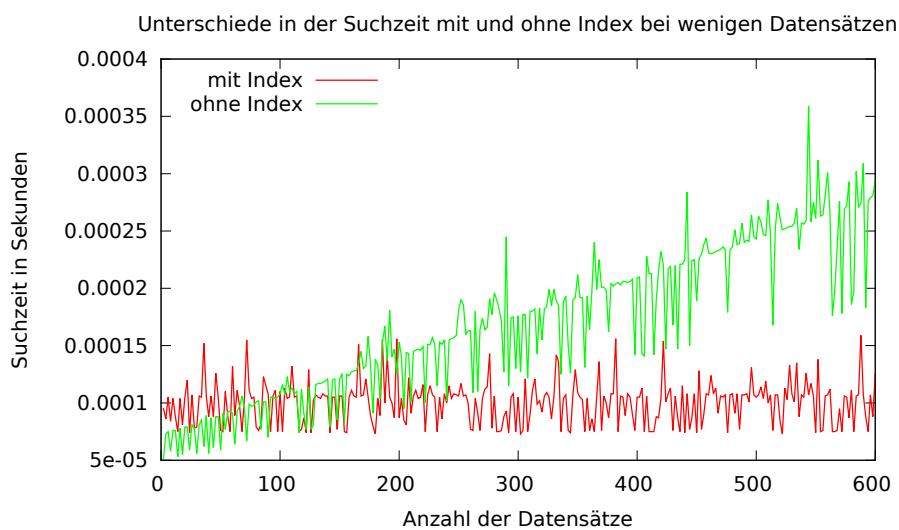


Bild 6.2 Zeitdauer von Such- und Sortieroperationen mit und ohne Index bei kleinen Mengen

³ Ich möchte hier darauf verzichten, den B-Baum genauer zu erläutern. Sein Verständnis bringt nicht zwingend ein besseres Anwendungswissen bzgl. SQL. Trotzdem empfehle ich die Literatur, da B-Bäume ein schönes Stück Informatik sind.

6.1.1 Wann wird ein Index automatisch erstellt?

Die Spalte einer Tabelle, nach der sicherlich am häufigsten gesucht wird, ist der Primärschlüssel. Besonders dann, wenn er in einer anderen Tabelle als Fremdschlüssel Verwendung findet.

Und weil dem so ist, wird automatisch mit der Deklaration zum Primärschlüssel ein Index angelegt. Hilfreich ist hier der Befehl `SHOW INDEX`:

```
1 mysql> SHOW INDEX FROM kunde\G
2 *************************** 1. row ****
3   Table: kunde
4   Non_unique: 0
5     Key_name: PRIMARY
6 Seq_in_index: 1
7   Column_name: kunde_id
8     Collation: A
9   Cardinality: 5
10    Sub_part: NULL
11    Packed: NULL
12    Null:
13  Index_type: BTREE
14    Comment:
15 Index_comment:
16   Expression: NULL
17   Visible: YES
18 *************************** 2. row ****
19   Table: kunde
20   Non_unique: 1
21     Key_name: rechnung_adresse_id
22 Seq_in_index: 1
23   Column_name: rechnung_adresse_id
24     Collation: A
25   Cardinality: 5
26    Sub_part: NULL
27    Packed: NULL
28    Null: YES
29  Index_type: BTREE
30    Comment:
31 Index_comment:
32   Expression: NULL
33   Visible: YES
34 *************************** 3. row ****
35   Table: kunde
36   Non_unique: 1
37     Key_name: liefer_adresse_id
38 Seq_in_index: 1
39   Column_name: liefer_adresse_id
40     Collation: A
41   Cardinality: 5
42    Sub_part: NULL
43    Packed: NULL
44    Null: YES
45  Index_type: BTREE
46    Comment:
47 Index_comment:
48   Expression: NULL
49   Visible: YES
```

Die Bedeutungen der Angaben sind:

- **Table:** Name der Tabelle, aus der der Index gebildet wird.
- **Non_unique:** Gibt an, ob die Werte des Index einmalig sind. Dies ist bei allen Schlüsseln der Fall (siehe [Definition 8 auf Seite 17](#)). Der Wert 0 steht dafür, dass keine Doublets vorkommen dürfen, der Wert 1 eben doch. Deshalb wird bei PRIMARY KEY-Indizes hier immer eine 0 stehen.
- **Key_name:** Name des Index. Dieser kann frei vergeben werden, wenn man den Index per CREATE INDEX anlegt. Bei Primärschlüsseln heißt dieser automatisch PRIMARY, weshalb dieser Name nicht manuell vergeben werden kann.
- **Seq_in_index:** Hier wird angegeben, an welcher Stelle die Spalte Column_name im Index steht. Bei Indizes, die aus mehreren Spalten zusammengesetzt sind (z.B. bei bankverbindung), gibt es auch eine Spalte mit der Sequenznummer 2.
- **Column_name:** Ein Index wird entweder durch einen Spaltennamen oder einen Ausdruck (Expression) festgelegt. Falls er durch einen Spaltennamen festgelegt wird, steht dieser hier, ansonsten NULL.
- **Collation:** Da ein Index die Daten in einer Sortierung vorhalten soll, kann diese angegeben werden. Bei MySQL können hier die Werte A für *aufsteigend*, D für *absteigend* oder NULL für *nicht sortiert* auftreten.
- **Cardinality:** Je mehr eindeutige Werte in einem Index vorkommen, desto sinnvoller ist es, danach zu suchen. Der Wert, der hier angegeben wird, schätzt die Anzahl der eindeutigen Werte. Der Wert muss somit nicht exakt sein. Je höher der Wert im Verhältnis zur Gesamtanzahl der Datensätze ist, desto höher ist die Wahrscheinlichkeit, dass für Suchoperationen der Index verwendet wird.
- **Sub_part:** Wird ein Index manuell erstellt, kann man angeben, dass er beispielsweise nur auf den ersten zehn Zeichen eines Nachnamens basieren soll. Ist dies nicht der Fall, steht hier NULL, ansonsten wird hier eine Zahl angegeben.
- **Packed:** Ein Schlüssel kann gepackt (komprimiert) werden. Steht hier NULL, ist er nicht komprimiert.
- **NULL:** Ist die Spalte nicht mit NOT NULL deklariert, steht hier YES, ansonsten nichts oder NO.
- **Index_type:** Es gibt verschiedene Typen von Indizes. Bei MySQL sind derzeit folgende möglich: BTREE, FULLTEXT, HASH und RTREE.
- **Comment:** Kommentare, die beispielsweise bei der Erstellung dem Index mitgegeben wurden.
- **Visible:** Legt fest, ob der Index dem Optimierer bekannt gemacht wird oder nicht.
- **Expression:** Ein Index wird entweder durch einen Spaltennamen (Column_name) oder einem Ausdruck festgelegt. Falls er durch einen Ausdruck festgelegt wird, steht dieser hier, ansonsten NULL.



Aufgabe 6.1: Untersuchen Sie mit SHOW INDEX die anderen Tabellen. Versuchen Sie, die Ergebnisse zu interpretieren und den Angaben im CREATE TABLE zuzuordnen. Warum werden wohl nicht nur bei den Primärschlüsseln automatisch Indizes angelegt?

6.1.2 Wie kann man einen Index manuell erstellen?



MySQL/MariaDB, PostgreSQL, T-SQL

```
CREATE INDEX indexname  
    ON tabellenname (spaltenname [, spaltenname]*)  
;
```



Hinweis: In SQL:2016 gibt es keinen Befehl zum Anlegen oder Löschen von Indizes. In vielen DBMSen, wie beispielsweise PostgreSQL und MS SQL Server, können Indizes aber mit der hier vorgestellten Syntax angelegt werden.

Zunächst stellt sich aber die Frage, wann man einen manuell erstellten Index braucht. Grundsätzlich dann, wenn man Such- und Sortieroperationen beschleunigen will⁴.

Die Such- und Sortieroperationen über die Primär- und Fremdschlüssel brauchen wir nicht weiter zu untersuchen, da die entsprechenden Indizes automatisch mit dem CREATE TABLE angelegt werden.

Gehen wir jetzt die Tabellen durch und überlegen dabei, welche Such- und Sortieroperationen wir zusätzlich gebrauchen könnten.

- Tabelle **adresse** kein neuer Index: -
- Tabelle **kunde** ein neuer Index auf: (**nachname, vorname**)
- Tabelle **bank** ein neuer Index auf: (**bankname**)
- Tabelle **bankverbindung** zwei neue Indizes auf: (**bank_id, kontonummer**) und (**iban**)
- Tabelle **artikel** ein neuer Index auf: (**bezeichnung**)
- Tabelle **warengruppe** ein neuer Index auf: (**bezeichnung**)
- Tabelle **lieferant** ein neuer Index auf: (**firmenname**)
- Tabelle **bestellung** ein neuer Index auf: (**kunde_id, datum**)
- Tabelle **bestellung_position** kein neuer Index: -
- Tabelle **rechnung** ein neuer Index auf: (**kunde_id, datum**)
- Tabelle **rechnung_position** kein neuer Index: -
- Tabelle **artikel_nm_warengruppe** kein neuer Index: -
- Tabelle **artikel_nm_lieferant** kein neuer Index: -

Es müssen somit neun neue Indizes angelegt werden. Fangen wir mit dem ersten an:

```
1 mysql> CREATE INDEX idx_kunde_nachname_vorname  
2      ->     ON kunde (nachname, vorname);
```

Man sollte bei der Benennung eines Index eine gewisse Konvention einhalten. Zuerst kommt der Präfix **idx** und dann der Tabellenname oder eine sinnvolle Abkürzung. Die folgenden Angaben sind die Indexspalten in der richtigen Reihenfolge.

Und zuletzt überprüfen wir, ob der Index auch angekommen ist:

⁴ Aber nicht nur dann: Gruppierungen, DISTINCT, ...

```

1 mysql> SHOW INDEX FROM kunde\G
2 *************************** 1. row ****
3 [...]
4 *************************** 2. row ****
5 [...]
6 *************************** 3. row ****
7 [...]
8 *************************** 4. row ****
9         Table: kunde
10        Non_unique: 1
11          Key_name: idx_kunde_nachname_vorname
12        Seq_in_index: 1
13          Column_name: nachname
14            Collation: A
15 [...]
16 *************************** 5. row ****
17         Table: kunde
18        Non_unique: 1
19          Key_name: idx_kunde_nachname_vorname
20        Seq_in_index: 2
21          Column_name: vorname
22            Collation: A
23 [...]

```

Der Teil 1. row bis 3. row ist uns schon aus obiger Betrachtung bekannt und hat sich nicht geändert. Es sind zwei weitere Einträge hinzugekommen, für jede Indexspalte eine. Sie können auch gut erkennen, dass die Angabe Seq_in_index in der [Zeile 12](#) für die erste Indexspalte 1 und in [Zeile 21](#) für die zweite Indexspalte 2 ist.

Noch zwei weitere Beispiele, und dann müssen Sie selbst ran:

```

1 CREATE INDEX idx_bank_bankname
2   ON bank (bankname);
3
4 CREATE INDEX idx_bankverbindung_bankid_kontonummer
5   ON bankverbindung (bank_id, kontonummer);
6
7 CREATE INDEX idx_bankverbindung_iban
8   ON bankverbindung (iban);

```



Aufgabe 6.2: Erstellen Sie die verbleibenden Indizes zu den Tabellen laut der Aufstellung auf [Seite 97](#).

Mit dieser Maßnahme sollten wir unsere Datenbank schon für die meisten Anforderungen gut vorbereitet haben.

■ 6.2 Und jetzt etwas genauer



Es lohnt sich, Indizes genauer zu betrachten:

- Vertiefendes
 - Index mit UNIQUE
 - Indexselektivität
 - Dubletten
 - Nachteile durch Ausbalancieren der B-Bäume
 - Löschen eines Index mit DROP INDEX
 - Aktivieren und Deaktivieren mit ENABLE KEYS und DISABLE KEYS

Der Befehl CREATE INDEX kann ein bisschen mehr als das, was in der Version auf [Seite 97](#) angegeben wurde:



MySQL/MariaDB, PostgreSQL, T-SQL

```
CREATE [UNIQUE] INDEX indexname
    ON tabellenname (spaltenangabe, [spaltenangabe]*)
;
spaltenangabe:
spaltenname [(länge)] [ASC|DESC]
```

Der *spaltenangabe* kann man also noch eine Längenangabe und eine Sortierreihenfolge mitgeben. Im Folgenden wird auf verschiedene Optionen des Befehls eingegangen.

6.2.1 Wie kann ich die Schlüsseleigenschaft erzwingen?

Nach [Definition 8 auf Seite 17](#) müssen bei Schlüsseln Spaltenwerte eindeutig sein. Bei Primärschlüsseln wird dies durch PRIMARY KEY automatisch vom Server kontrolliert. Aber was ist mit anderen Schlüsselkandidaten?

Hier hilft die Option UNIQUE. Für unsere Tabelle bankverbindung gilt beispielsweise, dass die Spalte iban ein Schlüssel ist.

```
1 CREATE UNIQUE INDEX idx_bankverbindung_iban
2   ON bankverbindung (iban);
```

Würden Sie nun versuchen eine Zeile hinzuzufügen, die eine schon vorhandene IBAN beinhaltet, erhielten Sie folgende Fehlermeldung:

```
1 ERROR 1062 (23000): Duplicate entry '1' for key 'idx_bankverbindung_iban'
```



Aufgabe 6.3: Interpretieren Sie die Fehlermeldung. Verwenden Sie ggf. eine Online-Hilfe.

6.2.2 Wie kann ich Dubletten verhindern?

Was ist eine Dublette? Schnell eine Definition dazu:



Definition 31: Dublette

Sind zwei Zeilen einer Tabelle inhaltlich gleich, spricht man von einer *Dublette*.

Die inhaltliche Gleichheit muss sich nicht zwingend durch die Gleichheit aller Spaltenwerte ergeben. Es reicht, dass signifikante Angaben inhaltlich identisch sind.

Eine Dublette liegt insbesondere dann vor, wenn die Werte der signifikanten Spalten nicht nur inhaltlich gleich sind, sondern auch syntaktisch.

Na, das hilft aber weiter ... doch langsam, es ist nicht so schwer, wie es sich anhört. Nehmen wir mal an, in unserer Tabelle `adresse` gäbe es zwei Zeilen, die genau die gleichen Angaben in den Spalten `strasse`, `hnr` und `plz` hätten. Unterschiede gäbe es nur in den Spalten `adresse_id` und `ort`. Stellte sich doch die Frage, ob das nicht eigentlich die gleiche Adresse wäre?

Da der Ort ungenauer als die Postleitzahl ist (z.B. einmal mit Ortsteilangabe und einmal ohne), kann man sicherlich davon ausgehen, dass diese beiden Zeilen auf die gleiche Adresse verweisen. Überprüfen wir dies anhand der **Definition 31**: Wir haben keine Gleichheit in allen Spaltenwerten, aber bei den *signifikanten* Angaben wie Postleitzahl etc. Wir hätten also eine Dublette.

Grundsätzlich sind Dubletten kein Beinbruch, wenn sie nicht überhandnehmen oder die Verarbeitung einer Adresse nicht zu teuer ist. Damit ist nicht die DV-technische Verarbeitung gemeint. Stellen Sie sich vielmehr vor, es muss ein Postversand bezahlt werden.

Neben der Möglichkeit, durch eigene Programme die Daten auf Dubletten zu überprüfen, könnte man auch die Annahme auf Seiten der Tabelle verweigern. Man baut einen Index auf alle signifikanten Felder mit der Option `UNIQUE`. Da ein Indexschlüssel in MySQL maximal 3072 Bytes lang sein darf, legen wir sinnvolle Spaltenlängen fest. Für unser Beispiel sähe dies wie folgt aus:

```
1 CREATE UNIQUE INDEX idx_adresse_dublette  
2   ON adresse (strasse(100), hnr(100), plz);
```



Hinweis: Ich möchte aber darauf hinweisen, dass diese Vorgehensweise nicht zu extensiv eingesetzt werden sollte. Machen Sie einen Kosten-Nutzen-Vergleich.

Wenn Indizes so ein Segen sind, warum indizieren wir dann nicht vorsorglich so viele Spalten und Spaltenkombinationen wie möglich – am besten noch in verschiedenen Sortierreihenfolgen?

Nun, weil alles seinen Preis hat. Wir wissen, dass Such- und Sortieroperationen durch einen Index erheblich beschleunigt werden können. Aber was ist mit Einfügen, Ändern und Lösen? Grundsätzlich muss einem klar sein, dass ein Index parallel zur eigentlichen Tabelle mit gepflegt werden muss.

- *Einfügen*: Wenn neue Datensätze hinzukommen, muss für jeden Datensatz die neue Position innerhalb der Indizierung gefunden werden. Falls es dabei zu einem nicht ausbalancierten B-Baum kommt, muss dieser umgebaut werden⁵.
- *Löschen*: Wenn ein Datensatz gelöscht wird, muss dieser auch aus dem Index entfernt werden. Auch dabei kann es passieren, dass der B-Baum ins Ungleichgewicht kommt und umgebaut werden muss.
- *Ändern*: Falls die Änderung ein Feld betrifft, welches Teil des Index ist, kann auch dadurch ein Umbau notwendig sein (Mischung zwischen Einfügen und Löschen).
- Zusätzlich verbraucht ein Index auch noch Speicherplatz; ein glücklicherweise immer unwichtiger werdendes Argument.

Ist aber die Einfüge-Operation wirklich messbar langsamer? Da ist wohl mal wieder eine Performancemessung fällig: Es werden jeweils 20.000 Datensätze in eine Tabelle ohne Index und eine mit Index importiert.

In Bild 6.3 erkennt man deutlich, dass das Einfügen mit Index mehr Zeit verbraucht als ohne⁶.

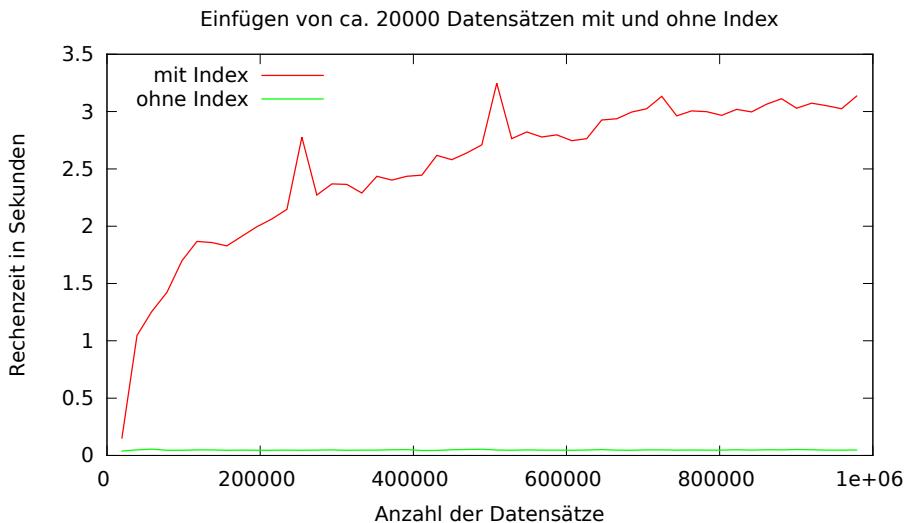


Bild 6.3 Zeitdauer Einfüge-Operationen mit und ohne Index

Sie müssen deshalb abwägen, ob und welche Indizes Sie verwenden wollen. Die Tabelle 6.1 auf der nächsten Seite soll Ihnen bei der Entscheidung helfen.

Lässt sich das Einfügen, Ändern oder Löschen großer Datenmengen zeitlich eingrenzen, dann kann es sinnvoll sein, für die Dauer dieser Operationen den Index abzuschalten und anschließend neu zu erstellen. Das Einfügen, Ändern und Löschen kann dadurch schnell verarbeitet werden, und anschließend sind wegen der Indizes Such- und Sortieroperationen wieder schnell verarbeitbar.

⁵ Siehe [Wik19a]

⁶ Ich habe hier die MyISAM verwendet. Bei der InnoDB war das Ergebnis überraschend anders. Näheres siehe Abschnitt 27.5 auf Seite 449

Tabelle 6.1 Entscheidungshilfe Index oder nicht

Kriterium	Index: ja	Index: Nein
Suchhäufigkeit	oft	selten
Änderungsmenge pro Zeiteinheit	wenig	viel
Sortierhäufigkeit	oft	selten
Gruppierungsmerkmal	oft	selten
Löschenmenge pro Zeiteinheit	wenig	viel
Wert als Fremdschlüssel verwendet	oft	selten
Einfügemenge pro Zeiteinheit	wenig	viel

6.2.3 Was bedeutet Indexselektivität?

Wir haben den Index bei der Dublettenüberprüfung in der Länge eingeschränkt. Ohne eine solche Einschränkung wird der gesamte Inhalt der Spalte in die Indexbildung einbezogen.

Bei numerischen Datentypen (siehe die Abschnitte [26.1.1.1 auf Seite 397](#) und [26.1.1.2 auf Seite 398](#)) kann die gesamte Spalte verwendet werden, da der Vergleich von Zahlen sehr schnell – oft in einem Taktzyklus der CPU – durchgeführt werden kann.

Bei zeichenbasierten Datentypen (siehe [Tabelle 26.1.2 auf Seite 399](#)) muss Zeichen für Zeichen verglichen werden. Besteht der zu suchende String aus vielen Zeichen und sind die indizierten Elemente ebenfalls lang, ergeben sich dadurch lange Suchzeiten.

Es stellt sich nun die Frage, ob man nicht die ganze Länge eines Feldes indiziert, sondern nur die ersten n Zeichen.

Ein einfaches Beispiel: In einer VARCHAR-Spalte gibt es die Werte Aachen, Berlin und Bochum. Ein Index dieser Spalte, der über die ganze Länge geht, würde 18 Zeichen speichern, um drei Werte zu unterscheiden. Ein Index über das erste Zeichen müsste nur drei Zeichen speichern, könnte aber schon/nur zwei Werte eindeutig identifizieren.



Definition 32: Indexselektivität

Die Indexselektivität S ergibt sich aus der Anzahl der unterscheidbaren Indexeinträge U und der Anzahl der Zeilen der Tabelle A wie folgt:

$$S = \frac{U}{A}$$

Demnach ist die Selektivität unseres Beispiels bei voller Länge: $S = \frac{U}{A} = \frac{3}{3} = 1$ und bei der Länge 1: $S = \frac{U}{A} = \frac{2}{3} = 0, \overline{6}$.

Die Indexselektivität von 1 ist somit das Optimum, da es bedeutet, dass jede Zeile der Tabelle durch den Index identifiziert werden kann. Unser Beispiel liefert aber auch eine Indexselektivität von 1, wenn man nur die ersten beiden Zeichen nimmt. Offensichtlich können diese beiden Zeichen viel schneller verglichen und abgespeichert/geladen werden als bei voller Länge.

Kann ich also durch einen kleineren und damit schnelleren Index genau die gleiche Ergebnisqualität bekommen? Ich möchte das Ganze an einem echten Beispiel näher untersu-

chen. Die Datei `blz_20120305.csv` enthält die Liste aller deutscher Banken mit Namen und Bankleitzahl zum Zeitpunkt 03.05.2012⁷.

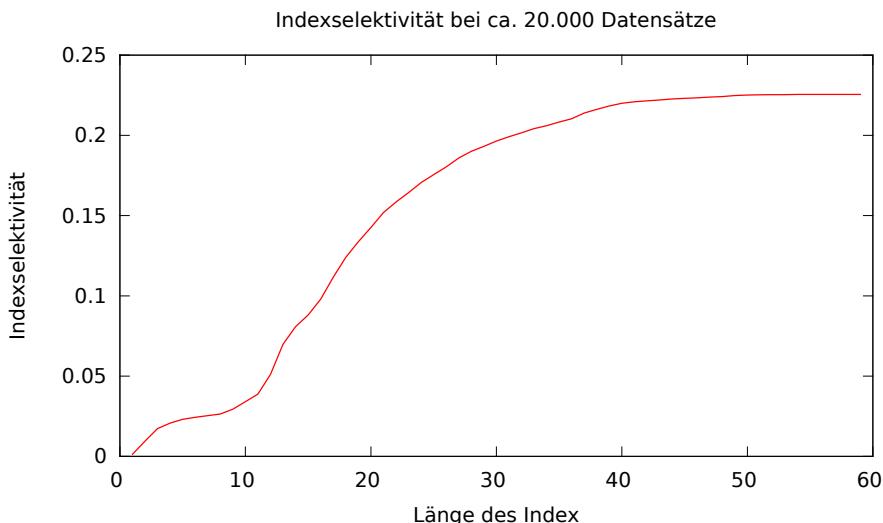


Bild 6.4 Indexselektivität am Beispiel Bankbezeichnung

Die maximale Länge einer Bankbezeichnung ist 62. In Bild 6.4 können Sie sehen, wie sich die Indexselektivität bei diesem realen Beispiel gestaltet⁸. Leider erreichen wir die Indexselektivität von 1 bei Weitem nicht. Nach kurzem Nachdenken wird klar, dass dies auch nur bei Schlüsseln zu erwarten ist. Noch viel spannender ist aber, dass wir bis zu einer Indexlänge von ca. 30 einen steilen Anstieg der Indexselektivität haben und ab diesem Wert nur noch ein sehr flaches Anwachsen erkennen können.

Mit anderen Worten, ab 30 müssen wir uns geringe Verbesserungen der Indexselektivität mit einer teuren Verlängerung des Index erkaufen. Schlussfolgerung: Wir beschränken den Index auf eine Länge von 30 Zeichen:

```

1 mysql> CREATE INDEX idx_testdaten_bank_bezeichnung
2      -> ON testdaten_bank (bezeichnung(30));
3
4 mysql> SHOW INDEX FROM testdaten_bank\G
5 *************************** 1. row ****
6   Table: testdaten_bank
7   Non_unique: 1
8   Key_name: idx_testdaten_bank_bezeichnung
9   Seq_in_index: 1
10  Column_name: bezeichnung
11  Collation: A
12  Cardinality: 194
13  Sub_part: 5265
14  [...]

```

⁷ Die Originalquelle [Bun16] ist aktuell – 07.2019 – so nicht mehr erreichbar.

⁸ Die genaue Messung können Sie in Abschnitt 27.6 auf Seite 452 nachlesen.

In der [Zeile 13](#) erkennen Sie die Einschränkung auf die ersten 30 Zeichen. MySQL schätzt, dass dies bei 19575 Zeilen zu 5265 Indexeinträgen führt ([Zeile 12](#))⁹.

Es ist somit sinnvoll, sich bei hinreichend großen Datenmengen mit der Indexselektivität zu beschäftigen, wenn die Indexfelder zeichenbasierte Datentypen enthalten.

6.2.4 Wie kann man einen Index löschen?



MySQL/MariaDB

```
DROP INDEX indexname ON tabellenname
;
```

PostgreSQL, T-SQL

```
DROP INDEX [IF EXISTS] indexname ON tabellenname
;
```

Der Index – und damit seine Spezifikation – wird vollständig entfernt. Möchte man später den Index wieder verwenden, muss man mit `CREATE INDEX` diesen wieder erstellen.



MySQL/MariaDB

```
ALTER TABLE tabellenname {ENABLE|DISABLE} KEYS
;
```

Der Nachteil des `DROP INDEX` ist, dass man die Spezifikation des Index verliert. Will man den Index endgültig löschen, ist der Befehl gut; will aber nur für umfangreiche Einfüge-, Änderungs- oder Löschoperationen den Index deaktivieren, braucht man `DISABLE KEYS`. Für die Tabelle `bankverbindung` sähe dies wie folgt aus:

```
1 ALTER TABLE bankverbindung DISABLE KEYS;
```

Aktivieren kann man den Index mit `ENABLE KEYS`. Für die Tabelle `bankverbindung` sähe dies wie folgt aus:

```
1 ALTER TABLE bankverbindung ENABLE KEYS;
```

Doch jetzt die schlechte Nachricht: Diese Option steht nur für bei MyISAM zur Verfügung; die InnoDB liefert Warnungen, die man sich mit `SHOW WARNINGS\G` anschauen kann:

```
1 **** 1. row ****
2 Level: Note
3 Code: 1031
4 Message: Table storage engine for 'bankverbindung' doesn't have this option
```

Auch werden nur die Indizes ausgeschaltet, die nicht mit `UNIQUE` erstellt wurden.

⁹ Interessanterweise schätzt MariaDB hier 3908.

7

Werte in Tabellen einfügen

■ 7.1 Daten importieren



Massenhaft neue Daten und wie man sie in Tabellen verstaut

- Grundkurs
 - CSV-Dateien
 - CSV-Dateien mit LOAD DATA INFILE importieren
 - Tabelleninhalte komplett ausgeben
- Vertiefendes
 - Spezialfälle von CSV-Dateien
 - CSV-Feldern den richtigen Spalten zuordnen
 - Ausdrücke für Spalteninhalte während des Imports festlegen
 - Kollisionsstrategien bei Gleichheit der Primärschlüsselwerte
 - Binäre Daten einfügen



Die Quelltexte dieses Kapitels stehen in der Datei `listing04.sql` (siehe [Listing 29.4 auf Seite 468](#), [Listing 29.40 auf Seite 592](#) und [Listing 29.25 auf Seite 543](#)).

Sie brauchen auch die Dateien `artikel01.csv` und `artikel02.csv`.

Unser Online-Shop braucht Daten, damit er arbeiten kann. Diese Daten können auf verschiedenen Wegen in die Tabellen gelangen:

- Die Daten werden aus fremden Datenquellen im CSV-Format importiert.
- Die Daten werden manuell angelegt.
- Die Daten werden aus einer Tabelle in die andere kopiert.

Alle drei Methoden sollen hier vorgestellt werden; fangen wir mit dem Import an.

7.1.1 Das CSV-Format

Die Artikelstammdaten für unseren Online-Shop werden nur selten manuell eingegeben, da diese in der Regel durch externe Anbieter zur Verfügung gestellt werden. Wenn Sie beispielsweise Kinderspielzeug anbieten wollen, bekommen Sie von den Herstellern Artikelstammdaten inklusive Fotos, z.B. per CD oder Online-Download.

Diese Daten liegen in einem CSV-Format¹ vor und müssen in die Artikeltabelle importiert werden.



Definition 33: CSV-Format

Das *CSV-Format* (Character Separated Values) ist ein anwendungsneutrales Austauschformat für nicht-binäre Daten. Die Daten sind wie folgt strukturiert:

- Ein Datensatz endet mit den beiden Steuerzeichen \r\n.
- Bei der letzten Datenzeile kann das Endezeichen entfallen.
- Die erste Zeile kann optional die Feldnamen enthalten.
- Es gibt ein Trennzeichen, welches die optionalen Feldnamen in der ersten Zeile und die Werte in den Datensätzen voneinander trennt. Dieses Trennzeichen ist ein Komma.
- In den Zeilen muss immer die gleiche Anzahl von Trennzeichen verwendet werden.
- Werte können durch Hochkommas ' oder Gänsefußchen " eingeschlossen werden.

Die beiden Steuerzeichen \r\n sind der ASCII-Code 0x0D und 0x0A. Unter Windows-Systemen entspricht dies dem Zeilenumbruch. Auf UNIX-artigen Systemen ist der Zeilenumbruch durch nur ein Zeichen gekennzeichnet: \n und unter MAC: \r. Grundsätzlich ist aber das Ende des Datensatzes mit jedem beliebigen Zeichen markierbar, sofern es sich eindeutig von den Daten und anderen Steuerzeichen unterscheiden lässt. Man verwendet deshalb gerne den Zeilenumbruch, weil sich die Daten dadurch leichter in einem Editor darstellen lassen.

Das Trennzeichen kann jedes beliebige Zeichen sein, sofern es sich eindeutig von den Daten und anderen Steuerzeichen unterscheiden lässt. Neben dem Komma , sind das Semikolon ;, das kaufmännische Und &, der Lattenzaun #, der Slash /, das Leerzeichen und der Tabulator sehr beliebt.

Dass in jeder Zeile immer die gleiche Anzahl von Trennzeichen stehen muss, bedeutet letztlich, dass jede Zeile immer gleich viele Felder enthält. Leere Werte werden beispielsweise so , , dargestellt.

Das Einschließen von Werten mit dem Hochkomma ist dann notwendig, wenn die Werte unvermeidbar auch Sonderzeichen oder das Trennzeichen enthalten können.

Ich möchte nun die Artikelstammdaten importieren. Die Datei artikel01.csv sieht so aus:

```
1 artikel_id;bezeichnung;einzelpreis;waehrung
2 7856,Silberzwiebel;0.41;EUR
```

¹ Siehe [Sha05]

```
3 7863;Tulpenzwiebel;3.29;EUR
4 9010;Schaufel;15.00;USD
5 9015;Spaten;19.90;EUR
6 3001;Papier (100);2.30;EUR
7 3005;Tinte (gold);55.70;EUR
8 3006;Tinte (rot);6.20;EUR
9 3010;Feder;5.00;EUR
```

Trennsymbol ist das Semikolon, und das Ende des Datensatzes wird mit \n markiert.

7.1.2 LOAD DATA INFILE

Diese Daten wollen wir natürlich nicht per Hand in die Tabelle artikel einfügen. Hier hilft der Befehl LOAD DATA INFILE, der hier verkürzt angegeben wird:



```
MySQL/MariaDB
LOAD DATA [LOCAL] INFILE 'dateiname'
  INTO TABLE tabellenname
  [FIELDS
    [TERMINATED BY 'zeichenfolge']]
  [LINES
    [TERMINATED BY 'zeichenfolge']]
;
```



Hinweis: In SQL:2016 gibt es keine Anweisung zum Importieren von externen Daten. In anderen DBMSen gibt es Entsprechungen: z.B. COPY in PostgreSQL oder BULK INSERT in MSSQL.

Zunächst muss angegeben werden, aus welcher CSV-Datei die Daten importiert werden sollen. Es reicht aber nicht aus, den Dateinamen ggf. mit seinem Pfad anzugeben. Die Datei artikel.csv kann schließlich an zwei Orten liegen: auf dem Rechner, wo der MySQL oder MariaDB Server läuft, oder auf dem Client, über den der Befehl zum Import gerade versendet wird. Dies wird durch die Angabe oder das Weglassen von LOCAL entschieden. Wird LOCAL angegeben, werden die Daten zuerst vom Client zum Server transferiert und erst dann importiert. Lässt man LOCAL weg, sucht er die Datei in einem Serverpfad.

Die Zieltabelle ist artikel, und das Trennzeichen ist das Semikolon. Die Datensätze werden nur durch \n getrennt, sodass sich folgende erste Version ergibt:

```
1 mysql> LOAD DATA LOCAL INFILE 'artikel01.csv'
2     ->     INTO TABLE oshop.artikel
3     ->     FIELDS
4     ->     TERMINATED BY ';'
5     ->     LINES
6     ->     TERMINATED BY '\n'
7     -> ;
8 ERROR 1148 (42000): The used command is not allowed with this MySQL version
```

Unser erster Versuch ging glatt schief²: Die Fehlermeldung in [Zeile 8](#) kann mehrere Gründe haben. Der wahrscheinlichste ist, dass es bei Ihrer Installation nicht gestattet ist, dass der Client lokale Dateien zwecks Import hochlädt.

Versuchen Sie es erneut, nachdem Sie den MySQL Client mit `--local-infile` gestartet haben (siehe [Seite 390](#)):

```

1 ralf@localhost:~/>mysql -uroot --local-infile
2 mysql> SET GLOBAL local_infile = 1;
3 Query OK, 0 rows affected (0.00 sec)
4
5 mysql> LOAD DATA LOCAL INFILE 'artikel01.csv'
6     -> INTO TABLE oshop.artikel    -- Spare mir ein USE oshop
7     -> FIELDS
8     -> TERMINATED BY ';'
9     -> LINES
10    -> TERMINATED BY '\n'
11    -> ;
12 Query OK, 9 rows affected, 12 warnings (0.03 sec)
13 Records: 9 Deleted: 0 Skipped: 0 Warnings: 10

```

Die zwölf Warnungen machen uns auch nicht glücklich. Schauen wir uns die mal genauer an:

```

1 mysql> SHOW WARNINGS;[label={lst04_003}]
2 +-----+-----+-----+-----+-----+-----+
3 | Level | Code | Message |          |          |          |
4 +-----+-----+-----+-----+-----+-----+
5 | Warning | 1366 | Incorrect integer value: 'artikel_id' for column 'artikel_id' at row 1 |          |          |          |
6 | Warning | 1366 | Incorrect decimal value: 'einzelpreis' for column 'einzelpreis' at row 1 |          |          |          |
7 | Warning | 1265 | Data truncated for column 'waehrung' at row 1 |          |          |          |
8 | Warning | 1261 | Row 1 doesn't contain data for all columns |          |          |          |
9 | Warning | 1261 | Row 2 doesn't contain data for all columns |          |          |          |
10 | Warning | 1261 | Row 3 doesn't contain data for all columns |          |          |          |
11 | Warning | 1261 | Row 4 doesn't contain data for all columns |          |          |          |
12 | Warning | 1261 | Row 5 doesn't contain data for all columns |          |          |          |
13 | Warning | 1261 | Row 6 doesn't contain data for all columns |          |          |          |
14 | Warning | 1261 | Row 7 doesn't contain data for all columns |          |          |          |
15 | Warning | 1261 | Row 8 doesn't contain data for all columns |          |          |          |
16 | Warning | 1261 | Row 9 doesn't contain data for all columns |          |          |          |
17 +-----+-----+-----+-----+-----+-----+
18 12 rows in set (0.00 sec)

```



Aufgabe 7.1: Die drei Warnungen in den [Zeilen 5 ff.](#) beziehen sich auf die erste Zeile (row 1). Was ist denn mit der ersten Zeile?

Aufgabe 7.2: Die Warnungen in den [Zeilen 8 ff.](#) haben auch eine gemeinsame Ursache. Vergleichen Sie die Anzahl der Angaben pro Zeile in der csv-Datei und die Anzahl der Spalten der Tabelle. Was fällt auf? ■

² Falls nicht, lesen Sie trotzdem weiter ;-).

Wir brauchen eine neue Version von LOAD DATA INFILE:



MySQL/MariaDB

```
LOAD DATA [LOW_PRIORITY|CONCURRENT] [LOCAL] INFILE 'dateiname'
[REPLACE|IGNORE]
INTO TABLE tabellenname
[FIELDS
  [TERMINATED BY 'zeichenfolge']
  [[OPTIONALLY] ENCLOSED BY 'zeichen']
  [ESCAPED BY 'zeichen'
  ]
[LINES
  [STARTING BY 'zeichenfolge']
  [TERMINATED BY 'zeichenfolge']
  ]
[IGNORE anzahl LINES]
[(spaltenname,...)]
[SET spaltenname = ausdruck,...]]
;
```

Die erste Aufgabe können wir mit IGNORE lösen. Mit dieser Option können wir festlegen, wie viele Zeilen (*anzahl*) zu Beginn der Datei übersprungen werden sollen. Die zweite Aufgabe entsteht dadurch, dass ich nicht angegeben habe, welche Felder ich in der CSV-Datei erwarte.

Auf zum nächsten Versuch:

```
1 mysql> DELETE FROM oshop.artikel;
2 Query OK, 8 rows affected (0.08 sec)
3
4 mysql> LOAD DATA LOCAL INFILE 'artikel01.csv'
5     ->     INTO TABLE oshop.artikel
6     ->     FIELDS
7     ->         TERMINATED BY ';'
8     ->     LINES
9     ->         TERMINATED BY '\n'
10    ->     IGNORE 1 LINES
11    ->     (artikel_id, bezeichnung, einzelpreis, waehrung)
12    ->     SET deleted=0;
13 Query OK, 8 rows affected (0.03 sec)
14 Records: 8 Deleted: 0 Skipped: 0 Warnings: 0
```

Keine Warnungen, keine Fehler :-). Die Zeile 10 löst die erste Aufgabe, indem sie anweist, die erste Zeile zu überspringen. Aber was macht Zeile 1? Im Vorgriff auf spätere Kapitel³ muss hier die Tabelle erst einmal wieder geleert werden. Dies erreiche ich durch das DELETE.

Die zweite Aufgabe wird durch zwei Angaben gelöst. In der Zeile 11 wird eine Spaltenliste angegeben. Diese Spaltenliste weist den Feldern der CSV-Datei die Spalten der Tabelle zu. Das erste Feld wird der Spalte artikel_id zugewiesen usw. Fehlt diese Spaltenliste,

³ Siehe Abschnitt 9.3 auf Seite 150

versucht er, alle Spalten der Tabelle in der CSV-Datei zu finden. In der [Zeile 12](#) wird der fehlenden Spalte `deleted` die 0 zugewiesen.



Aufgabe 7.3: Warum ist [Zeile 12](#) beim `LOAD DATA INFILE` eigentlich überflüssig?
Tipp: DESCRIBE artikel.

Ein Blick in die Tabelle `artikel` bestätigt es: Die Werte wurden korrekt übernommen.

```
1 mysql> SELECT * FROM oshop.artikel;
2 +-----+-----+-----+-----+
3 | artikel_id | bezeichnung | einzelpreis | waehrung | deleted |
4 +-----+-----+-----+-----+
5 | 3001 | Papier (100) | 2.300000 | EUR | 0 |
6 | 3005 | Tinte (gold) | 55.700000 | EUR | 0 |
7 | 3006 | Tinte (rot) | 6.200000 | EUR | 0 |
8 | 3010 | Feder | 5.000000 | EUR | 0 |
9 | 7856 | Silberwiebel | 0.410000 | EUR | 0 |
10 | 7863 | Tulpenzwiebel | 3.290000 | EUR | 0 |
11 | 9010 | Schaufel | 15.000000 | USD | 0 |
12 | 9015 | Spaten | 19.900000 | EUR | 0 |
13 +-----+-----+-----+-----+
```

BULK LOAD in T-SQL



T-SQL

```
BULK INSERT '[tabellenname.schemaname.]tabellenname'
    FROM dateiname
    [WITH (
        [FIRSTROW='zeilennummer']
        [FORMATFILE='dateiname'])]
;
```

In T-SQL gibt es eine sehr mächtige Importschnittstelle, deren Möglichkeiten ich hier nur anreißen möchte⁴. Dort können Sie über eine Formatdatei mehr oder weniger frei die Art des Imports steuern. Der Import der Artikeldaten sähe so aus:

```
1 >1 BULK INSERT artikel[label={lst04_006}]
2 >2     FROM '/home/[...]/src/mssql/artikel01.csv'
3 >3     WITH (FIRSTROW=2, FORMATFILE='/home/[...]/src/mssql/artikel.fmt');
4 GO
```

Die Formatdatei hätte für diesen simplen Fall folgenden Aufbau:

```
1 11.0
2 4
3 1   SQLCHAR   0    12    ";"      1   artikel_id      ""
4 2   SQLCHAR   0    510   ";"      2   bezeichnung    German_PhoneB[...]
5 3   SQLCHAR   0    41    ";"      3   einzelpreis    ""
6 4   SQLCHAR   0    6     "\n"    4   waehrung      German_PhoneB[...]
```

⁴ Siehe <https://docs.microsoft.com/de-de/sql/t-sql/statements/bulk-insert-transact-sql?view=sql-server-2017>

7.1.3 Was ist, wenn ich geänderte Werte importieren will?

Nehmen wir an, wir bekämen eine neue Version unserer Artikelstammdaten auf CD zugeschickt. Würden wir jetzt die Daten wieder importieren, stellt sich die Frage, was mit CD-Datensätzen passiert, die den gleichen Primärschlüsselwert haben? *Beispiel:* Die Preise unserer Artikelstammdaten haben sich geändert, und es sind neue hinzugekommen:

```

1 artikel_id;bezeichnung;einzelpreis;waehrung
2 7856;Silberzwiebel;0.51;EUR
3 7863;Tulpenzwiebel;3.39;EUR
4 9010;Schaufel;14.95;USD
5 9015;Spaten;19.90;EUR
6 3001;Papier (100);2.30;EUR
7 3005;Tinte (gold);55.70;EUR
8 3006;Tinte (rot);6.20;EUR
9 3007;Tinte (blau);4.13;EUR
10 3010;Feder;5.00;EUR

```

Was passiert jetzt mit den Artikeln, die den gleichen Primärschlüsselwert haben, aber sich im Preis unterscheiden (7856, 7863 und 9010)? Das entscheidet die Option REPLACE|IGNORE:

- IGNORE: Datensätze, die den gleichen Primärschlüsselwert haben, werden nicht importiert. Dies soll verhindern, dass man versehentlich schon vorhandene Datensätze überschreibt. Man erwartet eigentlich keine geänderten, sondern nur neue Datensätze.
- REPLACE: Datensätze, die den gleichen Primärschlüsselwert haben, werden komplett ersetzt, d.h., der alte Datensatz wird gelöscht und der neue eingefügt.

Probieren wir das Ganze mal aus, zunächst mit IGNORE:

```

1 mysql> LOAD DATA LOCAL INFILE 'artikel02.csv' IGNORE
2 [...]
3      -> (artikel_id, bezeichnung, einzelpreis, waehrung);
4 Query OK, 1 row affected (0.06 sec)
5 Records: 9 Deleted: 0 Skipped: 8 Warnings: 0

```

Beachten Sie bitte [Zeile 5](#): Von neun Datensätzen wurden acht übersprungen. Dies entspricht auch unseren Erwartungen. Alle Datensätze mit gleichem Primärschlüsselwert wurden wegen IGNORE nicht importiert. Der neue Datensatz 3007 wurde hingegen anstandslos importiert:

```

1 mysql> SELECT * FROM artikel;
2 +-----+-----+-----+-----+
3 | artikel_id | bezeichnung | einzelpreis | waehrung | deleted |
4 +-----+-----+-----+-----+
5 | 3001 | Papier (100) | 2.300000 | EUR | 0 |
6 | 3005 | Tinte (gold) | 55.700000 | EUR | 0 |
7 | 3006 | Tinte (rot) | 6.200000 | EUR | 0 |
8 | 3007 | Tinte (blau) | 4.130000 | EUR | 0 |
9 | 3010 | Feder | 5.000000 | EUR | 0 |
10 | 7856 | Silberwiebel | 0.410000 | EUR | 0 |
11 | 7863 | Tulpenzwiebel | 3.290000 | EUR | 0 |
12 | 9010 | Schaufel | 15.000000 | USD | 0 |
13 | 9015 | Spaten | 19.900000 | EUR | 0 |
14 +-----+-----+-----+-----+

```

Und jetzt mit REPLACE:

```

1 mysql> DELETE FROM artikel;
2 Query OK, 8 rows affected (0.08 sec)
3
4 mysql> LOAD DATA LOCAL INFILE 'artikel01.csv'
5      -> INTO TABLE artikel
6 [...]
7 Query OK, 8 rows affected (0.08 sec)
8 Records: 8 Deleted: 0 Skipped: 0 Warnings: 0
9
10 mysql> LOAD DATA LOCAL INFILE 'artikel02.csv' REPLACE
11 [...]
12 Query OK, 17 rows affected (0.07 sec)
13 Records: 9 Deleted: 8 Skipped: 0 Warnings: 0

```

Zuerst müssen wir wieder die Tabelle leeren ([Zeile 1](#)). Anschließend mit der ersten Version der Artikelstammdaten füllen ([Zeile 4](#)). In [Zeile 10](#) wird die zweite Version der Artikelstammdaten mit der Option REPLACE importiert. Das Ergebnis: 17 Datensätze sind betroffen. Acht wurden gelöscht und einer neu eingefügt:

```

1 mysql> SELECT * FROM artikel;
2 +-----+-----+-----+-----+-----+
3 | artikel_id | bezeichnung | einzelpreis | waehrung | deleted |
4 +-----+-----+-----+-----+-----+
5 | 3001 | Papier (100) | 2.300000 | EUR | 0 |
6 | 3005 | Tinte (gold) | 55.700000 | EUR | 0 |
7 | 3006 | Tinte (rot) | 6.200000 | EUR | 0 |
8 | 3007 | Tinte (blau) | 4.130000 | EUR | 0 |
9 | 3010 | Feder | 5.000000 | EUR | 0 |
10 | 7856 | Silberwiebel | 0.510000 | EUR | 0 |
11 | 7863 | Tulpenzwiebel | 3.390000 | EUR | 0 |
12 | 9010 | Schaufel | 14.950000 | USD | 0 |
13 | 9015 | Spaten | 19.900000 | EUR | 0 |
14 +-----+-----+-----+-----+-----+

```

■ 7.2 Daten anlegen



Erlesene Daten, liebevoll und einzeln eingefügt

- Grundkurs
 - Manuelles Anlegen einzelner Zeilen mit `INSERT INTO ... SET`
 - Manuelles Anlegen mehrerer Zeilen mit `INSERT INTO ... VALUES`
- Vertiefendes
 - Binäre Daten einfügen
 - Einfügereihenfolgen wegen der referenziellen Integrität

Schauen wir uns noch einmal das ER-Modell zur Artikelverwaltung an. Jedem Artikel muss mindestens eine Warenguppe zugeordnet werden, eine Warenguppe kann beliebig viele Artikel enthalten. Wir wissen darüber hinaus, dass die *n:m*-Verknüpfung eine Hilfstabelle

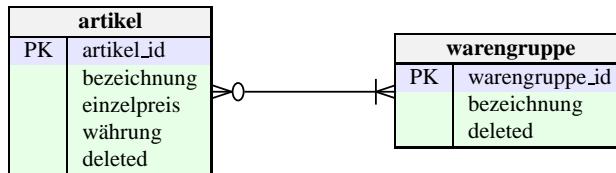


Bild 7.1 ER-Modell: Artikel zu Warengruppe

impliziert (siehe [Definition 21 auf Seite 30](#)). Wir wollen nun dem Artikel die Warengruppe zuordnen. Ich beziehe mich dabei auf die Daten in `artikel02.csv`.

Die Artikel mit der führenden 3 sind sicherlich leicht als Bürobedarf zu kategorisieren. Die Artikel mit der führenden 7 gehören zu Pflanzen und Gartenbedarf, die mit der führenden 9 zu Werkzeug und Gartenbedarf.

Unser Vorgehen ist in zwei Phasen aufgeteilt. In Phase 1 werden die Warengruppen in der Tabelle `warengruppe` angelegt. In Phase 2 wird in der Hilfstabelle die Verknüpfung zwischen diesen Warengruppen vorgenommen.

7.2.1 Wie legt man mehrere Zeilen mit einem Befehl an?

Der Befehl zum Anlegen einer Zeile heißt `INSERT INTO`. Hier ein Syntax-Auszug:



```

SQL:2016, MySQL/MariaDB, PostgreSQL, T-SQL ,
INSERT INTO tabellenname [(spaltenliste)]
VALUES
    (werteliste)
    [, (werteliste)]*
;
  
```

Die *werteliste* muss genau so viele Werte enthalten, wie Spalten in der *spaltenliste* angegeben wurden. Die *spaltenliste* ist optional. Wird sie nicht angegeben, müssen zu allen Spalten der Tabelle in der Reihenfolge des `CREATE TABLE` in der *werteliste* Werte angegeben werden.

```

1 INSERT INTO warengruppe (bezeichnung, deleted)
2 VALUES
3     ('Bürobedarf', 0)
4     ,('Pflanzen', 0)
5     ,('Gartenbedarf', 0)
6     ,('Werkzeug', 0)
7 ;
```

Das Ergebnis:

```

1 mysql> SELECT * FROM warengruppe;
2 +-----+-----+-----+
3 | warengruppe_id | bezeichnung | deleted |
4 +-----+-----+-----+
5 |          1 | Bürobedarf   |      0 |
6 |          2 | Pflanzen     |      0 |
```

7		3		Gartenbedarf		0	
8		4		Werkzeug		0	
9	+-----+-----+-----+						

Wir können mehrere Dinge hier erkennen:

- Zeichenketten werden in SQL immer von Hochkommata ' umschlossen, numerische Werte nicht.
- In der *spaltenliste* sind nur die Spalten angegeben, zu denen neue Werte bekannt sind.
- Jede *werteliste* enthält gleich viele Angaben und immer in der gleichen Reihenfolge.
- Obwohl keine Angaben über die Spalte `warengruppe_id` gemacht wurden, werden diese automatisch generiert.



Aufgabe 7.4: Warum werden die Werte von der Spalte `warengruppe_id` automatisch generiert?

Aufgabe 7.5: Dieser Quelltext würde ebenfalls das gleiche Ergebnis liefern. Warum?

```
INSERT INTO warengruppe (warengruppe_id, bezeichnung)
VALUES
    (1, 'Bürobedarf')
    ,(2, 'Pflanzen')
    ,(3, 'Gartenbedarf')
    ,(4, 'Werkzeug')
;
```

Tipp: SHOW CREATE TABLE warengruppe;

Aufgabe 7.6: Wenn Sie beide Quelltexte hintereinander ausführen, erhalten Sie folgende Fehlermeldung:

ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'

Interpretieren Sie diese Fehlermeldung.

7.2.2 Wie kann man eine einzelne Zeile anlegen?

Wir wollen jetzt die Verknüpfungen in die Hilfstabelle eintragen. Dazu möchte ich die Zeilen einzeln anlegen, um bei der Zuordnung die Übersicht nicht zu verlieren. Und wie geht das? Zuerst die einfache Antwort: Indem Sie nur eine *werteliste* im oben vorgestellten `INSERT INTO` angeben.

Eine andere Variante sieht so aus:



MySQL/MariaDB

```
INSERT INTO tabellenname
SET
    spaltenname=wert
    [,spaltenname=wert]*
;
```

Bevor wir die neuen Daten einfügen, sollten wir zuerst die Daten einzeln notieren, um das Verhältnis zwischen Artikelstammdaten und Warengruppen besser zu verstehen.

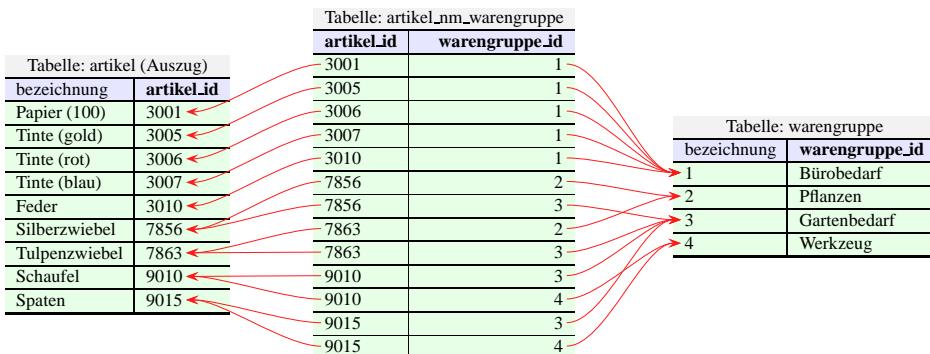


Bild 7.2 Hilfstabelle: Artikel zu Warengruppe

In Bild 7.2 wird die Funktionsweise der Hilfstabelle deutlich. Jede Verknüpfung in der Hilfstabelle verweist auf einen Artikel und eine Warengruppe. Als Folge kann ein Artikel zu mehreren Warengruppen gehören (z.B. 7863) und eine Warengruppe mehrere Artikel beinhalten (z.B. 4).

Der nächste Schritt besteht nun darin, für jede Verknüpfung ein eigenes `INSERT INTO` zu erstellen:

```

1  INSERT INTO artikel_nm_warengruppe
2    SET
3      artikel_id=3001
4      ,warengruppe_id=1
5 ;

```

Man erkennt schneller, welcher Wert welcher Spalte zugeordnet wird.



Aufgabe 7.7: Erstellen Sie die `INSERT INTO`s für alle Zuordnungen, sodass jeder Artikel seine Warengruppe kennt und jede Warengruppe alle entsprechenden Artikel enthält.

7.2.3 Vorsicht Constraints!

Zuerst eine kleine Fehlermeldung:

```

1  mysql> SELECT bestellung_id, kunde_id, adresse_id, datum FROM bestellung;
2  +-----+-----+-----+-----+
3  | bestellung_id | kunde_id | adresse_id | datum          |
4  +-----+-----+-----+-----+
5  |           1 |        1 |           1 | 2012-03-24 17:41:00 |
6  |           2 |        2 |           2 | 2012-03-23 16:11:00 |
7  +-----+-----+-----+-----+
8
9  mysql> INSERT INTO bestellung (datum) VALUES (NOW());

```

```
10 ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint
   fails ('oshop'.'bestellung', CONSTRAINT 'bestellung_ibfk_2' FOREIGN KEY
   ('adresse_id') REFERENCES 'adresse' ('adresse_id'))
```

Die Fehlermeldung besagt übersetzt soviel wie: Ich konnte den Datensatz nicht anlegen, da der Constraint einen NULL-Wert verbietet. Also 2. Versuch:

```
1 mysql> INSERT INTO bestellung (adresse_id, datum) VALUES (10, NOW());
2 ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint
   fails ('oshop'.'bestellung', CONSTRAINT 'bestellung_ibfk_2' FOREIGN KEY
   ('adresse_id') REFERENCES 'adresse' ('adresse_id'))
```

Ging schon wieder schief, da es keine Adresse 10 gibt. Also zuerst eine Adresse 10 einfügen:

```
1 mysql> INSERT INTO adresse (adresse_id, strasse) VALUES (10, 'Oberer Weg');
2 Query OK, 1 row affected (0.09 sec)
3
4 mysql> INSERT INTO bestellung (adresse_id, datum) VALUES (10, NOW());
5 ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint
   fails ('oshop'.'bestellung', CONSTRAINT 'bestellung_ibfk_1' FOREIGN KEY
   ('kunde_id') REFERENCES 'kunde' ('kunde_id'))
```

Schön, wenigstens eine neue Fehlerquelle. Aber leider die gleiche Art von Ursache. Sie sehen, dass bei Tabellen, die Constraints beachten, die Reihenfolge wichtig ist, in der die Daten importiert, angelegt oder kopiert werden.

Die fehlgeschlagenen Versuche bedeuten natürlich nicht, dass man keine neue Bestellung anlegen kann. Es muss aber darauf geachtet werden, dass der neue Datensatz zu allen geforderten Constraints eine gültige Angabe enthält.



Hinweis: Bei Constraints muss beim Anlegen neuer Zeilen in mehreren Tabellen auf die Reihenfolge geachtet werden, um die referentielle Integrität zu wahren (siehe [Abschnitt 2.3 auf Seite 32](#)). ■

7.2.4 Einfügen von binären Daten über einen C#-Client

Mit dem Datentyp BLOB und dessen Verwandten steht uns die Möglichkeit offen, binäre Daten wie Bilder, Musik, PDF-Dateien etc. in Tabellen abzulegen. Stellt sich nur die Frage, wie man die Dinger da reinbekommt.



Aufgabe 7.8: Wir könnten für unsere Artikel eine Bildergalerie anlegen wollen.

1. Modellieren Sie die Tabelle bild. Sie soll mindestens die Bilddaten, einen Dateinamen, die Dateigröße und den Dateityp enthalten.
2. Erweitern Sie das ER-Modell derart, dass man zu jedem Artikel beliebig viele Bilder ablegen kann.
3. Erstellen Sie das passende CREATE TABLE.

Das Einfügen erfolgt in der Regel über einen Client. Ich verwende hier als Beispiel einen C#-Client; PHP-Beispiele finden Sie zu Zehntausenden im Internet.



Wir brauchen die Dateien: spaten01.png, spaten02.png, spaten03.png und HauptfensterInsert.cs.

Hier die Tabelle:

```
1 mysql> CREATE TABLE bild
2      -> (
3          -> bild_id          INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
4          -> bild              BLOB,
5          -> dateiname        VARCHAR(255),
6          -> dateityp          VARCHAR(255),
7          -> dateigroesse     BIGINT UNSIGNED,
8          -> artikel_id        INT UNSIGNED REFERENCES artikel(artikel_id)
9      );
```

Zuerst wird mithilfe eines OpenFileDialogs die Datei ausgewählt: spaten01.png. Nur wenn der OpenFileDialog mit OK beendet wurde, werden die nachfolgenden Anweisungen ausgeführt.

Listing 7.1 HauptfensterInsert.cs, Teil 1

```
18     private void BttnBild_Click(object sender, EventArgs e) {
19         OpenFileDialog dlg = new OpenFileDialog();
20         dlg.DefaultExt = "png";
21         dlg.Filter = "png-Dateien|*.png|jpg-Dateien|*.jpg|Alle Dateien|*.*";
22         if (dlg.ShowDialog() == System.Windows.Forms.DialogResult.OK) {
```

Jetzt werden einige Objekte deklariert und initialisiert, die wir später brauchen. Das Objekt MySqlConn repräsentiert die Verbindung zwischen dem SQL Server und der Anwendung. MySqlCommand enthält alle Informationen, die man für die Ausführung eines Befehls braucht. Dazu gehört der Befehl selbst (`strInsert`) und die Verbindung, über die der Befehl ausgeführt werden soll.

In `iDateigroesse` wird die Größe der Datei in Bytes gemessen abgespeichert. `binBilddaten` ist ein Array vom Datentyp `Byte`. In dieses Array werden die Bilddaten abgespeichert, die wir aus der Datei auslesen werden.

Listing 7.2 HauptfensterInsert.cs, Teil 2

```
23     MySqlConnection MySqlConn = new MySqlConnection();
24     MySqlCommand MySqlCommand = new MySqlCommand();
25     String strInsert;
26     Int32 iDateigroesse;
27     Byte[] binBilddaten;
28     FileStream streamDatei;
29
30     MySqlConn.ConnectionString = "server=127.0.2.1";
31     MySqlConn.ConnectionString += ";uid=root";
32     MySqlConn.ConnectionString += ";pwd=";
33     MySqlConn.ConnectionString += ";database=oShop; charset=utf8";
```

Die Datei selbst wird über `streamDatei` angesprochen. Bleibt noch das Setzen der Verbindungsparameter. Dazu wird ein geläufiges Verfahren gewählt: der `ConnectionString`. Bitte achten Sie darauf, dass Sie ggf. andere Parameter einstellen müssen.

Listing 7.3 HauptfensterInsert.cs, Teil 3

```

35     try {
36         streamDatei = new FileStream(Path.GetFullPath(dlg.FileName),
37             FileMode.Open, FileAccess.Read);
38         if (streamDatei.Length > Int32.MaxValue) {
39             throw new Exception("Datei zu gross");
40         } // if Ende
41         iDateigroesse = Convert.ToInt32(streamDatei.Length);

```

Die Datei wird lesend geöffnet. Ich verwende dabei die statische Klasse `Path`. Diese hilft dabei, Dateizugriffe unabhängig vom verwendeten Betriebssystem zu programmieren. Besonders die unterschiedlichen Verzeichnisbegrenzer⁵ werden bequem verwaltet.

Problem: Die Größe der Datei `Length` ist vom Datentyp `long`. Ich kann später mit der Methode `Read()` aber nur Blöcke mit einer maximalen Größe von `MaxInt32` einlesen. Ich verbiete größere Dateien, indem ich ggf. eine Ausnahme auslöse.



Hinweis: Wenn Sie große Dateien einfügen möchten, müssen diese blockweise eingelesen werden. Beachten Sie dabei, dass ggf. die Variable `max_allowed_packet` auf dem Server verändert werden muss.

Jetzt wird der Inhalt der Datei eingelesen und im Array `binBilddaten` abgelegt. Da wir die Datei nicht mehr brauchen, kann sie geschlossen werden.

Listing 7.4 HauptfensterInsert.cs, Teil 4

```

42     binBilddaten = new byte[iDateigroesse];
43     streamDatei.Read(binBilddaten, 0, iDateigroesse);
44     streamDatei.Close();

```

Der eigentliche Befehl wird nun gebastelt. Dabei wird ein ganz normaler `INSERT` verwendet. Anstelle der eigentlichen Daten werden aber Platzhalter verwendet. Die Nummer `9015` ist fest vorgegeben, da es sich um die Artikelnummer des Spatens handelt.

Listing 7.5 HauptfensterInsert.cs, Teil 5

```

46     strInsert = "INSERT INTO ";
47     strInsert += "bild (bild, dateiname, dateityp, dateigroesse,
48                 artikel_id)";
49     strInsert += "VALUES";
50     strInsert += " (@bild, @dateiname, @dateityp, @dateigroesse, 9015)";

```

Und los geht's: Die Verbindung zum Server wird geöffnet. Läuft da was schief, wird eine Ausnahme ausgelöst. Vielleicht ist der `ConnectionString` nicht richtig? Die gerade aufgebaute Verbindung wird dem `MySqlCommand` mitgeteilt, ebenso der SQL-Befehl. Mithilfe von `Parameter.AddWithValue()` werden die Platzhalter durch die konkreten Werte ersetzt, u.a. auch die binären Bilddaten⁶.

⁵ Linux/Unix: `/`, Windows: `\`

⁶ Um SQL-Injection-Angriffen vorzubeugen, sollten Sie sich mit `Prepare()` vertraut machen.

Da wir keine Ergebnisse vom Server zurückerwarten, wird mit `ExecuteNonQuery()` der SQL-Befehl zum Server gesendet und ausgeführt. Da wir die Verbindung zum Server nicht mehr brauchen, wird diese geschlossen.

Listing 7.6 HauptfensterInsert.cs, Teil 6

```

51     MySqlConn.Open();
52     MySqlCommand.Connection = MySqlConn;
53     MySqlCommand.CommandText = strInsert;
54     MySqlCommand.Parameters.AddWithValue("@dateiname", Path.GetFileName(dlg.
      FileName));
55     MySqlCommand.Parameters.AddWithValue("@dateigroesse", iDateigroesse);
56     MySqlCommand.Parameters.AddWithValue("@dateityp", Path.GetExtension(dlg.
      FileName));
57     MySqlCommand.Parameters.AddWithValue("@bild", binBilddaten);
58     MySqlCommand.ExecuteNonQuery();
59     MySqlConn.Close();

```

Zum Schluss noch eine Erfolgsmeldung und der `catch()` zur Fehlerbehandlung.

Listing 7.7 HauptfensterInsert.cs, Teil 7

```

61     MessageBox.Show("Und wieder ein neuer Spaten",
62                     "Spatenbild importiert!", MessageBoxButtons.OK,
63                     MessageBoxIcon.Asterisk);
64
65     } // try Ende
66     catch (Exception ex) {
67         MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK,
68                         MessageBoxIcon.Error);
69     } // catch Ende
70     } // if Ende
71     } // BttnBild_Click Ende
72 } // Hauptfenster Ende

```

7.2.5 Einfügen von binären Daten LOAD FILE

Eine andere Methode, BLOB-Daten in die Tabelle einzufügen, ist die Option `LOAD FILE`. Bei `INSERT`- oder `UPDATE`-Anweisungen können Dateiinhalte eingelesen und Spalten zugeordnet werden.

```

1 mysql> INSERT INTO bild
2 -> SET
3 ->   bild = LOADFILE('/var/lib/mysql/interchange/spaten01.png'),
4 ->   dateiname = 'spaten01.png',
5 ->   dateityp = 'png',
6 ->   dateigroesse = 7168,
7 ->   artikel_id = 9015
8 -> ;

```

In Zeile 3 können Sie sehen, wie einfach die entsprechende Syntax ist. Innerhalb der Option wird der Dateiname mit vollständigem Datenpfad angegeben. Damit auf diesen zugegriffen werden kann, muss er allerdings in der Konfigurationsdatei des MySQL/MariaDB Servers (z.B. `/etc/mysql/my.cnf`) als Datenverzeichnis angemeldet werden. Diese Angabe ist zur Laufzeit des Servers nicht veränderbar:

```

1 [...]
2 # The MySQL Server configuration file.
3 [...]
4 [mysqld]
5 [...]
6 # secure-file-priv= NULL
7 secure-file-priv= /var/lib/mysql/interchange/
8 [...]

```

Der Parameter `secure-file-priv` enthält die Verzeichnisse, aus denen der Server Dateien importieren oder in denen er Dateien exportieren darf (siehe [Zeile 7](#)).



Hinweis: Diese Möglichkeit ist in einer Produktivumgebung mit Vorsicht zu genießen! Besser ist es, diese Variable mit dem Wert `NULL` zu belegen, damit keine Dateien vom Server im- oder exportiert werden können. Wie man diese Variable für einen SQL-Injection-Angriff ausnutzen kann, finden Sie gut in [[Mik10](#)] dargestellt.

■ 7.3 Daten kopieren



Aus eins mach zwei: Daten aus Tabellen kopieren

- Vertiefendes
 - Daten aus bestehenden Tabellen mit `INSERT INTO ... SELECT` übernehmen

Beim Anlegen temporärer Tabellen hatten wir die Situation (siehe [Seite 90](#)), dass man für eine Marketingaktion nur Auszüge der Daten aus der Tabelle `kunde` benötigte.

Wir möchten alle Kunden, die zur Familie *Beutlin* gehören, separat auswerten. Dazu sind zwei Schritte notwendig: das Erstellen der temporären Tabelle und das Kopieren aller Kunden mit Nachnamen *Beutlin*.



SQL:2016, MySQL/MariaDB, PostgreSQL

```

INSERT INTO tabellenname_ziel [(spaltenliste)]
  SELECT auswahl
;

```

T-SQL

```

SELECT auswahl_der_spalten
  INTO tabellenname_ziel [(spaltenliste)]
  FROM
    SELECT auswahl_der_datenquelle
;

```

In unserem Fall ist die *auswahl* eine einfache Abfrage auf den Nachnamen:

```

1 mysql> CREATE TEMPORARY TABLE kunde_beutlin LIKE kunde;
2 Query OK, 0 rows affected (0.04 sec)
3
4 mysql> INSERT INTO kunde_beutlin
5      -> SELECT * FROM kunde WHERE nachname='Beutlin';
6 Query OK, 2 rows affected (0.04 sec)
7 Records: 2  Duplicates: 0  Warnings: 0
8
9 mysql> SELECT * FROM kunde_beutlin;
10 +-----+-----+-----+ [...] ++
11 | kunde_id | nachname | vorname | [...] |
12 +-----+-----+-----+ [...] ++
13 |      2 | Beutlin   | Frodo    | [...] |
14 |      3 | Beutlin   | Bilbo    | [...] |
15 +-----+-----+-----+ [...] ++

```

Den Aufbau des Befehls `SELECT` werden wir noch später ab [Kapitel IV auf Seite 155](#) behandeln. Aber mit ein wenig Phantasie ist klar, wie hier der `SELECT` arbeitet. Das Sternchen hinter dem `SELECT` gibt an, dass alle Spalten der Tabelle ausgegeben werden sollen. Da die Tabelle `kunde` und die temporäre Tabelle `kunde_beutlin` die gleiche Anzahl, Reihenfolge und Typisierung der Spalten haben, passt das `SELECT *` genau zu `kunde_beutlin`, und die Daten können sofort importiert werden.

Sind die Tabellen nicht exakte strukturelle Kopien, so muss der `SELECT` so aufgebaut werden, dass das Ergebnis in die Tabellenstruktur oder *spaltenliste* der Zieltabelle passt.

Beispiel: Hier werden nur die Vor- und Nachnamen aller Kunden in eine neue temporäre Tabelle kopiert.

```

1 mysql> CREATE TEMPORARY TABLE kunde_namen LIKE kunde;
2 Query OK, 0 rows affected (0.04 sec)
3
4 mysql> INSERT INTO kunde_namen (vorname, nachname)
5      -> SELECT vorname, nachname FROM kunde
6      -> ;
7 Query OK, 5 rows affected (0.05 sec)
8 Records: 5  Duplicates: 0  Warnings: 0
9
10 mysql> SELECT * FROM kunde_namen;
11 +-----+-----+-----+-----+-----+ [...] ++
12 | kunde_id | nachname     | vorname | rechnung_adresse_id | [...] |
13 +-----+-----+-----+-----+-----+ [...] ++
14 |      1 | Beutlin       | Bilbo   |                 NULL | [...] |
15 |      2 | Beutlin       | Frodo   |                 NULL | [...] |
16 |      3 | Earendillionn | Elrond   |                 NULL | [...] |
17 |      4 | Gamdschie     | Samweis |                 NULL | [...] |
18 |      5 | Telcontar     | Elessar  |                 NULL | [...] |
19 +-----+-----+-----+-----+-----+ [...] ++

```


TEIL III

Datenbank ändern

8

Datenbank und Tabellen umbauen

■ 8.1 Eine Datenbank ändern



Erstaunlich, wie wenig man von einem Schema ändern kann.

- Grundkurs
 - ALTER SCHEMA, ALTER DATABASE
 - Ein neuer Zeichensatz
 - Eine neue Sortierung
- Vertiefendes: SHOW CREATE TABLE

Beim Anlegen der Datenbank gab es nur vier Angaben, die für eine Datenbank festgelegt werden konnten: Datenbankname, Zeichensatz, Sortierung und ob verschlüsselt werden soll. Der Datenbankname kann nicht geändert werden, aber Zeichensatz, Sortierung und der Verschlüsselungsschalter.

In SQL:2016 gibt es keinen Befehl zum Ändern einer Datenbank. Eine Änderung kann hier nur durch Anlegen einer neuen Datenbank mit den geänderten Parametern erfolgen. Anschließend werden die Datenbankobjekte einzeln in die neue Datenbank überführt.



MySQL/MariaDB

```
ALTER {DATABASE | SCHEMA} datenbankname
[DEFAULT CHARACTER SET zeichensatz]
[DEFAULT COLLATE sortierung]
[DEFAULT ENCRYPTION {'Y' | 'N'}]
;
```

Wie Sie die zur Verfügung stehenden Zeichensätze ermitteln, entnehmen Sie entsprechenden Abschnitten ab [Seite 69](#) und denen über Sortierungen ab [Seite 70](#).

Spannend sind jetzt folgende Fragen:

- Werden die Tabellenoptionen der schon vorhandenen Tabellen mit geändert oder werden nur neue Tabellen mit den geänderten Datenbankoptionen erstellt?
- Werden die Zeichensätze konvertiert oder nur anders interpretiert?

- Werden die Indizes automatisch neu gebildet oder muss man dies manuell tun?

Die erste Frage lässt sich leicht durch Ausprobieren beantworten:

```

1 mysql> SHOW CREATE DATABASE oshop\G
2 **** 1. row ****
3 Database: oshop
4 Create Database: CREATE DATABASE 'oshop' /*!40100 DEFAULT CHARACTER SET utf8
   COLLATE utf8_unicode_ci
5 */ /*!80016 DEFAULT ENCRYPTION='N' */
6
7 mysql> ALTER DATABASE oshop DEFAULT CHARACTER SET latin1;
8
9 mysql> SHOW CREATE DATABASE oshop\G
10 **** 1. row ****
11 Database: oshop
12 Create Database: CREATE DATABASE 'oshop' /*!40100 DEFAULT CHARACTER SET latin1
   */
13 /*!80016 DEFAULT ENCRYPTION='N' */
14 1 row in set (0.00 sec)
```



Hinweis: Mit /*!40100 [...] */ wird ein bedingter Kommentar (Conditional Comment) programmiert. Hier: Bei einem Server der Version kleiner 4.1 ist dies ein Kommentar, ansonsten nicht. So können abwärtskompatible Befehle erstellt werden.

Mit SHOW CREATE DATABASE konnten wir uns davon überzeugen, dass die Datenbank einen neuen DEFAULT-Zeichensatz hat. Wie sieht es aber für die Tabellen aus? Hier hilft der gute alte Bekannte SHOW CREATE TABLE:

```

1 mysql>SHOW CREATE TABLE kunde\G
2 **** 1. row ****
3 Table: kunde
4 Create Table: CREATE TABLE 'kunde' (
5   'kunde_id' int(10) unsigned NOT NULL AUTO_INCREMENT,
6   'nachname' varchar(255) COLLATE utf8_unicode_ci NOT NULL DEFAULT '',
7   [...]
8 ) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
```

In der Zeile 8 wird der Zeichensatz mit utf8 angegeben. Die Tabellen behalten somit ihren Zeichensatz und werden nicht automatisch in den neuen konvertiert.

Bei neuen Tabellen sieht das anders aus:

```

1 mysql> CREATE TABLE bla (name VARCHAR(255));
2
3 mysql> SHOW CREATE TABLE bla\G
4 **** 1. row ****
5 Table: bla
6 Create Table: CREATE TABLE 'bla' (
7   'name' varchar(255) DEFAULT NULL
8 ) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Der Zeichensatz hier ist der neu eingestellte der Datenbank: latin1. Wenn die alten Tabellen bei dem alten Zeichensatz bleiben, wird auch die letzte Frage beantwortet: Die Indizes brauchen nicht neu gebildet werden.



Aufgabe 8.1: Ermitteln Sie selber, ob eine geänderte Sortierung in den Tabellen übernommen wird.

■ 8.2 Ein Schema löschen



Nothing lasts forever. Mit einem Handstreich alles löschen.

- Grundkurs: Löschen mit DROP DATABASE
- Vertiefendes: Was beim Löschen so alles passiert

Das Löschen einer Datenbank mit `DROP DATABASE` ist erfrischend einfach. Die Datenbank und alle ihre Elemente werden aus dem DBMS entfernt. Dazu gehören beispielsweise die Tabellen, Ansichten, Prozeduren und Trigger.



SQL:2016

```
DROP DATABASE datenbankname  
;
```

MySQL/MariaDB

```
DROP {SCHEMA|DATABASE} [IF EXISTS] datenbankname  
;
```

PostgreSQL, T-SQL

```
DROP DATABASE [IF EXISTS] datenbankname  
;
```

Damit wir uns nichts kaputt machen, legen wir eine neue Datenbank an und überprüfen ihre Existenz.

```
1 mysql> CREATE DATABASE wurstbrot;  
2  
3 mysql> SHOW DATABASES;  
4 +-----+  
5 | Database      |  
6 +-----+  
7 | information_schema |  
8 | mysql          |  
9 | oshop          |  
10 | performance_schema |  
11 | sys            |  
12 | wurstbrot      |  
13 +-----+  
14 6 rows in set (0.00 sec)
```

Jetzt löschen wir sie wieder:

```

1 mysql> DROP DATABASE wurstbrot;
2 mysql> SHOW DATABASES;
3 +-----+
4 | Database      |
5 +-----+
6 | information_schema |
7 | mysql          |
8 | oshop          |
9 | performance_schema |
10 | sys            |
11 +-----+
12 5 rows in set (0.00 sec)

```

Nun hat MySQL aus dem Unterverzeichnis `wurstbrot` alle Dateien mit den Endungen `.BAK`, `.DAT`, `.HSH`, `.MRG`, `.MYD`, `.ISD`, `.MYI`, `.db` und `.frm` gelöscht. Ebenso wird dort die Datei `db.opt` gelöscht.

Abschließend wird das Verzeichnis selbst gelöscht. Sollten in dem Verzeichnis noch andere Dateien liegen – z.B. solche, die durch `SELECT ... INTO OUTFILE1` entstanden sind –, kann das Verzeichnis nicht gelöscht werden. Es erscheint eine entsprechende Fehlermeldung.

```

1 mysql> SELECT * FROM warengruppe INTO OUTFILE 'bla.txt';
2 Query OK, 4 rows affected (0.00 sec)
3
4 mysql> DROP DATABASE oshop;
5 ERROR 1010 (HY000): Error dropping database (can't rmdir './oshop/', [...]

```

Bei einigen DBMS können Sie die Datenbank nicht löschen, wenn diese gerade in einer Sitzung verwendet wird. Hier ein Beispiel in TSQL:

```

1 1> CREATE DATABASE wurstbrot;
2 2> GO
3 1> USE wurstbrot;
4 2> GO
5 Changed database context to 'wurstbrot'.
6 1> DROP DATABASE wurstbrot;
7 2> GO
8 Msg 3702, Level 16, State 3, Server 86e3151cc665, Line 1
9 Cannot drop database "wurstbrot" because it is currently in use.
10 1> USE master;
11 2> GO
12 Changed database context to 'master'.
13 1> DROP DATABASE wurstbrot;
14 2> GO
15 1>

```

Der Löschversuch in Zeile 9 scheitert mit der entsprechenden Fehlermeldung. Erst der Wechsel zu einer anderen Datenbank (siehe Zeile 12) gibt die Datenbank wieder frei und sie kann daher auch gelöscht werden, falls nicht noch andere Sitzungen auf sie zugreifen.



Hinweis: Bevor Sie eine Datenbank löschen, sollten Sie unbedingt eine Sicherheitskopie der Datenbank gemacht haben! Nach dem DROP ist nämlich alles weg! Es erfolgt keine Rückmeldung, und es gibt auch kein *undo*.

¹ Siehe [Abschnitt 10.6.1 auf Seite 179](#)

■ 8.3 Eine Tabelle ändern



Tabellen müssen sich anpassen lassen. Daher frei nach dem Motto des Zweiten Vatikanischen Konzils: Tabula semper reformanda est.

- Grundkurs
 - Tabelle mit ALTER TABLE ... RENAME umbenennen
 - Spalte mit ALTER TABLE ADD hinzufügen
 - Spaltenspezifikation mit ALTER TABLE ... MODIFY ändern
 - Spalten mit ALTER TABLE ... DROP entfernen
- Vertiefendes
 - Umbenennen der Fremdschlüsselverweise
 - Spalten einsortieren
 - Was passiert bei Längenänderungen?
 - Was passiert bei Zeichensatzänderungen?
 - Von Zahl nach Zeichen konvertieren
 - Wertebereich einer Zahl ändern
 - Datum- und Uhrzeitwerte



Die Quelltexte dieses Kapitels stehen in der Datei `Listing05.sql` (siehe [Listing 29.5 auf Seite 473](#), [Listing 29.41 auf Seite 595](#) und [Listing 29.26 auf Seite 545](#)).

Eine Tabelle kann in folgenden Eigenschaften nachträglich verändert werden:

- Name der Tabelle
- Spaltenspezifikation bzgl. des Namens, des Datentyps und der Zusätze
- Constraints
- Tabellenoptionen

Das ist so ziemlich alles, was eine Tabelle ausmacht. Je umfangreicher eine Tabellenänderung ist, desto eher sollte darüber nachgedacht werden, eine neue Tabelle anzulegen, die Datensätze einzeln zu lesen, umzubauen und in die neue hineinzuschreiben. Dazu später mehr.

Der entscheidende Befehl zum Ändern einer Tabelle ist `ALTER TABLE`. Da dieser Befehl sehr umfangreich ist, wird er hier anhand von Fallbeispielen eingeführt. Dabei werden nicht alle Varianten des Befehls berücksichtigt.

8.3.1 Wie kann ich den Namen der Tabelle ändern?



MySQL/MariaDB, PostgreSQL

```
ALTER TABLE tabellename_alt  
    RENAME TO tabellename_neu
```

```
;
```

T-SQL

```
EXEC sp_rename tabellename_alt, tabellename_neu
;
```

Das Umbenennen ist schnell durchgeführt und überprüft.

1 mysql> SHOW TABLES;	15 mysql> SHOW TABLES;
2 +-----+ <td>16 +-----+</td>	16 +-----+
3 Tables_in_oshop <td>17 Tables_in_oshop </td>	17 Tables_in_oshop
4 +-----+ <td>18 +-----+</td>	18 +-----+
5 adresse <td>19 artikel </td>	19 artikel
6 artikel <td>20 artikel_nm_lieferant </td>	20 artikel_nm_lieferant
7 [...] <td>21 [...] </td>	21 [...]
8 rechnung_position <td>22 warengruppe </td>	22 warengruppe
9 warengruppe <td>23 wurstbrot </td>	23 wurstbrot
10 +-----+ <td>24 +-----+</td>	24 +-----+
11	25 14 rows in set (0.00 sec)
12 mysql> ALTER TABLE adresse	26
13 -> RENAME TO wurstbrot;	27 mysql>
14	

In [Zeile 23](#) können Sie sehen, dass die Tabelle `adresse` nun `wurstbrot` heißt.

Stellt sich doch die Frage, was bei einem `ALTER TABLE ... RENAME` mit den Fremdschlüssel-Constraints passiert?

1 mysql> SHOW CREATE TABLE kunde\G	15 mysql> SHOW TABLES;
2 ***** 1. row *****	16 +-----+
3 Table: kunde	17 Tables_in_oshop
4 Create Table: CREATE TABLE 'kunde' (18 +-----+
5 'kunde_id' int(10) unsigned NOT NULL AUTO_INCREMENT,	19 artikel
6 [...]	20 artikel_nm_lieferant
7 CONSTRAINT 'kunde_ibfk_1' FOREIGN KEY ('rechnung_adresse_id') REFERENCES 'wurstbrot' ('adresse_id') ON UPDATE CASCADE,	21 [...]
8 CONSTRAINT 'kunde_ibfk_2' FOREIGN KEY ('liefer_adresse_id') REFERENCES 'wurstbrot' ('adresse_id') ON DELETE SET NULL ON UPDATE CASCADE	22 warengruppe
9) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci	23 wurstbrot
10 1 row in set (0.00 sec)	24 +-----+
11	25 14 rows in set (0.00 sec)
12 mysql> ALTER TABLE wurstbrot RENAME TO adresse;	26
	27 mysql>

Es haben diverse Fremdschlüssel auf eine Tabelle mit dem Namen `adresse` verwiesen. Eine Tabelle, die Fremdschlüssel auf `adresse` enthält, ist `kunde`. In [Zeile 7](#) ist zu sehen, dass die Verweise auf den neuen Namen `wurstbrot` aktualisiert wurden. Puh :-)



Hinweis: In SQL:2016 gibt es kein `ALTER TABLE ... RENAME TO`-Befehl. In den verschiedenen RDBMS-Implementierungen finden sich aber entsprechende Anweisungen wie beispielsweise in TSQL die Systemprozedur der `sp_rename`.

```
1 1> EXEC sp_rename 'adresse', 'wurstbrot';
2 2> GO
3 Caution: Changing any part of an object name could break scripts and
           stored procedures.
```

8.3.2 Wie kann ich eine Spalte hinzufügen?



SQL:2016

PostgreSQL, T-SQL

```
ALTER TABLE tabellenname
    ADD [COLUMN] spaltenspezifikation
;
```

MySQL/MariaDB

```
ALTER TABLE tabellenname
    ADD [COLUMN] spaltenspezifikation [[FIRST|AFTER] spaltenname]
;
```

Die *spaltenspezifikation* in `ALTER TABLE ... ADD` entspricht der, die bei einem `CREATE TABLE` (siehe [Abschnitt 5.3 auf Seite 72](#)) möglich ist. Wir wollen in der Tabelle `kunde` neben Nach- und Vornamen auch einen Firmennamen erfassen können.



Aufgabe 8.2: Wie lautet die Spaltenspezifikation für den Firmennamen? Begründen Sie Ihre Wahl.

Der aktuelle Stand der Tabelle `kunde` enthält keine Spalte zum Firmennamen. Dies könnten wir uns mit `SHOW CREATE TABLE` bestätigen lassen. Ein etwas weniger geschwätziger Befehl liefert uns auch alle notwendigen Informationen: `DESCRIBE`.

```
1 mysql> DESCRIBE kunde;
2 +-----+-----+-----+
3 | Field      | Type       | [...] |
4 +-----+-----+-----+
5 | kunde_id   | int(10) unsigned | [...] |
6 | nachname   | varchar(255)  | [...] |
7 | vorname    | varchar(255)  | [...] |
8 | rechnung_adresse_id | int(10) unsigned | [...] |
9 | liefer_adresse_id | int(10) unsigned | [...] |
10 | bezahlart  | int(10) unsigned | [...] |
11 | art         | enum('unb','prv','gsch') | [...] |
12 | deleted    | tinyint(3) unsigned | [...] |
13 +-----+-----+-----+
14 8 rows in set (0.00 sec)
```

Hinter dem Vornamen wird jetzt die Spalte `firmenname` eingefügt:

```
1 ALTER TABLE kunde
2     ADD firmenname VARCHAR(255) NOT NULL DEFAULT '' AFTER vorname
3 ;
```

In [Zeile 8](#) taucht unser Firmenname auf – genau da, wo wir ihn haben wollten: nach dem Vornamen. Wird weder `FIRST` noch `AFTER` verwendet, wird die neue Spalte hinter der letzten eingefügt.

```
1 mysql> DESCRIBE kunde;
2 +-----+-----+-----+
3 | Field      | Type       | [...] |
4 +-----+-----+-----+
5 | kunde_id   | int(10) unsigned | [...] |
```

6	nachname	varchar(255)	[...]
7	vorname	varchar(255)	[...]
8	firmenname	varchar(255)	[...]
9	rechnung_adresse_id	int(10) unsigned	[...]
10	liefer_adresse_id	int(10) unsigned	[...]
11	bezahlart	int(10) unsigned	[...]
12	art	enum('unb','prv','gsch')	[...]
13	deleted	tinyint(3) unsigned	[...]
14			[...]

Nachträglich eine neue NOT NULL-Spalte hinzuzufügen, kann problematisch sein, wenn die Tabelle nicht leer ist. Was soll denn in die schon angelegten Zeilen hineingeschrieben werden, wenn jetzt eine neue Spalte daher kommt?

MySQL und MariaDB belegen die Spalten mit Standardwerten (siehe Abschnitt [26.1.7 auf Seite 405](#)). Sinnvoller wäre es, sich selbst eigene Standardwerte zu überlegen und diese in die Spaltenspezifikation per DEFAULT einzubauen.



Aufgabe 8.3: Verändern Sie die Spaltenspezifikationen in den entsprechenden CREATE TABLE-Anweisungen so, dass den Spalten sinnvolle Standardwerte zugewiesen werden.

8.3.3 Wie kann ich die Spezifikation einer Spalte ändern?



SQL:2016

```
ALTER TABLE tabellenname
    ALTER COLUMN spaltenname SET DATA TYPE datentyp
;
```

MySQL/MariaDB, PostgreSQL

```
ALTER TABLE tabellenname
    MODIFY [COLUMN] spaltenname spaltenspezifikation
;
```

T-SQL

```
ALTER TABLE COLUMN tabellenname spaltenname spaltenspezifikation
;
```

In SQL:2016 kann nur der Datentyp auf diese Art verändert werden. Für Zusätze wie NOT NULL müssen jeweils eigene ALTER TABLE ... ALTER COLUMN-Varianten verwendet werden.

In MySQL oder MariaDB steht bei ALTER TABLE ... MODIFY hinter dem *spaltenname* die neue Spaltenspezifikation, die genauso wie im CREATE TABLE aufgebaut ist.

Wir haben bisher die Bezahlart über eine ganze Zahl kodiert. Wie bei der Kundenart ist es aber viel aussagekräftiger, einen ENUM zu verwenden. Wir wollen die Bezahlarten per Rechnung, per Bankeinzug und per Nachname anbieten. Hier der aktuelle Zustand der Tabelle kunde:

```

1 mysql> DESCRIBE kunde\G
2 *************************** 1. row ****
3 [...]
4 *************************** 7. row ****
5   Field: bezahlart
6     Type: int(10)
7     Null: YES
8     Key:
9   Default: 0
10    Extra:
11 **************************** 8. row ****
12 [...]

```

Die Spalte **bezahlart** ist eine vorzeichenlose ganze Zahl mit der 0 als Vorbelegung. Jetzt wird die Spalte angepasst:

```

1 ALTER TABLE kunde
2   MODIFY
3     bezahlart ENUM('rechnung', 'bankeinzug', 'nachname') DEFAULT 'rechnung';

```

Das Ergebnis ist selbsterklärend:

```

1 mysql> DESCRIBE kunde\G
2 *************************** 1. row ****
3 [...]
4 *************************** 7. row ****
5   Field: bezahlart
6     Type: enum('rechnung','bankeinzug','nachname')
7     Null: YES
8     Key:
9   Default: rechnung
10    Extra:
11 **************************** 8. row ****
12 [...]

```

So weit, so einfach. Die Änderung der Spaltenspezifikation ist in vielen Fällen ohne großes Nachdenken möglich. So wird die Änderung von einem VARCHAR(255) auf einen CHAR(50) keine Probleme bereiten, oder?

8.3.4 Zeichenbasierte Spalten in der Länge verändern

Hat die neue Spaltenspezifikation eine geringere Länge als die alte, wurden in früheren Serverversionen die Daten, die nicht in die neue passen, abgeschnitten. In den aktuellen Versionen liefern MySQL und MariaDB eine Fehlermeldung (siehe [Zeile 2](#)). Als Beispiel begrenzen wir die Spalte **bezeichnung** in einer Kopie der Tabelle **artikel** auf zehn Zeichen:

```

1 mysql> ALTER TABLE bla MODIFY bezeichnung CHAR(10);
2 Query OK, 9 rows affected, 6 warnings (0.24 sec)
3 ERROR 1406 (22001): Data too long for column 'bezeichnung' at row 1

```

Es ist sinnvoll, die Längen der Werte in den Spalten vorab zu ermitteln:

```

1 mysql> SELECT MAX(CHAR_LENGTH(bezeichnung)) FROM artikel;
2 +-----+
3 | MAX(CHAR_LENGTH(bezeichnung)) |
4 |                               13 |
5 +-----+
6

```

Dieser Befehl liefert den Wert 13. Um noch ein wenig Luft nach oben zu haben, wäre die Länge 20 sicherlich ausreichend.



Aufgabe 8.4: Überprüfen Sie auf einer Kopie von Artikel (verwenden Sie CREATE TEMPORARY TABLE ... LIKE und INSERT INTO ... SELECT), ob die Länge 20 ohne Warnungen übernommen wird.

Eine Verlängerung der maximalen Länge einer zeichenbasierenden Spalte ist ohne Datenverlust möglich.

8.3.5 Zeichensatz verändern

Das kann richtig kompliziert werden. Zunächst sollte man sich darüber im Klaren sein, dass es keinen Sinn hat, Zeichensätze wie Kyrillisch (cp1251) nach Arabisch (cp1265) zu konvertieren. Aber die Konvertierungen zwischen latin1, utf8 und cp850 sind schon wahrscheinlich.

Das Konvertieren der Daten sollte m.E. nicht in den Tabellen durch einen SQL-Befehl erfolgen. Ich empfehle, die Daten zu exportieren, sie mit einem Konvertierungsprogramm² umzuwandeln und sie in die geänderte Tabelle zu importieren.

Der Vorteil dieses Vorgehens ist, dass Sie jeden einzelnen Schritt unter Kontrolle haben und die Ergebnisse überprüfen können. Lassen Sie den Datenbankserver dies tun, erhalten Sie ggf. Warnungen, aber die Daten sind dann vielleicht schon verfälscht.

8.3.6 Zeichenbasierte Spalten in numerische Spalten verändern

Weil wir eine intensive Auswertung über die örtliche Verteilung unserer Kunden mithilfe der Postleitzahl vornehmen wollen, werden wir in einer Kopie der Tabelle `adresse` die Postleitzahl in ein numerisches Feld umwandeln. Vergleiche und Sortierungen lassen sich dann erheblich schneller durchführen.

```

1 mysql> SELECT * FROM wurstbrot;
2 +-----+-----+-----+-----+-----+-----+
3 | adresse_id | strasse      | hnr | lkz | plz   | [...] |
4 +-----+-----+-----+-----+-----+-----+
5 |     1 | Beutelhaldenweg | 5   | AL  | 67676 | [...] |
6 |     2 | Beutelhaldenweg | 1   | AL  | 67676 | [...] |
7 |     3 | Auf der Feste   | 1   | GO  | 54786 | [...] |
8 |     4 | Letztes Haus    | 4   | ER  | 87567 | [...] |
9 |     5 | Baradur        | 1   | MO  | 62519 | [...] |
10 |    10 | Hochstrasse    | 4a  | DE  | 44879 | [...] |
11 |    11 | Industriegebiet | 8   | DE  | 44878 | [...] |
12 +-----+-----+-----+-----+-----+-----+
13
14 mysql> ALTER TABLE wurstbrot MODIFY plz INT;
```

² Unter Linux beispielsweise iconv oder recode

```

15 Query OK, 7 rows affected (0.00 sec)
16 Records: 7  Duplicates: 0  Warnings: 0
17
18 mysql> SELECT * FROM wurstbrot;
19 +-----+-----+-----+-----+-----+
20 | adresse_id | strasse      | hnr | lkz | plz   | [...] |
21 +-----+-----+-----+-----+-----+
22 | 1 | Beutelhaldenweg | 5 | AL | 67676 | [...] |
23 | 2 | Beutelhaldenweg | 1 | AL | 67676 | [...] |
24 | 3 | Auf der Feste   | 1 | GO | 54786 | [...] |
25 | 4 | Letztes Haus    | 4 | ER | 87567 | [...] |
26 | 5 | Baradur        | 1 | MO | 62519 | [...] |
27 | 10 | Hochstrasse    | 4a | DE | 44879 | [...] |
28 | 11 | Industriegebiet | 8 | DE | 44878 | [...] |
29 +-----+-----+-----+-----+-----+

```



Hinweis: Die Daten werden hier problemlos automatisch konvertiert, weil es keine führende 0 gibt.

Probleme gäbe es nur, wenn die ermittelte Zahl nicht in den Zieldatentyp passt:

```

1 mysql> ALTER TABLE wurstbrot MODIFY plz TINYINT;
2 ERROR 1264 (22003): Out of range value for column 'plz' at row 1

```

Was aber, wenn die Werte nicht nur numerisch sind? Konvertieren wir die Hausnummer in eine numerische Spalte:

```

1 mysql> ALTER TABLE wurstbrot MODIFY hnr INT;
2 ERROR 1265 (01000): Data truncated for column 'hnr' at row 6

```

8.3.7 Numerische Spalten im Wertebereich verändern

Wird der Wertebereich bei ganzzahligen Datentypen erweitert (z.B. INT nach BIGINT), ergeben sich keine Probleme.

Ein ALTER TABLE von negativen Werten nach UNSIGNED wird mit einer Fehlermeldung abgebrochen. Konvertierungen von DOUBLE nach FLOAT bzw. DECIMAL nach DOUBLE oder FLOAT werden unter Ausgabe einer Warnung gerundet. Dabei gehen Nachkommastellen verloren, aber nicht der Wert der Zahl als solcher³.

8.3.8 Datum- oder Zeitspalten verändern

Intern werden Datums- und Zeitangaben als ganzzahlige Werte abgespeichert. Insofern erwarte ich keine Konvertierungsprobleme. Aber *schau'n mer mal*, wie schon der Kaiser sagte:

Legen wir zuerst eine temporäre Tabelle an, fügen einen Testdatensatz ein und konvertieren die Spalte zu BIGINTs:

³ Natürlich lassen sich perverse Beispiele von DECIMALS erstellen, die bei einer Konvertierung nach FLOAT oder INT den Zahlenwert verlieren.

```

1 CREATE TEMPORARY TABLE wurstbrot
2 (
3     a DATETIME,
4     b TIME,
5     c YEAR
6 );
7
8 INSERT INTO wurstbrot (a, b, c)
9     VALUES ('2012-03-24 14:57:00', '14:57:00', '2012');
10
11 ALTER TABLE wurstbrot
12     MODIFY a BIGINT,
13     MODIFY b BIGINT,
14     MODIFY c BIGINT
15 ;
16
17 mysql> SELECT * FROM wurstbrot;
18 +-----+-----+-----+
19 | a      | b      | c      |
20 +-----+-----+-----+
21 | 20120324145700 | 145700 | 2012 |
22 +-----+-----+-----+

```

Die Konvertierung scheint keine Probleme gemacht zu haben. Dies erscheint einigermaßen sinnvoll, da Datums- und Uhrzeitwerte intern als Zahlen kodiert werden. Konsequenterweise liefert eine entsprechende *Rückkonvertierung* auch wieder die ursprünglichen Werte:

```

1 mysql> ALTER TABLE wurstbrot
2 ->     MODIFY a DATETIME,
3 ->     MODIFY b TIME,
4 ->     MODIFY c YEAR
5 -> ;
6 Query OK, 1 row affected (0.01 sec)
7 Records: 1  Duplicates: 0  Warnings: 0
8
9 mysql> SELECT * FROM wurstbrot;
10 +-----+-----+-----+
11 | a      | b      | c      |
12 +-----+-----+-----+
13 | 2012-03-24 14:57:00 | 14:57:00 | 2012 |
14 +-----+-----+-----+

```

Der gleiche Effekt kann auch bei der Konvertierung in einen zeichenbasierenden Datentyp beobachtet werden:

```

1 mysql> ALTER TABLE wurstbrot
2 ->     MODIFY a VARCHAR(255),
3 ->     MODIFY b VARCHAR(255),
4 ->     MODIFY c VARCHAR(255)
5 -> ;
6 Query OK, 1 row affected (0.00 sec)
7 Records: 1  Duplicates: 0  Warnings: 0
8
9 mysql> SELECT * FROM wurstbrot;
10 +-----+-----+-----+
11 | a      | b      | c      |
12 +-----+-----+-----+
13 | 2012-03-24 14:57:00 | 14:57:00 | 2012 |

```

```

14 +-----+-----+-----+
15
16 mysql> ALTER TABLE wurstbrot
17 -> MODIFY a DATETIME,
18 -> MODIFY b TIME,
19 -> MODIFY c YEAR
20 -> ;
21 Query OK, 1 row affected (0.00 sec)
22 Records: 1 Duplicates: 0 Warnings: 0
23
24 mysql> SELECT * FROM wurstbrot;
25 +-----+-----+-----+
26 | a      | b      | c      |
27 +-----+-----+-----+
28 | 2012-03-24 14:57:00 | 14:57:00 | 2012 |
29 +-----+-----+-----+

```

Die Angaben werden einfach als Zeichenkette dargestellt. Zwar sind jetzt keine Datums- und Zeitoperationen mehr direkt möglich, aber der Wert ist erhalten geblieben, und mit Konvertierungsfunktionen wie DATE() lassen sich diese auch weiterverarbeiten.

8.3.9 Wie kann ich aus einer Tabelle Spalten entfernen?



SQL:2016
`ALTER TABLE tabellenname DROP COLUMN spaltenname [RESTRICT|CASCADE] ;`

MySQL/MariaDB, PostgreSQL
`ALTER TABLE tabellenname DROP [COLUMN] spaltenname ;`

T-SQL
`ALTER TABLE tabellenname DROP COLUMN spaltenname ;`

Mit `ALTER TABLE ... DROP` werden Spalten aus den Tabellen entfernt. Dabei gehen natürlich auch alle darin enthaltenen Werte verloren.

Aus einer Kopie von Adresse möchte ich diverse Spalten entfernen, da ich sie nicht brauche. Zuerst der aktuelle Zustand:

```

1 mysql> DESCRIBE wurstbrot;
2 +-----+-----+-----+-----+-----+
3 | Field      | Type          | Null | Key | Default | Extra          |
4 +-----+-----+-----+-----+-----+
5 | adresse_id | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
6 | strasse     | varchar(255)   | NO   | MUL |          |                 |
7 | hnr         | varchar(255)   | NO   |      |          |                 |
8 | lkz         | char(2)        | NO   |      |          |                 |
9 | plz         | char(9)        | NO   |      |          |                 |
10 | ort         | varchar(255)   | NO   |      |          |                 |
11 | deleted     | tinyint(3) unsigned | NO   |      | 0       |                 |
12 +-----+-----+-----+-----+-----+

```

Und weg damit:

```

1 ALTER TABLE wurstbrot
2     DROP deleted, DROP strasse, DROP ort;

```

Nur noch mal nachschauen, ob die Spalten wirklich entfernt wurden:

```
1 mysql> DESCRIBE wurstbrot;
2 +-----+-----+-----+-----+-----+
3 | Field      | Type       | Null | Key | Default | Extra       |
4 +-----+-----+-----+-----+-----+
5 | adresse_id | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
6 | hnr        | varchar(255)   | NO   | MUL |          |               |
7 | lkz        | char(2)       | NO   |      |          |               |
8 | plz        | char(9)       | NO   |      |          |               |
9 +-----+-----+-----+-----+-----+
```

In SQL:2016 hat man die Möglichkeit, mithilfe der Optionen RESTRICT und CASCADE zu steuern, was mit Daten passieren soll, welche mit den zu löschen Spalten – beispielsweise über einen Constraint – verknüpft sind. Bei RESTRICT – der Default – wird das Löschen verweigert, und bei CASCADE werden alle verknüpften Daten ebenfalls gelöscht. MySQL und MariaDB verhalten sich wie die Option RESTRICT.



Hinweis: Sie erhalten keine Warnung oder Rückfrage. Die Spalten und ihre Inhalte sind ... weg!

■ 8.4 Eine Tabelle löschen



Requiescat in pace. Eine Tabelle weniger.

- Grundkurs: Tabelle mit DROP TABLE löschen
- Vertiefendes
 - Löschreihenfolge bei Constraints
 - CASCADE und RESTRICT

Wie das Löschen der Datenbank (siehe [Seite 127](#)) ist auch das Löschen von Tabellen mit **DROP TABLE** einfach.



SQL:2016

`DROP TABLE tabellename [RESTRICT|CASCADE] ;`

MySQL/MariaDB, PostgreSQL, T-SQL

`DROP TABLE [IF EXISTS] tabellename [RESTRICT|CASCADE] ;`



Hinweis: Die Tabellen werden ohne Rückfrage oder Warnung endgültig entfernt. Es gibt kein Undo, um die Tabellen wieder herzustellen.

8.4.1 Einfach löschen

Schauen wir uns an, was wir an Tabellen haben:

```
1 mysql> SHOW TABLES;
2 +-----+
3 | Tables_in_oshop      |
4 +-----+
5 | adresse              |
6 | artikel               |
7 | artikel_nm_lieferant |
8 | artikel_nm_warengruppe|
9 | bank                 |
10| bankverbindung        |
11| bestellung            |
12| bestellung_position   |
13| bild                 |
14| kunde                |
15| lieferant             |
16| rechnung              |
17| rechnung_position     |
18| warengruppe           |
19 +-----+
20 14 rows in set (0.00 sec)
```

Jetzt löschen wir nicht nur eine, sondern gleich mehrere Tabellen.

```
1 mysql> DROP TABLE
2       > rechnung_position, rechnung, bestellung_position,bestellung
3       > ;
4
5 mysql> SHOW TABLES;
6 +-----+
7 | Tables_in_oshop      |
8 +-----+
9 | adresse              |
10| artikel               |
11| artikel_nm_lieferant |
12| artikel_nm_warengruppe|
13| bank                 |
14| bankverbindung        |
15| bild                 |
16| kunde                |
17| lieferant             |
18| warengruppe           |
19 +-----+
20 10 rows in set (0.00 sec)
```

Was passiert eigentlich mit den Indizes der entsprechenden Tabellen? Diese werden mit gelöscht. Durch das Löschen einer Tabelle werden alle damit verbundenen Indizes, Constraints etc. gelöscht.

8.4.2 Was bedeuten die Optionen CASCADE und RESTRICT?



Aufgabe 8.5: Was für eine Fehlermeldung erwarte ich, wenn die Tabellen in folgender Reihenfolge zu löschen versucht werden: rechnung, rechnung_position?

Besteht ein Constraint zwischen zwei Tabellen – z.B. eine Primär-Fremdschlüsselbeziehung –, wird dieser verletzt, wenn man beispielsweise die Primärschlüsseltabelle löscht.

Wird die Option RESTRICT angegeben, scheitert der Löschversuch. Wird die Option CASCADE angegeben, werden alle verbundenen Datenobjekte wie Constraints oder Ansichten auch gelöscht. Schauen wir uns das mal in PostgreSQL⁴ an:

```

1 oshop=# \dt
2 List of relations
3 Schema | Name | Type | Owner
4 -----+-----+-----+-----+
5 public | adresse | table | postgres
6 public | artikel | table | postgres
7 public | artikel_nm_lieferant | table | postgres
8 public | artikel_nm_warengruppe | table | postgres
9 public | bank | table | postgres
10 public | bankverbindung | table | postgres
11 public | bestellung | table | postgres
12 public | bestellung_position | table | postgres
13 public | bild | table | postgres
14 public | kunde | table | postgres
15 public | lieferant | table | postgres
16 public | rechnung | table | postgres
17 public | rechnung_position | table | postgres
18 public | warengruppe | table | postgres
19 (14 rows)
20
21 oshop=# DROP TABLE kunde RESTRICT;
22 ERROR: cannot drop table kunde because other objects depend on it
23 DETAIL: constraint bankverbindung_kunde_id_fkey on table bankverbindung
          depends on table kunde
24 constraint bestellung_kunde_id_fkey on table bestellung depends on table kunde
25 constraint rechnung_kunde_id_fkey on table rechnung depends on table kunde
26 HINT: Use DROP ... CASCADE to drop the dependent objects too.
27
28 oshop=# DROP TABLE kunde CASCADE;
29 NOTICE: drop cascades to 3 other objects
30 DETAIL: drop cascades to constraint bankverbindung_kunde_id_fkey on table
          bankverbindung
31 drop cascades to constraint bestellung_kunde_id_fkey on table bestellung
32 drop cascades to constraint rechnung_kunde_id_fkey on table rechnung

```

⁴ In MySQL und MariaDB werden diese Optionen ignoriert und führen somit auch zu keiner Fehlermeldung.

9

Werte in Tabellen verändern

■ 9.1 WHERE-Klausel



Mit WHERE nur die Zeilen auswählen, die betroffen sind.

- Grundkurs
 - Bedingung und Wahrheitswerte
 - Verknüpfung von Teilbedingungen
- Vertiefendes: Groß- und Kleinschreibung



Die Quelltexte dieses Kapitels stehen in der Datei `Listing06.sql` (siehe [Listing 29.6 auf Seite 475](#), [Listing 29.42 auf Seite 598](#) und [Listing 29.27 auf Seite 547](#)).

Wir werden in den nachfolgenden Kapiteln Befehle kennenlernen, die Tabelleninhalte auswerten, verändern und löschen. Diese Befehle werden mal auf alle Zeilen der Tabelle angewendet, mal auf eine Teilmenge.

Damit SQL unterscheiden kann, auf welche Teilmenge der Befehl eingeschränkt wird, müssen wir für die Zeilen Bedingungen formulieren können.



Definition 34: Bedingung

Eine *Bedingung* (engl: condition) ist eine Frage, die der Computer mit TRUE oder FALSE beantworten muss.

In SQL¹ kann ein boolescher Ausdruck auch noch einen dritten Wert annehmen: UNKNOWN. Dieser tritt in der Regel auf, wenn wir mit NULL-Werten vergleichen.

Für die Befehle UPDATE (siehe [Abschnitt 9.2 auf Seite 146](#)), DELETE (siehe [Abschnitt 9.3 auf Seite 150](#)) und SELECT (siehe [Kapitel IV auf Seite 155](#)) werden die Bedingungen durch das Schlüsselwort WHERE eingeleitet.

¹ Der Datentyp BOOL ist eigentlich nur ein anderer Ausdruck für ein TINYINT mit der Domäne 0 (=FALSE), 1 (=TRUE) und NULL (=UNKNOWN).



SQL:2016
MySQL/MariaDB, PostgreSQL, T-SQL

```
{UPDATE | DELETE | SELECT} befehlsinhalt
    WHERE
        bedingung
;
```



Definition 35: WHERE-Klausel

Die *WHERE-Klausel* schränkt die voranstehende Operation auf die Zeilen ein, für welche die Bedingung TRUE ist.

9.1.1 Wie formuliert man eine einfache Bedingung?

Es gibt zwei *sehr* einfache Bedingungen².

- 1 `SELECT * FROM artikel WHERE TRUE;`
- 2 `SELECT * FROM artikel WHERE FALSE;`



Aufgabe 9.1: Der Befehl `SELECT` liefert mir den Inhalt einer Tabelle. Welche Ausgabe erwarte ich bei [Zeile 1](#) und welche bei [2?](#) Verwenden Sie in Ihrer Begründung die [Definition 35](#).

So triviale Bedingungen kommen normalerweise nicht vor. Vielmehr wollen wir Bedingungen bauen, die die inhaltlichen Eigenschaften einer Zeile untersucht: Ist `delete` auch mit 0 gefüllt? Ist der Kunde volljährig? Kommen die Kunden aus einem bestimmten Postleitzahlbereich? Etc.

Dazu muss man Spalteninhalte miteinander oder gegen Konstanten oder Ausdrücke vergleichen. Eine Liste der in SQL möglichen Vergleichsoperatoren finden Sie in [Abschnitt 26.3.1 auf Seite 414](#).



Aufgabe 9.2: Führen Sie folgende Befehle aus und interpretieren Sie die Ergebnisse:

```
SELECT * FROM artikel WHERE artikel_id = 3010;
SELECT * FROM artikel WHERE artikel_id != 3010;
SELECT * FROM artikel WHERE einzelpreis < 5.0;
SELECT * FROM artikel WHERE einzelpreis <= 5.0;
SELECT * FROM artikel WHERE einzelpreis > 5.0;
SELECT * FROM artikel WHERE einzelpreis >= 5.0;
SELECT * FROM artikel WHERE einzelpreis BETWEEN 1.00 AND 15.00;
SELECT * FROM artikel WHERE einzelpreis NOT BETWEEN 1.00 AND 15.00;
SELECT * FROM artikel WHERE artikel_id IN (3001, 7856, 9015);
SELECT * FROM artikel WHERE artikel_id NOT IN (3001, 7856, 9015);
SELECT * FROM artikel WHERE bezeichnung LIKE 'Tinte%';
SELECT * FROM artikel WHERE bezeichnung = 'Tinte%';
```

² In TSQL gibt es keine booleschen Datentypen; man behilft sich mit dem Datentyp BIT. Deshalb kann auch nicht auf TRUE oder FALSE verglichen werden.

```

SELECT * FROM kunde WHERE liefer_adresse_id = NULL;
SELECT * FROM kunde WHERE liefer_adresse_id IS NULL;
SELECT * FROM kunde WHERE liefer_adresse_id IS NOT NULL;

```

Aufgabe 9.3: Ermitteln Sie die passende WHERE-Klausel zum SELECT für folgende Anforderungen:

- Geben Sie den Kunden mit dem Namen 'Beutlin' aus.
- Geben Sie alle Kunden, die nicht 'Beutlin' heißen, aus.
- Geben Sie alle Kunden mit einer Kundennummer kleiner als 4 aus.
- Geben Sie alle Kunden mit einer Kundennummer größer oder gleich 4 aus.
- Geben Sie alle Kunden mit einer Kundennummer von 2 bis 4 aus.
- Geben Sie alle Kunden mit einer Kundennummer kleiner 2 oder größer 4 aus.
- Geben Sie alle Kunden, deren Nachname ein 'n' enthalten, aus.

9.1.2 Wird zwischen Groß- und Kleinschreibung unterschieden?

Ein Versuch sagt mehr als 1000 Worte³:

```

1 mysql> CREATE TEMPORARY TABLE wurstbrot (
2      ->     name VARCHAR(255)
3      -> );
4
5 mysql> INSERT INTO wurstbrot VALUES
6      ->     ('Jaqueline')
7      ->     ,('Kevin')
8      ->     ,('kevin')
9      ->     ,('jaqueline');
10
11 mysql> SELECT * FROM wurstbrot WHERE name = 'kevin';
12 +-----+
13 | name   |
14 +-----+
15 | Kevin  |
16 | kevin  |
17 +-----+

```

Offensichtlich wird in MySQL und MariaDB standardmäßig nicht zwischen Groß- und Kleinschreibung unterschieden.⁴ Möchte man zwischen Groß- und Kleinschreibung unterscheiden, was beispielsweise bei Passwortinhalten⁵ notwendig wäre, so muss der Zusatz **BINARY** verwendet werden.

```

1 mysql> DROP TABLE wurstbrot;
2
3 mysql> CREATE TEMPORARY TABLE wurstbrot (
4      ->     name VARCHAR(255) BINARY

```

³ Dabei ist ein Quelltext auch nur ein Haufen Worte. Darüber sollte man mal nachdenken ...

⁴ In PostgreSQL allerdings schon. Dort hätte man schreiben müssen:
SELECT * FROM wurstbrot WHERE LOWER(name) = 'kevin';

⁵ Die natürlich NIE NIE NIE klartextlich abgespeichert werden.

```

5      -> );
6
7 mysql> INSERT INTO wurstbrot VALUES
8      -> ('Jaqueline')
9      -> ,('Kevin')
10     -> ,('kevin')
11     -> ,('jaqueline');
12
13 mysql> SELECT * FROM wurstbrot WHERE name = 'kevin';
14 +-----+
15 | name   |
16 +-----+
17 | kevin  |
18 +-----+

```

In T-SQL wird dies durch die Angabe _CS bzw. _CI in der Sortierung (siehe Zeile 26) gesteuert:

```

1  > DROP TABLE IF EXISTS #wurstbrot;
2  2> GO
3  1> CREATE TABLE #wurstbrot (
4  2>   name NVARCHAR(15)
5  3> );
6  4> GO
7  1> INSERT INTO #wurstbrot VALUES
8  2> ('Jaqueline'),
9  3> ('Kevin'),
10 4> ('kevin'),
11 5> ('jaqueline')
12 6> ;
13 7> SELECT * FROM #wurstbrot WHERE name = 'kevin';
14 8> GO
15
16 (4 rows affected)
17 name
18 -----
19 Kevin
20 kevin
21
22 (2 rows affected)
23 1> DROP TABLE #wurstbrot;
24 2> GO
25 1> CREATE TABLE #wurstbrot (
26 2>   name NVARCHAR(15) COLLATE SQL_Latin1_General_CI_AS
27 3> );
28 4> GO
29 1> INSERT INTO #wurstbrot VALUES
30 2> ('Jaqueline'),
31 3> ('Kevin'),
32 4> ('kevin'),
33 5> ('jaqueline')
34 6> ;
35 7> GO
36
37 (4 rows affected)
38 1> SELECT * FROM #wurstbrot WHERE name = 'kevin';
39 2> GO
40 name
41 -----

```

```

42 kevin
43
44 (1 rows affected)
45 1>

```



Hinweis: In SQL:2016 gibt es keine Möglichkeit, zwischen Groß- und Kleinschreibung zu unterscheiden.

9.1.3 Wie formuliert man eine zusammengesetzte Bedingung?

Oft sind die fachlichen Anforderungen, die für die Operation zutreffen müssen, aus mehreren Bedingungen zusammengesetzt: Bochumer Kunden mit mehr als 10 Bestellungen im letzten Jahr, nicht gelöschte Bestellpositionen mit einer Menge von 0 usw.

Teilbedingungen müssen daher zu Gesamtbedingungen zusammengesetzt werden. Eine Liste der verfügbaren Verknüpfungsoperatoren finden Sie in [Abschnitt 26.3.2 auf Seite 416](#).



Hinweis: PostgreSQL und TSQL kennen keinen logischen XOR-Operator. Dieser kann aber anhand folgender Gleichung simuliert werden:

$$A \otimes B = (A \vee B) \wedge \neg(A \wedge B)$$



Aufgabe 9.4: Führen Sie folgende Befehle aus und interpretieren Sie die Ergebnisse:

```

SELECT * FROM artikel
WHERE
    waehrung = 'EUR' AND einzelpreis > 10.0;

SELECT * FROM artikel
WHERE
    waehrung = 'USD' OR einzelpreis <= 10.0;

SELECT * FROM artikel
WHERE
    waehrung != 'EUR';

SELECT * FROM kunde
WHERE
    (rechnung_adresse_id IS NULL) XOR (liefer_adresse_id IS NOT NULL);

```

Um die Ausführungsreihenfolge festzulegen und die Übersichtlichkeit zu erhöhen, werden Klammerungen verwendet. Verwenden Sie auch dann die Klammern, wenn die Ausführungsreihenfolge klar ist.



Aufgabe 9.5: Führen Sie folgende Befehle aus und interpretieren Sie die Ergebnisse:

```

SELECT * FROM artikel
WHERE
    artikel_id > 3006 AND einzelpreis > 10.0 OR waehrung = 'EUR';

```

```

SELECT * FROM artikel
WHERE
    (artikel_id > 3006 AND einzelpreis > 10.0) OR (waehrung = 'EUR');

SELECT * FROM artikel
WHERE
    (artikel_id > 3006) AND (einzelpreis > 10.0 OR waehrung = 'EUR');

```

Aufgabe 9.6: Ermitteln Sie die passende WHERE-Klausel zum SELECT für folgende Anforderungen:

- Geben Sie alle Kunden, die 'Beutlin' heißen oder eine Lieferadresse haben, aus.
- Geben Sie alle Kunden mit dem Nachnamen 'Beutlin' und dem Vornamen 'Frodo' aus.
- Geben Sie alle Kunden, die entweder eine Lieferadresse haben oder Geschäftskunden sind, aus.

Der Aufbau der WHERE-Klausel kann beliebig kompliziert werden. Wir haben jetzt nur die einfachen Grundgerüste kennengelernt. So wird der Inhalt der Werteliste bei dem Operator IN oft durch eigene SELECT⁶ ermittelt. Auch der obere und untere Wert beim Operator BETWEEN wird meist dynamisch aus Tabelleninhalten berechnet.

Freuen wir uns also auf viel kompliziertere WHERE-Klauseln ;-).

■ 9.2 Tabelleninhalte verändern



Alles ändert sich, nichts bleibt, wie es ist.

- Grundkurs
 - Stammdaten und Bewegungsdaten
 - Einfache Wertzuweisung mit UPDATE
 - Werte berechnen
- Vertiefendes
 - LOW_PRIORITY
 - IGNORE



Die Quelltexte dieses Kapitels stehen in der Datei `listing06.sql` (siehe Listing 29.6 auf Seite 475, Listing 29.42 auf Seite 598 und Listing 29.27 auf Seite 547).

⁶ Siehe Unterabfragen in Abschnitt 13 auf Seite 225

Wir haben Tabelleninhalte mit `INSERT INTO` oder `LOAD DATA INFILE` erstellt. Viele Tabelleninhalte werden selten bis nie verändert, andere häufiger.



Definition 36: Stammdaten/Bewegungsdaten

Tabellen, deren Inhalte eine geringe Änderungswahrscheinlichkeit aufweisen und in den Geschäftsprozessen eine nicht triviale Bedeutung haben, heißen *Stammdaten*.

Tabellen, deren Inhalte eine hohe Änderungswahrscheinlichkeit besitzen und in den Geschäftsprozessen eine nicht triviale Bedeutung haben, heißen *Bewegungsdaten*.

In unserem Beispiel oshop würde ich die Tabellen wie folgt kategorisieren:

Tabelle 9.1 Stamm- und Bewegungsdaten im oshop

Stammdaten	Bewegungsdaten
adresse	bestellung
artikel	bestellung_position
artikel_nm_warengruppen	lagerbestand
artikel_nm_lieferant	rechnung
bank	rechnung_position
bankverbindung	
kunde	
bild	
lieferant	
warengruppe	

Wir werden in den Bewegungsdaten mit dem Befehl `UPDATE` Änderungen vornehmen:



SQL:2016, PostgreSQL, T-SQL

```
UPDATE tabellenname
    SET
        spaltenname=ausdruck
        [,spaltenname=ausdruck]*
        [WHERE bedingung]
;
```

MySQL/MariaDB

```
UPDATE [LOW_PRIORITY] [IGNORE] tabellenname
    SET
        spaltenname=ausdruck
        [,spaltenname=ausdruck]*
        [WHERE bedingung]
;
```

9.2.1 Szenario 1: Einfache Wertzuweisung

Der Kunde ändert die Mengenangabe in einer Bestellung. In der Bestellung 1 sollen zwei anstelle von einem Spaten aufgegeben werden.

Die Daten in der Tabelle `bestellung_position` sind folgende:

```

1 mysql> SELECT * FROM bestellung_position;
2 +-----+-----+-----+-----+
3 | bestellung_id | position_nr | artikel_id | menge      | deleted |
4 +-----+-----+-----+-----+
5 [...]          1 |           3 |         9015 | 1.000000 |       0 |
6 [...]          1 |           3 |         9015 | 1.000000 |       0 |
7 [...]          1 |           3 |         9015 | 1.000000 |       0 |
8 +-----+-----+-----+-----+

```

In der Zeile 6 befindet sich der zu ändernde Datensatz. Um den Datensatz eindeutig zu identifizieren, eignet sich am besten der Primärschlüssel.



Aufgabe 9.7: Wie bekommt man heraus, welche Spalten den Primärschlüssel einer Tabelle bilden?

```

1 mysql> UPDATE bestellung_position
2   -> SET menge = 2
3   -> WHERE bestellung_id = 1 AND position_nr = 3;
4
5 mysql> SELECT * FROM bestellung_position;
6 +-----+-----+-----+-----+
7 | bestellung_id | position_nr | artikel_id | menge      | deleted |
8 +-----+-----+-----+-----+
9 [...]          1 |           3 |         9015 | 2.000000 |       0 |
10 [...]          1 |           3 |         9015 | 2.000000 |       0 |
11 [...]          1 |           3 |         9015 | 2.000000 |       0 |
12 +-----+-----+-----+-----+

```

In der Zeile 10 kann der geänderte Wert überprüft werden.

9.2.2 Szenario 2: Berechnete Werte

Die Preise sollen pauschal um 1% erhöht werden.

```

1 mysql> SELECT * FROM artikel;
2 +-----+-----+-----+-----+
3 | artikel_id | bezeichnung | einzelpreis | waehrung | deleted |
4 +-----+-----+-----+-----+
5 |     3001 | Papier (100) | 2.300000 | EUR      |       0 |
6 |     3005 | Tinte (gold) | 55.700000 | EUR      |       0 |
7 |     3006 | Tinte (rot)  | 6.200000 | EUR      |       0 |
8 |     3007 | Tinte (blau) | 4.130000 | EUR      |       0 |
9 |     3010 | Feder       | 5.000000 | EUR      |       0 |
10 |    7856 | Silberwiebel | 0.510000 | EUR      |       0 |
11 |    7863 | Tulpenzwiebel | 3.390000 | EUR      |       0 |
12 |    9010 | Schaufel    | 14.950000 | USD      |       0 |
13 |    9015 | Spaten      | 19.900000 | EUR      |       0 |
14 +-----+-----+-----+-----+

```

Jetzt werden die Preise aktualisiert. Da alle Preise aktualisiert werden sollen, kann auf eine WHERE-Klausel verzichtet werden.

```
1 UPDATE artikel SET einzelpreis = einzelpreis + einzelpreis / 100.0;
```

Wir haben hier erstmalig die *rechnerischen* Fähigkeiten von SQL kennengelernt. Eine Liste der mathematischen Operationen finden Sie in [Abschnitt 26.2.1 auf Seite 408](#) und von mathematischen Funktionen in [Abschnitt 26.2.2 auf Seite 408](#).

```
1 mysql> SELECT * FROM artikel;
2 +-----+-----+-----+-----+
3 | artikel_id | bezeichnung | einzelpreis | waehrung | deleted |
4 +-----+-----+-----+-----+
5 | 3001 | Papier (100) | 2.323000 | EUR | 0 |
6 | 3005 | Tinte (gold) | 56.257000 | EUR | 0 |
7 | 3006 | Tinte (rot) | 6.262000 | EUR | 0 |
8 | 3007 | Tinte (blau) | 4.171300 | EUR | 0 |
9 | 3010 | Feder | 5.050000 | EUR | 0 |
10 | 7856 | Silberwiebel | 0.515100 | EUR | 0 |
11 | 7863 | Tulpenzwiebel | 3.423900 | EUR | 0 |
12 | 9010 | Schaufel | 15.099500 | USD | 0 |
13 | 9015 | Spaten | 20.099000 | EUR | 0 |
14 +-----+-----+-----+-----+
```

Wir haben es nun mit sehr unschönen Preisen zu tun. Besser wäre ein Runden auf die zweite Stelle hinter dem Komma.

```
1 UPDATE artikel SET einzelpreis = ROUND(einzelpreis, 2);
```



Aufgabe 9.8: Schauen Sie sich die Preise an, und ermitteln Sie die Rundungsregeln.

9.2.3 Szenario 3: Gebastelte Zeichenketten

Für den Briefverkehr soll eine Anrede verwendet werden. In der Tabelle kunde müssen wir diese Spalte zuerst einfügen.

```
1 ALTER TABLE kunde ADD anrede VARCHAR(255) AFTER nachname;
```

Jetzt können wir aus dem Nachnamen eine Anrede generieren. Da wir keine Unterscheidungsmöglichkeit für Männer und Frauen haben, muss diese Anrede neutral gehalten sein. Dabei sollen alle Datensätze ausgespart werden, für die kein Nachname erfasst ist.

```
1 UPDATE kunde
2 SET anrede = CONCAT('Sehr geehrte/r Frau/Herr ', nachname)
3 WHERE nachname <> '';
```



Aufgabe 9.9: Betrachten Sie das Ergebnis, und überlegen Sie sich die Funktionsweise von CONCAT().

9.2.4 Was bedeutet die Option LOW_PRIORITY?

Ein Update kann recht lange dauern. Entweder weil es auf einer großen Datenmenge arbeitet, oder weil es schwer zu berechnen ist. In beiden Fällen würden konkurrierende Zugriffe auf die Tabelle ausgebremst oder gar blockiert werden.

Mit der Option `LOW_PRIORITY` wird der `UPDATE` so lange verzögert, bis keine Verbindung mehr auf der Tabelle arbeitet.

9.2.5 Was bedeutet die Option IGNORE?

Eine Aktualisierung der Daten bricht ab, sobald ein Fehler auftritt. Man könnte beispielsweise allen Zeilen den gleichen Primärschlüssel zuweisen wollen oder in einem Fremdschlüssel einen Wert eintragen, der einem Constraint widerspricht.

Will man, dass das Aktualisieren aber weitergehen soll, gibt man die Option `IGNORE` an. Dubletten in Schlüsselspalten werden dabei nicht angelegt.

■ 9.3 Tabelleninhalte löschen



Die verbotene Frucht. Fluch und Segen des Löschens von Daten.

- Grundkurs
 - Löschen mit `DELETE`
 - Tabellen mit einem Schlag leeren
- Vertiefendes
 - Constraints
 - `AUTO_INCREMENT`
 - `LOW_PRIORITY`
 - `QUICK`
 - `IGNORE`



Die Quelltexte dieses Kapitels stehen in der Datei `Listing06.sql` (siehe [Listing 29.6 auf Seite 475](#), [Listing 29.42 auf Seite 598](#) und [Listing 29.27 auf Seite 547](#)).

Wir wollen eine Bestellposition löschen. Schauen wir uns vorher den Bestand an Bestellpositionen an:

```
1 mysql> SELECT * FROM bestellung_position;
2 +-----+-----+-----+-----+
3 | bestellung_id | position_nr | artikel_id | menge      | deleted |
4 +-----+-----+-----+-----+
5 |           1 |           1 |       7856 | 30.000000 |       0 |
```

6	1	2	7863	50.000000	0
7	1	3	9015	1.000000	0
8	2	1	7856	10.000000	0
9	2	2	9010	5.000000	0
10					

Der Befehl `DELETE` ist einfach. Das Komplizierteste an ihm ist die `WHERE`-Klausel.



SQL:2016, PostgreSQL, T-SQL

```
DELETE FROM tabellenname
[WHERE bedingung]
;
```

MySQL/MariaDB

```
DELETE FROM [LOW_PRIORITY] [QUICK] [IGNORE] tabellenname
[WHERE bedingung]
;
```

Löschen wir die Positionen der Bestellung 1:

```
1 DELETE FROM bestellung_position WHERE bestellung_id = 1;
```

Das Ergebnis ist wie erwartet:

```
1 mysql> SELECT * FROM bestellung_position;
2 +-----+-----+-----+-----+-----+
3 | bestellung_id | position_nr | artikel_id | menge      | deleted |
4 +-----+-----+-----+-----+-----+
5 |           2 |           1 |       7856 | 10.000000 |       0 |
6 |           2 |           2 |       9010 | 5.000000  |       0 |
7 +-----+-----+-----+-----+-----+
```

Bitte beachten Sie, dass drei Zeilen aus der Tabelle gelöscht werden.



Hinweis: Die Daten werden ohne Rückfrage gelöscht! Nach der Drei-Schichten-Architektur (siehe [Seite 80](#)) wird das Löschen durch andere Schichten der Anwendung plausibilisiert.

9.3.1 Und was passiert bei Constraints?

Können Sie sich noch an die Lösch- und Änderungsweitergabe erinnern (siehe [Seite 85](#))? Wäre unsere Tabellen mit Löschweitergabe erstellt worden, würde Folgendes passieren:

```
1 mysql> DELETE FROM kunde WHERE kunde_id = 1;
2 Query OK, 1 row affected (0.02 sec)
3
4 mysql> SELECT * FROM bestellung;
5 +-----+-----+-----+-----+-----+-----+-----+
6 | bestellung_id | kunde_id | adresse_id | datum          | [...] |
7 +-----+-----+-----+-----+-----+-----+-----+
8 |           2 |         2 |           2 | 2012-03-23 16:11:00 | [...] |
9 +-----+-----+-----+-----+-----+-----+-----+
10 1 rows in set (0.00 sec)
11
```

```

12 mysql> SELECT * FROM bestellung_position;
13 +-----+-----+-----+-----+
14 | bestellung_id | position_nr | artikel_id | menge      | deleted |
15 +-----+-----+-----+-----+
16 |           1 |           1 |       7856 | 30.000000 |       0 |
17 |           1 |           2 |       7863 | 50.000000 |       0 |
18 |           1 |           3 |       9015 | 1.000000 |       0 |
19 +-----+-----+-----+-----+
20 3 rows in set (0.00 sec)

```

Bitte beachten Sie, dass in [Zeile 2](#) nur ein Hinweis auf eine gelöschte Zeile erscheint. Ebenso erscheint keine Warnung. Ein anschließender Blick in die Tabelle `bestellung` zeigt mir, dass alle Bestellungen des Kunden 1 ebenfalls gelöscht wurden ([Zeile 4ff](#)). Und nicht nur das: In der Tabelle `bestellung_position` sind alle Positionen der Bestellung gelöscht worden.

Jetzt sind unsere Tabellen aber nicht mit `ON DELETE CASCADE`, sondern `ON DELETE RESTRICT` erstellt worden. Der Löschversuch wird daher zurückgewiesen:

```

1 mysql> DELETE FROM kunde WHERE kunde_id = 1;
2 ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key
   constraint fails ('oshop'.'bestellung', CONSTRAINT 'bestellung_ibfk_1'
   FOREIGN KEY ('kunde_id') REFERENCES 'kunde' ('kunde_id') ON DELETE
   RESTRICT ON UPDATE RESTRICT)

```

9.3.2 Was passiert mit dem AUTO_INCREMENT?

Ein immer wieder vorgetragenes Missverständnis: Der `AUTO_INCREMENT` nimmt als neuen Wert $id_{neu} = \text{Max}(id_{alt}) + 1$ an. Man könnte sogar annehmen, dass Lücken in der Zahlenfolge aufgefüllt werden.

Schauen wir uns das an einem Beispiel an:

```

1 mysql> SELECT kunde_id, nachname FROM kunde;
2 +-----+-----+
3 | kunde_id | nachname      |
4 +-----+-----+
5 |       3 | Beutlin      |
6 |       2 | Beutlin      |
7 |       5 | Earendilionn |
8 |       1 | Gandschie    |
9 |       4 | Telcontar    |
10 +-----+-----+
11
12 mysql> DELETE FROM kunde WHERE kunde_id IN (3,5);
13
14 mysql> INSERT INTO
15   >   kunde (nachname, vorname)
16   >   VALUES ('Eichenschild', 'Thorin')
17   > ;
18
19 mysql> SELECT kunde_id, nachname FROM kunde;
20 +-----+-----+
21 | kunde_id | nachname      |
22 +-----+-----+
23 |       2 | Beutlin      |

```

```
24 |      6 | Eichenschild |
25 |      1 | Gamdschie   |
26 |      4 | Telcontar   |
27 +-----+-----+
```

Es werden somit keine Lücken gefüllt oder die obige `Max()`-Formel verwendet. Die Tabellen müssen sich irgendwo den zuletzt erzeugten oder nächsten `AUTO_INCREMENT`-Wert merken. Und tatsächlich: Mit `SHOW CREATE TABLE` kriegt man heraus, welchen Wert der nächste `AUTO_INCREMENT` liefern wird (siehe [Zeile 7](#)).

```
1 mysql> SHOW CREATE TABLE kunde\G
2 *----- 1. row -----
3     Table: kunde
4 Create Table: CREATE TABLE `kunde` (
5   `kunde_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
6   [...]
7 ) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
```



Hinweis: Dies bedeutet auch, dass ein Löschen des gesamten Inhalts nicht dazu führt, dass wieder mit 1 angefangen wird.

9.3.3 Was bedeutet LOW_PRIORITY?

Ein Löschvorgang kann recht lange dauern. Entweder weil er auf einer großen Datenmenge arbeitet oder weil er eine schwer auszuwertende WHERE-Klausel hat. In beiden Fällen würden konkurrierende Zugriffe auf die Tabelle ausgebremst oder gar blockiert werden.

Mit der Option `LOW_PRIORITY` wird der Löschvorgang solange verzögert, bis keine Verbindung mehr auf der Tabelle arbeitet.

9.3.4 Was bedeutet QUICK?

Das Löschen von Datensätzen kann die Reorganisation von Indizes auslösen, was recht langsam ist. Bei MyISAM-Tabellen kann die Option `QUICK` das Löschen so organisieren, dass es schneller ist.

9.3.5 Was bedeutet IGNORE?

Ein Löschen der Daten bricht ab, sobald ein Fehler auftritt. So könnte durch eine Löschoperation ein Constraint verletzt werden. Will man, dass das Löschen mit der nächsten Zeile fortgesetzt werden soll, gibt man die Option `IGNORE` an.

9.3.6 Wie kann man eine Tabelle komplett leeren?

Dazu gibt es zwei Möglichkeiten. Die erste ist ein `DELETE` ohne WHERE-Klausel. Da die WHERE-Klausel eine Einschränkung darstellt, bedeutet das Fehlen der WHERE-Klausel, dass das Löschen ohne Einschränkung erfolgt⁷:

```
1  DELETE FROM bestellung_position;
```

Damit wird die Tabelle aber nicht wieder in den *Startzustand* versetzt. Hierfür gibt es eine Alternative:



SQL:2016, T-SQL

```
TRUNCATE TABLE tabellename ;
```

MySQL/MariaDB, PostgreSQL

```
TRUNCATE [TABLE] tabellename ;
```

```
1  TRUNCATE bestellung_position;
```

Außer in der InnoDB-Engine wird die Tabelle komplett gelöscht und wieder neu erzeugt. Dies ist in der Regel sehr viel schneller, als jede Zeile einzeln zu löschen, wie es die InnoDB u.U. tut. Bei jeder Engine in MySQL und MariaDB wird der Zähler vom `AUTO_INCREMENT` wieder auf 1 gesetzt.

Wie bei `DELETE` werden – sofern die Engine es unterstützt – bei einem `TRUNCATE` die Constraints überprüft.



Hinweis: Bei einem `TRUNCATE` werden die Tabelleninhalte ohne Rückfrage endgültig gelöscht.

⁷ Aha!

TEIL IV

Datenbank auswerten

10

Einfache Auswertungen



Wie, SQL ist kein WRITE ONLY-Baustein? Wie wir die Daten wieder auslesen können.

- Grundkurs
 - Konstanten
 - Mathematische Operatoren
 - Auswahl von Spalten
 - Verwendung der WHERE-Klausel
 - Sortieren mit ORDER BY
 - Sortierreihenfolge mit ASC und DESC
 - Groß- und Kleinschreibung bei der Sortierung
 - Sortieren von Datum- und Uhrzeitwerten
 - Mehrfachausgabe mit DISTINCT unterbinden
 - Teilergebnisse mit LIMIT
- Vertiefendes
 - Operatorenpriorität
 - Zufallszahlen
 - Variablen
 - Collations
 - Einfluss von Indizes auf Sortierung
 - EXPLAIN
 - Einfluss von Indizes auf DISTINCT
 - Daten exportieren mit SELECT ... INTO OUTFILE
 - Binäre Daten auslesen

Der Befehl SELECT ist das Herz von SQL. Alle anderen Befehle haben natürlich ihre Berechtigung, aber SELECT stellt mir die Daten aufbereitet zur Verfügung. Dabei kann der SELECT sowohl Säbel als auch Florett sein. Von kleinen trivialen Auswertungen bis zu mehrseitigen Analysen ist mit diesem Wunderkind alles möglich. Für mich als Programmierer ist es schon sehr erstaunlich, dass 90% der Programmierarbeit mit Varianten eines Befehls erfolgen – und eben genau das macht die Eleganz und den Reiz der SQL-Programmierung aus.



Die Quelltexte dieses Kapitels stehen in der Datei listing07.sql (siehe Listing 29.7 auf Seite 477, Listing 29.43 auf Seite 600 und Listing 29.28 auf Seite 549).

Die Dateien perfIndexOrderBy01.sql und blz_20120305.csv werden verwendet.

■ 10.1 Ausdrücke



SQL:2016, MySQL/MariaDB, PostgreSQL, T-SQL

SELECT ausdruck

;

10.1.1 Konstanten

Der einfachste Ausdruck ist eine Konstante. Dies kann eine numerische oder eine zeichenbasierte Konstante sein.



Definition 37: Konstante

Eine Konstante ist ein Ausdruck, der zur Compilezeit festgelegt wird und zur Laufzeit seinen Wert nicht verändern kann.

Was heißen jetzt schon wieder zur Laufzeit und Compilezeit? Der Quelltext – hier SQL-Quelltext – wird durch einen Compiler zu einem ausführbaren Code¹ umgewandelt. Zur Compilezeit bedeutet also, dass der Wert während dieser Umwandlung festgelegt wird. Laufzeit ist der Zeitraum, wo das Programm – der SQL-Befehl – verarbeitet wird. Bei einer Konstanten kann also während der Ausführung des Befehls keine Veränderung vorgenommen werden.

Beispiel: Wir wollen in einem SQL-Skript auf der Konsole angeben, bei welchem Bearbeitungsschritt wir gerade sind.

```
1 mysql> SELECT 5, 'Beginn der Auswertung';
2 +-----+-----+
3 | 5 | Beginn der Auswertung |
4 +-----+-----+
5 | 5 | Beginn der Auswertung |
6 +-----+-----+
```

Sowohl die 5 als auch die Zeichenkette Beginn der Auswertung sind Konstanten, da sie während der Laufzeit nicht verändert werden. Konstanten sind schwer zu erklären und super einfach zu verwenden.

¹ Wobei SQL-Quelltext anders als beispielsweise ein C-Programm nicht in eine EXE-Datei umgewandelt wird. Aber die SQL-Befehle werden in ein Format gebracht, welches von einer Laufzeitumgebung bzw. einem Interpreter ausgeführt werden kann.

10.1.2 Wie kann man Berechnungen vornehmen?

Man kann mit SQL auch rechnen. Eine Übersicht der mathematischen Operatoren finden Sie in [Abschnitt 26.2.1 auf Seite 408](#).

```
1 mysql> SELECT -5, 9 + 4, 9 - 4, 9 * 4, 9 / 4, 9 DIV 4, 9 % 4, 9 MOD 4;
2 +-----+-----+-----+-----+-----+-----+
3 | -5 | 9 + 4 | 9 - 4 | 9 * 4 | 9 / 4 | 9 DIV 4 | 9 % 4 | 9 MOD 4 |
4 +-----+-----+-----+-----+-----+-----+
5 | -5 | 13 | 5 | 36 | 2.2500 | 2 | 1 | 1 |
6 +-----+-----+-----+-----+-----+-----+
```

Das unäre Minus setzt das Vorzeichen der nachfolgenden Zahl. Dieser Operator hat die höchste Ausführungsriorität, er wird deshalb vor allen anderen ausgeführt. Natürlich dürfen anstelle von Konstanten als Operanden auch Spaltennamen oder Variablen verwendet werden.²



Definition 38: Operatorenpriorität

Unter der *Operatorenpriorität* oder *Operatorenrangfolge* versteht man die Festlegung darüber, welche Operation in einer Anweisung vor anderen ausgeführt wird. Haben zwei Operatoren die gleiche *Operatorenpriorität*, so werden sie von links nach rechts ausgeführt.

Die Reihenfolge der Auswertung kann durch Klammern festgelegt werden. Da die Operatorenpriorität nicht von jedem auswendig gewusst wird und der besseren Lesbarkeit wegen, empfehle ich immer zu klammern. Hier ein Beispiel:

```
1 mysql> SELECT 3 * 4 DIV 3, (3 * 4) DIV 3, 3 * (4 DIV 3);
2 +-----+-----+-----+
3 | 3 * 4 DIV 3 | (3 * 4) DIV 3 | 3 * (4 DIV 3) |
4 +-----+-----+-----+
5 | 4 | 4 | 3 |
6 +-----+-----+
```

Neben den einfachen mathematischen Operatoren gibt es eine Vielzahl von mathematischen Funktionen. In [Abschnitt 26.2.2 auf Seite 408](#) finden Sie die Funktionen und ihre Bedeutung³. Bei der Verwendung von mathematischen Operatoren oder Funktionen ist aber auf den Wertebereich zu achten.

```
1 mysql> SELECT 9 / 0, 9 * NULL, SQRT(-5);
2 +-----+-----+-----+
3 | 9 / 0 | 9 * NULL | SQRT(-5) |
4 +-----+-----+-----+
5 | NULL | NULL | NULL |
6 +-----+-----+
```

In MySQL oder MariaDB liefert ein Funktionsparameter, der nicht im Definitionsbereich liegt, das Ergebnis NULL. Bei anderen Systemen wie beispielsweise PostgreSQL und T-SQL wird die Verarbeitung mit einer Fehlermeldung abgebrochen.

² In PostgreSQL und T-SQL ergibt $9/4$ das gleiche Ergebnis wie $\text{DIV}(9, 4)$. Sind die Werte der Division Integer, wird die ganzzahlige Division genommen. Ein $9 . 0/4 . 0$ bringt das gewünschte Ergebnis.

³ Ergänzende Beispiele finden Sie unter [[MyS18b](#)].

10.1.3 Wie ermittelt man Zufallszahlen?

Die Funktion zur Ermittlung von Zufallszahlen ist `RAND()` mit einem Wertebereich von $[0.0, 1.0]$. Dieser Funktion kann ein sogenannter *seed* mitgegeben werden.

Die Berechnung von Zufallszahlen ist nicht wirklich dem Zufall überlassen. Vielmehr muss man sich das Ganze als einen riesigen Kreis vorstellen. Auf dem Kreis liegen die *Zufallszahlen*, und jede hat einen festen Vorgänger und einen festen Nachfolger. Ruft man `RAND()` oft genug auf, so würde sich der Kreis schließen und sich die Zahlenfolge wiederholen.

Die Qualität eines Zufallszahlengenerators ergibt sich aus der Größe dieses Kreises (ab wann sich die Zahlenfolge wiederholt), der Häufigkeitsverteilung der Zahlen auf dem Kreis (möglichst gleich verteilt) und der Vorhersagbarkeit der nächsten Zahl. Wer sich näher damit beschäftigen möchte, dem sei das epochale Werk von Donald E. Knuth [Knu81] empfohlen.

Der *seed* legt indirekt den Startpunkt auf diesem Kreis fest. Wird der gleiche *seed* verwendet, erhält man das gleiche Ergebnis.

```

1 mysql> SELECT RAND(1);
2 +-----+
3 | RAND(1)      |
4 +-----+
5 | 0.40540353712197724 |
6 +-----+
7
8 mysql> SELECT RAND(1);
9 +-----+
10 | RAND(1)      |
11 +-----+
12 | 0.40540353712197724 |
13 +-----+
14
15 mysql> SELECT RAND(1) FROM artikel;
16 +-----+
17 | RAND(1)      |
18 +-----+
19 | 0.40540353712197724 |
20 | 0.8716141803857071 |
21 | 0.1418603212962489 |
22 | 0.09445909605776807 |
23 | 0.04671454713373868 |
24 | 0.9501954782290342 |
25 | 0.6108337804776 |
26 | 0.2035824984345422 |
27 | 0.18541118147355615 |
28 +-----+

```

Der Verweis auf die Artikeltabelle ist nur notwendig, damit mehrere Zufallszahlen in einer Anweisung generiert werden.



Aufgabe 10.1: Würde man den Befehl noch mal ausführen, kämen dann die gleichen oder andere Zufallszahlen?

Zufallszahlen im Intervall $[a, b]$, für $a \leq b$, gehen so: `FLOOR(a + RAND() * (b - a))`.

10.1.4 Wie steckt man das Berechnungsergebnis in eine Variable?

In MySQL oder MariaDB werden Variablen, die außerhalb von gekapselten Anweisungen liegen⁴, mit einem Klammeraffen @ gekennzeichnet. Mit `SELECT ... INTO` kann man Ergebnisse in eine Variable schreiben. Die Variablen lassen sich in späteren Berechnungen und Bedingungen weiterverwenden:

```
1 mysql> SELECT POWER(2,8) INTO @x;
2
3 mysql> SELECT @x - 1;
4 +-----+
5 | @x - 1 |
6 +-----+
7 |      255 |
8 +-----+
```

Aber:

```
1 mysql> SELECT einzelpreis FROM artikel INTO @y;
2 ERROR 1172 (42000): Result consisted of more than one row
```



Aufgabe 10.2: Interpretieren Sie die Fehlermeldung. Überlegen Sie sich, was `SELECT` liefert und worin Sie das Ergebnis abspeichern wollen.

Man kann Variablen auch direkt mit einem Wert versehen:

```
1 mysql> SET @artnr=3010;
2
3 mysql> SELECT * FROM artikel WHERE artikel_id=@artnr;
4 +-----+-----+-----+-----+
5 | artikel_id | bezeichnung | einzelpreis | waehrung | deleted |
6 +-----+-----+-----+-----+
7 |      3010 | Feder       |    5.050000 | EUR        |      0 |
8 +-----+-----+-----+-----+
```

Variablen sind immer dann sehr hilfreich, wenn Informationen von einem SQL-Befehl an den anderen weitergereicht werden müssen. So könnte man in einem `SELECT` eine bestimmte Bestellung ermitteln und in einem zweiten die Positionen zu genau dieser Bestellung und in einem dritten Befehl vielleicht die Lieferadresse der Bestellung. Bei den letzten beiden `SELECT`-Anweisungen würde man in der `WHERE`-Klausel die `bestellung_id` auf Gleichheit mit dem Variablenwert testen.

⁴ Siehe Kapitel V auf Seite 313

■ 10.2 Zeilen- und Spaltenwahl



SQL:2016, MySQL/MariaDB, PostgreSQL, T-SQL

```
SELECT
  {*|spaltenliste|ausdruck}
  FROM tabellenname
  [WHERE bedingung]
;
```

Um den Inhalt einer Tabelle – z.B. `artikel` – komplett auszugeben, haben wir in den vorangegangenen Abschnitten eine einfache Version des SELECT verwendet:

```
1  SELECT * FROM artikel;
```

Der `*` steht hier als Platzhalter für alle Spalten einer Tabelle. Folgender Befehl hätte das gleiche Ergebnis:

```
1  SELECT
2    artikel_id, bezeichnung, einzelpreis, waehrung, deleted
3    FROM artikel;
```

Diese *spaltenliste* kann sehr flexibel zusammengesetzt werden:

- Die Reihenfolge der Spalten kann beliebig sein.
OK: `deleted, bezeichnung, einzelpreis, artikel_id, waehrung`
- Die Spalten können mehrfach vorkommen.
OK: `deleted, artikel_id, deleted`
- Es muss aber mindestens eine Spalte angegeben werden.
NOT OK: `SELECT FROM artikel`
- Es können Konstanten und Ausdrücke vorkommen.
OK: `RAND(), deleted, bezeichnung, artikel_id + 5000`

Eine weitere schöne Sache ist, dass man den Spaltennamen oder Ausdrücken neue *Namen* – einen Alias – zuweisen kann, wobei das Schlüsselwort AS optional ist:

```
1  mysql> SELECT
2    -> artikel_id + 10000 AS Unsinn, deleted AS Löschkennzeichen
3    -> FROM artikel;
4  +-----+-----+
5  | Unsinn | Löschkennzeichen |
6  +-----+-----+
7  | 13010 | deleted          |
8  [...]
9  | 17863 | deleted          |
10 +-----+-----+
```

In [Zeile 5](#) werden nicht mehr die ursprünglichen Spaltennamen oder Ausdrücke als Überschriften verwendet. Der neue Name sollte selbsterklärend oder eine sinnvolle Abkürzung sein. Diese Namensvergabe wird uns noch häufiger begegnen.

Zur Auswahl der Zeilen wird wie beim UPDATE und DELETE die WHERE-Klausel verwendet. Dabei entsteht leicht die Annahme, dass die Spalten, die in der WHERE-Klausel stehen, auch in der *spaltenliste* vorkommen müssen⁵. Hier ein Gegenbeispiel:

```

1 mysql> SELECT
2      -> artikel_id, bezeichnung
3      -> FROM artikel
4      -> WHERE einzelpreis BETWEEN 10 AND 100;
5 +-----+-----+
6 | artikel_id | bezeichnung |
7 +-----+-----+
8 |      3005 | Tinte (gold) |
9 |      9010 | Schaufel   |
10 |     9015 | Spaten    |
11 +-----+-----+

```

Weitere Informationen zur Zeilenauswahl finden Sie in der Besprechung der WHERE-Klausel in [Abschnitt 9.1 auf Seite 141](#).

■ 10.3 Sortierung



SQL:2016, MySQL/MariaDB, PostgreSQL, T-SQL

```

SELECT
  {*|spaltenliste|ausdruck}
  FROM tabellenname
  [WHERE bedingung]
  [ORDER BY spaltenname [ASC|DESC] [.spaltenname [ASC|DESC]]*]
;
```

Die Ausgabe von Daten in einer fachlich oder ergonomisch sinnvollen Reihenfolge ist eine oft gestellte Forderung an eine Auswertung. Beispiele: Kunden nach Umsatz absteigend, Artikelpreise nach Einzelpreis aufsteigend, Namenslisten nach Nach- und Vorname etc. Die entsprechende Option heißt ORDER BY.

Geben wir die Artikel nach Einzelpreis aufsteigend aus:

```

1 mysql> SELECT
2      -> artikel_id, bezeichnung, einzelpreis
3      -> FROM artikel
4      -> ORDER BY einzelpreis ASC;
5 +-----+-----+-----+
6 | artikel_id | bezeichnung | einzelpreis |
7 +-----+-----+-----+
8 |      7856 | Silberzwiebel |    0.520000 |
9 |      3001 | Papier (100) |    2.320000 |
10 |     7863 | Tulpenzwiebel |    3.420000 |
11 |      3007 | Tinte (blau) |    4.170000 |

```

⁵ Vermutlich entsteht der Fehler dadurch, dass man bei der imperativen Programmierung immer die Datenobjekte angeben muss, mit denen man gerade arbeiten will.

12	3010	Feder		5.050000
13	3006	Tinte (rot)		6.260000
14	9010	Schaufel		15.100000
15	9015	Spaten		20.100000
16	3005	Tinte (gold)		56.260000
17 +-----+-----+-----+				

Das Schlüsselwort ASC sortiert die Daten aufsteigend, DESC absteigend. Wird keine der beiden Angaben gemacht, werden die Daten aufsteigend sortiert.



Aufgabe 10.3: Führen Sie den Befehl einmal ohne ASC und anschließend mit einem DESC aus. Vergleichen Sie die Ergebnisse mit dem obigen.

Die Sortierung kann auch nach mehreren Spalten erfolgen:

```

1 mysql> SELECT
2      -> nachname, vorname
3      -> FROM kunde
4      -> ORDER BY nachname, vorname
5      -> ;
6 +-----+-----+
7 | nachname | vorname |
8 +-----+-----+
9 | Beutlin   | Bilbo   |
10 | Beutlin   | Frodo   |
11 | Earendilionn | Elrond |
12 | Eichenschild | Thorin |
13 | Gamdschie   | Samweis |
14 | Telcontar   | Elessar |
15 +-----+-----+

```

Bitte beachten Sie, dass die Liste in erster Priorität nach dem Nachnamen sortiert ist. Nur dort, wo im Nachnamen der gleiche Wert steht, wird nach Vorname sortiert. Deshalb steht der **Bilbo** vor dem **Frodo**. Die Sortierreihenfolge ist aber für beide Spalten unabhängig einstellbar:

```

1 mysql> SELECT
2      -> nachname, vorname
3      -> FROM kunde
4      -> ORDER BY nachname DESC, vorname ASC
5      -> ;
6 +-----+-----+
7 | nachname | vorname |
8 +-----+-----+
9 | Telcontar | Elessar |
10 | Gamdschie | Samweis |
11 | Eichenschild | Thorin |
12 | Earendilionn | Elrond |
13 | Beutlin   | Bilbo   |
14 | Beutlin   | Frodo   |
15 +-----+-----+

```

Die Liste wird jetzt nach dem Nachnamen absteigend sortiert, aber die Vornamen wiederum aufsteigend⁶.

⁶ Ich habe hier das Schlüsselwort ASC nur des besseren Verständnisses wegen angegeben.



Hinweis: Ein gerne gemachter Fehler bei folgender Aufgabenstellung: *Geben Sie die Namensliste für Nach- und Vorname absteigend sortiert aus.* Häufig wird folgende Lösung angeboten:

```
SELECT
    nachname, vorname
  FROM kunde
  ORDER BY nachname, vorname DESC;
```

Diese Lösung ist aber falsch, da sich das DESC hier nur auf die letzte Spalte auswirkt und der Nachname aufsteigend sortiert wird. Denken Sie daran, dass ASC immer dann verwendet wird, wenn man für diese Spalte keine Angaben macht.

Würde man für die Spaltennamen Aliase verwenden, kann man diese in die ORDER BY-Klausel einsetzen.

```
1 SELECT nachname n, vorname v FROM kunde ORDER BY n DESC, v DESC;
```

Die ORDER BY-Klausel kann anstelle des Spaltennamens oder eines Alias auch eine Zahl enthalten. Die Zahl ergibt sich aus einer 1 basierenden Indizierung der *spaltenliste*:

```
1 --      1      2
2 SELECT nachname, vorname FROM kunde ORDER BY 1 DESC, 2 ASC;
```

Und so haben wir auch das Kommentarzeichen - - für einzeilige Kommentare eingeführt ([Zeile 1](#)). Bitte beachten Sie, dass ein Leerzeichen nach den zwei Strichen stehen muss.

10.3.1 Was muss bei der Sortierung von Texten beachtet werden?

Zunächst stellt sich die Frage, wann eine Zeichenkette größer als eine andere ist. Denn jede Sortierung basiert auf einer Festlegung darüber, wie eine Reihenfolge ermittelt werden kann. In der Informatik verwendet man dazu den Begriff der *Ordnung*:



Definition 39: Ordnung

Eine Menge M ist *wohlgeordnet*, wenn sie folgende Eigenschaften erfüllt:

- Trichotomie
Für die Elemente $a, b \in M$ gilt immer genau eine der folgenden Aussagen:
 $a = b$, $a < b$ oder $a > b$
- Transitivität
Für die Elemente $a, b, c \in M$ gilt, wenn $a \leq b$ und $b \leq c$, dann muss $a \leq c$ sein.
- Minimumexistenz
Es gibt ein $m \in M$, für welches gilt, dass für alle $x \in M$ $m \leq x$ ist.

Wann immer eine Menge wohlgeordnet ist, kann man die Menge sortieren. Machen wir uns die [Definition 39](#) anhand der natürlichen Zahlen klar⁷.

⁷ Ich weiß selber, dass das hier kein mathematischer Beweis ist :-).

- Trichotomie: Für zwei Zahlen muss gelten, dass sie entweder gleich oder unterschiedlich sind. Wenn sie unterschiedlich sind, muss eine der beiden die kleinere sein.
- Transitivität: Wenn $5 < 10$ und $10 < 12$ ist, dann ist auch $5 < 12$.
- Minimumexistenz: Wir behaupten, dass 0 die gesuchte Zahl ist.

Wie ist das aber jetzt mit den Texten? Oft bekomme ich die Antwort, dass man nach der Länge des Textes sortieren kann. Und ja, das wäre möglich:

- Trichotomie: Die Längen zweier Texte sind entweder gleich oder unterschiedlich. Wenn sie unterschiedlich sind, muss ein Text der kürzere sein.
- Transitivität: Wenn ein Text kürzer als ein zweiter ist und der zweite kürzer als ein dritter Text ist, dann muss der erste auch kürzer als der dritte sein.
- Minimumexistenz: Wir behaupten, dass dies der leere Text mit einer Länge 0 ist.

Aber wer braucht das? Textsortierungen werden in der Regel als alphabetische Sortierungen betrachtet. Und jetzt fängt es an, kompliziert zu werden. Was ist ein Alphabet und wer legt die Reihenfolge fest? Es hat mich beispielsweise immer erstaunt, dass wir in Deutschland als Kinder ein Alphabet lernen, in dem die Umlaute und das ß nicht vorkommen. Schließlich sind diese Buchstaben ja keine Unfälle, sondern genauso wichtig und elementar wie ein X.

Eine textliche Sortierung verlangt somit, dass es eine Festschreibung über die Reihenfolge der gültigen Zeichen gibt. Historisch hat man die ASCII-Zeichenfolge mit ihrem Nummerncode verwendet. Was zu folgendem unglücklichen Ergebnis führt:

```

1 mysql> SET NAMES cp850;
2
3 mysql> CREATE TEMPORARY TABLE wurstbrot (
4     ->     name VARCHAR(255) CHARACTER SET 'cp850'
5     -> );
6
7 mysql> INSERT INTO wurstbrot
8     ->     VALUES
9     ->     ('winfried')
10    ->     ,('achim')
11    ->     ,('olga')
12    ->     ,('zechine')
13    ->     ,('ägidius')
14    -> ;
15
16 mysql> SELECT * FROM wurstbrot ORDER BY name;
17 +-----+
18 | name   |
19 +-----+
20 | achim  |
21 | olga   |
22 | winfried |
23 | zechine |
24 | ägidius |
25 +-----+

```

In [Zeile 1](#) wird durch `SET NAMES` sichergestellt, dass die Konsoleneingabe in der hier gewünschten Kodierung zum Server geschickt wird. [Zeile 4](#) erstellt eine Spalte mit dem `CHARACTER SET cp850`. Übrigens sehen Sie hier ein Beispiel dafür, dass Sie innerhalb einer Tabelle für Textspalten unterschiedliche Zeichensätze verwenden können (zum Thema Zeichensätze siehe [Seite 68](#)).

In Zeile 13 wird ein Name mit einem Umlaut eingefügt, der nach einer Sortierung in Zeile 24 ganz an das Ende gestellt wird. Warum? Der ASCII hat in den ersten 128 Zeichen nur die Buchstaben für das amerikanische Alphabet. Länderspezifische Sonderzeichen haben demnach einen Code > 127. Wird jetzt dieser Code als Basis der Sortierung verwendet, landen die Umlaute immer ganz am Ende.

Es war also nötig, eine von der Zeichenkodierung unabhängige Spezifikation für die Sortierung zu entwickeln. Das Ergebnis dieser Bemühungen sind die Collations (siehe Seite 70). Im Prinzip wird dabei jedem Zeichen oder sogar einer Zeichenkombination eine eindeutige Nummer zugewiesen. Anhand dieser Nummer kann dann die Sortierung erfolgen. Da es für UTF-8 nur eine Sortierung gibt, muss diese nicht weiter angegeben werden:

```

1 mysql> SET NAMES utf8;
2
3 mysql> CREATE TEMPORARY TABLE wurstbrot (
4     ->     name VARCHAR(255) CHARACTER SET 'utf8'
5     -> );
6
7 mysql> INSERT INTO wurstbrot
8     -> VALUES
9     ->     ('winfried')
10    ->     ,('achim')
11    ->     ,('olga')
12    ->     ,('zechine')
13    ->     ,('ägidius')
14    -> ;
15
16 mysql> SELECT * FROM wurstbrot ORDER BY name;
17 +-----+
18 | name   |
19 +-----+
20 | achim  |
21 | ägidius |
22 | olga   |
23 | winfried |
24 | zechine |
25 +-----+

```

Wenn Sie eine Sortierung auf zeichenbasierten Spalten machen, sollten Sie unbedingt auf den eingestellten Zeichensatz und die entsprechende Sortierung achten. So ist es beispielsweise unter Windows bis *Windows 10* nicht möglich, Unicode-Zeichen über den MySQL Client einzugeben, da der Windows-COMMAND kein Unicode verarbeiten kann. Die Verarbeitung kann dann zu unerwünschten Resultaten führen.

Verwenden Sie dann lieber den Query-Browser der Workbench oder noch besser: Schreiben Sie in einer Sprache Ihrer Wahl (C#, PHP, Perl ...) einen eigenen Query-Browser. Sie glauben gar nicht, wie lehrreich das ist!

10.3.2 Wird zwischen Groß- und Kleinschreibung unterschieden?

Ändern wir das obige Beispiel ein wenig ab, um diese Frage zu beantworten. Zuerst die Testdaten:

```
1 mysql> CREATE TEMPORARY TABLE wurstbrot (name VARCHAR(255));
```

```

2
3 mysql> INSERT INTO wurstbrot
4      -> VALUES
5      ->     ('winfried')
6      ->     ,('achim')
7      ->     ,('olga')
8      ->     ,('Olga')
9      ->     ,('zechine');

```

Jetzt einmal aufsteigend und einmal absteigend sortiert.

```

1 mysql> SELECT * FROM wurstbrot ORDER BY name;
2 +-----+
3 | name   |
4 +-----+
5 | achim  |
6 | Olga   |
7 | olga   |
8 | winfried |
9 | zechine |
10+-----+
11
12 mysql> SELECT * FROM wurstbrot ORDER BY name DESC;
13 +-----+
14 | name   |
15 +-----+
16 | zechine |
17 | winfried |
18 | Olga   |
19 | olga   |
20 | achim  |
21+-----+

```

In beiden Sortierreihenfolgen erscheint die große Olga vor der kleinen. Würde zwischen Groß- und Kleinschreibung unterschieden, müsste die große Olga entweder als erste Zeile oder als letzte Zeile oder mal vor und mal nach der kleinen Olga erscheinen, aber nicht in jedem Fall gleich.

Möchte man zwischen Groß- und Kleinschreibung einen Unterschied machen, muss die zeichenbasierte Spalte mit BINARY als Zusatz deklariert sein (siehe [Zeile 1](#)).

```

1 mysql> CREATE TEMPORARY TABLE wurstbrot (name VARCHAR(255) BINARY);
2
3 mysql> INSERT INTO wurstbrot
4      -> VALUES
5      ->     ('winfried')
6      ->     ,('achim')
7      ->     ,('olga')
8      ->     ,('Olga')
9      ->     ,('zechine');
10
11 mysql> SELECT * FROM wurstbrot ORDER BY name;
12 +-----+
13 | name   |
14 +-----+
15 | Olga   |
16 | achim  |
17 | olga   |
18 | winfried |
19 | zechine |

```

```

20 +-----+
21
22 mysql> SELECT * FROM wurstbrot ORDER BY name DESC;
23 +-----+
24 | name   |
25 +-----+
26 | zechine |
27 | winfried |
28 | olga    |
29 | achim   |
30 | Olga    |
31 +-----+

```

Wie erwartet, wird die große Olga jetzt anders einsortiert.



Hinweis: In SQL:2016 gibt es keine Möglichkeit, zwischen Groß- und Kleinschreibung zu unterscheiden.

10.3.3 Wie werden Datums- und Uhrzeitwerte sortiert?

Datums- und Uhrzeitwerte werden nach ihrem Alter sortiert. Das älteste Datum, die älteste Uhrzeit kommt als Erstes, das jüngste Datum bzw. die jüngste Uhrzeit als Letztes.

```

1 mysql> SELECT
2      -> bestellung_id, datum
3      -> FROM bestellung
4      -> ORDER BY datum;
5 +-----+-----+
6 | bestellung_id | datum          |
7 +-----+-----+
8 |           2 | 2012-03-23 16:11:00 |
9 |           1 | 2012-03-24 17:41:00 |
10 +-----+-----+

```



Hinweis: Der aktuelle Zeitpunkt bleibt dabei völlig unberücksichtigt. So kann ein in der Zukunft liegendes Datum das älteste sein, wenn alle anderen noch weiter in der Zukunft liegen.

Die Uhrzeit wird mit 00:00:00 Uhr beginnend sortiert. Die späteste Uhrzeit kommt als Letztes:

```

1 mysql> SELECT
2      -> bestellung_id, TIME(datum) Uhrzeit
3      -> FROM bestellung
4      -> ORDER BY uhrzeit;
5 +-----+-----+
6 | bestellung_id | Uhrzeit   |
7 +-----+-----+
8 |           2 | 16:11:00 |
9 |           1 | 17:41:00 |
10 +-----+-----+

```



Aufgabe 10.4: Ermitteln Sie die Bestellung, die als Letztes – bezogen auf Datum und Uhrzeit – aufgegeben wurde.

10.3.4 Wie kann man das Sortieren beschleunigen?

Grundsätzlich gibt es zwei Methoden, wie die Sortierung vorgenommen wird:

- **Index:** Die zu sortierenden Felder sind in der gewünschten Reihenfolge in einem Index. SQL braucht dann nur die Vorsortierung des Index auszugeben, was sehr schnell geht.
- **Filesort**⁸: Die zu sortierenden Felder werden mithilfe integrierter Sortierroutinen wie gefordert geordnet.

Damit wir überhaupt einen Unterschied feststellen können, bauen wir eine Tabelle auf, die tausend Datensätze hat. Laden Sie das Experiment in [Abschnitt 27.7 auf Seite 454](#) und führen Sie Folgendes aus (kann ein paar Minuten dauern):

```
1 mysql> CALL datenerzeugen(1000);
```

Um herauszufinden, welche der beiden Methoden verwendet wird, kann man in MySQL den Befehl EXPLAIN verwenden. Dieser sehr hilfreiche Befehl beschreibt den Ausführungsplan eines Befehls.

```
1 mysql> EXPLAIN
2   -> SELECT * FROM name_mit ORDER BY nachname, vorname\G
3 **** 1. row ****
4   id: 1
5   select_type: SIMPLE
6     table: name_mit
7     type: index
8   possible_keys: NULL
9     key: idx_namemit_nnvn
10    key_len: 1536
11    ref: NULL
12    rows: 1141
13   Extra: Using index
14
15 mysql> EXPLAIN
16   -> SELECT * FROM name_ohne ORDER BY nachname, vorname\G
17 **** 1. row ****
18   id: 1
19   select_type: SIMPLE
20     table: name_ohne
21     type: ALL
22   possible_keys: NULL
23     key: NULL
24     key_len: NULL
25     ref: NULL
26     rows: 853
27   Extra: Using filesort
```

⁸ Die Daten werden nicht zwingend in Dateien sortiert. Bis zu einer bestimmten Datenmenge kann das auch im RAM geschehen.

In Zeile 13 wird angezeigt, dass hier ein passender Index für das ORDER BY gefunden wurde und dieser verwendet wird. Etwas anders sagt Zeile 27: Ein Filesort wird benötigt. Im gleichen Listing gibt es ein Experiment zur Performancemessung, welches dem Bild 10.1 zugrunde liegt.

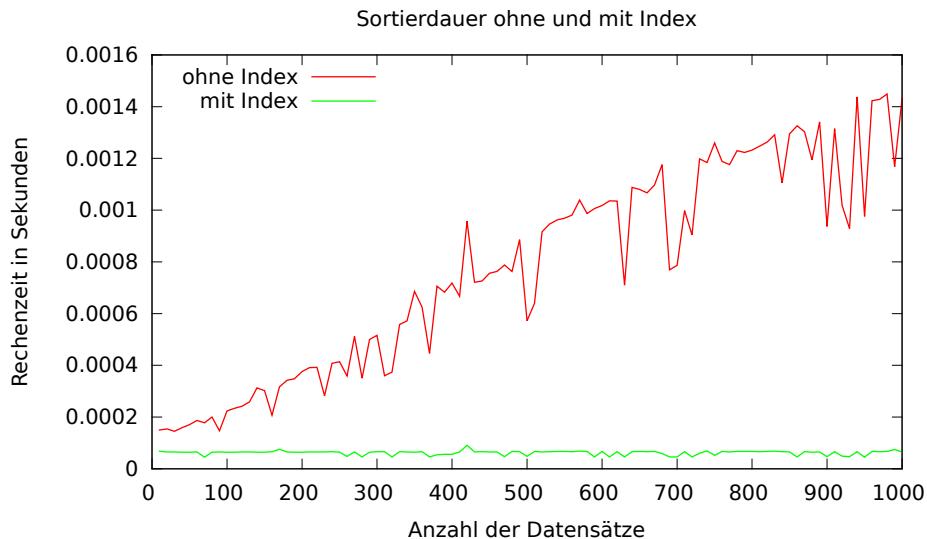


Bild 10.1 Unterschiedlicher Zeitverbrauch von Sortieroperationen ohne und mit Index

Es fällt auf, dass der Zeitverbrauch ohne Index deutlich ansteigt und der mit Index nahezu lineare Wachstum der Sortierdauer ohne Index stimmt nicht ganz. Ich vermute, dass es sich um einen Performanceverbrauch der Größenordnung $O(n \log n)$ handelt. Dies ist das durchschnittliche Zeitverhalten eines guten Sortieralgoritmus wie Quicksort. Das fast konstante Zeitverhalten mit Index ergibt sich aus der Tatsache, dass die Daten schon sortiert vorliegen und nur ein *look up* erfolgt.

Es lohnt sich, beim Erzeugen der Indizes (siehe [Abschnitt 6.1 auf Seite 93](#)) sehr genau den Bedarf zu planen und beim Formulieren der ORDER BY-Klausel gut aufzupassen, denn leider verliert man sehr schnell die Möglichkeit, den Index zu verwenden.

```

1 mysql> EXPLAIN
2      -> SELECT * FROM name_mit ORDER BY nachname\G
3 **** 1. row ****
4      id: 1
5      select_type: SIMPLE
6      table: name_mit
7      [...]
8      Extra: Using index
9
10 EXPLAIN EXTENDED
11      -> SELECT * FROM name_mit ORDER BY nachname DESC, vorname DESC\G
12 **** 1. row ****
13      id: 1

```

⁹ Der genaue Versuchsaufbau kann in [Abschnitt 27.7 auf Seite 454](#) nachgelesen werden.

```

14      select_type: SIMPLE
15          table: name_mit
16      [...]
17          Extra: Using index

```

Das erste Beispiel kann den Index verwenden (Zeile 8), obwohl nur ein Teil des Index in der ORDER BY-Klausel vorkommt. Auch das zweite Beispiel benutzt den Index, obwohl beide Spalten in unterschiedlicher Reihenfolge gebraucht werden.

```

1  mysql> EXPLAIN
2      -> SELECT * FROM name_mit ORDER BY id, nachname \G
3  **** 1. row ****
4      id: 1
5      select_type: SIMPLE
6          table: name_mit
7      [...]
8          Extra: Using index; Using filesort

```

Zuerst wird ein Index verwendet (Zeile 8), weil die `id` als Primärschlüssel automatisch einen Index hat. Anschließend wird ein Filesort benötigt, obwohl auf `nachname` ein Index liegt. Man kann also nicht zwei Indizes kombinieren¹⁰.

```

1  mysql> EXPLAIN EXTENDED
2      -> SELECT * FROM name_mit ORDER BY vorname, nachname\G
3  **** 1. row ****
4      id: 1
5      select_type: SIMPLE
6          table: name_mit
7          partitions: NULL
8          type: index
9      [...]
10         Extra: Using index; Using filesort

```

Beim letzten Beispiel wird der Vorname mit dem Filesort sortiert. Innerhalb einer Gruppe gleicher Vornamen kann der Index auf Nachname benutzt werden.

Was auch interessant ist, dass bei kleinen Datenmengen trotz Index der Filesort verwendet wird. Vermutlich kann dieser bei kleinen Datenmengen sehr schnell im RAM durchgeführt werden. Als Beispiel sei hier die Tabelle `kunde` mit nur sieben Datensätzen genannt:

```

1  mysql> EXPLAIN
2      -> SELECT * FROM kunde ORDER BY nachname, vorname\G
3  **** 1. row ****
4      id: 1
5      select_type: SIMPLE
6          table: kunde
7          partitions: NULL
8          type: ALL
9      [...]
10         Extra: Using filesort

```

¹⁰ Natürlich ist Ihnen aufgefallen, dass es völlig sinnlos ist, auch noch nach dem Nachnamen zu sortieren. Wie, es ist Ihnen nicht aufgefallen? Warum ist es denn sinnlos?

■ 10.4 Mehrfachausgaben unterbinden



SQL:2016, MySQL/MariaDB, PostgreSQL, T-SQL

```
SELECT [DISTINCT]
{*>|spaltenliste|ausdruck}
FROM tabellenname
[WHERE bedingung]
[ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
;
```

Wenn wir wissen wollen, in welchen Städten wir Adressen haben, so können wir jetzt schon eine entsprechende Auswertung programmieren:

```
1 mysql> SELECT
2      ->     ort FROM adresse ORDER BY ort;
3 +-----+
4 | ort   |
5 +-----+
6 | Bochum |
7 | Bochum |
8 | Bruchtal |
9 | Hobbingen |
10 | Hobbingen |
11 | Lugburz |
12 | Minas Tirith |
13 +-----+
```

Das ORDER BY ist dringend nötig, da wir sonst nicht erkennen würden, welche Orte mehrfach vorkommen. Bei unserem Datenbestand ist das noch sehr übersichtlich. Stellen wir uns einen Adressbestand von mehreren Hundert Adressen vor, so wäre der obige SELECT sicherlich nicht verwendbar. Und warum nicht? Weil er mich mit vielen überflüssigen Informationen zuschüttet.

Hier hilft eine Option des SELECT, der DISTINCT:

```
1 mysql> SELECT DISTINCT
2      ->     ort FROM adresse ORDER BY ort;
3 +-----+
4 | ort   |
5 +-----+
6 | Bochum |
7 | Bruchtal |
8 | Hobbingen |
9 | Lugburz |
10 | Minas Tirith |
11 +-----+
```

Nun werden die Orte nur einmal angezeigt.

Der DISTINCT vergleicht *nicht* die komplette Zeile, sondern nur die Inhalte, der in *spaltenliste* angegebenen Spalten oder Ausdrücke, wie dieses Beispiel zeigt:

```
1 mysql> SELECT DISTINCT
2      ->     nachname
3      ->     FROM kunde
4      ->     ORDER BY nachname;
```

```

5 +-----+
6 | nachname      |
7 +-----+
8 | Beutlin       |
9 | Earendilionn |
10 | Eichenschild |
11 | Gamdschie    |
12 | Telcontar    |
13 +-----+

```



Hinweis: Ein DISTINCT, der Schlüsselwerte unverändert auswählt, ist zwar schnell, aber sinnlos.



Aufgabe 10.5: Begründen Sie diesen Hinweis mithilfe der [Definition 8 auf Seite 17](#).

10.4.1 Fallstudie: Datenimport von Bankdaten

Unsere Tabelle bank ist bisher leer. Die Datei blz_20120305.csv enthält die Liste aller deutscher Banken mit Namen und Bankleitzahl zum Zeitpunkt 03.05.2012¹¹. Ziel ist es, diese Daten für den Aufbau unserer bank-Tabelle zu verwenden.

Ein Blick in die Datei zeigt uns, welche Informationen uns hier angeboten werden. Für uns sind folgende Informationen interessant: Bankleitzahl und Bezeichnung. Die anderen Informationen sind sicherlich nicht unwichtig, werden hier aber nicht weiter berücksichtigt. Wir wollen diese Daten importieren, auswerten, vorbereiten und in die Tabelle bank kopieren.

Schritt 1: Importieren

Wir legen dazu eine temporäre Tabelle an, die die gewünschten Informationen aufnehmen kann:

```

1 CREATE TEMPORARY TABLE bank_import (
2   bankleitzahl  CHAR(8),
3   bezeichnung   VARCHAR(255)
4 );

```

Nun werden die Daten mithilfe von LOAD DATA INFILE importiert (siehe [Abschnitt 7.1 auf Seite 105](#)):

```

1 LOAD DATA LOCAL INFILE 'blz_20120305.csv'
2 INTO TABLE bank_import
3 FIELDS
4   TERMINATED BY ';'
5 LINES
6   TERMINATED BY '\n'
7 IGNORE 1 LINES
8 (bankleitzahl, @dummy, bezeichnung);

```

¹¹ Die Originalquelle [Bun16] ist aktuell – 09.2019 – so nicht mehr erreichbar.

Die @dummy-Variable wird dazu verwendet, die nicht benötigten Informationen beim Einlesen zu ignorieren. Die Warnungen weisen darauf hin, dass nach der Spalte `bezeichnung` Daten einfach abgeschnitten werden. Das ist aber in Ordnung, denn schließlich brauchen wir diese nicht.

Schritt 2: Auswerten

Aus dem Import wissen wir, dass in der Tabelle 19574 Zeilen sind. Wie viele davon sind unterschiedliche Banken?

```
1 SELECT DISTINCT * FROM bank_import;
```

Wir erhalten 8927 unterschiedliche Zeilen. Unterscheiden die sich auch in der Bankleitzahl oder kommt die mehrfach vor?

```
1 SELECT DISTINCT bankleitzahl FROM bank_import;
```

Nun sind es nur noch 4120. Haben die alle die gleiche Bezeichnung?

```
1 SELECT DISTINCT bankleitzahl, bezeichnung FROM bank_import;
```

Leider nein, es sind jetzt wieder 6106 verschiedene Einträge. Somit kommen für eine Bankleitzahl ggf. mehrere Bezeichnungen in Frage. So kann die Bankleitzahl nicht als Schlüssel verwendet werden. Muss denn die Bankleitzahl der Schlüssel sein? Können wir nicht die Tabelle `bank` und `bankverbindung` so umbauen, dass mit einer laufenden Nummer als Primärschlüssel gearbeitet wird?

Klar, geht das. Also ran an den nächsten Schritt:

Schritt 3: Vorbereiten

Unter der Annahme, dass beide Tabellen noch leer sind, müssen wir die Tabelle `bankverbindung` so präparieren, dass die Constraints keine Schwierigkeiten mehr machen. Alle Fremdschlüssele und Indizes werden deshalb gelöscht.

```
1 ALTER TABLE bankverbindung
2 DROP FOREIGN KEY bankverbindung_ibfk_1,
3 DROP FOREIGN KEY bankverbindung_ibfk_2,
4 DROP INDEX idx_bankverbindung_bankid_kontonummer,
5 DROP PRIMARY KEY
6 ;
```

Nun wird die Tabelle `bank` umgebaut. Da das Löschen eines Primärschlüssels und das Anlegen eines neuen Primärschlüssels nicht mit *einem* `ALTER TABLE` möglich ist, müssen hier *zwei* programmiert werden. Zusätzlich wird jetzt eine neue Spalte mit dem Namen `blz` hinzugefügt. Der Optik wegen wird diese direkt nach dem Primärschlüssel eingesortiert.

```
1 ALTER TABLE bank
2 DROP PRIMARY KEY
3 ;
4
5 ALTER TABLE bank
6 MODIFY bank_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
7 ADD blz CHAR(12) NOT NULL DEFAULT '' AFTER bank_id
8 ;
```

Und zum Schluss werden die Indizes und Schlüssele in der Tabelle `bankverbindung` wieder hergestellt:

```

1 ALTER TABLE bankverbindung
2   MODIFY bank_id INT UNSIGNED,
3   ADD PRIMARY KEY(kunde_id, bankverbindung_nr),
4   ADD INDEX idx_bankverbindung_bankid_kontonummer (bank_id, kontonummer),
5   ADD FOREIGN KEY (kunde_id)
6     REFERENCES kunde(kunde_id)
7     ON UPDATE RESTRICT
8     ON DELETE RESTRICT,
9   ADD FOREIGN KEY (bank_id)
10    REFERENCES bank(bank_id)
11    ON UPDATE RESTRICT
12    ON DELETE RESTRICT
13 ;

```

Schritt 4: Kopieren

Und auf zum letzten Schritt:

```

1 INSERT INTO bank (blz, bankname)
2   SELECT DISTINCT bankleitzahl, bezeichnung FROM bank_import
3 ;
4
5 CREATE INDEX idx_bank_blzbankname
6   ON bank (blz, bankname)
7 ;

```

Zum einen haben wir den DISTINCT für Untersuchungen verwendet. Auch wenn wir hier nur an der Anzahl interessiert waren¹², lieferte er uns doch eine entscheidende Information. Ohne den Test auf Dubletten hätte wir 19574 anstelle von 6106 Zeilen in die Tabelle bank importiert. Na, wenn sich das nicht gelohnt hat!

Zum anderen wurde der DISTINCT aktiv dafür verwendet, beim Import nur die einmaligen Datensätze zu verwenden.

10.4.2 Was ist beim DISTINCT bzgl. der Performance zu beachten?

Im Prinzip das Gleiche wie beim ORDER BY (siehe [Abschnitt 10.3 auf Seite 163](#)). Das Erkennen von Dubletten kann bei sortierten Listen erheblich schneller ablaufen als bei nicht sortierten. Liegt ein Index auf den DISTINCT-Spalten, kann der Zugriff direkt über den Index abgearbeitet werden.

Wie oben bereits angesprochen, ist es aber sinnlos, auf Indizes mit der Eigenschaft UNIQUE einen DISTINCT abzusetzen.

¹² Was wir mit COUNT(*) etwas bequemer hätten können (siehe [Abschnitt 26.2.3 auf Seite 411](#)).

■ 10.5 Ergebnismenge ausschneiden

**MySQL/MariaDB**

```
SELECT [DISTINCT]
{*<|spaltenliste|ausdruck}
FROM tabellenname
[WHERE bedingung]
[ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
[LIMIT [offset,] anzahl]
;
```

Unter der Überschrift *Befehle, die man nicht im MySQL Client ausführen sollte:*

```
1 SELECT * FROM bank;
```

Wie sinnlos, über 6000 Datensätze an sich vorbeiziehen zu sehen. In MySQL gibt es eine einfache Möglichkeit, die Ausgabemenge zu beschränken: LIMIT.

10.5.1 Wie kann man sich die ersten *n* Datensätze ausschneiden?

Wir wollen uns die ersten drei Zeilen der Tabelle bank anschauen:

```
1 mysql> SELECT
2      -> blz, bankname
3      -> FROM bank
4      -> LIMIT 3;
5
6 +-----+
7 | blz      | bankname          |
8 +-----+
9 | 10000000 | Bundesbank        |
10 | 10010010 | Postbank           |
11 | 10010111 | SEB                |
12 +-----+
```

Durch die Angabe *anzahl* hinter dem Schlüsselwort LIMIT werden nur die ersten drei Datensätze ausgegeben. Dabei werden ggf. vorhandene ORDER BY-Klauseln berücksichtigt.

Gerade in Zusammenarbeit mit dem ORDER BY lassen sich häufige vorkommende Auswertungen erstellen. Wann immer man das erste oder letzte Element einer Liste braucht, wird mit LIMIT 1 gearbeitet, z.B. der Kunde mit dem höchsten bzw. geringsten Umsatz oder der teuerste bzw. billigste Artikel oder die Bank mit der höchsten Bankleitzahl:

```
1 mysql> SELECT
2      -> blz, bankname
3      -> FROM bank
4      -> ORDER BY blz DESC
5      -> LIMIT 1;
6
7 +-----+
8 | blz      | bankname          |
9 +-----+
10 | 87096214 | Volksbank Chemnitz (Gf P2) |
11 +-----+
```

Beschränkt man die Ausgabemenge nicht nur auf eine Zeile, sondern auch auf eine Spalte, lässt sich das Ergebnis in eine Variable schreiben, um es beispielsweise weiter verarbeiten zu können:

```

1 mysql> SELECT
2      -> blz
3      -> FROM bank
4      -> ORDER BY blz DESC
5      -> LIMIT 1
6      -> INTO @blzmin;
7 mysql> SELECT
8      -> bankname
9      -> FROM bank
10     -> WHERE blz = @blzmin;
11
12 +-----+
13 | bankname
14 +-----+
15 | Volksbank Chemnitz
16 | Volksbank Chemnitz (Gf P2)
17 +-----+

```

10.5.2 Wie kann man Teilmengen mittendrin ausschneiden?

Besonders bei Online-Anwendungen werden größere Datenmengen seitenweise z.B. in 20er Paketen ausgegeben. Dazu muss aus der Ergebnismenge 20 Zeilen ab einem bestimmten *offset* ausgeschnitten werden. Wir wandeln das Beispiel so ab, dass immer nur zwei Zeilen ausgeschnitten werden:

```

1 mysql> SELECT blz, bankname FROM bank LIMIT 1, 2;
2 +-----+-----+
3 | blz      | bankname |
4 +-----+-----+
5 | 10010010 | Postbank |
6 | 10010111 | SEB       |
7 +-----+-----+

```



Aufgabe 10.6: Was ist hier falsch? Bitte vergleichen Sie das Ergebnis mit der Ausgabe bei `LIMIT 3`.

Der *offset* ist ein 0 basierender Zähler. So ergeben folgende Befehle das gleiche Ergebnis:

```

1 mysql> SELECT blz, bankname FROM bank LIMIT 1;
2 +-----+-----+
3 | blz      | bankname |
4 +-----+-----+
5 | 10000000 | Bundesbank |
6 +-----+-----+
7 mysql> SELECT blz, bankname FROM bank LIMIT 0, 1;
8 +-----+-----+
9 | blz      | bankname |
10 +-----+-----+
11 | 10000000 | Bundesbank |
12 +-----+-----+

```



Aufgabe 10.7: Geben Sie die Bankleitzahl und den Banknamen der Zeilen 1000 und 1005 aus. Vergleichen Sie die Ergebnisse:

```

1 +-----+
2 | blz      | bankname
3 +-----+
4 | 28062560 | Volksbank Lohne-Mühlen (Gf P2)
5 | 28062740 | Volksbank Bookholzberg-Lemwerder
6 | 28062740 | Volksbank Bookholzberg-Lemwerder (Gf P2)
7 | 28062913 | Volksbank Bösel
8 | 28062913 | Volksbank Bösel (Gf P2)
9 +-----+

```



Hinweis: Leider ergibt der Quelltext unten Fehlermeldungen, da im MySQL oder MariaDB Client LIMIT nur Konstanten akzeptiert werden. Aber was wird hier versucht?

```

1 SET @offset=0;
2 SET @anzahl=2;
3 SELECT blz, bankname FROM bank ORDER BY blz, bankname LIMIT '@offset', '@anzahl';
4 SELECT @offset + @anzahl INTO @offset;
5 SELECT blz, bankname FROM bank ORDER BY blz, bankname LIMIT @offset, @anzahl;
6 SELECT @offset + @anzahl INTO @offset;

```



Hinweis: LIMIT ist kein Standard-SQL, sondern kommt so z.B. in MySQL, MariaDB und PostgreSQL vor. Wobei PostgreSQL eine etwas andere Syntax hat: LIMIT 2 OFFSET 0 anstelle von beispielsweise LIMIT 0,2. Andere Systeme haben aber andere Lösungen dafür: siehe [[Wik19o](#)].

■ 10.6 Ergebnisse exportieren

10.6.1 Wie legt man eine Exportdatei auf dem Server an?



MySQL/MariaDB

```

SELECT [DISTINCT] {*}|spaltenliste|ausdruck}
FROM tabellenname
[WHERE bedingung]
[ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
[LIMIT [offset,] anzahl]
[INTO OUTFILE 'dateiname' exportoptionen]
;
```

exportoptionen: siehe Optionen zum LOAD DATA INFILE auf [Seite 109](#)

Wir haben in [Abschnitt 7.1 auf Seite 105](#) gesehen, wie man Daten per `LOAD DATA INFILE` in eine Tabelle importieren kann. Es gibt auch eine entsprechende Variante des `SELECT`, der Daten in CSV-Format exportiert.

Wenn wir beispielsweise Bankdaten wieder in ein neutrales Datenaustauschformat bringen möchten, würde man eine CSV-Datei `INTO OUTFILE` wie folgt erzeugen:

```

1  SELECT
2    blz Bankleitzahl, bankname Bankname
3    INTO OUTFILE '/var/lib/mysql/interchange/bank.csv'
4      FIELDS
5        TERMINATED BY ';'
6        ENCLOSED BY ""
7        LINES
8        TERMINATED BY '\r\n'
9    FROM
10   bank
11 ORDER BY
12   Bankleitzahl, Bankname;
```

Es wird die Datei `bank.csv` auf dem Server mit folgendem Inhalt erstellt:

```

1 "10000000";"Bundesbank"
2 "10010010";"Postbank"
3 "10010111";"SEB"
4 "10010222";"The Royal Bank of Scotland, Niederlassung Deutschland"
5 [...]
```



Hinweis: Bitte beachten Sie den Sicherheitshinweis zu SQL-Injection in [Abschnitt 7.2.5 auf Seite 119](#).

10.6.2 Wie legt man eine Exportdatei auf dem Client an?

Grundsätzlich ist das keine so gute Idee, die CSV-Datei auf dem Server anzulegen. Man kann den MySQL Client aber gut dafür verwenden, eine lokale Datei anzulegen:

```
1 ralf@localhost:~/> mysql -D oshop -e "SELECT blz, bankname FROM bank;" > bank.txt
```

Nun ist eine lokale Datei erstellt worden. Diese ist nicht so frei konfigurierbar erstellt wie bei einem `INTO OUTFILE`, aber mit gängigen Importassistenten oder Programmen wie `awk` lässt sich eine neutrale CSV-Datei erstellen.

Wenn Sie den `SELECT` – oder noch weitere SQL-Befehle – in eine Datei packen, so lässt sich das Ganze noch weiter vereinfachen¹³:

```
1 ralf@localhost:~/> mysql -D oshop < befehle.sql > bank.txt
```

¹³ Das klappt übrigens auch in der COMMAND-Box unter Windows.

10.6.3 Wie liest man mithilfe eines C#-Client binäre Daten aus?

In [Abschnitt 7.2.4](#) haben wir Bilder in die Tabelle `bild` eingefügt. Diese Daten wollen wir nun wieder auslesen und in Bilddateien auf dem Client ablegen. Auch hier möchte ich dazu einen C#-Client verwenden.

Zunächst werden ein paar benötigte Objekte gebastelt. Mit `dlg` wird das Verzeichnis ausgewählt, in welches die Bilder abgespeichert werden sollen. Die `MySqlCommand`- und `MySqlConnection`-Objekte sollten noch aus dem Obigen klar sein. Da wir nun Daten empfangen wollen, brauchen wir einen Container, der diese Daten aufnimmt: der `MySqlDataReader`. Die restlichen Objekte sollten ebenfalls in ihrer Bedeutung bekannt sein.

Listing 10.1 HauptfensterSelect.cs, Teil 1

```

18     private void btnBild_Click(object sender, EventArgs e) {
19         FolderBrowserDialog dlg = new FolderBrowserDialog();
20         MySqlConnection MySqlConn = new MySqlConnection();
21         MySqlCommand MySqlCmd = new MySqlCommand();
22         MySqlDataReader MySqlData;
23
24         string strSelect;
25         Int32 iDateigroesse;
26         byte[] binBilddaten;
27         FileStream streamDatei;
28         string strDateiname;
```

Nun werden die Verbindungsparameter angegeben. Bitte beachten Sie, dass Sie ggf. andere Werte eintragen müssen. Besonders die Angabe `server` muss überprüft werden.

Listing 10.2 HauptfensterSelect.cs, Teil 2

```

30     MySqlConn.ConnectionString = "server=10.0.2.2";
31     MySqlConn.ConnectionString += ";uid=root";
32     MySqlConn.ConnectionString += ";pwd=";
33     MySqlConn.ConnectionString += ";database=oshop;charset=utf8";
```

Mit diesem `SELECT` werden alle relevanten Bilddaten ausgewählt.

Listing 10.3 HauptfensterSelect.cs, Teil 3

```
35     strSelect = "SELECT bild, dateiname, dateigroesse FROM bild";
```

Nur wenn ein Verzeichnis ausgewählt wurde, soll das Auslesen der Daten begonnen werden.

Listing 10.4 HauptfensterSelect.cs, Teil 4

```

37     if (dlg.ShowDialog() != DialogResult.OK) {
38         return;
39     } // if Ende
```

Jetzt wird versucht, eine Verbindung zum Server herzustellen und den `SELECT` auszuführen. Man könnte hier ggf. noch eine Warnung einbauen, falls keine Daten gefunden wurden.

Listing 10.5 HauptfensterSelect.cs, Teil 5

```

41     try {
42         MySqlConn.Open();
43
44         MySqlCommand.Connection = MySqlConn;
45         MySqlCommand.CommandText = strSelect;
46         MySqlData = MySqlCommand.ExecuteReader();

```

Solange Daten vom `MySqlDataReader`-Objekt gefunden werden, läuft die `while`-Schleife. Durch Methoden des `MySqlDataReader`-Objekts wie `GetOrdinal()` und `GetString()` werden die Ergebnisse der Auswahl mit `SELECT` in passende C#-Variablen kopiert. Dabei wird bei jedem Schleifendurchlauf eine entsprechende Datei im ausgewählten Verzeichnis angelegt, und die Binärdaten werden mit `Write()` dort hineingeschrieben.

Listing 10.6 HauptfensterSelect.cs, Teil 6

```

48     while (MySqlData.Read()) {
49         iDateigroesse = MySqlData.GetInt32(MySqlData.GetOrdinal("dateigroesse"));
50         binBilddaten = new byte[iDateigroesse];
51         MySqlData.GetBytes(MySqlData.GetOrdinal("bild"), 0, binBilddaten, 0,
52                             iDateigroesse);
53
54         strDateiname = dlg.SelectedPath + Path.DirectorySeparatorChar +
55                         MySqlData.GetString("dateiname");
56         streamDatei = new FileStream(Path.GetFullPath(strDateiname),
57                                       FileMode.OpenOrCreate, FileAccess.Write);
58         streamDatei.Write(binBilddaten, 0, iDateigroesse);
59         streamDatei.Close();
60         lbDateien.Items.Insert(0, Path.GetFullPath(strDateiname));
61     } // while Ende

```

Zum Schluss noch alle offenen Verbindungen schließen.

Listing 10.7 HauptfensterSelect.cs, Teil 7

```

60     MySqlData.Close();
61     MySqlConn.Close();
62 } // try Ende

```

Verbleibt nur noch das Auffangen der Ausnahmen, und alles ist gut.

Listing 10.8 HauptfensterSelect.cs, Teil 8

```

63     catch (Exception ex) {
64         MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK,
65                         MessageBoxIcon.Error);
66     } // catch Ende

```

11

Tabellen verbinden



Daten für Auswertungen verbinden.

- Grundkurs
 - Kartesisches Produkt, CROSS JOIN
 - INNER JOIN mit zwei und mehr Tabellen
 - LEFT und RIGHT OUTER JOIN
- Vertiefendes
 - Abkürzung mit USING
 - NATURAL JOIN
 - JOIN als Datenquelle
 - Verknüpfung über Nichtschlüsselpalten
 - EQUI JOIN
 - OUTER JOIN und referentielle Integrität
 - SELF JOIN
 - Common Table Expression (WITH)
 - Einfluss von Indizes auf JOIN

Spätestens jetzt sind wir im nichttrivialen Bereich von SELECT angekommen. Daten aus mehreren Tabellen zusammenzuführen, ist Alltagsgeschäft und wird von vielen DBMS-Tools unterstützt. Trotzdem muss man das Handwerk dahinter verstehen, denn jeder DB-Designer muss wissen, wie die Daten später wieder zusammengebastelt werden müssen. Ohne den Aufwand zu kennen, lässt sich kein seriöses ER-Modell erstellen¹.



Die Quelltexte dieses Kapitels stehen in der Datei `listing08.sql` (siehe [Listing 29.8 auf Seite 482](#), [Listing 29.44 auf Seite 604](#) und [Listing 29.29 auf Seite 552](#)).

¹ So, wie wir es in den ersten Kapiteln getan haben ;-).

■ 11.1 Heiße Liebe: Primär-Fremdschlüsselpaare



SQL:2016, MySQL/MariaDB, PostgreSQL, T-SQL

```
SELECT [DISTINCT]
  {*[spaltenliste|ausdruck]}
  FROM tabellenname[, , tabellenname]*
  [WHERE bedingung]
  [ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
;
```

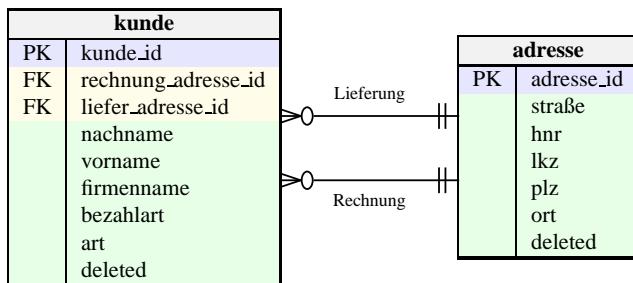


Bild 11.1 ER-Modell: kunde und adresse

Die bisher durchgeführten Auswertungen fanden immer auf *einer* Tabelle statt. Wenn wir uns die ER-Modelle für den Online-Shop anschauen, erkennen wir, dass inhaltlich zusammenhängende Informationen oft auf mehrere Tabellen verteilt sind.²

So stehen beispielsweise die Adressdaten eines Kunden in einer anderen Tabelle als sein Name. Für ein Rechnungsschreiben werden beide Informationen wieder miteinander zu verknüpfen sein. Eine Variante des SELECT erlaubt die Verwendung mehrerer Tabellen in einem SELECT. Wollen wir mal schauen, was dabei herauskommt:

```
1 mysql> SELECT
2      -> kunde_id, nachname, vorname, rechnung_adresse_id, adresse_id
3      -> FROM
4      -> kunde, adresse
5      -> ;
6 +-----+-----+-----+-----+
7 | kunde_id | nachname | vorname | rechnung_adresse_id | adresse_id |
8 +-----+-----+-----+-----+
9 |      1 | Gamdschie | Samweis |                  1 |      1 |
10 |      2 | Beutlin   | Frodo   |                  2 |      1 |
11 |      3 | Beutlin   | Bilbo   |                  2 |      1 |
12 |      4 | Telcontar | Elessar |                  3 |      1 |
13 |      5 | Earendillion | Elrond |                  4 |      1 |
14 |      6 | Eichenschild | Thorin |          NULL |      1 |
15 |      1 | Gamdschie | Samweis |                  1 |      2 |
16 |      2 | Beutlin   | Frodo   |                  2 |      2 |
```

² Dies ist gerade der entscheidende Unterschied zu objektorientierten Datenbanken oder NoSQL.

17	3	Beutlin	Bilbo		2	2
18	4	Telcontar	Elessar		3	2
19	5	Earendillionn	Elrond		4	2
20	6	Eichenschield	Thorin		NULL	2
21	1	Gamdschie	Samweis		1	3
22	2	Beutlin	Frodo		2	3
23	3	Beutlin	Bilbo		2	3
24	4	Telcontar	Elessar		3	3
25	5	Earendillionn	Elrond		4	3
26	6	Eichenschield	Thorin		NULL	3
27	1	Gamdschie	Samweis		1	4
28	2	Beutlin	Frodo		2	4
29	3	Beutlin	Bilbo		2	4
30	4	Telcontar	Elessar		3	4
31	5	Earendillionn	Elrond		4	4
32	6	Eichenschield	Thorin		NULL	4
33	1	Gamdschie	Samweis		1	5
34	2	Beutlin	Frodo		2	5
35	3	Beutlin	Bilbo		2	5
36	4	Telcontar	Elessar		3	5
37	5	Earendillionn	Elrond		4	5
38	6	Eichenschield	Thorin		NULL	5
39	1	Gamdschie	Samweis		1	10
40	2	Beutlin	Frodo		2	10
41	3	Beutlin	Bilbo		2	10
42	4	Telcontar	Elessar		3	10
43	5	Earendillionn	Elrond		4	10
44	6	Eichenschield	Thorin		NULL	10
45	1	Gamdschie	Samweis		1	11
46	2	Beutlin	Frodo		2	11
47	3	Beutlin	Bilbo		2	11
48	4	Telcontar	Elessar		3	11
49	5	Earendillionn	Elrond		4	11
50	6	Eichenschield	Thorin		NULL	11
51 +-----+-----+-----+-----+-----+-----+-----+						
52 42 rows in set (0.00 sec)						

In Zeile 4 werden zwei Tabellen hinter dem FROM angegeben. Das ist neu; bisher stand hier immer nur *die eine* Tabelle.

Im Ergebnis werden mir 42 Zeilen einer *neuen* Tabelle angezeigt. Aber wie sind diese Zeilen gebildet worden? Es fällt auf, dass die kunde_id sich nach dem Schema 1,2,3,4,5,6 wiederholt. Ebenso interessant ist, dass die adresse_id mit jeweils sechs gleichen Werten auftaucht.

Betrachtet man nur die Werte von kunde_id und adresse_id, so hat man Datenpaare, die wie folgt aufgebaut sind: Jede Adresse ist mit allen Kunden kombiniert worden. Adresse 1 mit Kunde 1 bis 6, Adresse 2 mit Kunde 1 bis 6 usw. Da es sechs Kunden und sieben Adressen gibt, erhalten wir 42 Kombinationen, das *Kartesische Produkt*.



Definition 40: Kartesisches Produkt

Seien A und B zwei endliche Mengen, $a \in A$ und $b \in B$, dann ist die Menge aller unterschiedlichen Paare (a, b) das *Kartesische Produkt* K der Mengen A und B .

Diese Definition lässt $A = B$ zu. Wenn n die Anzahl der Elemente in A ist und m die Anzahl der Elemente in B , dann hat K $n \times m$ viele Elemente.

**Definition 41: CROSS JOIN**

Das Kartesische Produkt zweier Mengen wird auch *CROSS JOIN* oder *Kreuzprodukt* genannt.

Toll, ein Kartesisches Produkt³, ja und?



Aufgabe 11.1: Betrachten Sie genau die Zahlenpaare `rechnung_adresse_id` und `adresse_id`. Formulieren Sie eine WHERE-Klausel, die genau die Zeilen übrig lässt, die zu einem Kunden die richtige Adressnummer angeben.

In den [Zeilen 9, 16, 17, 24](#) und [31](#) stehen jeweils die gleichen Werte. Ausformuliert bedeutet dies: Im Fremdschlüssel `rechnung_adresse_id` steht der gleiche Wert wie im Primärschlüssel `adresse_id`. Das sind genau die Elemente des Kartesischen Produkts, die eine gültige Verknüpfung zwischen den beiden Tabellen darstellen. Die WHERE-Klausel sollte somit nur diese Zeilen übrig lassen:

```

1 mysql> SELECT
2     -> kunde_id, nachname, vorname, rechnung_adresse_id, adresse_id
3     -> FROM
4     -> kunde, adresse
5     -> WHERE
6     -> rechnung_adresse_id = adresse_id
7     -> ;
8 +-----+-----+-----+-----+
9 | kunde_id | nachname      | vorname | rechnung_adresse_id | adresse_id |
10+-----+-----+-----+-----+
11 |      1 | Gamdschie    | Samweis |          1 |      1 |
12 |      2 | Beutlin       | Frodo   |          2 |      2 |
13 |      3 | Beutlin       | Bilbo   |          2 |      2 |
14 |      4 | Telcontar     | Elessar |          3 |      3 |
15 |      5 | Earendilionn | Elrond  |          4 |      4 |
16+-----+-----+-----+-----+
17 5 rows in set (0.00 sec)

```



Aufgabe 11.2: Was ist mit den Adressen mit den Primärschlüsselwerten 5, 10 und 11 passiert? Wie kann man das inhaltlich interpretieren? Und was ist mit dem Kunden *Thorin Eichenschild*?

Die Reduzierung des Kartesischen Produkts auf die Zeilen mit passenden Schlüsselpaaren ist schon ein `INNER JOIN`. Was uns nun noch fehlt, ist ein *richtiger Befehl* dazu.

³ Wird von den Azubis gerne *Orgienjoin* genannt.

■ 11.2 INNER JOIN zwischen zwei Tabellen



Definition 42: INNER JOIN

Der **INNER JOIN** zweier Tabellen ist die Teilmenge des kartesischen Produkts, für welche gilt, dass die Fremdschlüsselwerte zu den Primärschlüsselwerten passen.

Die Formulierung über das Kartesische Produkt mit der passenden WHERE-Klausel ist für die Programmierung ein wenig sperrig. Stellen Sie sich vor, dass die Ergebnismenge weitere Bedingungen erfüllen muss oder keine echten Tabellen verwendet werden, sondern Unterabfragen⁴. Deshalb gibt es eine eigene Syntax für den INNER JOIN:



SQL:2016, MySQL/MariaDB, PostgreSQL, T-SQL

```
SELECT [DISTINCT]
  {*|spaltenliste|ausdruck}
  FROM
    tabellennamefk INNER JOIN tabellennamepk
      ON tabellennamefk.fk = tabellennamepk.pk
  [WHERE bedingung]
  [ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
;
```

Der obige Befehl sähe umgebaut so aus:

```
1 SELECT
2   kunde_id, nachname, vorname, rechnung_adresse_id, adresse_id
3   FROM
4     kunde INNER JOIN adresse ON rechnung_adresse_id = adresse_id
5 ;
```



Aufgabe 11.3: Bauen Sie den letzten Befehl so um, dass folgende Ausgabe erzeugt wird:

nachname	vorname	strasse	hnr	ort
Gamschie	Samweis	Beutelhaldenweg	5	Hobbingen
Beutlin	Frodo	Beutelhaldenweg	1	Hobbingen
Beutlin	Bilbo	Beutelhaldenweg	1	Hobbingen
Telcontar	Elessar	Auf der Feste	1	Minas Tirith
Earendillionn	Elrond	Letztes Haus	4	Bruchtal

⁴ Siehe Kapitel 13 auf Seite 225

11.2.1 Bauanleitung für einen INNER JOIN

Die Programmierung eines INNER JOIN fällt meinen Schülerinnen und Schülern oft schwer. Deshalb will ich hier ein wenig ausführlicher beschreiben, wie man einen INNER JOIN zusammenbauen kann. Die passende Aufgabe: Wir wollen zu einer Bankverbindung die Bankleitzahl und den Banknamen wissen.

- Ermitteln der beteiligten Tabellen:** In unserem Fall sind es die Tabellen bankverbindung und bank. Schreiben Sie sich diese Tabellen auf: eine auf die linke Seite vom Blatt und eine auf die rechte Seite.
- Ermitteln der Primär- und Fremdschlüssel:** Schreiben Sie unter den Tabellennamen jeweils den Primärschlüssel und alle vorkommenden Fremdschlüssel.

bankverbindung		bank
kunde_id		bank_id
bankverbindung_id		
bank_id		

- Fremdschlüssel festlegen:** In einer der Tabellen muss ein Fremdschlüssel vorkommen, der auf die andere Tabelle zeigt. Wenn Sie beim Design alles richtig gemacht haben und die Namenskonvention beachten, finden Sie diesen sehr schnell. Es kommen zwei Fremdschlüssel infrage: kunde_id und bank_id. Da eine der Tabellen bank heißt und wir der Namenskonvention gefolgt sind, muss es bank_id sein. Markieren Sie den Fremdschlüssel z.B. mit einem Textmarker.

bankverbindung		bank
kunde_id		bank_id
bankverbindung_id		
bank_id		

- Primärschlüssel festlegen:** Jetzt markieren Sie in der gleichen Farbe in der anderen Tabelle den dazugehörigen Primärschlüssel. Hier ist es die Spalte bank_id

bankverbindung		bank
kunde_id		bank_id
bankverbindung_id		
bank_id		

- Schablone benutzen:** Jetzt schreiben Sie auf das Blatt die Schablone für den INNER JOIN. Das Ganze sollte jetzt so aussehen:

bankverbindung
 kunde_id
 bankverbindung_id
 bank_id

bank
 bank_id

SELECT
 kunde_id, kontonummer, blz, bankname
 FROM
 bankverbindung INNER JOIN bank
 ON bankverbindung.bank_id = bank.bank_id

6. **Fremdschlüsseltabelle eintragen:** Fügen Sie links vom INNER JOIN den Tabellennamen für die Tabelle mit dem markierten Fremdschlüssel ein.

bankverbindung
 kunde_id
 bankverbindung_id
 bank_id

bank
 bank_id

SELECT
 kunde_id, kontonummer, blz, bankname
 FROM
 bankverbindung INNER JOIN bank
 ON bankverbindung.bank_id = bank.bank_id

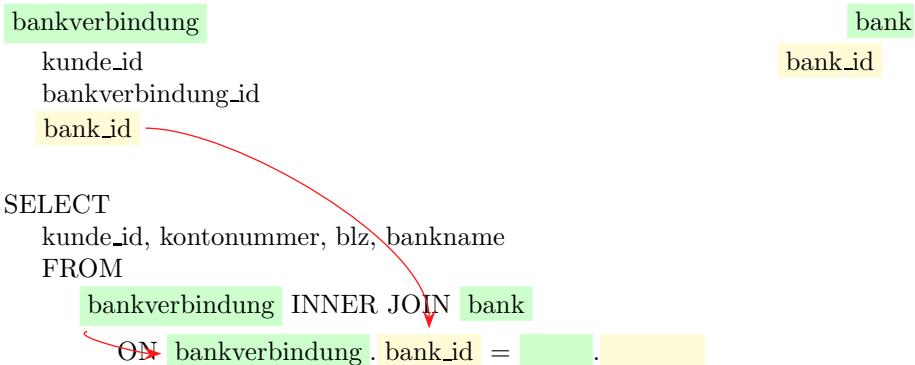
7. **Primärschlüsseltabelle eintragen:** Fügen Sie rechts vom INNER JOIN den Tabellennamen für die Tabelle mit dem markierten Primärschlüssel ein.

bankverbindung
 kunde_id
 bankverbindung_id
 bank_id

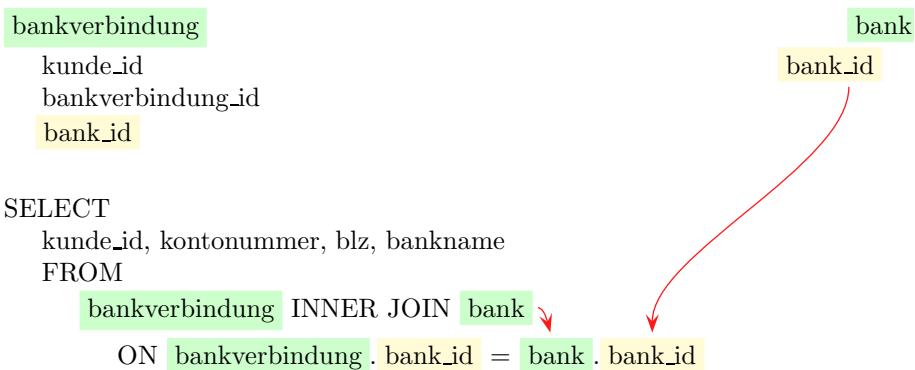
bank
 bank_id

SELECT
 kunde_id, kontonummer, blz, bankname
 FROM
 bankverbindung INNER JOIN bank
 ON bankverbindung.bank_id = bank.bank_id

8. **Fremdschlüssel eintragen:** Fügen Sie zwischen dem ON und dem Gleichheitszeichen den Namen der Fremdschlüsseltabelle, einen Punkt und den Namen des Fremdschlüssels ein.



9. **Primärschlüssel eintragen:** Fügen Sie nach dem = den Namen der Primärschlüsseltabelle, einen Punkt und den Namen des Primärschlüssels ein.



10. Fertig!

```

1 mysql> SELECT
2     ->     kunde_id, kontonummer, blz, bankname
3     ->     FROM
4     ->     bankverbindung INNER JOIN bank
5     ->     ON bankverbindung.bank_id = bank.bank_id
6     ->     ORDER BY kunde_id, kontonummer;
7 +-----+-----+-----+
8 | kunde_id | kontonummer | blz      | bankname   |
9 +-----+-----+-----+
10 |       1 | 1111111111 | 10010010 | Postbank   |
11 |       1 | 1111111112 | 10010010 | Postbank   |
12 |       2 | 2222222221 | 10060198 | Pax-Bank   |
13 |       3 | 3333333331 | 10060198 | Pax-Bank   |
14 |       4 | 4444444441 | 12070000 | Deutsche Bank |
15 |       5 | 5555555551 | 12070000 | Deutsche Bank |
16 +-----+-----+-----+
  
```

Ein zweites Beispiel: Zu einem Kunden werden die Kontonummern ausgegeben.

- Ermitteln der beteiligten Tabellen:** Es sind kunde und bankverbindung.
- Ermitteln der Primär- und Fremdschlüssel:** In der Tabelle bankverbindung gibt es den zusammengesetzten Primärschlüssel mit kunde_id und bankverbindung_nr und den Fremdschlüssel bank_id. In der Tabelle kunde gibt es den Primärschlüssel

kunde_id und die beiden Fremdschlüssel rechnung_adresse_id und liefer_adresse_id.

3. **Fremdschlüssel festlegen:** Da wir die Namenskonvention beachtet haben, brauchen wir nur nach einem Fremdschlüssel suchen, der so wie die andere Tabelle heißt. Dies ist in unserem Fall kunde_id.
4. **Primärschlüssel festlegen:** Jetzt wird die Spalte kunde_id in der Tabelle kunde markiert.
5. **Schablone benutzen:** Jetzt schreiben Sie auf das Blatt die Schablone für den INNER JOIN.
6. **Fremdschlüsseltabelle eintragen:** Fügen Sie links vom INNER JOIN den Tabellennamen für die Tabelle mit dem markierten Fremdschlüssel ein.
7. **Primärschlüsseltabelle eintragen:** Fügen Sie an die passende Stelle den Tabellennamen für die Tabelle mit dem markierten Primärschlüssel ein.
8. **Fremdschlüssel eintragen:** Fügen Sie an die passende Stelle den Spaltennamen des markierten Fremdschlüssels ein.
9. **Primärschlüssel eintragen:** Fügen Sie an die passende Stelle den Spaltennamen des markierten Primärschlüssels ein.

10. Fertig!

```

1 mysql> SELECT
2      ->    nachname, vorname, kontonummer
3      ->  FROM
4      ->  bankverbindung b INNER JOIN kunde k ON b.kunde_id = k.kunde_id
5      -> ORDER BY nachname, vorname, kontonummer;
6 +-----+-----+-----+
7 | nachname | vorname | kontonummer |
8 +-----+-----+-----+
9 | Beutlin   | Bilbo   | 3333333331 |
10 | Beutlin   | Frodo   | 2222222221 |
11 | Earendilionn | Elrond | 5555555551 |
12 | Gamdschie  | Samweis | 1111111111 |
13 | Gamdschie  | Samweis | 1111111112 |
14 | Telcontar   | Elessar  | 4444444441 |
15 +-----+-----+-----+

```

Haben Sie gesehen, dass in Zeile 4 die Tabellennamen durch Aliase ersetzt wurden?



Aufgabe 11.4: Geben Sie zu allen Kunden Namen und Rechnungsadresse aus.

Aufgabe 11.5: Geben Sie zu allen Kunden den Namen und die Lieferadresse aus. Interpretieren Sie das Ergebnis.

Aufgabe 11.6: Geben Sie zu jedem Kunden den Namen und die Bestellungen nach Kundennamen und Bestelldatum sortiert aus. Die letzte Bestellung des Kunden soll zuerst erscheinen.

Aufgabe 11.7: Geben Sie zu jeder Bestellung die Kundennummer, das Bestelldatum und die Positionen aus. Die Sortierung soll nach Kundennummer, Bestellnummer und Position erfolgen.

Aufgabe 11.8: Geben Sie zu jeder Position den Artikelnamen aus. Es soll nach Artikelname sortiert werden.

11.2.2 Abkürzende Schreibweisen



SQL:2016, MySQL/MariaDB

```
SELECT [DISTINCT]
  {*|spaltenliste|ausdruck}
  FROM
    tabellennamefk INNER JOIN tabellennamepk
      [{ON tabellennamefk.fk = tabellennamepk.pk|USING(spaltenname)}]
    [WHERE bedingung]
    [ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
;
```

Falls der Name des Fremdschlüssels genau der gleiche ist wie der des Primärschlüssels, kann man die ON-Klausel durch eine kürzere Variante mit USING ersetzen:

```
1 SELECT
2   kunde_id, kontonummer, blz, bankname
3   FROM
4     bankverbindung INNER JOIN bank USING (bank_id)
5 ;
```

Bei den Fremdschlüsseln, die anders als die Primärschlüssel heißen, wie beispielsweise rechnung_adresse_id, ist eine solche Abkürzung nicht möglich.



Aufgabe 11.9: Bauen Sie Ihre Lösungen zu den Aufgaben um, wenn ein USING möglich ist.

Sind die Fremdschlüssel-/Primärschlüssel Spalten die einzigen Spalten, die in beiden Tabellen gleich sind und die Verknüpfung definieren, spricht man von einem NATURAL JOIN.



Definition 43: NATURAL JOIN

Werden zwei Tabellen über die Spalten verknüpft, die den gleichen Namen und Inhalt haben, spricht man von einem NATURAL JOIN.

```
1 SELECT
2   kunde_id, kontonummer, blz, bankname
3   FROM
4     bankverbindung NATURAL JOIN bank
5 ;
```



Hinweis: Bitte beachten Sie, dass bei unseren Tabellen in einem NATURAL JOIN auch die Spalte deleted in die Verknüpfung mit einfließt.

11.2.3 Als Datenquelle für temporäre Tabellen

Wir haben oben die Bankverbindung zu einem Kunden ermittelt. Annahme: Wir wollen jetzt viele Auswertungen der Kunden inklusive der Bankverbindung machen, dann müs-

sen jedes Mal im entsprechenden SELECT die Informationen mit INNER JOIN verknüpft werden.

Obwohl INNER JOIN-Anweisungen durch passend gewählte Indizes sehr beschleunigt werden, entsteht trotzdem eine Rechnerlast auf dem Server, die jedes Mal das gleiche – oder fast das gleiche – Ergebnis liefert⁵.

Da es plausibel ist anzunehmen, dass sich die Kundenstammdaten (siehe [Definition 36](#)) für einen gewissen Auswertungszeitraum nicht ändern, könnte man statt dessen das Ergebnis des INNER JOIN in eine (temporäre) Tabelle ablegen und mit dieser weiterarbeiten.

Als Beispiel wollen wir die Kundendaten mit Rechnungsanschrift und allen Bestellungen ausgeben und danach die Kundendaten mit Rechnungsanschrift mit allen Bankverbindungen.

In beiden Fällen werden die Kundendaten mit Rechnungsanschrift benötigt. Das können wir schon:

```

1  SELECT
2    k.kunde_id, k.nachname, k.vorname, a.strasse, a.hnr, a.plz, a.ort
3    FROM
4      kunde k INNER JOIN adresse a
5        ON k.rechnung_adresse_id = a.adresse_id
6  ;

```

Sie bemerken bitte, dass hier abkürzende Alias ([Zeile 4](#)) verwendet werden. Das ist gerade bei Verknüpfungen sehr beliebt, da hier oft zwischen den Tabellen unterschieden werden muss. Diese Abfrage wird jetzt in eine Variante des CREATE TABLE eingebaut.

The screenshot shows the MySQL/MariaDB interface. On the left, there's a toolbar with icons for database, table, column, key, and other database operations. The main area has a title bar "MySQL/MariaDB". Below it, the SQL command is displayed:

```

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tabellename
  SELECT auswahl
;

```

```

1  mysql> CREATE TEMPORARY TABLE tmp_kadresse
2    -> SELECT
3    ->   k.kunde_id, k.nachname, k.vorname, a.strasse, a.hnr, a.plz, a.ort
4    ->   FROM
5    ->     kunde k INNER JOIN adresse a
6    ->       ON k.rechnung_adresse_id = a.adresse_id;
7
8  Query OK, 5 rows affected (0.09 sec)
9  Records: 5  Duplicates: 0  Warnings: 0
10
11 mysql> SELECT kunde_id, nachname, ort FROM tmp_kadresse;
12 +-----+-----+-----+
13 | kunde_id | nachname | ort      |
14 +-----+-----+-----+
15 |         1 | Gamdschie | Hobbingen |
16 |         2 | Beutlin    | Hobbingen |
17 |         3 | Beutlin    | Hobbingen |
18 |         4 | Telcontar  | Minas Tirith |

```

⁵ Wobei davon auszugehen ist, dass exakt gleiche Ergebnisse durch eine Cache-Strategie beschleunigt zur Verfügung gestellt werden können.

```

19 |      5 | Earendilionn | Bruchtal      |
20 +-----+-----+-----+

```

Jetzt zur ersten Auswertung: Alle Kunden mit Adressdaten und ihren Bestellungen:

```

1 mysql> SELECT
2     -> t.kunde_id, t.nachname, t.ort, b.bestellung_id, DATE(b.datum)
3     -> FROM
4     -> bestellung b INNER JOIN tmp_kadresse t USING (kunde_id);
5
6 +-----+-----+-----+-----+
7 | kunde_id | nachname | ort       | bestellung_id | DATE(b.datum) |
8 +-----+-----+-----+-----+
9 |      1 | Gamdschie | Hobbingen |           1 | 2012-03-24   |
10 |      2 | Beutlin   | Hobbingen |           2 | 2012-03-23   |
11 +-----+-----+-----+-----+

```



Aufgabe 11.10: Erzeugen Sie mit der Spaltenliste t.* , b.bestellung_id , datum eine neue temporäre Tabelle und verknüpfen Sie diese mit den Positionen der Bestellungen. Achten Sie auf eine sinnvolle Sortierung.

Jetzt zur zweiten Auswertung: Alle Kunden mit Adressdaten und allen Bankverbindungen. Das wollen wir jetzt besonders schön machen. Zuerst eine temporäre Tabelle für die Bankleitzahl und den Banknamen, und dann werden diese beiden temporären Tabellen wieder verknüpft. Und weil wir es jetzt können, wird am Ende noch eine temporäre Tabelle gebaut.

```

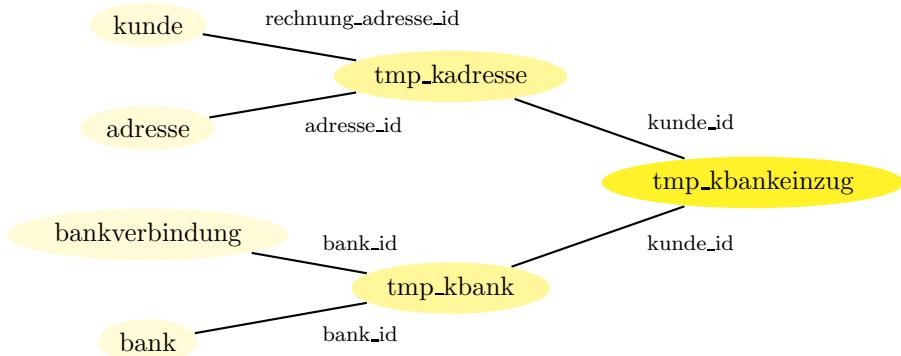
1 mysql> CREATE TEMPORARY TABLE tmp_kbank
2     -> SELECT
3     -> bv.kunde_id, bv.bankverbindung_nr, bv.kontonummer,
4     -> bv.iban, ba.blz, ba.bankname
5     -> FROM
6     -> bankverbindung bv INNER JOIN bank ba USING (bank_id);
7
8 mysql> CREATE TEMPORARY TABLE tmp_kbankeinzug
9     -> SELECT
10    -> ka.*, kb.bankverbindung_nr, kb.kontonummer,
11    -> kb.iban, kb.blz, kb.bankname
12    -> FROM
13    -> tmp_kadresse ka INNER JOIN tmp_kbank kb USING(kunde_id);
14
15 mysql> SELECT kunde_id, nachname, ort, bankname FROM tmp_kbankeinzug;
16 +-----+-----+-----+
17 | kunde_id | nachname | ort       | bankname   |
18 +-----+-----+-----+
19 |      1 | Gamdschie | Hobbingen | Postbank   |
20 |      1 | Gamdschie | Hobbingen | Postbank   |
21 |      2 | Beutlin   | Hobbingen | Pax-Bank   |
22 |      3 | Beutlin   | Hobbingen | Pax-Bank   |
23 |      4 | Telcontar | Minas Tirith | Deutsche Bank |
24 |      5 | Earendilionn | Bruchtal | Deutsche Bank |
25 +-----+-----+-----+

```

Das Zusammenspiel der tatsächlichen und temporären Tabellen sei hier in einer Baumstruktur dargestellt. Die Blätter⁶ sind die Ursprungstabellen. Zwei davon werden in je-

⁶ die Knoten ganz links

weils eine temporäre Tabelle mit INNER JOIN zusammengefasst. Die beiden neu erzeugten temporären Tabellen werden wiederum in eine temporäre Tabelle vereinigt.



Jede der Tabellen kann nun unabhängig voneinander verwendet werden, da die Datensätze in den temporären Tabellen keine Verweise auf die ursprünglichen Datensätze sind, sondern Kopien. Diese *Unabhängigkeit* ist sehr wichtig, wenn man mit vielen konkurrierenden Zugriffen auf die Tabellen kunde etc. rechnet.

Bei der Betrachtung von Transaktionen (siehe [Kapitel 19 auf Seite 319](#)) werden wir noch sehen, dass Tabellen für Operationen gesperrt werden. Durch die temporären Tabellen kann aber auf der Tabelle kunde gearbeitet werden, während Auswertungen auf tmp_kadresse stattfinden.



Hinweis: Ich habe oben erwähnt, dass es sich um Stammdaten handelt und somit während der Auswertungen eine geringe Änderungswahrscheinlichkeit besteht. Bei Bewegungsdaten ist das Verfahren genau abzuwagen. Vielleicht lassen sich ja Änderungen auf eine gewisse Uhrzeit festmachen, und die Auswertungen passieren außerhalb dieses Zeitfensters.

11.2.4 Ist Ihnen was aufgefallen?

Beim Bau der letzten temporären Tabelle gab es keinen Primärschlüssel!

```

1 mysql> DESCRIBE tmp_kadresse;
2 +-----+-----+-----+-----+-----+
3 | Field      | Type           | Null | Key | Default | Extra |
4 +-----+-----+-----+-----+-----+
5 | kunde_id   | int(10) unsigned | NO   | NO  | 0       | NULL  |
6 | nachname   | varchar(255)     | NO   | NO  |          | NULL  |
7 | vorname    | varchar(255)     | NO   | NO  |          | NULL  |
8 | strasse    | varchar(255)     | NO   | NO  |          | NULL  |
9 | hnr         | varchar(255)     | NO   | NO  |          | NULL  |
10 | plz        | char(9)          | NO   | NO  |          | NULL  |
11 | ort        | varchar(255)     | NO   | NO  |          | NULL  |
12 +-----+-----+-----+-----+-----+
13
14
15
  
```

```

16 mysql> DESCRIBE tmp_kbank;
17 +-----+-----+-----+-----+-----+
18 | Field      | Type       | Null | Key | Default | Extra |
19 +-----+-----+-----+-----+-----+
20 | kunde_id   | int(10) unsigned | NO   |   | NULL    | NULL  |
21 | bankverbindung_nr | int(10) unsigned | NO   |   | NULL    | NULL  |
22 | kontonummer | char(25)     | NO   |   |         | NULL  |
23 | iban        | char(34)     | NO   |   |         | NULL  |
24 | blz         | char(12)     | NO   |   |         | NULL  |
25 | bankname    | varchar(255) | NO   |   |         | NULL  |
26 +-----+-----+-----+-----+-----+

```

Und tatsächlich: Für einen `INNER JOIN` ist es nicht nötig, Primär-Fremdschlüsselpaare zu bilden. Dem Befehl ist es völlig egal, was bei einem `INNER JOIN` in der `ON-` oder `USING-`Klausel steht. Es wird nur die Verträglichkeit der Datentypen untersucht.

Natürlich erfolgt die überwiegende Anzahl der Verknüpfungen auf Primär-Fremdschlüsselpaaren. Aber hier haben wir ein Beispiel dafür, dass auch die Verknüpfung über Nicht-schlüsselpalten⁷ sinnvoll sein kann.

Es geht sogar noch weiter. Bei der `ON`-Klausel ist das Gleichheitszeichen nicht zwingend vorgeschrieben:

```

1 mysql> SELECT
2   ->   k.kunde_id, k.vorname, a.strasse, a.hnr, a.plz, a.ort
3   ->   FROM
4   ->   kunde k INNER JOIN adresse a
5   ->   ON k.rechnung_adresse_id <= a.adresse_id;
6 +-----+-----+-----+-----+-----+
7 | kunde_id | vorname | strasse      | hnr | plz  | ort   |
8 +-----+-----+-----+-----+-----+
9 |      1 | Samweis | Beutelhaldenweg | 5  | 67676 | Hobbingen |
10 |     1 | Samweis | Beutelhaldenweg | 1  | 67676 | Hobbingen |
11 |     2 | Frodo   | Beutelhaldenweg | 1  | 67676 | Hobbingen |
12 |     3 | Bilbo   | Beutelhaldenweg | 1  | 67676 | Hobbingen |
13 |     1 | Samweis | Auf der Feste   | 1  | 54786 | Minas Tirith |
14 |     2 | Frodo   | Auf der Feste   | 1  | 54786 | Minas Tirith |
15 |     3 | Bilbo   | Auf der Feste   | 1  | 54786 | Minas Tirith |
16 |     4 | Elessar  | Auf der Feste   | 1  | 54786 | Minas Tirith |
17 |     1 | Samweis | Letztes Haus   | 4  | 87567 | Bruchtal  |
18 |     2 | Frodo   | Letztes Haus   | 4  | 87567 | Bruchtal  |
19 |     3 | Bilbo   | Letztes Haus   | 4  | 87567 | Bruchtal  |
20 [...]          |
21 |     2 | Frodo   | Baradur        | 1  | 62519 | Lugburz   |
22 |     3 | Bilbo   | Baradur        | 1  | 62519 | Lugburz   |
23 |     4 | Elessar  | Baradur        | 1  | 62519 | Lugburz   |
24 |     5 | Elrond   | Baradur        | 1  | 62519 | Lugburz   |
25 |     1 | Samweis | Hochstrasse   | 4a | 44879 | Bochum    |
26 |     2 | Frodo   | Hochstrasse   | 4a | 44879 | Bochum    |
27 |     3 | Bilbo   | Hochstrasse   | 4a | 44879 | Bochum    |
28 |     4 | Elessar  | Hochstrasse   | 4a | 44879 | Bochum    |
29 |     5 | Elrond   | Hochstrasse   | 4a | 44879 | Bochum    |
30 |     1 | Samweis | Industriegebiet | 8  | 44878 | Bochum    |
31 |     2 | Frodo   | Industriegebiet | 8  | 44878 | Bochum    |
32 |     3 | Bilbo   | Industriegebiet | 8  | 44878 | Bochum    |
33 |     4 | Elessar  | Industriegebiet | 8  | 44878 | Bochum    |

```

⁷ Immerhin sind es Fremdschlüssel.

```

34 |      5 | Elrond | Industriegebiet | 8   | 44878 | Bochum      |
35 +-----+-----+-----+-----+-----+-----+
36 28 rows in set (0.00 sec)

```

In Zeile 5 ist anstelle des `=` ein `<=` verwendet worden. Ich kann mir zwar keine vernünftige Anwendung dafür ausdenken, will aber nicht bestreiten, dass es sie irgendwo gibt.

Wird aber die Verknüpfung über die Gleichheit hergestellt, hat das Ganze einen schönen Namen:



Definition 44: EQUI JOIN

Wird bei INNER JOIN auf die Gleichheit von Fremdschlüsselwert und Primärschlüsselwert getestet, spricht man von einem *EQUI JOIN*.

Sie haben sicher bei der [Definition 42 auf Seite 187](#) bemerkt, dass nur allgemein von *passen* gesprochen wird. Was immer dieses *passen* auch bedeutet. Der Test auf Gleichheit ist aber so gängig, dass der Begriff INNER JOIN synonym für EQUI JOIN verwendet wird.



Hinweis: Lassen Sie sich nicht verwirren. Erst mal nach Primär-Fremdschlüsselpaaren suchen und diese mit `=` verknüpfen. In den absolut meisten Fällen sind Sie auf der sicheren Seite. Erst, wenn das so überhaupt nicht klappen sollte, denken Sie über Alternativen nach.

■ 11.3 INNER JOIN über mehr als zwei Tabellen

Eine Möglichkeit, mehr als zwei Tabellen zu verknüpfen, haben Sie oben auf [Seite 193](#) kennengelernt. Die Verwendung von temporären Tabellen bietet sich aber nur bei Stammdaten an oder wenn die Änderungen unerheblich für das Gesamtergebnis sind.

Trotzdem können wir aus der Episode mit den temporären Tabellen eine wichtige Schlussfolgerung ziehen: Das Ergebnis eines INNER JOINs ist wieder eine Tabelle. Wie kann ich damit die Beschränkung umgehen, dass der INNER JOIN nur zwei Tabellen verknüpfen kann?

Betrachten wir dazu den Klassiker für die Verknüpfung von mehr als zwei Tabellen: das Auflösen einer *n:m*-Verknüpfung (siehe [Abschnitt 2.2.4 auf Seite 29](#)). Wir haben eine *n:m*-Verknüpfung zwischen den Tabellen artikel und warengruppe.

Unser erster Versuch besteht darin, wie oben beschrieben vorzugehen, indem wir zwei *1:n*-Verknüpfungen erstellen:

1. **Ermitteln der beteiligten Tabellen:** artikel_nm_warengruppe und artikel.
2. **Ermitteln der Primär- und Fremdschlüssel:** In der Tabelle artikel_nm_warengruppe gibt es keine *echten* Primärschlüssel, aber die beiden Fremdschlüssel warengruppe_id und artikel_id. In der Tabelle artikel gibt es nur den Primärschlüssel artikel_id.
3. **Fremdschlüssel festlegen:** Laut Namenskonvention muss artikel_id in der Tabelle artikel_nm_warengruppe der gesuchte Fremdschlüssel sein.

4. **Primärschlüssel festlegen:** Wir markieren `artikel_id` in `artikel`.
5. **Schablone benutzen:** Jetzt schreiben Sie auf das Blatt die Schablone für den INNER JOIN.
6. **Fremdschlüsseltabelle eintragen:** Fügen Sie links vom INNER JOIN den Tabellennamen für die Tabelle mit dem markierten Fremdschlüssel ein.
7. **Primärschlüsseltabelle eintragen:** Fügen Sie an die passende Stelle den Tabellennamen für die Tabelle mit dem markierten Primärschlüssel ein.
8. **Fremdschlüssel eintragen:** Fügen Sie an die passende Stelle den Spaltennamen des markierten Fremdschlüssels ein.
9. **Primärschlüssel eintragen:** Fügen Sie an die passende Stelle den Spaltennamen des markierten Primärschlüssels ein.
10. **Fertig!**

```

1 mysql> SELECT
2      -> a.bezeichnung, nm.warenguppe_id
3      -> FROM
4      -> artikel_nm_warenguppe nm NATURAL JOIN artikel a;
5 +-----+-----+
6 | bezeichnung | warenguppe_id |
7 +-----+-----+
8 | Feder          | 1 |
9 | Papier (100)   | 1 |
10 | Schaufel        | 3 |
11 | Schaufel        | 4 |
12 | Silberzwiebel   | 2 |
13 | Silberzwiebel   | 3 |
14 | Spaten          | 3 |
15 | Spaten          | 4 |
16 | Tinte (blau)    | 1 |
17 | Tinte (gold)    | 1 |
18 | Tinte (rot)     | 1 |
19 | Tulpenzwiebel   | 2 |
20 | Tulpenzwiebel   | 3 |
21 +-----+-----+

```

Das ganze Prozedere mit den Tabellen `artikel_nm_warenguppe` und `warenguppe` führt zu einer zweiten Verknüpfung:

```

1 mysql> SELECT
2      -> w.bezeichnung, nm.artikel_id
3      -> FROM
4      -> artikel_nm_warenguppe nm NATURAL JOIN warenguppe w;
5 +-----+-----+
6 | bezeichnung | artikel_id |
7 +-----+-----+
8 | Bürobedarf   | 3001 |
9 | Bürobedarf   | 3005 |
10 | Bürobedarf  | 3006 |
11 | Bürobedarf   | 3007 |
12 | Bürobedarf   | 3010 |
13 | Gartenbedarf | 7856 |
14 | Gartenbedarf | 7863 |
15 | Gartenbedarf | 9010 |
16 | Gartenbedarf | 9015 |
17 | Pflanzen      | 7856 |

```

```

18 | Pflanzen      |      7863 |
19 | Werkzeug      |      9010 |
20 | Werkzeug      |      9015 |
21 +-----+-----+

```

Und jetzt kommt's: Wir nehmen einfach den letzten SELECT, klammern die Verknüpfung und verwenden nicht mehr NATURAL, sondern INNER JOIN:

```

1 SELECT
2   w.bezeichnung, nm.artikel_id
3   FROM
4     (artikel_nm_warengruppe nm INNER JOIN warengruppe w USING(warengruppe_id))
5 ;

```

Der Inhalt der Klammer ist eine (!) Tabelle, und diese könnte der linke Teil einer neuen Verknüpfung sein:

```

1 SELECT
2   w.bezeichnung, a.bezeichnung
3   FROM
4     (artikel_nm_warengruppe nm INNER JOIN warengruppe w USING (warengruppe_id))
5           INNER JOIN artikel a USING(artikel_id)
6 ;

```

Und eine weitere gute Nachricht ist, dass man die Klammern nicht braucht; sie dienten nur dem besseren Verständnis:

```

1 SELECT
2   w.bezeichnung Warengruppe, a.bezeichnung Artikel
3   FROM
4     artikel_nm_warengruppe nm INNER JOIN warengruppe w USING (warengruppe_id)
5           INNER JOIN artikel a USING(artikel_id)
6 ;

```

Dieser SELECT liefert das gewünschte Ergebnis:

```

1 +-----+-----+
2 | Warengruppe | Artikel      |
3 +-----+-----+
4 | Bürobedarf  | Papier (100)  |
5 | Bürobedarf  | Tinte (gold)  |
6 | Bürobedarf  | Tinte (rot)   |
7 | Bürobedarf  | Tinte (blau)  |
8 | Bürobedarf  | Feder        |
9 | Gartenbedarf | Silberwiebel |
10 | Gartenbedarf| Tulpenzwiebel|
11 | Gartenbedarf| Schaufel    |
12 | Gartenbedarf| Spaten      |
13 | Pflanzen    | Silberwiebel |
14 | Pflanzen    | Tulpenzwiebel|
15 | Werkzeug    | Schaufel    |
16 | Werkzeug    | Spaten      |
17 +-----+-----+

```

Die Spezifikation des SELECT lässt sich somit erweitern:



SQL:2016, PostgreSQL, T-SQL

```
SELECT [DISTINCT]
  { * | spaltenliste | ausdruck }
  FROM
    tabname [INNER JOIN tabname
      ON tabname.spaltenname = tabname.spaltenname ]*
  [WHERE bedingung]
  [ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
```

MySQL/MariaDB

```
SELECT [DISTINCT]
  { * | spaltenliste | ausdruck }
  FROM
    tabname [INNER JOIN tabname
      ON tabname.spaltenname = tabname.spaltenname | USING (spaltenname) ]*
  [WHERE bedingung]
  [ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
;
```

Der Abschnitt rechts vom ersten *tabname* kann beliebig oft wiederholt werden, wobei beliebig auch bedeuten kann, dass er gar nicht vorkommt. Es wäre dann ein normaler SELECT.



Aufgabe 11.11: Warum konnte beim letzten SELECT kein NATURAL JOIN verwendet werden?

Aufgabe 11.12: Erweitern Sie diesen SELECT um die Positionen, in denen der Artikel vorkommt.

Aufgabe 11.13: Erweitern Sie das Ergebnis der letzten Aufgabe um die Daten der Bestellung.

Aufgabe 11.14: Erweitern Sie das Ergebnis der letzten Aufgabe um die Daten des Kunden.

Aufgabe 11.15: Erweitern Sie das Ergebnis der letzten Aufgabe um die Rechnungsadresse des Kunden.

■ 11.4 Es muss nicht immer heiße Liebe sein: OUTER JOIN

Wir wollen die Kunden mit ihren ggf. vorhandenen Bestellungen wissen:

```
1 mysql> SELECT
2       ->   k.kunde_id, k.nachname, k.vorname, b.datum
3       ->   FROM
```

```

4      -> bestellung b INNER JOIN kunde k USING (kunde_id)
5      -> ORDER BY
6      -> k.nachname, k.vorname;
7 +-----+-----+-----+
8 | kunde_id | nachname | vorname | datum          |
9 +-----+-----+-----+
10 |       2 | Beutlin   | Frodo    | 2012-03-23 16:11:00 |
11 |       1 | Gamdschie  | Samweis  | 2012-03-24 17:41:00 |
12 +-----+-----+-----+

```



Aufgabe 11.16: Es werden zwar alle Bestellungen ausgegeben, aber nicht alle Kunden. Warum?

Möchte man aber alle Kunden sehen, auch wenn diese keine Bestellungen aufgegeben haben, kann man keinen INNER JOIN verwenden; dazu wird ein OUTER JOIN, hier ein RIGHT OUTER JOIN, notwendig sein.

```

1 mysql> SELECT
2     -> k.kunde_id, k.nachname, k.vorname, b.datum
3     -> FROM
4     -> bestellung b RIGHT OUTER JOIN kunde k USING (kunde_id)
5     -> ORDER BY
6     -> k.nachname, k.vorname;
7 +-----+-----+-----+
8 | kunde_id | nachname   | vorname | datum          |
9 +-----+-----+-----+
10 |       3 | Beutlin    | Bilbo   | NULL           |
11 |       2 | Beutlin    | Frodo   | 2012-03-23 16:11:00 |
12 |       5 | Earendillion | Elrond  | NULL           |
13 |       6 | Eichenschild | Thorin | NULL           |
14 |       1 | Gamdschie  | Samweis | 2012-03-24 17:41:00 |
15 |       4 | Telcontar  | Elessar | NULL           |
16 +-----+-----+-----+

```

Zuerst fällt auf, dass nicht zwei, sondern sechs Zeilen ausgegeben werden, denn jetzt werden auch die Kunden angezeigt, für die keine Bestellungen vorliegen ([Zeilen 10, 12, 13 und 15](#)). Da SQL nicht weiß, was es in den entsprechenden Spalten an Werten eintragen soll, wird hier NULL verwendet.



Definition 45: OUTER JOIN

Der **OUTER JOIN** zweier Tabellen ist der INNER JOIN dieser beiden Tabellen, der um folgende Zeilen erweitert wird: Zeilen der rechten (*RIGHT OUTER JOIN*) oder linken (*LEFT OUTER JOIN*) Tabelle, für welche keine passenden Paarung gefunden wurde.

Wird der INNER JOIN um Zeilen aus der linken und der rechten Tabelle erweitert, spricht man von einem **FULL OUTER JOIN**.

Keine Sorge, die Sache ist komplizierter zu erklären als zu benutzen. Betrachten wir noch einmal obiges Beispiel. Der INNER JOIN liefert uns nur die Zeilen, für welche ein passendes Primär-Fremdschlüsselpaar gefunden wird. Der RIGHT JOIN in [Zeile 4](#) erweitert jetzt das Ergebnis des INNER JOIN. Um welche Zeilen? Laut **Definition 45** um die Zeilen der rechts vom JOIN stehenden Tabelle, für die keine passenden Primär-Fremdschlüsselpaare gefunden werden. Und tatsächlich, es sind dies die Kunden ohne Bestellung; für diese Pri-

märschlüsselwerte kann kein passender Fremdschlüsselwert und bestellung gefunden werden.

Und das mit dem LEFT und RIGHT ist einfach nur banal. Bei LEFT wird um die Zeilen der Tabelle, die links vom Wort JOIN steht, erweitert. Bei RIGHT um die Tabelle, die rechts vom Wort JOIN steht. Vertauscht man die beiden Tabellennamen und macht aus RIGHT ein LEFT, kommt genau das gleiche Ergebnis heraus:

```

1  SELECT
2    k.kunde_id, k.nachname, k.vorname, b.datum
3  FROM
4    Kunde k LEFT OUTER JOIN bestellung b USING (kunde_id)
5  ORDER BY
6    k.nachname, k.vorname;
```

Ein LEFT OUTER JOIN oder RIGHT OUTER JOIN wird immer dann verwendet, wenn nicht nur die Zeilen einer Tabelle interessant sind, für die es eine Paarung gibt. Nehmen Sie beispielsweise eine Liste von Vertretern und die von den Vertretern abgeschlossenen Verträge. Wollte man nun die Anzahl der abgeschlossenen Verträge pro Vertreter wissen, würde ein INNER JOIN alle Vertreter unterdrücken, die noch keinen Vertrag abgeschlossen haben. In einer Übersichtsauswertung wäre dies sicherlich fehlerhaft.

Hurra, wir haben eine neue Variante des SELECT:



SQL:2016, PostgreSQL, T-SQL

```

SELECT [DISTINCT]
  {*[|spaltenliste|ausdruck]}
  FROM
    tabname [[RIGHT OUTER|LEFT OUTER|FULL OUTER|INNER] JOIN tabname
      ON tabname.spaltenname = tabname.spaltenname]*]
  [WHERE bedingung]
  [ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
  ;
```

MySQL und MariaDB kennen keinen FULL OUTER JOIN. Wie man diesen simulieren kann, kommt später. Bei beiden ist das Schlüsselwort OUTER optional, und es gilt: Wird nur JOIN angegeben, wird ein INNER JOIN verwendet.



MySQL/MariaDB

```

SELECT [DISTINCT]
  {*[|spaltenliste|ausdruck]}
  FROM
    tabname [[RIGHT [OUTER]|LEFT [OUTER]|INNER] JOIN tabname
      {ON tabname.spaltenname = tabname.spaltenname|USING(spaltenname)}]*]
  [WHERE bedingung]
  [ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
  [LIMIT [offset,]anzahl]
  [INTO OUTFILE 'dateiname' exportoptionen]
  ;
```

Ein weiteres Beispiel: Wir möchten die Lieferanten und ihre Artikel wissen⁸.

```

1 mysql> SELECT
2     -> l.lieferant_id, l.firmenname, a.artikel_id, a.bezeichnung
3     -> FROM
4     -> artikel_nm_lieferant nm INNER JOIN artikel a   USING(artikel_id)
5     ->                               INNER JOIN lieferant l USING(lieferant_id)
6     -> ORDER BY
7     ->   firmenname;
8 +-----+-----+-----+-----+
9 | lieferant_id | firmenname          | artikel_id | bezeichnung |
10+-----+-----+-----+-----+
11 |      3 | Bürohengst GmbH           |    3001 | Papier (100)  |
12 |      3 | Bürohengst GmbH           |    3005 | Tinte (gold) |
13 |      3 | Bürohengst GmbH           |    3006 | Tinte (rot)  |
14 |      3 | Bürohengst GmbH           |    3007 | Tinte (blau) |
15 |      3 | Bürohengst GmbH           |    3010 | Feder        |
16 |      1 | Gartenbedarf AllesGrün   |    7856 | Silberwiebel |
17 |      1 | Gartenbedarf AllesGrün   |    7863 | Tulpenzwiebel|
18 |      1 | Gartenbedarf AllesGrün   |    9010 | Schaufel     |
19 |      1 | Gartenbedarf AllesGrün   |    9015 | Spaten       |
20 +-----+-----+-----+-----+

```

Altes Problem: Wir sehen nur die Lieferanten, die aktuell Ware liefern. Wir wollen aber alle Lieferanten ausgegeben bekommen:

```

1 mysql> SELECT
2     -> l.lieferant_id, l.firmenname, a.artikel_id, a.bezeichnung
3     -> FROM
4     -> artikel_nm_lieferant nm INNER JOIN artikel a   USING(artikel_id)
5     ->                               RIGHT JOIN lieferant l USING(lieferant_id)
6     -> ORDER BY
7     ->   firmenname;
8 +-----+-----+-----+-----+
9 | lieferant_id | firmenname          | artikel_id | bezeichnung |
10+-----+-----+-----+-----+
11 |      3 | Bürohengst GmbH           |    3001 | Papier (100)  |
12 |      3 | Bürohengst GmbH           |    3005 | Tinte (gold) |
13 |      3 | Bürohengst GmbH           |    3006 | Tinte (rot)  |
14 |      3 | Bürohengst GmbH           |    3007 | Tinte (blau) |
15 |      3 | Bürohengst GmbH           |    3010 | Feder        |
16 |      1 | Gartenbedarf AllesGrün   |    7856 | Silberwiebel |
17 |      1 | Gartenbedarf AllesGrün   |    7863 | Tulpenzwiebel|
18 |      1 | Gartenbedarf AllesGrün   |    9010 | Schaufel     |
19 |      1 | Gartenbedarf AllesGrün   |    9015 | Spaten       |
20 |      2 | Office International        |    NULL  | NULL         |
21 +-----+-----+-----+-----+

```

In Zeile 20 wird jetzt die Firma ausgegeben, welche keinen Artikel beliefert.

Umgekehrt kann der OUTER JOIN dafür verwendet werden, gerade die Datensätze herauszufiltern, für welche keine passenden Paarungen gefunden werden. Wir wollen alle Lieferanten wissen, die keine Ware liefern:

```

1 mysql> SELECT l.firmenname
2     -> FROM
3     -> artikel_nm_lieferant nm INNER JOIN artikel a USING(artikel_id)

```

⁸ In der Datei listing08.sql werden diese angelegt.

```

4      ->          RIGHT JOIN lieferant l USING(lieferant_id)
5      -> WHERE artikel_id IS NULL;
6 +-----+
7 | firmenname      |
8 +-----+
9 | Office International |
10+-----+

```

Durch das `IS NULL` in [Zeile 5](#) werden alle Zeilen aus der Ergebnismenge entfernt, die eine Artikelnummer haben. Übrig bleiben die, für welche keine Artikelnummer gefunden wird.

Ein weiteres Beispiel für diese Verwendung ist, wenn man die Kunden ohne Bestellungen wissen möchte.

```

1 mysql> SELECT k.kunde_id, k.nachname, k.vorname
2      -> FROM
3      -> bestellung b RIGHT JOIN kunde k USING(kunde_id)
4      -> WHERE b.bestellung_id IS NULL;
5 +-----+-----+-----+
6 | kunde_id | nachname   | vorname  |
7 +-----+-----+-----+
8 |      3  | Beutlin     | Bilbo    |
9 |      5  | Earendilionn | Elrond   |
10 |      6 | Eichenschild | Thorin   |
11 |      4 | Telcontar    | Elessar  |
12+-----+-----+-----+

```



Aufgabe 11.17: Welche Kunden haben keine eigene Lieferadresse?

Aufgabe 11.18: Welcher Artikel ist noch nie bestellt worden?



Hinweis: Ist der OUTER JOIN Teil einer Kette von JOINS, muss man darauf achten, dass in der Kette das Ergebnis des OUTER JOINs nicht wieder durch einen INNER JOIN verloren geht.



Aufgabe 11.19: Warum verschwindet hier die Firma Office International?

```

SELECT DISTINCT l.firmenname
  FROM
    artikel_nm_lieferant nm INNER JOIN artikel a USING(artikel_id)
                           RIGHT JOIN lieferant l USING(lieferant_id)
                           INNER JOIN bestellung_position USING(artikel_id)
;

```

Aufgabe 11.20: Unter der Annahme, dass Sie keine Constraints verwenden: Wie kann man mit einem OUTER JOIN verletzte referentielle Integritäten ([siehe Definition 22 auf Seite 33](#)) ermitteln?

Zum Schluss noch ein Beispiel für die FULL OUTER JOIN-Syntax, wie sie in PostgreSQL angewendet werden kann:

```

1 oshop=# SELECT
2 oshop=#   kunde_id, nachname, vorname, bestellung_id, datum
3 oshop=#   FROM

```

```

4 oshop=# bestellung FULL OUTER JOIN kunde USING(kunde_id)
5 oshop=# ;
6
7 kunde_id | nachname | vorname | bestellung_id | datum
8 -----+-----+-----+-----+
9      1 | Gamdschie   | Samweis  |           1 | 2012-03-24 17:41:00
10     2 | Beutlin     | Frodo    |           2 | 2012-03-23 16:11:00
11     5 | Earendillion | Elrond   |           |
12     6 | Eichenschild | Thorin   |           |
13     4 | Telcontar   | Elessar   |           |
14     3 | Beutlin     | Bilbo    |           |
15 (6 rows)

```

Mangels fachlich sinnvollem Beispiel ist dies hier das gleiche Ergebnis wie bei einem RIGHT OUTER JOIN.

■ 11.5 Narzissmus pur: SELF JOIN

Wir wollen den Kunden die Möglichkeit bieten, in einem Forum Beiträge zu formulieren. Mit unseren bisherigen Tabellen ist das nicht möglich. Ein kurzes Brainstorming zeigt uns, dass wir eine Tabelle mit Anmeldedaten und eine Tabelle mit Beiträgen brauchen (siehe ER-Modell in Bild 11.2).

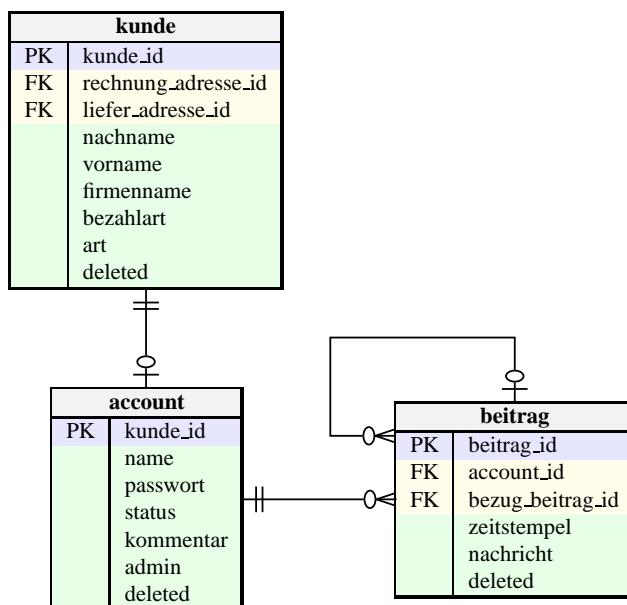


Bild 11.2 ER-Modell: Kundenforum

Zunächst die einfachen Dinge: Jeder Kunde kann einen Account haben, muss er aber nicht. Umgekehrt muss aber ein Account auf einen Kunden zeigen. Wir haben also eine 1:1-Verknüpfung.

Jeder Account kann mehrere Beiträge erfassen, muss er aber nicht. Ein Beitrag hingegen muss auf genau einen Account verweisen: eine *1:n*-Verknüpfung.

In Foren ist es üblich, dass man auf Beiträge antworten kann. Dazu muss ein Beitrag wissen, auf welchen anderen Beitrag er sich bezieht. Deshalb gibt es in der Tabelle `beitrag` den Fremdschlüssel `bezug_beitrag_id`. In diesen Fremdschlüssel trage ich den Wert von `beitrag_id` ein, den die ursprüngliche Nachricht hat.

Aus Sicht der Tabelle ist der Fremdschlüssel `bezug_beitrag_id` ein Fremdschlüssel, der auf eigene Datensätze verweist. Eine solche Verknüpfung nennt man SELF JOIN.



Definition 46: SELF JOIN

Enthält eine Tabelle einen Fremdschlüssel, der Primärschlüsselwerte der eigenen Tabelle enthält, so nennt man diese Art der Verknüpfung *SELF JOIN*.

Bitte beachten Sie, dass ein SELF JOIN auch ein INNER JOIN, OUTER JOIN oder CROSS JOIN sein kann!

In [Bild 11.2 auf der vorherigen Seite](#) können Sie sehen, wie man einen SELF JOIN sofort erkennt. Er verweist eben auf sich selbst.



Aufgabe 11.21: Erstellen Sie zu den beiden Tabellen `account` und `beitrag` passende CREATE TABLE- und ggf. CREATE INDEX-Befehle. Füllen Sie die beiden Tabellen mit passenden Inhalten. Beachten Sie dabei folgende Hinweise:

- `account`: Der Primärschlüssel `kunde_id` hat keinen eigenen Zähler.
- `account`: Der Inhalt der Spalte `name` muss den Schlüsseleigenschaften genügen.
- `account`: Der Status kann zwei Werte haben: `aktiv` und `gesperrt`.
- `account`: Der Kommentar muss einen umfangreichen Text aufnehmen können.
- `account`: Die Spalte `admin` ist vom Typ B00L.
- `beitrag`: Die Spalte `account_id` ist der Fremdschlüssel auf die Spalte `kunde_id` in der Tabelle `account`.
- `beitrag`: Der Selbstbezug soll den Default 1 haben. Wir werden sicherstellen müssen, dass es eine leere Nachricht mit dem Primärschlüsselwert 1 geben wird. Auf diesen werden alle Nachrichten verweisen, die keine Antworten auf eine andere Nachricht sind.
- `beitrag`: Der Nachrichtentext muss genügend Platz für längere Texte haben.
- `beitrag`: Auf eine Thread-Verwaltung wird hier verzichtet.

Nun stehen uns Testdaten zur Verfügung⁹, deren Nachrichtentexte ich aus Platzmangel auf 20 Stellen in der Ausgabe begrenze:

```
1 mysql> SELECT
2      -> kunde_id, name, status, admin
3      -> FROM
4      -> account;
```

⁹ Die entsprechenden Befehle stehen in `listing08.sql`.

```

5 +-----+-----+-----+
6 | kunde_id | name   | status  | admin |
7 +-----+-----+-----+
8 |      1 | admin   | aktiv    |     1 |
9 |      2 | frodo   | aktiv    |     0 |
10 |     3 | bilbo   | aktiv    |     0 |
11 |     5 | elle    | aktiv    |     0 |
12 +-----+-----+-----+
13
14 mysql> SELECT
15     -> beitrag_id, account_id, bezug_beitrag_id, LEFT(nachricht, 20)
16     -> FROM
17     -> beitrag;
18 +-----+-----+-----+
19 | beitrag_id | account_id | bezug_beitrag_id | LEFT(nachricht, 20) |
20 +-----+-----+-----+
21 |      1 |         1 |                 1 | |
22 |      2 |         2 |                 1 | Der Lieferservice is |
23 |      3 |         3 |                 2 | Das finde ich auch. |
24 |      4 |         5 |                 2 | Aber ein wenig lang |
25 |      5 |         2 |                 4 | Finde ich nicht. |
26 |      6 |         5 |                 1 | Angebot könnte besse |
27 +-----+-----+-----+

```

Jetzt kommt der SELF JOIN: Wir wollen die Antworten auf Nachricht 2 wissen:

```

1 mysql> SELECT nachricht
2     -> FROM
3     -> beitrag INNER JOIN beitrag
4     -> ON bezug_beitrag_id = beitrag_id
5     -> WHERE
6     -> beitrag_id = 2;
7 ERROR 1066 (42000): Not unique table/alias: 'beitrag'

```

Die Fehlermeldung in [Zeile 7](#) besagt, dass die Tabelle `beitrag` nicht eindeutig ist. Klar, die Tabelle kommt in der Verknüpfung ja auch zweimal vor. Für SQL ist das ein Problem. Dieses wird besonders in [Zeile 4](#) deutlich. Wenn er die beiden Spalteninhalte vergleichen soll, ist unklar, ob mit `beitrag_id` jetzt die Spalte der linken oder rechten Tabelle `beitrag` gemeint ist.

Spätestens jetzt ist die Vergabe eines Alias kein *nice to have* mehr, sondern ein *must be*. Indem man der Tabelle jeweils einen eindeutigen Alias – links `ant` für `beitrag` in der Rolle einer Antwort und rechts `orig` für `beitrag` in der Rolle der Originalnachricht – gibt und diesen bei der Angabe der Spalten auch verwendet, sind alle Unklarheiten beseitigt.

```

1 mysql> SELECT ant.nachricht 'Antwort'
2     -> FROM
3     -> beitrag ant INNER JOIN beitrag orig
4     -> ON ant.bezug_beitrag_id = orig.beitrag_id
5     -> WHERE
6     -> orig.beitrag_id = 2;
7 +-----+
8 | Antwort          |
9 +-----+
10 | Das finde ich auch. |
11 | Aber ein wenig langsam. |
12 +-----+

```

Es sei bemerkt, dass es keinen eigenen Befehl SELF JOIN gibt. Man verwendet dazu einfach einen INNER JOIN oder eine andere JOIN-Variante, die auf beiden Seiten die gleiche Tabelle stehen hat.

Mithilfe der *common table expression*¹⁰ wird eine virtuelle Ergebnismenge (Tabelle) aufgebaut, die sich auch selbst als Datenquelle verwenden kann. Hier ein Quelltextbeispiel mit WITH RECURSIVE:

```

1  mysql> WITH RECURSIVE ant_auf AS
2  -> (
3  ->   -- nicht rekuriver Teil
4  ->   SELECT *
5  ->   FROM beitrag
6  ->   WHERE bezug_beitrag_id = 2
7  -> UNION ALL
8  ->   -- rekuriver Teil
9  ->   SELECT ant.*
10 ->    FROM
11 ->      beitrag ant INNER JOIN ant_auf orig
12 ->    ON ant.bezug_beitrag_id = orig.beitrag_id
13 -> )
14 ->   SELECT beitrag_id, bezug_beitrag_id, nachricht FROM ant_auf;
15 +-----+-----+-----+
16 | beitrag_id | bezug_beitrag_id | nachricht          |
17 +-----+-----+-----+
18 |          3 |                  2 | Das finde ich auch. |
19 |          4 |                  2 | Aber ein wenig langsam. |
20 |          5 |                  4 | Finde ich nicht. |
21 +-----+-----+-----+

```

Ihnen fällt sicherlich auf, dass nun auch der Beitrag 5 als indirekte Antwort auf Beitrag 2 ausgegeben wird.

■ 11.6 Eine Verknüpfung beschleunigen

Sind die Daten auf mehrere Tabellen verteilt, kann eine Verknüpfung mithilfe von Indizes beschleunigt werden. Ein Primärschlüssel hat automatisch einen passenden Index. Das Gleiche gilt für Fremdschlüssel, wenn sie denn durch FOREIGN KEY ... REFERENCES deklariert sind.

Anders sieht es bei Verknüpfungen aus, die nicht über indizierte Spalten durchgeführt werden. Denken Sie beispielsweise an den Zusammenbau der letzten temporären Tabelle auf Seite 195. Beide verwendeten die Spalte kunde_id, aber diese war in den temporären Tabellen nicht als Primär- oder Fremdschlüssel deklariert. Eine Verknüpfung über diese beiden Spalten kann somit sehr teuer werden.

Grundsätzlich ist aber der Zusammenbau von Daten aus verschiedenen Tabellen teurer als das Auslesen der Daten aus einer Tabelle. Mit temporären Tabellen – wie oben beschrie-

¹⁰ Eine nähere Betrachtung von CTE muss hier leider aus Platzgründen entfallen. Eine gute Einführung finden Sie unter [Tut19].

ben – kann man übliche Verbindungen vorbauen, damit diese nicht jedes Mal neu erstellt werden müssen. Eine weitere Möglichkeit sind redundante Daten.

Redundante Daten sind nach [Definition 13 auf Seite 20](#) Daten, die mehrfach im System abgespeichert sind. Grundsätzlich versucht man, redundante Daten zu vermeiden. Neben dem Speicherplatzverbrauch muss man Daten an vielen Orten aktualisieren, was fehlerträchtig ist.

Aber redundante Daten können auch sinnvoll sein. Wir könnten die Tabelle kunde um die Spalten für die Rechnungs- und Lieferadresse erweitern. Ebenso um zwei Spalten, die mir markieren, ob die beiden Adressen noch aktuell sind.

```

1 ALTER TABLE kunde
2 ADD r_strasse VARCHAR(255),
3 ADD r_ort VARCHAR(255),
4 ADD r_aktuell BOOL NOT NULL DEFAULT TRUE,
5 ADD l_strasse VARCHAR(255),
6 ADD l_ort VARCHAR(255),
7 ADD l_aktuell BOOL NOT NULL DEFAULT TRUE
8 ;

```

Straße und Ort sollen Zusammenbauten der Spalten strasse mit hnr und lkz mit plz mit ort sein¹¹. In periodischen Abständen werden die Daten aus der Adresstabelle in die Kundentabelle kopiert.

```

1 mysql> UPDATE kunde INNER JOIN adresse ON rechnung_adresse_id = adresse_id
2      -> SET
3      ->   r_strasse = CONCAT(strasse, ' ', hnr),
4      ->   r_ort = CONCAT(lkz, '-', plz, ' ', ort),
5      ->   r_aktuell = TRUE;
6
7 mysql> SELECT
8      ->   kunde_id, r_strasse, r_ort, r_aktuell
9      ->   FROM kunde;
10
11 +-----+-----+-----+
12 | kunde_id | r_strasse       | r_ort          | r_aktuell |
13 +-----+-----+-----+
14 |     1 | Beutelhaldenweg 5 | AL-67676 Hobbingen |      1 |
15 |     2 | Beutelhaldenweg 1 | AL-67676 Hobbingen |      1 |
16 |     3 | Beutelhaldenweg 1 | AL-67676 Hobbingen |      1 |
17 |     4 | Auf der Feste 1   | GO-54786 Minas Tirith |      1 |
18 |     5 | Letztes Haus 4   | ER-87567 Bruchtal |      1 |
19 |     6 | NULL             | NULL           |      1 |
20 +-----+-----+-----+

```

Das ist scharf, oder? Der `INNER JOIN` wird gar nicht in einem `SELECT` verwendet, sondern in einem `UPDATE!` Erinnern Sie sich? Das Ergebnis einer Verknüpfung ist wieder eine Tabelle. Deshalb können Sie an vielen Stellen, wo in der SQL-Referenz der Tabellename steht, eine Verknüpfung einsetzen.



Aufgabe 11.22: Überlegen Sie mal, lässt sich beim `UPDATE` eine geschickte `WHERE`-Klausel einbauen?

¹¹ Eine von mir willkürlich gefällte Designentscheidung, um eine bessere Übersicht zu haben

Jetzt wird jedes Mal, wenn sich eine Adresse innerhalb der Periode ändert, die Spalte `r_aktuell` oder `l_aktuell` auf FALSE gesetzt. Dies könnte man beispielsweise mit einem Trigger erreichen¹². Mithilfe eines Events¹³ wird jetzt in periodischen Abständen nachgeschaut, ob sich die Adressdaten geändert haben. Falls ja, werden die redundanten Daten neu aufgebaut. Falls man die Änderung sofort in den redundanten Daten ändern will, kann man dies ebenfalls im Trigger machen.



Aufgabe 11.23: Erweitern Sie das UPDATE so, dass gleichzeitig auch die Lieferadresse gesetzt wird. Vorsicht beim zweiten JOIN und der WHERE-Klausel!

¹² Siehe [Kapitel 22 auf Seite 363](#)

¹³ Siehe [Kapitel 23 auf Seite 371](#)

12

Differenzierte Auswertungen



Auswertungen mithilfe von Aggregatfunktionen erstellen.

- Grundkurs
 - Einfache Statistik mit MIN(), MAX(), SUM(), COUNT() und AVG()
 - Tabellen mit GROUP BY in Gruppen zerlegen
 - Aggregatfunktionen auf Gruppen anwenden
 - Gruppenergebnisse mit HAVING aussortieren
- Vertiefendes
 - Aggregatfunktionen
 - Summenbildung mit WITH ROLLUP
 - Gruppieren nach Ausdrücken
 - Gruppieren nach mehr als einer Spalte
 - Einfluss von Indizes auf Gruppierungen



Die Quelltexte dieses Kapitels stehen in der Datei `Listing09.sql` (siehe [Listing 29.9 auf Seite 488](#), [Listing 29.45 auf Seite 610](#) und [Listing 29.30 auf Seite 558](#)).

■ 12.1 Statistisches mit Aggregatfunktionen

Die Daten, die bisher aus einem SELECT kamen, sind immer Originaldaten gewesen. Soll heißen, es wurden nur Texte oder Zahlen angezeigt, die in den Tabellen so abgelegt waren. Mithilfe von Aggregatfunktionen werden nun Auswertungen über die Daten erstellt. Eine Übersicht der verfügbaren Aggregatfunktionen finden Sie im Anhang [26.2.3 auf Seite 411](#). Der *ausdruck* im Anhang [26.2.3](#) ist meist ein Spaltenname oder eine mathematische Kombination aus Spaltennamen.

Die wichtigsten Aggregatfunktionen sind: MIN(), MAX(), SUM(), COUNT() und AVG(). Mit MIN() und MAX() lassen sich der minimale und maximale Wert einer Liste ermitteln. SUM() addiert die Werte einer Liste auf, und COUNT() zählt die Anzahl von Werten. Mit AVG() wird das arithmetische Mittel einer Werteliste berechnet.

Diese Funktionen lassen sich überall da einbauen, wo man mit Werten arbeitet. Beim SELECT beispielsweise in der Spaltenliste oder beim WHERE in den Vergleichen usw. Wir wollen mal die meisten Aggregatfunktionen an unseren Daten ausprobieren:

Was ist der durchschnittliche Preis unserer Artikel?

Der Artikelpreis steht in der Spalte `einzelpreis`. Da dieser mit der Option NOT NULL erstellt wurde, erwarten wir keine Schwierigkeiten.

```

1  SELECT AVG(einzelpreis) FROM artikel;
2  -----
3  | AVG(einzelpreis) |
4  +-----+
5  |      12.577777778 |
6  +-----+

```

Wie viele Zeilen hat eine Tabelle?

Wir verwenden hier die Tabelle `kunde`, es könnte aber auf diese Art und Weise von jeder beliebigen Tabelle die Anzahl der Zeilen ermittelt werden.

```

1  SELECT COUNT(*) FROM kunde;
2  -----
3  | COUNT(*) |
4  +-----+
5  |      6 |
6  +-----+

```

Wie viele Kunden haben eine eigene Lieferadresse?

Da der Inhalt des Fremdschlüssels `liefer_adresse_id` den Wert NULL hat, wenn keine eigene Lieferadresse erfasst ist, kann über diese Spalte die Anzahl ermittelt werden.

```

1  SELECT COUNT(liefer_adresse_id) FROM kunde;
2  -----
3  | COUNT(liefer_adresse_id) |
4  +-----+
5  |      1 |
6  +-----+

```

Wie viele unterschiedliche Rechnungssadressen gibt es?

Die Tabelle `kunde` hat in der Spalte `rechnung_adresse_id` die Fremdschlüsselwerte zu den Rechnungssadressen abgelegt. Die Anzahl unterschiedlicher Werte liefert mir das gewünschte Ergebnis.

```

1  SELECT COUNT(DISTINCT(rechnung_adresse_id)) FROM kunde;
2  -----
3  | COUNT(DISTINCT(rechnung_adresse_id)) |
4  +-----+
5  |      4 |
6  +-----+

```

Was ist unser teuerster und unser billigster Artikel?

Die Tabelle artikel enthält in der Spalte einzelpreis unsere Preise.

```
1 SELECT MAX(einzelpreis), MIN(einzelpreis) FROM artikel;
2 +-----+-----+
3 | MAX(einzelpreis) | MIN(einzelpreis) |
4 +-----+-----+
5 |      56.260000 |      0.520000 |
6 +-----+-----+
```

Wie viele Einzelartikel sind bestellt worden?

Die Bestellmenge steht in der Tabelle bestellung_position in der Spalte menge.

```
1 SELECT SUM(menge) FROM bestellung_position;
2 +-----+
3 | SUM(menge) |
4 +-----+
5 | 96.000000 |
6 +-----+
```

Wie hoch ist das Bestellvolumen?

Hier werden zwei Tabellen zur Auswertung benötigt: bestellung_position und artikel.
Bilden Sie zuerst den INNER JOIN wie auf [Seite 188](#) beschrieben.

```
1 mysql> SELECT
2      -> bp.menge, a.einzelpreis
3      -> FROM
4      -> bestellung_position bp INNER JOIN artikel a USING(artikel_id);
5 +-----+-----+
6 | menge      | einzelpreis |
7 +-----+-----+
8 | 30.000000 |    0.520000 |
9 | 50.000000 |    3.420000 |
10 | 1.000000 |   20.100000 |
11 | 10.000000 |    0.520000 |
12 | 5.000000 |   15.100000 |
13 +-----+-----+
```

Der Wert einer Position lässt sich nun einfach dadurch ermitteln, dass die beiden Werte menge und einzelpreis miteinander multipliziert werden:

```
1 mysql> SELECT
2      -> bp.menge * a.einzelpreis 'Positionswert'
3      -> FROM
4      -> bestellung_position bp INNER JOIN artikel a USING(artikel_id);
5 +-----+
6 | Positionswert      |
7 +-----+
8 | 15.60000000000000 |
9 | 171.0000000000000 |
10 | 20.10000000000000 |
11 | 5.20000000000000 |
12 | 75.50000000000000 |
13 +-----+
```

Zum Schluss werden die Positionswerte summiert.

```

1  SELECT
2    SUM(bp.menge * a.einzelpreis)
3    FROM
4      bestellung_position bp INNER JOIN artikel a USING(artikel_id);
5 +-----+
6 | SUM(bp.menge * a.einzelpreis) |
7 +-----+
8 |          287.4000000000000 |
9 +-----+

```

Wir haben hier ein Beispiel dafür, dass als Parameter einer Aggregatfunktion nicht nur Spaltennamen, sondern auch Ausdrücke vorkommen können.



Aufgabe 12.1: Hier ein paar Übungsausgaben. Für die Übungsausgaben habe ich in *listing09.sql* noch einen Lagerbestand aufgebaut.

- Ergänzen Sie das ER-Modell in [Bild 3.2 auf Seite 43](#) um die Lagerverwaltung in *listing09.sql*.
- Ermitteln Sie die durchschnittliche Bestellmenge.
- Ermitteln Sie die Anzahl der Artikel mit der Währung USD .
- Ermitteln Sie die Anzahl der Privatkunden.
- Ermitteln Sie die Anzahl der Kunden, die noch keine Bestellung aufgegeben haben.
- Was ist unsere kleinste und was unsere größte Bestellmenge?
- Ermitteln Sie, wie viele *ml* Tinte noch auf Lager sind.
- Ermitteln Sie den Wert des Lagers.

■ 12.2 Tabelle in Gruppen zerlegen

Die Aggregatfunktionen liefern uns bis jetzt ihre Ergebnisse bezogen auf die ganze Tabelle, also *eine* Zahl. Es kommt aber viel häufiger vor, dass man die Auswertung pro Kunde, pro Postleitzahl, pro Bestellung, pro Artikel etc. vornehmen möchte, z.B. Anzahl der Bestellpositionen pro Artikel. Wir brauchen ein Werkzeug, was die Anwendung von Aggregatfunktionen auf Teiltabellen (Gruppen) ermöglicht.



SQL:2016, MySQL/MariaDB, PostgreSQL, T-SQL

```

SELECT [DISTINCT]
  {*|spaltenliste|ausdruck}
  FROM
    from_ausdruck
  [WHERE bedingung]
  [GROUP BY spaltenliste|ausdruck]
  [ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
;

```

Wie viele Positionen pro Bestellungen gibt es?

Da es irgendwas mit *Anzahl* zu tun hat, ist COUNT() unser Mann¹.

```
1 mysql> SELECT COUNT(*) FROM bestellung_position;
2 +-----+
3 | COUNT(*) |
4 +-----+
5 |      5 |
6 +-----+
```

Jetzt wissen wir die Anzahl der Positionen überhaupt. Wir wollen aber wissen, wie viele es pro Bestellung sind. Wir müssen also ein GROUP BY einfügen. Aber was steht in *spaltenliste* hinter dem GROUP BY? Die Spalte, die bestimmt, zu welcher Gruppe die Zeile gehört. In unserem Fall der Fremdschlüssel auf die Tabelle bestellung.

```
1 mysql> SELECT
2     -> bestellung_id Bestellnummer, COUNT(*) 'Anzahl der Positionen'
3     -> FROM
4     -> bestellung_position
5     -> GROUP BY
6     -> bestellung_id;
7 +-----+-----+
8 | Bestellnummer | Anzahl der Positionen |
9 +-----+-----+
10|          1 |            3 |
11|          2 |            2 |
12+-----+-----+
```

Noch mal: Schauen wir uns die erste Version an, so wird die Aggregatfunktion COUNT(*) auf die gesamte Tabelle angewendet. In der zweiten Version weisen wir an, dass die Tabelle in Gruppen zerlegt wird. Als Unterscheidungsmerkmal wird in Zeile 6 die Spalte bestellung_id angegeben. Das bedeutet, dass alle Zeilen, die in dieser Spalte den gleichen Wert stehen haben, zur gleichen Gruppe gehören. Daher gibt es jetzt 2 Gruppen, für jede bestellung_id eine.

Ist eine Tabelle in Gruppen zerlegt worden, so werden die Aggregatfunktionen immer pro Gruppe ausgeführt. In unserem Fall bedeutet das, dass pro Bestellung COUNT(*) die Zeilen in bestellung_position zählt.



Hinweis: In Zeile 2 wird ein Alias mit Leerzeichen verwendet. Deshalb muss um den Alias eine Stringbegrenzung erfolgen. Die *normalen* Hochkomma ' oder Gänsefußchen " können in MySQL/MariaDB nicht verwendet werden, da an dieser Stelle auch Stringkonstanten stehen könnten. Als neues Zeichen wird auch ein Hochkomma ' verwendet, aber das auf der Taste links neben dem Backspace.

Wir wollen wissen, wie oft ein Artikel bestellt wurde

Dabei soll der Artikelname mit ausgegeben werden. Zuerst wenden wir bzgl. der beiden Tabellen bestellung_position und artikel das auf Seite 188 beschriebene Verfahren an, um den Artikelname zu erfahren.

¹ unsere Frau

```

1 mysql> SELECT a.bezeichnung, bp.menge
2      -> FROM
3      -> bestellung_position bp INNER JOIN artikel a USING (artikel_id);
4 +-----+-----+
5 | bezeichnung | menge |
6 +-----+-----+
7 | Silberzwiebel | 30.000000 |
8 | Tulpenzwiebel | 50.000000 |
9 | Spaten | 1.000000 |
10 | Silberzwiebel | 10.000000 |
11 | Schaufel | 5.000000 |
12 +-----+-----+

```

Jetzt heißt es herauszufinden, nach welcher Spalte die Gruppen gebildet werden sollen. Da hilft ein Tipp: Wenn die Aufgabenstellung Formulierungen wie *pro Artikel* oder *für jeden Kunden* enthält, dann sind dies in der Regel die Gruppierungsmerkmale. In unserem Beispiel wird demnach nach der Spalte `artikel_id` gruppiert. Die Aggregatfunktion, um die Anzahl der Artikel zu ermitteln, ist hier `SUM()` und nicht `COUNT()`, da pro Zeile mehr als ein Artikel verkauft wird.

```

1 mysql> SELECT
2      -> a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
3      -> FROM
4      -> bestellung_position bp INNER JOIN artikel a USING (artikel_id)
5      -> GROUP BY
6      -> artikel_id;
7 +-----+-----+
8 | Artikelname | Anzahl bestellter Artikel |
9 +-----+-----+
10 | Silberzwiebel | 40.000000 |
11 | Tulpenzwiebel | 50.000000 |
12 | Schaufel | 5.000000 |
13 | Spaten | 1.000000 |
14 +-----+-----+

```

Jetzt ist es aber so, dass hier die Artikel, die in keiner Bestellung vorkommen, gar nicht aufgeführt werden. Ein `RIGHT OUTER JOIN` könnte helfen:

```

1 mysql> SELECT
2      -> a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
3      -> FROM
4      -> bestellung_position bp RIGHT JOIN artikel a USING (artikel_id)
5      -> GROUP BY
6      -> artikel_id;
7 +-----+-----+
8 | Artikelname | Anzahl bestellter Artikel |
9 +-----+-----+
10 | Papier (100) | NULL |
11 | Tinte (gold) | NULL |
12 | Tinte (rot) | NULL |
13 | Tinte (blau) | NULL |
14 | Feder | NULL |
15 | Silberzwiebel | 40.000000 |
16 | Tulpenzwiebel | 50.000000 |
17 | Schaufel | 5.000000 |
18 | Spaten | 1.000000 |
19 +-----+-----+

```

Wie Sie die Ausgabe NULL in eine schöne Ausgabe verwandeln können, erfahren Sie in [Kapitel 15 auf Seite 261](#). Ach was, ich kann mich nicht beherrschen:

```

1 mysql> SELECT
2     -> a.bezeichnung 'Artikelname',
3     -> CASE
4     -> WHEN SUM(bp.menge) IS NULL THEN 0
5     -> ELSE SUM(bp.menge)
6     -> END AS 'Anzahl bestellter Artikel'
7     -> FROM
8     -> bestellung_position bp RIGHT JOIN artikel a USING (artikel_id)
9     -> GROUP BY
10    -> artikel_id
11    -> ;
12 +-----+-----+
13 | Artikelname | Anzahl bestellter Artikel |
14 +-----+-----+
15 | Papier (100) | 0 |
16 | Tinte (gold) | 0 |
17 | Tinte (rot) | 0 |
18 | Tinte (blau) | 0 |
19 | Feder | 0 |
20 | Silberzwiebel | 40.000000 |
21 | Tulpenzwiebel | 50.000000 |
22 | Schaufel | 5.000000 |
23 | Spaten | 1.000000 |
24 +-----+-----+

```

Mit dem CASE kann man eine Fallunterscheidung auf Werte vornehmen und die Ausgabe danach anpassen. Den Rest erzähle ich Ihnen wirklich erst später ;-).



Hinweis: Im SQL:2016-Standard muss das Gruppierungsmerkmal in der Spaltenliste hinter dem SELECT auftauchen (siehe Zeile 10). PostgreSQL und T-SQL setzen diese Forderung um.

```

1 oshop=# SELECT
2 oshop-# a.bezeichnung Artikelname,
3 oshop-# CASE
4 oshop-# WHEN SUM(bp.menge) IS NULL THEN 0
5 oshop-# ELSE SUM(bp.menge)
6 oshop-# END AS Anzahl_bestellter_Artikel
7 oshop-# FROM
8 oshop-# bestellung_position bp RIGHT JOIN artikel a USING (artikel_id)
9 oshop-# GROUP BY
10 oshop-# a.bezeichnung;

```

Ein schönes Feature ist die Summenbildung bei Aggregatfunktionen mit `WITH ROLLUP`. Dabei wird am Ende der Auswertung eine zusätzliche Zeile eingefügt, welche die summierten Auswertungen enthält. Die Gruppierungsspalte (hier `artikel_id`) bekommt den Wert `NULL` zugewiesen.

```

1 mysql> SELECT
2     -> artikel_id,
3     -> CASE
4     -> WHEN SUM(bp.menge) IS NULL THEN 0
5     -> ELSE SUM(bp.menge)

```

```

6      -> END AS 'Anzahl bestellter Artikel'
7      -> FROM
8      -> bestellung_position bp RIGHT JOIN artikel a USING (artikel_id)
9      -> GROUP BY
10     -> artikel_id WITH ROLLUP
11     -> ;
12 +-----+-----+
13 | artikel_id | Anzahl bestellter Artikel |
14 +-----+-----+
15 |      3001 |          0 |
16 |      3005 |          0 |
17 |      3006 |          0 |
18 |      3007 |          0 |
19 |      3010 |          0 |
20 |      7856 | 40.000000 |
21 |      7863 | 50.000000 |
22 |      9010 | 5.000000 |
23 |      9015 | 1.000000 |
24 |      NULL | 96.000000 | ← Hier steht die Summe!
25 +-----+-----+

```

Werden mehr als ein Gruppierungselement benannt, werden Zwischenzeilen mit Zwischensummen ausgegeben.

■ 12.3 Gruppenergebnisse filtern



SQL:2016, MySQL/MariaDB, PostgreSQL, T-SQL

```

SELECT [DISTINCT]
  {*<|spaltenliste|ausdruck}
  FROM
    from_ausdruck
  [WHERE bedingung]
  [[GROUP BY spaltenliste|ausdruck]
    [HAVING bedingung]]
  [ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
;

```

Wir wollen nur solche Artikel sehen, die mehr als zehn Mal verkauft wurden. Intuitiv wollen wir dazu die WHERE-Klausel verwenden, müssen aber feststellen, dass das nicht geht.

```

1 mysql> SELECT
2     -> a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
3     -> FROM
4     -> bestellung_position bp INNER JOIN artikel a USING (artikel_id)
5     -> WHERE
6     -> SUM(bp.menge) > 10
7     -> GROUP BY
8     -> artikel_id;
9 ERROR 1111 (HY000): Invalid use of group function

```

Die [Definition 35 auf Seite 142](#) sagt aus, dass die Operation auf die Zeilen eingeschränkt wird, für welche die Bedingung der WHERE-Klausel TRUE ergibt. Das Problem ist aber, dass

wir gar nicht die Zeilen beschränken wollen, die in die Berechnungen einfließen, sondern die Ergebnisse, die aus den Berechnungen herausfließen. Dazu wird ein HAVING gebraucht.



Definition 47: HAVING

Durch ein *HAVING* werden die Ergebnisse eines GROUP BY verworfen, für welche die Bedingung nicht TRUE ergibt.

```

1 mysql> SELECT
2     -> a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
3     -> FROM
4     -> bestellung_position bp INNER JOIN artikel a USING (artikel_id)
5     -> GROUP BY
6     -> artikel_id
7     -> HAVING
8     -> SUM(bp.menge) > 10;
9 +-----+-----+
10 | Artikelname | Anzahl bestellter Artikel |
11 +-----+-----+
12 | Silberzwiebel |          40.000000 |
13 | Tulpenzwiebel |          50.000000 |
14 +-----+-----+

```

In [Zeile 7](#) wird der HAVING verwendet. Der Bau der Bedingung kann die gleichen Vergleichsoperatoren (siehe Anhang [26.3.1 auf Seite 414](#)) und Verknüpfungsoperatoren (siehe Anhang [26.3.2 auf Seite 416](#)) verwenden wie das WHERE.

Um den Unterschied zwischen WHERE und HAVING zu verdeutlichen, wird die obige Auswertung um solche Zeilen eingeschränkt, deren Artikelname mit Silber beginnen.

```

1 mysql> SELECT
2     -> a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
3     -> FROM
4     -> bestellung_position bp INNER JOIN artikel a USING (artikel_id)
5     -> WHERE
6     -> a.bezeichnung LIKE 'Silber%'
7     -> GROUP BY
8     -> artikel_id
9     -> HAVING
10    -> SUM(bp.menge) > 10;
11 +-----+-----+
12 | Artikelname | Anzahl bestellter Artikel |
13 +-----+-----+
14 | Silberzwiebel |          40.000000 |
15 +-----+-----+

```

Die WHERE-Klausel lässt nur Artikel zu, die das Präfix Silber haben. Deshalb steht das WHERE auch vor dem GROUP BY. Erst nach diesem Filter werden die restlichen Daten gruppiert und ausgewertet. Die Berechnungsergebnisse selbst werden dann noch mit dem HAVING weiter gefiltert.

■ 12.4 Noch Fragen?

12.4.1 Kann ich nach Ausdrücken gruppieren?

In der syntaktischen Beschreibung von GROUP BY auf Seite 218 steht *spaltenliste*, aber es können auch Ausdrücke verwendet werden.

Beispiel: Nach [Wik19b] steht die erste Ziffer der Bankleitzahl für das Clearinggebiet. Wir wollen wissen, wie viele Bankleitzahlen pro Clearinggebiet in der Tabelle bank sind.

```

1 mysql> SELECT
2      -> SUBSTRING(blz, 1, 1) 'Clearinggebiet', COUNT(*) 'Anzahl'
3      -> FROM
4      -> bank
5      -> GROUP BY
6      -> 'Clearinggebiet'
7      -> ORDER BY 'Anzahl' DESC;
8 +-----+-----+
9 | Clearinggebiet | Anzahl |
10+-----+-----+
11 | 7             | 1249  |
12 | 5             | 1193  |
13 | 6             | 1100  |
14 | 2             | 989   |
15 | 4             | 556   |
16 | 3             | 527   |
17 | 8             | 287   |
18 | 1             | 205   |
19 +-----+-----+

```

Um das Clearinggebiet zu ermitteln, verwende ich in Zeile 2 die Funktion SUBSTRING(). Diese schneidet mir aus einer Zeichenkette einen Abschnitt heraus. Der erste Buchstabe der Zeichenkette hat den Index 1. Als letzter Parameter der Funktion wird die Länge des Abschnitts festgelegt, hier ebenfalls 1.

Dadurch, dass ich den Alias Clearinggebiet vergeben habe, kann ich überall dort, wo der Ausdruck SUBSTRING(blz, 1, 1) gebraucht wird, den Alias verwenden. So auch in Zeile 6 beim GROUP BY.



Aufgabe 12.2: Wie viele Banken gibt es pro Bankengruppe (4. Ziffer der Bankleitzahl)?

12.4.2 Kann ich nach mehr als einer Spalte gruppieren?

In der syntaktischen Beschreibung von GROUP BY auf Seite 218 steht hinter dem GROUP BY das Wort *spaltenliste*. Wir können Gruppen also auch über mehrere Elemente definieren. Bisher waren es einfache Spalten wie die artikel_id.

Beispiel: Wir wollen die Anzahl der Bestellungen pro Jahr und Monat wissen²:

² Ich habe dazu in listing09.sql die Bestellungen um Daten erweitert.

```

1 mysql> SELECT
2     ->   CONCAT(YEAR(datum), '/', MONTHNAME(datum)) 'Monat', COUNT(*) 'Anzahl'
3     ->   FROM bestellung
4     ->   GROUP BY
5     ->   YEAR(datum), MONTH(datum)
6     ->   ORDER BY
7     ->   YEAR(datum) DESC, MONTH(datum) DESC;
8 +-----+-----+
9 | Monat      | Anzahl |
10+-----+-----+
11 | 2012/April |      1 |
12 | 2012/March |      2 |
13 | 2011/January|      3 |
14 +-----+-----+

```

Der entscheidende Teil ist die [Zeile 5](#). Die Gruppierung wird zuerst nach dem Jahr vorgenommen, anschließend nach dem Monat. Die Funktion YEAR() liefert mir aus einer Datumsangabe die vierstellige Darstellung des Jahrs als Zahl; MONTH() liefert die ein- oder zweistellige Darstellung des Monats als Zahl. Beide Funktionen verweise ich in [Zeile 7](#), um die Ausgabe nach dem Datum sinnvoll zu ordnen. Eine Übersicht mit vielen guten Beispiel finden Sie unter [\[MyS18c\]](#).

Der Optik wegen habe ich in [Zeile 2](#) die Angaben ein wenig komprimiert. Mit der Funktion CONCAT() werden Zeichenketten zu einer neuen Zeichenkette verklebt. Hier wird die Jahresangabe durch einen Slash mit dem Monatsnamen (MONTHNAME()) zu einer Ausgabe verbunden. Der Inhalt der Spalte Monat enthält das Resultat.



Aufgabe 12.3: Wie viele Banken gibt es pro Clearinggebiet und Bankengruppe
(4. Stelle der Bankleitzahl)?

12.4.3 Wie kann ich GROUP BY beschleunigen?

Bei der Ausführung eines GROUP BY muss für jeden Datensatz entschieden werden, zu welcher Gruppe er gehört. Dazu müssen die Spalten und Ausdrücke ausgewertet werden. Wie bei der Sortierung (siehe Kapitel [10.3](#) ab [Seite 163](#)) können Indizes diesen Vorgang erheblich beschleunigen.

Umgekehrt sind GROUP BY-Operationen, die keine Indexunterstützung haben, recht teuer. Besonders, wenn die Gruppierung über Ausdrücke erfolgt, muss der Ausdruck ausgewertet werden, was oft mit erheblicher Rechenleistung verbunden ist, da die Tabelle sequenziell durchlaufen werden muss. Schauen wir uns beispielsweise diesen Befehl an:

```

1 mysql> EXPLAIN
2     -> SELECT
3     -> SUBSTRING(blz, 1, 1) 'Clearinggebiet', COUNT(*) 'Anzahl'
4     -> FROM
5     -> bank
6     -> GROUP BY
7     -> 'Clearinggebiet'
8     -> ORDER BY
9     -> 'Anzahl' DESC\G
10 **** 1. row ****
11 id: 1

```

```

12      select_type: SIMPLE
13          table: bank
14      partitions: NULL
15          type: index
16  possible_keys: idx_bank_blzbankname
17          key: idx_bank_blzbankname
18      key_len: 803
19          ref: NULL
20      rows: 6067
21  filtered: 100.00
22      Extra: Using index; Using temporary; Using filesort

```

Unter der Überschrift [Extra](#) in [Zeile 22](#) wird angegeben, wie der Befehl ausgeführt wird. Da ich auf die Bankleitzahl einen Index gesetzt hatte, wird dieser verwendet. Nun verwende ich ein Gruppierungsmerkmal, welches zugegeben sinnlos ist, aber keinen Index verwenden kann:

```

1  mysql> EXPLAIN
2      -> SELECT
3      -> CONCAT(lkz, SUBSTRING(blz, 6, 3)) 'Sinnlos', COUNT(*) 'Anzahl'
4      -> FROM
5      -> bank
6      -> GROUP BY
7      -> 'Sinnlos'
8      -> ORDER BY
9      -> 'Sinnlos'\G
10 **** 1. row ****
11      id: 1
12      select_type: SIMPLE
13          table: bank
14      partitions: NULL
15          type: ALL
16  possible_keys: NULL
17          key: NULL
18      key_len: NULL
19          ref: NULL
20      rows: 6067
21  filtered: 100.00
22      Extra: Using temporary; Using filesort

```

Hier teilt mir MySQL mit, dass es eine temporäre Tabelle aufbauen muss und deren Inhalt mit Filesort sortiert. Keine Rede mehr davon, dass irgendwelche Indizes verwendet werden könnten.

Werden Gruppierungen nicht über Indizes unterstützt und Sie erwarten einen häufigen Zugriff auf nicht kleine Mengen, so sollten – falls möglich – entsprechende Indizes eingerichtet werden. Die Argumente für oder gegen das Anlegen von Indizes sind in [Tabelle 6.1 auf Seite 102](#) zusammengefasst.

Falls die Gruppierungen nicht permanent gebraucht werden und zum Zeitpunkt der Auswertung nicht oder nur mit wenigen Änderungen der Daten zu rechnen ist, bietet sich das Anlegen einer (temporären) Tabelle an.

12.4.4 Parallele Bearbeitung – unterschiedliche Ergebnisse?

Bei nicht transaktionsfähigen Engines wie MyISAM wird die Anzahl der Zeilen in den Infos über die Tabelle vom System mitgepflegt. Es müssen also nicht alle Zeilen gezählt werden, um die Anzahl zu ermitteln. Bei transaktionsfähigen Engines wie InnoDB können in offenen Transaktionen unterschiedlich viele Zeilen der Tabelle parallel vorhanden sein (Näheres siehe [Kapitel 19 auf Seite 319](#)).

Die Anzahl der Zeilen muss daher aktiv ermittelt werden und kann in jeder Sitzung unterschiedlich sein, weil jede Sitzung Zeilen hinzugefügt oder gelöscht haben kann, die in anderen Sitzungen noch nicht sichtbar sind.

■ 12.5 Haben Sie keine Aufgaben für mich?



Aufgabe 12.4: Klar, hab' ich:

- a) Ermitteln Sie pro Bankleitzahl die Anzahl der Banknamen. Sortieren Sie das Ergebnis nach Bankleitzahl.
- b) Ermitteln Sie pro Adresse die Anzahl der Verwendungen als Rechnungsanschrift. Die nicht verwendeten Adressen sollen auch angezeigt werden. Diese haben die Anzahl 0. Sortieren Sie nach der Anzahl absteigend.
- c) Geben Sie pro Warengruppe die Anzahl der Artikel aus. Es sollen auch die Warengruppen angezeigt werden, denen keine Artikel angehören.
- d) Geben Sie pro Lieferant die Anzahl der gelieferten Artikel aus. Lieferanten, die keine Artikel liefern, sollen ebenfalls angezeigt werden.
- e) Geben Sie den Lagerbestand pro Warengruppe aus. Sortieren Sie die Ausgabe nach Lagerbestand absteigend.
- f) Geben Sie die Kundennamen aus, die mehr als eine Bestellung abgegeben haben.
- g) Geben Sie für jeden Kundennamen die durchschnittliche Anzahl der Bestellungen aus.
- h) Geben Sie pro Kunde die durchschnittliche Menge an bestellten Artikeln aus. Die Ausgabe soll nach dem Durchschnitt aufsteigend sortiert werden.
 - i) Geben Sie den Wert der teuersten Einzelposition aus.
 - j) Geben Sie den durchschnittlichen Wert einer Einzelposition aus.
- k) Ermitteln Sie, welche Artikel bestellt sind, deren Mindestmenge im Lager unterschritten ist.

13

Auswertungen mit Unterabfragen



Und wenn Du denkst, es geht nicht mehr, dann kommt ein SUBSELECT daher.

- Vertiefendes
 - Warum eine Unterabfrage?
 - Korrelierende und nicht korrelierende Unterabfrage
 - Skalare Unterabfrage
 - Listenunterabfrage
 - Tabellenunterabfrage
 - IN(), ALL() und ANY()
 - EXISTS
 - Ablaufplan einer Unterabfrage



Die Quelltexte dieses Kapitels stehen in der Datei `listing10.sql` (siehe [Listing 29.10 auf Seite 491](#), [Listing 29.46 auf Seite 613](#) und [Listing 29.31 auf Seite 562](#)).

Sie benötigen ebenfalls die Datei `kunden01.csv`.

Unterabfragen gehören definitiv nicht mehr zu einem Grundkurs. Sie sind das Tor zu richtig schweren Auswertungen, da es jetzt möglich ist, in *einer* Abfrage *beliebig viele* Zwischenabfragen durchzuführen und weiterzuverwerten.

■ 13.1 Das Problem und die Lösung

Betrachten wir folgende Aufgabenstellung: Wir wollen den Umsatz pro Kunde wissen.

Schritt 1: Die Umsätze müssen sich aus den Bestellungen ergeben. Also ermitteln wir zuerst den Umsatz pro Bestellung. Dazu wird ein INNER JOIN zu der Tabelle `artikel` gebraucht, da dort der Preis abgelegt ist.

```
1 mysql> SELECT
2      ->     bp.bestellung_id, SUM(bp.menge * a.einzelpreis) 'Bestellwert'
```

```

3      -> FROM
4      -> bestellung_position bp INNER JOIN artikel a USING(artikel_id)
5      -> GROUP BY
6      -> bp.bestellung_id
7      -> ;
8 +-----+-----+
9 | bestellung_id | Bestellwert      |
10+-----+-----+
11 |          1 | 206.7000000000000 |
12 |          2 | 80.7000000000000 |
13 |          3 | 39.4000000000000 |
14 |          4 | 26.4600000000000 |
15 |          5 | 488.5500000000000 |
16 +-----+-----+

```

Schritt 2: Den Bestellungen den Kunden zuordnen. Dazu müssen Verknüpfungen zwischen den Tabellen `bestellung` und `kunde` geknüpft werden:

```

1 mysql> SELECT
2     -> k.nachname, k.vorname, SUM(bp.menge * a.einzelpreis) 'Bestellwert'
3     -> FROM
4     -> bestellung_position bp INNER JOIN artikel a    USING(artikel_id)
5     ->                      INNER JOIN bestellung b USING(bestellung_id)
6     ->                      INNER JOIN kunde k    USING(kunde_id)
7     -> GROUP BY
8     -> bp.bestellung_id
9     -> ;
10+-----+-----+-----+
11 | nachname | vorname | Bestellwert      |
12+-----+-----+-----+
13 | Gamdschie | Samweis | 206.7000000000000 |
14 | Beutlin   | Frodo   | 80.7000000000000 |
15 | Gamdschie | Samweis | 39.4000000000000 |
16 | Gamdschie | Samweis | 26.4600000000000 |
17 | Gamdschie | Samweis | 488.5500000000000 |
18 +-----+-----+-----+

```

Schritt 3: Wir müssen wiederum eine Gruppierung vornehmen, und zwar pro Kunde. Und für jeden Kunden die Aggregatfunktion `SUM(Bestellwert)` ausführen. Aber wie soll das hier funktionieren?

Eine Möglichkeit wären die schon weiter oben (siehe Seite 5.3.8) vorgestellten temporären Tabellen.

```

1 mysql> CREATE TEMPORARY TABLE tmp_bestellwert
2     -> SELECT
3     -> k.nachname, k.vorname, SUM(bp.menge * a.einzelpreis) 'bestellwert'
4     -> FROM
5     -> bestellung_position bp INNER JOIN artikel a USING(artikel_id)
6     ->                      INNER JOIN bestellung b USING(bestellung_id)
7     ->                      INNER JOIN kunde k USING(kunde_id)
8     -> GROUP BY
9     -> bp.bestellung_id
10    -> ;
11 Query OK, 5 rows affected (0.07 sec)
12 Records: 5  Duplicates: 0  Warnings: 0
13
14 mysql> SELECT * FROM tmp_bestellwert;
15 +-----+-----+-----+
16 | nachname | vorname | bestellwert      |

```

```

17 +-----+-----+-----+
18 | Gamdschie | Samweis | 206.7000000000000 |
19 | Beutlin   | Frodo   | 80.7000000000000 |
20 | Gamdschie | Samweis | 39.4000000000000 |
21 | Gamdschie | Samweis | 26.4600000000000 |
22 | Gamdschie | Samweis | 488.5500000000000 |
23 +-----+-----+-----+
24 5 rows in set (0.00 sec)
25
26 mysql> SELECT
27     -> nachname, vorname, SUM(bestellwert) 'Umsatz'
28     -> FROM tmp_bestellwert
29     -> GROUP BY nachname, vorname
30     -> ;
31 +-----+-----+-----+
32 | nachname | vorname | Umsatz      |
33 +-----+-----+-----+
34 | Beutlin  | Frodo   | 80.7000000000000 |
35 | Gamdschie | Samweis | 761.1100000000000 |
36 +-----+-----+-----+

```

Natürlich hat diese Lösung einen gewissen Charme. Besonders der Aspekt, dass man die Bestellwertdaten ggf. mehrfach auswerten kann. Aber es geht auch anders. Man kann die Abfrage von [Zeile 2 bis 9](#) in [Zeile 28](#) als Unterabfrage einbauen.

```

1 mysql> SELECT
2     -> bw.nachname, bw.vorname, SUM(bw.bestellwert) 'Umsatz'
3     -> FROM
4     -> (
5     ->     SELECT
6     ->         k.nachname, k.vorname, SUM(bp.menge * a.einzelpreis) 'bestellwert'
7     ->         FROM
8     ->             bestellung_position bp INNER JOIN artikel a USING(artikel_id)
9     ->                         INNER JOIN bestellung b USING(bestellung_id)
10    ->                         INNER JOIN kunde k USING(kunde_id)
11    ->         GROUP BY
12    ->             bp.bestellung_id
13    ->     ) AS 'bw'
14    ->     GROUP BY nachname, vorname
15    -> ;
16 +-----+-----+-----+
17 | nachname | vorname | Umsatz      |
18 +-----+-----+-----+
19 | Beutlin  | Frodo   | 80.7000000000000 |
20 | Gamdschie | Samweis | 761.1100000000000 |
21 +-----+-----+-----+

```

Die [Zeilen 4](#) bis [13](#) stellen eine Unterabfrage dar. In der Regel werden diese durch Klammerung besonders kenntlich gemacht; meist ist die Klammerung sogar aus syntaktischen Gründen zwingend. In [Zeile 13](#) haben wir dem Ergebnis noch einen Namen verpasst: *bw*. Dies erlaubt es uns, gezielt auf die Spalten der *neuen* Tabelle zuzugreifen, und verhindert Fehlermeldungen über mehrdeutige Spalten- oder Tabellennamen.



Definition 48: Unterabfrage

Kommt innerhalb eines SELECT, UPDATE, INSERT oder DELETE ein weiterer SELECT oder JOIN vor, so nennt man diesen eine *Unterabfrage* oder auch *SUBSELECT*.

Unterabfragen können wiederum Unterabfragen enthalten.

Die Anweisung, die die Unterabfrage enthält, wird *Hauptanweisung* oder *Oberanweisung* genannt.

Warum nennt man die *Oberanweisung* nicht *Oberabfrage*? Weil es auch ein INSERT, UPDATE oder DELETE sein kann, der sich einer Unterabfrage bedient. Ein Beispiel für eine solche Unterabfrage haben wir schon auf Seite 209 kennengelernt. Hier wurde eine Unterabfrage innerhalb eines UPDATES verwendet.

Wir wollen uns nun mit verschiedenen Typen von Unterabfragen beschäftigen. Diese unterscheiden sich hinsichtlich der Komplexität und der Verflechtung mit der Oberanweisung.

■ 13.2 Nicht korrelierende Unterabfrage



Definition 49: Korrelierende Unterabfrage

Kann eine Unterabfrage für sich alleine ausgeführt werden, so handelt es sich um eine *nicht korrelierende Unterabfrage*. Benötigt die Unterabfrage Ausdrücke oder Spalten der Oberanweisung, so handelt es sich um eine *korrelierende Unterabfrage*.

13.2.1 Skalarunterabfrage

13.2.1.1 Beispiel 1: Banken mit höchster BLZ

Fangen wir mit einem einfachen Beispiel an. Aus dem Import der Bankdaten wissen wir, dass es zu jeder Bankleitzahl mehrere Banknamen geben kann. Wir wollen die Banknamen der Bank wissen, die die höchste Bankleitzahl hat.

Schritt 1 – Unterabfrage ermitteln: Die Unterabfrage soll mir die höchste Bankleitzahl liefern.

```
1 mysql> SELECT MAX(blz) FROM bank;
2 +-----+
3 | MAX(blz) |
4 +-----+
5 | 87096214 |
6 +-----+
```

Schritt 2 – Bau der Oberanweisung ohne Unterabfrage: Die Oberanweisung will mir abhängig von einer WHERE-Klausel Banknamen ausgeben.

```
1 mysql> SELECT bankname FROM bank WHERE blz = '37010050';
2 +-----+
3 | bankname |
4 +-----+
5 | Postbank |
6 +-----+
```

Schritt 3 – Einbau der Unterabfrage in die Oberanweisung: Jetzt wird im Vergleich die Konstante in [Zeile 1](#) geklammert und durch die Unterabfrage ersetzt.

```

1 mysql> SELECT bankname
2      ->   FROM bank
3      -> WHERE blz =
4      -> (
5      ->   SELECT MAX(blz)
6      ->     FROM bank
7      -> )
8      -> ;
9 +-----+
10 | bankname           |
11 +-----+
12 | Volksbank Chemnitz |
13 | Volksbank Chemnitz (Gf P2) |
14 +-----+

```



Aufgabe 13.1: Führen Sie mal einen EXPLAIN aus und interpretieren Sie die Ausgabe.

13.2.1.2 Beispiel 2: Überdurchschnittlich teure Artikel

Wir wollen alle Artikel ermitteln, die einen überdurchschnittlichen Einzelpreis haben.

Schritt 1 – Unterabfrage ermitteln: Wir brauchen den durchschnittlichen Einzelpreis, um damit vergleichen zu können. Diesen können wir einfach ermitteln:

```

1 mysql> SELECT
2      ->   AVG(einzelpreis) AS 'durchschnittspreis'
3      ->   FROM artikel;
4 +-----+
5 | durchschnittspreis |
6 +-----+
7 |      12.577777778 |
8 +-----+

```

Da wir diesen SELECT ausführen können, handelt es sich nach [Definition 49 auf der vorherigen Seite](#) um eine nicht korrelierende Abfrage. Unsere Unterabfrage liefert eine sehr einfache Tabelle zurück, nämlich eine mit einer Zeile und einer Spalte. Eine solche Unterabfrage hat einen schönen Namen:



Definition 50: Skalarunterabfrage

Liefert die Unterabfrage eine Tabelle mit nur einer Zeile und nur einer Spalte – also einen einzigen Wert –, so nennt man diese *Skalarunterabfrage*.

Schritt 2 – Bau der Oberanweisung ohne Unterabfrage: Wie muss denn die Oberanweisung aussehen? Letztlich ein einfaches SELECT mit einer WHERE-Klausel. Als Platzhalter für die Unterabfrage verwende ich hier eine willkürliche Zahl (siehe [Zeile 4](#)).

```

1 SELECT *
2   FROM artikel
3  WHERE
4    einzelpreis > 30;

```

Schritt 3 – Einbau der Unterabfrage in die Oberanweisung: In der Zeile 4 steht eine 30 als Platzhalter für die Unterabfrage. Zuerst wird um die 30 ein Klammerpaar gesetzt und dann der SELECT aus Schritt 1 dort hineinkopiert. Etwas verschönert sieht das Ergebnis so aus:

```

1  mysql> SELECT *
2    ->   FROM artikel
3    ->   WHERE
4    ->     einzelpreis > (
5    ->       SELECT
6    ->         AVG(einzelpreis) AS 'durchschnittspreis'
7    ->         FROM artikel
8    ->       )
9    -> ;
10 +-----+-----+-----+-----+
11 | artikel_id | bezeichnung | einzelpreis | waehrung | deleted |
12 +-----+-----+-----+-----+
13 |      3005 | Tinte (gold) |  56.260000 | EUR      |      0 |
14 |      9010 | Schaufel      |  15.100000 | USD      |      0 |
15 |      9015 | Spaten        |  20.100000 | EUR      |      0 |
16 +-----+-----+-----+-----+

```

Der Vorteil gegenüber temporären Tabellen liegt auf der Hand. Wird durch ein INSERT, DELETE oder UPDATE der Durchschnittswert der Preise verändert, so muss die temporäre Tabelle komplett neu gebildet werden. Bei Unterabfragen wird erst zum Zeitpunkt der Ausführung die Unterabfrage ausgeführt und enthält damit automatisch¹ den aktuellen Wert.

13.2.1.3 Beispiel 3: Überdurchschnittlich wertvolle Bestellungen

Auch Skalarunterabfragen können kompliziert werden: wenn wir beispielsweise wissen wollen, welche Bestellungen einen überdurchschnittlichen Auftragswert haben.

Schritt 1 – Unterabfrage ermitteln: Zuerst müssen wir den Auftragswert ermitteln. Das haben wir schon auf Seite 225 gemacht.

```

1  SELECT
2    bp.bestellung_id, SUM(bp.menge * a.einzelpreis) 'bestellwert'
3    FROM
4    bestellung_position bp INNER JOIN artikel a USING(artikel_id)
5    GROUP BY
6    bp.bestellung_id
7  ;

```

Davon müssen wir jetzt den Durchschnitt ermitteln:

```

1  mysql> SELECT AVG(bw.bestellwert)
2    ->   FROM (
3    ->     SELECT
4    ->       bp.bestellung_id, SUM(bp.menge * a.einzelpreis) 'bestellwert'
5    ->       FROM
6    ->         bestellung_position bp INNER JOIN artikel a USING(artikel_id)
7    ->       GROUP BY
8    ->         bp.bestellung_id
9    ->     ) AS 'bw'
10   -> ;

```

¹ Hier sind Nebenläufigkeiten bei Transaktionen zu berücksichtigen. Davon später in Kapitel 19 auf Seite 319 mehr.

```

11 +-----+
12 | AVG(bw.bestellwert) |
13 +-----+
14 | 168.362000000000000000 |
15 +-----+

```

Jetzt haben wir eine Abfrage, die uns den durchschnittlichen Bestellwert angibt. Falls Sie den Alias bw in Zeile 9 vergessen haben sollten, bekommen Sie folgende schöne Fehlermeldung:

```
1 ERROR 1248 (42000): Every derived table must have its own alias
```

Warum schön? Weil sie noch mal verdeutlicht, dass das Ergebnis eines SELECT immer eine Tabelle ist.

Schritt 2 – Bau der Oberanweisung ohne Unterabfrage: Ich baue zuerst eine Version mit einem Platzhalter:

```

1 SELECT
2   bp1.bestellung_id, SUM(bp1.menge * a1.einzelpreis) 'bestellwert1'
3   FROM
4     bestellung_position bp1 INNER JOIN artikel a1 USING(artikel_id)
5   GROUP BY
6   bp1.bestellung_id
7   HAVING 'bestellwert1' > 100;

```

Da wir das Ergebnis eines GROUP BY einschränken wollen, muss in Zeile 7 das HAVING verwendet werden. In derselben Zeile steht auch der Platzhalter, der gleich umgebaut wird. Die Nummerierung mit a1 usw. habe ich nur für später eingebaut (siehe unten).

Schritt 3 – Einbau der Unterabfrage in die Oberanweisung: Anstelle der Konstante 100 wird jetzt in Klammern die Unterabfrage eingesetzt.

```

1 mysql> SELECT
2   ->   bp1.bestellung_id, SUM(bp1.menge * a1.einzelpreis) 'bwert1'
3   ->   FROM
4   ->     bestellung_position bp1 INNER JOIN artikel a1 USING(artikel_id)
5   ->   GROUP BY bp1.bestellung_id
6   ->   HAVING
7   ->     'bwert1' >
8   ->   (
9   ->     SELECT AVG(bw.bwert2)
10  ->       FROM
11  ->       (
12  ->         SELECT
13  ->           bp2.bestellung_id, SUM(bp2.menge * a2.einzelpreis) 'bwert2'
14  ->           FROM
15  ->             bestellung_position bp2 INNER JOIN artikel a2
16  ->               USING(artikel_id)
17  ->             GROUP BY bp2.bestellung_id
18  ->           ) AS 'bw'
19  ->   )
20  -> ;
21 +-----+-----+
22 | bestellung_id | bwert1      |
23 +-----+-----+
24 |          1 | 206.700000000000 |
25 |          5 | 488.550000000000 |
26 +-----+-----+

```

13.2.2 Listenunterabfrage

13.2.2.1 Beispiel 1: IN()

Es gibt nicht nur Skalarunterabfragen. Sie könnten auch *mehrere* Zeilen in *einer* Spalte zurückliefern.



Definition 51: Listenunterabfragen

Liefert die Unterabfrage eine Tabelle mit mehreren Zeilen und nur einer Spalte, so bezeichne ich dies als *Listenunterabfrage*.

Das Ergebnis kann dann natürlich nicht einfach mit dem Gleichheitsoperator in Bedingungen verwendet werden. Vielmehr brauchen wir Vergleichsoperatoren, die mit Mengen arbeiten. Ich habe dabei die Operatoren IN, ALL und ANY im Blick².

Wir wollen alle Rechnungen³, für die es auch eine Bestellung gibt.



Aufgabe 13.2: Erweitern Sie das ER-Modell in [Bild 3.2 auf Seite 43](#) um das Rechnungswesen.

Schritt 1 – Unterabfrage ermitteln: Die Unterabfrage muss mir die Primärschlüsselwerte der Bestellungen ermitteln:

```
1 mysql> SELECT bestellung_id FROM bestellung;
2 +-----+
3 | bestellung_id |
4 +-----+
5 |          1 |
6 |          3 |
7 |          4 |
8 |          5 |
9 |          2 |
10 |         6 |
11 +-----+
```

Schritt 2 – Bau der Oberanweisung ohne Unterabfrage: Die Oberanweisung soll die Primärschlüsselwerte von rechnung ausgeben, die in einer IN-Liste enthalten sind.

```
1 SELECT rechnung_id
2   FROM rechnung
3 WHERE bestellung_id IN (1, 2, 3)
4 ;
```

Schritt 3 – Einbau der Unterabfrage in die Oberanweisung: Die Liste im IN() ist oben eine Liste von Konstanten. Diese soll nun durch die Unterabfrage ersetzt werden:

```
1 mysql> SELECT rechnung_id
2      ->   FROM rechnung
3      -> WHERE bestellung_id IN
4      -> (
5      ->   SELECT bestellung_id FROM bestellung
```

² In [Abschnitt 26.3.1 auf Seite 414](#) finden Sie eine entsprechende Übersicht.

³ In [listing10.sql](#) wird der Datenbestand entsprechend erweitert.

```

6      ->  )
7      -> ;
8 +-----+
9 | rechnung_id |
10+-----+
11|       1 |
12|       2 |
13|       3 |
14|       4 |
15|       5 |
16|       6 |
17+-----+

```



Aufgabe 13.3: Bauen Sie das Beispiel so um, dass Sie gerade die Rechnungen ermitteln, zu denen es keine Bestellungen gibt.

13.2.2.2 Beispiel 2: ALL()

Wir wollen wissen, ob der Gartenbedarf unsere teuersten Artikel enthält. Anders: Welche Artikel sind teurer als die des Gartenbedarfs? Das IN() hilft uns hier nicht weiter. Was wir brauchen, ist ein Prädikat, welches uns mitteilt, ob ein Wert kleiner *oder* gleich *oder* größer als alle Elemente einer Liste ist. Das macht der ALL().

Dem ALL() geht immer ein Vergleichsoperator (siehe [Abschnitt 26.3.1 auf Seite 414](#)) voran: `menge <= ALL([...])`; dieser kann mit NOT verbunden werden.

Schritt 1 – Unterabfrage ermitteln: Die Unterabfrage ermittelt uns alle Preise der Gartenbedarfsartikel:

```

1 mysql> SELECT a.einzelpreis
2      -> FROM
3      -> artikel_nm_warengruppe INNER JOIN artikel a USING(artikel_id)
4      ->                               INNER JOIN warengruppe w USING(warengruppe_id)
5      -> WHERE
6      -> w.bezeichnung = 'Gartenbedarf';
7 +-----+
8 | einzelpreis |
9 +-----+
10|   0.520000 |
11|   3.420000 |
12|  15.100000 |
13|  20.100000 |
14+-----+

```

Schritt 2 – Bau der Oberanweisung ohne Unterabfrage: Die Oberanweisung muss alle Artikelpreise ermitteln:

```

1 SELECT
2   a.bezeichnung, a.einzelpreis
3   FROM
4     artikel a
5   WHERE
6     a.einzelpreis > ALL(SELECT 10)
7 ;

```

Leider kann ich in [Zeile 6](#) nicht so etwas schreiben wie (100, 200, 300), da hier explizit eine Unterabfrage erwartet wird.

Schritt 3 – Einbau der Unterabfrage in die Oberanweisung: Die Unterabfrage wird jetzt zwischen das Klammerpaar des ALL() eingefügt.

```

1 mysql> SELECT
2     ->     a.bezeichnung, a.einzelpreis
3     ->     FROM
4     ->     artikel a
5     ->     WHERE
6     ->     a.einzelpreis > ALL
7     ->     (
8     ->         SELECT a.einzelpreis
9     ->         FROM
10    ->             artikel_nm_warengruppe INNER JOIN artikel a USING(artikel_id)
11    ->                         INNER JOIN warengruppe w USING(warengruppe_id)
12    ->             WHERE
13    ->                 w.bezeichnung = 'Gartenbedarf'
14    ->         )
15    -> ;
16 +-----+-----+
17 | bezeichnung | einzelpreis |
18 +-----+-----+
19 | Tinte (gold) |      56.260000 |
20 +-----+-----+

```

13.2.2.3 Beispiel 3: ALL()

Die Beutlins haben einige Rechnungen. Es wäre doch interessant zu wissen, ob es irgend-einen Kunden gibt, der mehr Umsatz gemacht hat als die Beutlins.

Schritt 1 – Unterabfrage ermitteln: Die Unterabfrage muss uns die Summe aller Rechnungssummen der Beutlins liefern. Ermitteln wir zuerst die Rechnungssummen:

```

1 mysql> SELECT
2     ->     rechnung_id, kunde_id, SUM(einzelpreis*menge) AS 'umsatz'
3     ->     FROM
4     ->     rechnung_position INNER JOIN artikel  USING(artikel_id)
5     ->                         INNER JOIN rechnung USING(rechnung_id)
6     ->     GROUP BY
7     ->     rechnung_id
8     -> ;
9 +-----+-----+-----+
10 | rechnung_id | kunde_id | umsatz          |
11 +-----+-----+-----+
12 |           1 |       1 | 206.700000000000 |
13 |           2 |       2 | 80.700000000000 |
14 |           3 |       1 | 39.400000000000 |
15 |           4 |       1 | 26.460000000000 |
16 |           5 |       1 | 488.550000000000 |
17 |           7 |       3 | 67.860000000000 |
18 |           8 |       3 | 47.990000000000 |
19 |          10 |       5 | 87.350000000000 |
20 |          11 |       5 | 1125.200000000000 |
21 +-----+-----+-----+

```

Die Aufnahme der kunde_id ([Zeile 2](#)) wäre hier noch nicht nötig. Da ich die Kundennummer aber später noch brauche, muss ich zur Rechnung auch die Kundennummer ermitteln. Dazu erfolgt ein INNER JOIN auf die Tabelle rechnung ([Zeile 5](#)).

Als Nächstes müssen diese Rechnungssummen pro Kunde zu einer Umsatzsumme addiert werden.

```

1 mysql> SELECT SUM(umsatz) AS 'umsatzsumme'
2     -> FROM
3     -> (
4     ->   SELECT
5     ->     rechnung_id, kunde_id, SUM(einzelpreis*menge) AS 'umsatz'
6     ->   FROM
7     ->     rechnung_position INNER JOIN artikel USING(artikel_id)
8     ->           INNER JOIN rechnung USING(rechnung_id)
9     ->   GROUP BY
10    ->     rechnung_id
11    -> ) AS 'ksum'
12    -> GROUP BY
13    ->   kunde_id
14    -> ;
15 +-----+
16 | umsatzsumme      |
17 +-----+
18 | 761.110000000000 |
19 | 80.700000000000 |
20 | 115.850000000000 |
21 | 1212.550000000000 |
22 +-----+

```

Als Letztes erfolgt die Einschränkung auf die Beutlins ([Zeile 10](#)).

```

1 mysql> SELECT SUM(umsatz) AS 'umsatzsumme'
2     -> FROM
3     -> (
4     ->   SELECT
5     ->     rechnung_id, kunde_id, SUM(einzelpreis*menge) AS 'umsatz'
6     ->   FROM
7     ->     rechnung_position INNER JOIN artikel USING(artikel_id)
8     ->           INNER JOIN rechnung USING(rechnung_id)
9     ->           INNER JOIN kunde USING(kunde_id)
10    -> WHERE nachname = 'beutlin'
11    -> GROUP BY
12    ->     rechnung_id
13    -> ) AS 'ksum'
14    -> GROUP BY
15    ->   kunde_id
16    -> ;
17 +-----+
18 | umsatzsumme      |
19 +-----+
20 | 80.700000000000 |
21 | 115.850000000000 |
22 +-----+

```

Für eine Unterabfrage nicht schlecht, was?

Schritt 2 – Bau der Oberanweisung ohne Unterabfrage: Die Oberanweisung muss mir eine Liste von Kundennummern zurückliefern, deren Umsatzsummen alle größer sind als der Inhalt von der ALL ()-Liste. Die Oberanweisung sieht in weiten Teilen wie die Unterabfrage aus:

```

1 SELECT kunde_id, SUM(umsatz) AS 'umsatzsumme'
2   FROM

```

```

3  (
4    SELECT
5      rechnung_id, kunde_id, SUM(einzelpreis*menge) AS 'umsatz'
6      FROM
7        rechnung_position INNER JOIN artikel USING(artikel_id)
8          INNER JOIN rechnung USING(rechnung_id)
9      GROUP BY
10        rechnung_id
11  ) AS 'ksum'
12  GROUP BY
13    kunde_id
14  HAVING umsatzsumme > ALL (SELECT 100)
15 ;

```

Schritt 3 – Einbau der Unterabfrage in die Oberanweisung: Um die Sache noch schöner zu machen, habe ich den Kundennamen noch *drangejoint*.

```

1  mysql> SELECT
2    ->   kunde_id, kunde.nachname, kunde.vorname, SUM(umsatz) AS 'umsatzsumme'
3    ->   FROM
4    ->   (
5    ->     SELECT
6    ->       rechnung_id, kunde_id, SUM(einzelpreis*menge) AS 'umsatz'
7    ->       FROM
8    ->         rechnung_position INNER JOIN artikel USING(artikel_id)
9    ->           INNER JOIN rechnung USING(rechnung_id)
10   ->         GROUP BY rechnung_id
11  ) AS 'ksum'
12  ->   INNER JOIN kunde USING (kunde_id)
13  ->   GROUP BY kunde_id
14  ->   HAVING umsatzsumme > ALL
15  ->   (
16  ->     SELECT SUM(umsatz) AS 'umsatzsumme'
17  ->     FROM
18  ->     (
19  ->       SELECT
20  ->         rechnung_id, kunde_id, SUM(einzelpreis*menge) AS 'umsatz'
21  ->         FROM
22  ->           rechnung_position INNER JOIN artikel USING(artikel_id)
23  ->             INNER JOIN rechnung USING(rechnung_id)
24  ->               INNER JOIN kunde USING(kunde_id)
25  ->             WHERE nachname = 'beutlin'
26  ->             GROUP BY rechnung_id
27  ->       ) AS 'ksum'
28  ->       GROUP BY kunde_id
29  ->     )
30  -> ;
31
32 +-----+-----+-----+-----+
33 | kunde_id | nachname      | vorname | umsatzsumme |
34 +-----+-----+-----+-----+
35 |      1 | Gamdschie     | Samweis | 761.110000000000 |
36 |      5 | Earendilionn | Elrond  | 1212.550000000000 |
37 +-----+-----+-----+-----+

```

Schön, gell?

13.2.2.4 Beispiel 4: ANY()

Wir wollen alle Artikel wissen, die teurer sind als irgendein Artikel der Warengruppe Bürobedarf. Im Gegensatz zu ALL() liefert das ANY() schon dann ein TRUE, wenn mindestens einer der Vergleiche mit den Listenelementen stimmt.

Schritt 1 – Unterabfrage ermitteln: Die Unterabfrage muss mir alle Artikelpreise der Warengruppe Bürobedarf liefern.

```

1 mysql> SELECT a.einzelpreis
2      ->   FROM
3      ->   artikel_nm_warengruppe INNER JOIN artikel a USING(artikel_id)
4      ->                               INNER JOIN warengruppe w USING(warengruppe_id)
5      -> WHERE
6      ->   w.bezeichnung = 'Bürobedarf'
7      -> ;
8 +-----+
9 | einzelpreis |
10+-----+
11 |    2.320000 |
12 |    56.260000 |
13 |    6.260000 |
14 |    4.170000 |
15 |    5.050000 |
16 +-----+

```

Schritt 2 – Bau der Oberanweisung ohne Unterabfrage: Die Oberanweisung muss alle Artikel mit ihren Preisen ausgeben:

```

1 SELECT
2   a.bezeichnung, a.einzelpreis
3   FROM
4     artikel a
5   WHERE
6     a.einzelpreis > ANY(SELECT 10)
7 ;

```

Schritt 3 – Einbau der Unterabfrage in die Oberanweisung: Anstelle des SELECT 10 wird jetzt die Unterabfrage eingebaut.

```

1 mysql> SELECT
2      ->   a.bezeichnung, a.einzelpreis
3      ->   FROM
4      ->   artikel a
5      -> WHERE
6      ->   a.einzelpreis > ANY
7      ->
8      ->   (
9      ->     SELECT a.einzelpreis
10     ->       FROM
11     ->         artikel_nm_warengruppe INNER JOIN artikel a
12     ->                           USING(artikel_id)
13     ->                           INNER JOIN warengruppe w
14     ->                             USING(warengruppe_id)
15     -> WHERE
16     ->   w.bezeichnung = 'Bürobedarf'
17     ->
18 +-----+-----+
19 | bezeichnung | einzelpreis |
20 +-----+-----+

```

```

21 | Tinte (gold) | 56.260000 |
22 | Tinte (rot) | 6.260000 |
23 | Tinte (blau) | 4.170000 |
24 | Feder | 5.050000 |
25 | Tulpenzwiebel | 3.420000 |
26 | Schaufel | 15.100000 |
27 | Spaten | 20.100000 |
28 +-----+

```

Die Ausgabe ergibt sich daraus, dass es ausreicht, dass der Artikel größer als einer der Preise in der Warengruppe *Bürobedarf* ist. So ist die Tulpenzwiebel beispielsweise teurer als Papier und taucht deshalb hier auf.

Ärgerlich ist, dass hier die Artikel der Warengruppe Bürobedarf auch auftauchen. Hier hilft nur eine zweite Unterabfrage in den [Zeilen 6–14](#):

```

1 mysql> SELECT
2   -> a.bezeichnung, a.einzelpreis
3   -> FROM
4   -> artikel a
5   -> WHERE
6   -> a.artikel_id NOT IN
7   -> (
8   ->   SELECT
9   ->     artikel_id
10  ->   FROM
11  ->     artikel_nm_warengruppe nm  INNER JOIN warengruppe w
12  ->           USING(warengruppe_id)
13  ->   WHERE w.bezeichnung = 'Bürobedarf'
14  -> )
15  -> AND
16  -> a.einzelpreis > ANY
17  -> (
18  ->   SELECT a.einzelpreis
19  ->   FROM
20  ->     artikel_nm_warengruppe nm INNER JOIN artikel a
21  ->           USING(artikel_id)
22  ->           INNER JOIN warengruppe w
23  ->           USING(warengruppe_id)
24  ->   WHERE
25  ->     w.bezeichnung = 'Bürobedarf'
26  -> );
27 +-----+-----+
28 | bezeichnung | einzelpreis |
29 +-----+-----+
30 | Tulpenzwiebel | 3.420000 |
31 | Schaufel | 15.100000 |
32 | Spaten | 20.100000 |
33 +-----+

```

13.2.3 Also, der Unterschied zwischen IN(), ALL() und ANY() ist mir nicht ganz klar!

- **IN()** Diese Abfrage prüft, ob ein Wert in der Liste enthalten ist. Falls er enthalten ist, liefert sie TRUE, sonst FALSE. Beispiele:

[...] 5 IN (1, 2, 3, 4, 5)
 ⇒ TRUE

[...] 7 IN (1, 2, 3, 4, 5)
 ⇒ FALSE

Ein besonderer Fall entsteht, wenn NULL ins Spiel kommt:

[...] 5 IN (1, NULL, 3, 4, 5)
 ⇒ TRUE

[...] 7 IN (1, NULL, 3, 4, 5)
 ⇒ NULL

[...] NULL IN (1, 2, 3, 4, 5)
 ⇒ NULL

- **ALL()** Diese Abfrage prüft, ob ein Wert bezüglich eines Vergleichsoperators für alle Werte der Liste TRUE ergibt. Nur dann wird der Gesamtausdruck auch TRUE. Beispiele⁴:

[...] 1 <= ALL (1, 2, 3, 4, 5)
 ⇒ TRUE

[...] 3 <= ALL (1, 2, 3, 4, 5)
 ⇒ FALSE

Taucht in der Liste ein NULL auf, wird immer das Ergebnis auf NULL gesetzt, wenn nicht schon einer der Vergleiche ein FALSE liefert:

[...] 1 <> ALL (1, 2, 3, 4, NULL)
 ⇒ FALSE

[...] 1 <> ALL (NULL, 2, 3, 4, 5)
 ⇒ NULL

- **ANY()** Diese Abfrage prüft, ob ein Wert bezüglich eines Vergleichsoperators für mindestens einen Wert der Liste TRUE ergibt. Der Gesamtausdruck wird dann auch TRUE.

[...] 3 <= ANY (1, 2, 3, 4, 5)
 ⇒ TRUE

[...] 7 <= ANY (1, 2, 3, 4, 5)
 ⇒ FALSE

Auch hier ist der Fall NULL zu betrachten. Ergibt der Vergleich mit irgendeinem der Listenelemente TRUE, reicht das aus. Wird kein passender Vergleich gefunden und es kommt ein NULL vor, ist das Ergebnis auch NULL:

[...] 3 >= ANY (1, 2, 3, 4, NULL)
 ⇒ TRUE

[...] 1 >= ANY (NULL, 2, 3, 4, 5)
 ⇒ NULL

13.2.4 Gibt es einen Unterschied zwischen NOT IN() und <> ALL()?

Nein, siehe [MyS18d].

13.2.5 Tabellenunterabfrage

Unterabfragen können nicht nur auf *einer* Spalte arbeiten wie bei der Skalar- und der Listenunterabfrage, sondern auch mit Tabellen, die mehrere Spalten enthalten.



Definition 52: Tabellenunterabfragen

Liefert die Unterabfrage eine Tabelle mit mehreren Spalten, so bezeichne ich diese als **Tabellenunterabfrage**.

⁴ Die Beispiele sind eine Verkürzung von beispielsweise `SELECT 3 <= ALL (SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4 UNION SELECT 5)`.

Es fällt mir sehr schwer, ein einigermaßen sinnvolles Beispiel zu konstruieren. Also lasse ich es und zeige den Effekt anhand einer konstruierten Situation. Wieder wollen wir wissen, welcher Kunde – aber diesmal mit Namen – im Jahr 2011 und 2012 Rechnungen hatte.

```

1 mysql> SELECT DISTINCT k.nachname, k.vorname
2   ->   FROM rechnung r INNER JOIN kunde k USING(kunde_id)
3   -> WHERE
4   ->   YEAR(r.datum) = '2011' AND (k.nachname, k.vorname) IN
5   ->   (
6   ->     SELECT
7   ->       k.nachname, k.vorname
8   ->     FROM
9   ->       rechnung r INNER JOIN kunde k USING(kunde_id)
10  ->      WHERE YEAR(r.datum) = '2012'
11  ->   )
12  -> ;
13 +-----+-----+
14 | nachname | vorname |
15 +-----+-----+
16 | Gamdschie | Samweis |
17 +-----+-----+

```

In Zeile 4 geschieht etwas ganz Neues. Bisher haben wir immer nur skalare Werte miteinander verglichen. Hier werden zwei Listen miteinander verglichen. Auf der linken und der rechten Seite vom IN stehen Listen mit je zwei Spalten.



Hinweis: Beim Vergleich mit Tabellenunterabfragen muss man darauf achten, dass die Anzahl der Spalten auf der rechten und linken Seite der Vergleichsoperation gleich ist.

■ 13.3 Korrelierende Unterabfrage

In [Definition 49 auf Seite 228](#) wurden solche Unterabfragen eingeführt, die unabhängig von der Oberanweisung sind. Es gibt auch welche, die sich auf die Oberabfrage beziehen und deshalb nicht unabhängig von der Oberanweisung ausgeführt werden können.

13.3.1 Beispiel 1: Rechnungen mit vielen Positionen

Ich möchte alle Rechnungen mit mindestens drei Positionen.

```

1 mysql> SELECT rechnung_id
2   ->   FROM rechnung r
3   -> WHERE
4   ->   (
5   ->     SELECT
6   ->       COUNT(position_nr)
7   ->     FROM rechnung_position rp
8   ->     WHERE rp.rechnung_id = r.rechnung_id
9   ->   ) >= 3
10  -> ;

```

```
11 +-----+
12 | rechnung_id |
13 +-----+
14 |          1 |
15 |          5 |
16 +-----+
```

In der WHERE-Klausel der Unterabfrage wird in [Zeile 8](#) mit der Rechnungsnummer der Oberanweisung ([Zeile 1](#)) verglichen⁵.

Da die Unterabfrage jetzt von einem Spalteninhalt der Oberanweisung abhängt, wird sie nicht einmal ausgeführt, sondern für jeden Spalteninhalt der Oberanweisung neu. In unserem Fall wird für jede Rechnungsnummer – also 11-mal – die Anzahl der Rechnungspositionen berechnet.



Hinweis: Weil die korrelierende Unterabfrage mehrfach ausgeführt wird, ist sie ein Performancerisiko.

13.3.2 Beispiel 2: EXISTS

Ich möchte alle Rechnungen, zu denen es eine Bestellung gibt. Hier kann der EXISTS-Operator helfen, da er überprüft, ob das Ergebnis eines SELECT überhaupt eine Zeile zurückliefert.

```
1 mysql> SELECT rechnung_id
2     ->   FROM rechnung r
3     -> WHERE
4     ->   EXISTS
5     ->   (
6     ->     SELECT bestellung_id
7     ->       FROM bestellung b
8     ->     WHERE b.bestellung_id = r.bestellung_id
9     ->   )
10    -> ;
11 +-----+
12 | rechnung_id |
13 +-----+
14 |          1 |
15 |          2 |
16 |          3 |
17 |          4 |
18 |          5 |
19 |          6 |
20 +-----+
```

⁵ Genau diese Verknüpfung zwischen der Oberanweisung und der Unterabfrage ist die *Korrelation*, die dieser Abfrage den Namen gegeben hat.

■ 13.4 Fallstudie Datenimport

Ein anderer Online-Shop hat pleite gemacht. Sie haben im Rahmen einer Kundenübernahme die Kunden- und Adressdaten des Shops als CSV-Datei⁶ erhalten.

```
nachname;vorname;strasse;hnr;lkz;plz;ort;ktnr;blz;
Brandybock;Gormadoc;Brandyschloss;1;BL;57990;Bockenburg;9999999991;28063253
Starrkopf;Malva;Brandyschloss;1;BL;57990;Bockenburg;9999999992;28063253
Brandybock;Madoc;Brandyschloss;1;BL;57990;Bockenburg;9999999993;28063253
Goldwert;Hanna;Brandyschloss;1;BL;57990;Bockenburg;9999999994;28063253
Brandybock;Marmadoc;Brandyschloss;1a;BL;57990;Bockenburg;9999999995;28063253
Bolger;Adaldrida;Brandyschloss;1a;BL;57990;Bockenburg;9999999996;28063253
Brandybock;Gorbadoc;Brandyschloss;1;BL;57990;Bockenburg;9999999997;28063253
Tuk;Mirabella;Brandyschloss;1;BL;57990;Bockenburg;9999999998;28063253
Brandybock;Rorimac;Brandyschloss;1a;BL;57990;Bockenburg;9999999911;28063253
Guld;Menegilda;Brandyschloss;1a;BL;57990;Bockenburg;9999999912;28063253
Brandybock;Saradoc;Brandyschloss;1a;BL;57990;Bockenburg;9999999913;28063253
Tuk;Esmeralda;Brandyschloss;1a;BL;57990;Bockenburg;9999999914;28063253
Brandybock;Meriadoc;Brandyschloss;1;BL;57990;Bockenburg;9999999915;28063253
Brandybock;Merimac;Bocklandstraße;4;BL;57996;Krickloch;8888888816;44550045
Lochner;Rufus;Maggots Weg;2;BL;57980;Maggots Hof;7777777717;72160818
Tuk;Peregrin;Stockstraße;1;AL;45689;Buckelstadt;6666666618;44550045
Gamdschie;Samweis;Beutelhaldenweg;5;AL;67676;Hobbingen;1111111111;10010010
```

Schritt 1 – Daten importieren: Um die Daten auswerten und umbauen zu können, erstelle ich eine temporäre Tabelle, die die Daten erst mal unverändert importieren kann.

```
1 CREATE TEMPORARY TABLE tmp_import
2 (
3     nachname VARCHAR(255),
4     vorname  VARCHAR(255),
5     strasse   VARCHAR(255),
6     hnr       VARCHAR(255),
7     lkz       VARCHAR(255),
8     plz       VARCHAR(255),
9     ort       VARCHAR(255),
10    ktnr      VARCHAR(255),
11    blz       VARCHAR(255)
12 );
13
14 LOAD DATA LOCAL INFILE 'kunden01.csv'
15 INTO TABLE tmp_import
16 FIELDS
17 TERMINATED BY ';'
18 LINES
19 TERMINATED BY '\n'
20 IGNORE 1 LINES
21 (nachname, vorname, strasse, hnr, lkz, plz, ort, ktnr, blz)
22 ;
```



Aufgabe 13.4: Überprüfen Sie, dass keine Warnung vorkommt, und schauen Sie nach, ob die Daten genauso in der Tabelle angekommen sind, wie Sie es sich vor gestellt haben.

⁶ Die Daten sind in der Datei kunden01.csv.

Schritt 2 – Kunden ermitteln: Wir wollen jetzt alle neuen Kunden anhand ihres Namens identifizieren. Diese neuen Kunden sollen gleich in die Kundentabelle eingefügt werden.

```

1  INSERT INTO kunde (nachname, vorname)
2  SELECT DISTINCT nachname, vorname
3  FROM tmp_import tmp
4  WHERE
5      (tmp.vorname, tmp.nachname) NOT IN
6      (
7          SELECT vorname, nachname FROM kunde
8      )
9  ;

```

In der [Zeile 2](#) werden alle unterschiedlichen Namen ermittelt. Das DISTINCT soll verhindern, dass mehrfach vorkommende Namen mehrfach übernommen werden. In unserem Fall eine unnötige Vorsichtsmaßnahmen.

Durch das NOT IN in der [Zeile 5](#) wird ermittelt, ob der Name nicht schon in Tabelle kunde vorhanden ist.



Aufgabe 13.5: Welcher Kunde ist aussortiert worden? Erstellen Sie dazu ein passendes SELECT.

Aufgabe 13.6: Handelt es sich in [Zeile 5](#) um eine skalare Unterabfrage, eine Listenunterabfrage oder Tabellenunterabfrage?

Aufgabe 13.7: Handelt es sich in [Zeile 7](#) um eine korrelierende oder nicht korrelierende Unterabfrage?

Schritt 3 – Adressen ermitteln: Wie bei den Kunden muss hier überprüft werden, ob die Adresse schon vorhanden ist. Schließlich will ich keine sinnlosen Dubletten⁷.

```

1  INSERT INTO adresse (strasse, hnr, lkz, plz, ort)
2  SELECT DISTINCT strasse, hnr, lkz, plz, ort
3  FROM tmp_import tmp
4  WHERE
5      (tmp.strasse, tmp.hnr, tmp.lkz, tmp.plz, tmp.ort) NOT IN
6      (
7          SELECT strasse, hnr, lkz, plz, ort
8          FROM adresse
9      )
10 ;

```

Schritt 4 – Einbau der Fremdschlüssel in die Importtabelle: Wir haben jetzt die neuen Kunden- und Adressdaten eingefügt. Es fehlen aber die Verknüpfungen. Dazu könnte man jetzt verschiedene Wege gehen, aber ich möchte hier die Unterabfragen verwenden, die hier sehr effektiv sind.

Zunächst muss die Importtabelle um die Fremdschlüssel erweitert werden:

```

1  ALTER TABLE tmp_import ADD kunde_id    INT UNSIGNED,
2                      ADD adresse_id   INT UNSIGNED,
3                      ADD bank_id     CHAR(12);

```

⁷ Obwohl man das hier diskutieren kann. Zwei Personen können die gleiche Adresse haben, und trotzdem würde man diese nicht als *eine* Adresse im System abspeichern.

Sind die Spalten angelegt worden, wird jetzt für jeden Datensatz der Importtabelle der Fremdschlüssel ermittelt und eingetragen.

```

1 UPDATE tmp_import t SET
2   t.kunde_id =
3   (
4     SELECT kunde_id
5       FROM kunde k
6      WHERE t.vorname = k.vorname AND t.nachname = k.nachname
7   ),
8   t.adresse_id =
9   (
10    SELECT adresse_id
11      FROM adresse a
12     WHERE
13       t.strasse = a.strasse
14       AND t.hnr = a.hnr
15       AND t.lkz = a.lkz
16       AND t.plz = a.plz
17       AND t.ort = a.ort
18   ),
19   t.bank_id =
20   (
21     SELECT bank_id FROM bank WHERE t.blz = bank.blz LIMIT 1
22   )
23 ;

```

In der [Zeile 2](#) wird für jede Zeile die ermittelte Kundennummer eingetragen, in [Zeile 8](#) die ermittelte Adressnummer und in [Zeile 19](#) die ermittelte bank_id.



Aufgabe 13.8: Handelt es sich in den [Zeilen 2, 8](#) und [19](#) um korrelierende oder nicht korrelierende Abfragen?

Schritt 5 – Bankverbindung ermitteln: Jetzt können wir die Bankverbindungen in die entsprechende Tabelle eintragen.

```

1 INSERT INTO
2   bankverbindung (kunde_id, bankverbindung_nr, bank_id, kontonummer, iban)
3   SELECT DISTINCT kunde_id, 1, bank_id, ktNr, CONCAT(blz, ktNr)
4     FROM tmp_import tmp
5    WHERE
6      (tmp.kunde_id, 1, tmp.bank_id, tmp.ktNr) NOT IN
7      (
8        SELECT kunde_id, 1, bank_id, kontonummer
9          FROM bankverbindung
10         );

```

Schritt 6 – Einbau des Fremdschlüssels auf die Bankverbindung: Die Bankverbindung ist jetzt auch bekannt. Da der erste Teil des Primärschlüssels die Kundennummer ist und wir annehmen können, dass jeder der neuen Kunden nur eine Bankverbindung hat, brauchen wir nur eine 1 in die Spalte bankverbindung_nr einzutragen.

```

1 ALTER TABLE tmp_import
2   ADD bankverbindung_nr INT UNSIGNED NOT NULL DEFAULT 1;

```

Schritt 7 – Abschluss: Wir haben die neuen Kunden angelegt, wir haben die neuen Adressen angelegt, wir haben die neuen Bankverbindungen angelegt. Die Bankverbindungen

waren schnell über die Kundennummer und die Bankverbindungsnummer mit dem Kunden verknüpft (Schritt 6). Bleibt noch, die Kunden mit der Adresse zu verknüpfen. Es stehen zwei Möglichkeiten offen: die Rechnungs- oder Lieferadresse. Wir entscheiden uns für die Rechnungsadresse:

```
1 UPDATE kunde k SET
2   k.rechnung_adresse_id =
3     (SELECT adresse_id FROM tmp_import tmp WHERE tmp.kunde_id = k.kunde_id)
4 WHERE k.rechnung_adresse_id IS NULL;
```



Aufgabe 13.9: Ist Zeile 2 in Schritt 7 eine korrelierende oder nicht korrelierende Unterabfrage?

Zum Abschluss überprüfen wir, ob der Import geklappt hat, was natürlich nur wegen der wenigen Daten möglich ist (schließlich ist der Shop ja pleite gegangen).

```
1 mysql> SELECT kunde_id, nachname, vorname, strasse, ort
2      ->   FROM
3      ->   kunde LEFT JOIN adresse ON kunde.rechnung_adresse_id = adresse_id;
4 +-----+-----+-----+-----+-----+
5 | kunde_id | nachname    | vorname    | strasse    | ort      |
6 +-----+-----+-----+-----+-----+
7 |      1 | Gamdschie   | Samweis    | Beutelhaldenweg | Hobbingen |
8 |      2 | Beutlin      | Frodo      | Beutelhaldenweg | Hobbingen |
9 |      3 | Beutlin      | Bilbo      | Beutelhaldenweg | Hobbingen |
10 |     4 | Telcontar    | Elessar    | Auf der Feste  | Minas Tirith |
11 |     5 | Earendillionn| Elrond    | Letztes Haus  | Bruchtal  |
12 |     6 | Eichenschild | Thorin    | NULL        | NULL      |
13 |     7 | Brandybock   | Gormadoc   | Brandyschloss | Bockenburg |
14 |     8 | Starrkopf    | Malva      | Brandyschloss | Bockenburg |
15 |     9 | Brandybock   | Madoc      | Brandyschloss | Bockenburg |
16 |    10 | Goldwert     | Hanna      | Brandyschloss | Bockenburg |
17 |    11 | Brandybock   | Marmamadoc | Brandyschloss | Bockenburg |
18 |    12 | Bolger       | Adaldrida  | Brandyschloss | Bockenburg |
19 |    13 | Brandybock   | Gorbادoc   | Brandyschloss | Bockenburg |
20 |    14 | Tuk          | Mirabella  | Brandyschloss | Bockenburg |
21 |    15 | Brandybock   | Rorimac    | Brandyschloss | Bockenburg |
22 |    16 | Guld         | Menegilda  | Brandyschloss | Bockenburg |
23 |    17 | Brandybock   | Saradoc    | Brandyschloss | Bockenburg |
24 |    18 | Tuk          | Esmeralda  | Brandyschloss | Bockenburg |
25 |    19 | Brandybock   | Meriadoc   | Brandyschloss | Bockenburg |
26 |    20 | Brandybock   | Merimac    | Bocklandstraße | Krickloch |
27 |    21 | Lochner      | Rufus      | Maggots Weg   | Maggots Hof |
28 |    22 | Tuk          | Peregrin   | Stockstraße  | Buckelstadt |
29 +-----+-----+-----+-----+-----+
```

■ 13.5 Wie ticken Unterabfragen intern?

Zu Beispiel 3: Schauen wir uns den EXPLAIN dazu (siehe Seite 230) an:

```
1 mysql> SELECT
2       ->   bp1.bestellung_id, SUM(bp1.menge * a1.einzelpreis) 'bwert1'
```

```

3      -> FROM
4      -> bestellung_position bp1 INNER JOIN artikel a1 USING(artikel_id)
5      -> GROUP BY
6      -> bp1.bestellung_id
7      -> HAVING
8      -> 'bwert1' >
9      -> (
10     ->   SELECT AVG(bw.bwert2)
11     ->   FROM
12     ->   (
13     ->     SELECT
14     ->       bp2.bestellung_id, SUM(bp2.menge * a2.einzelpreis) 'bwert2'
15     ->       FROM
16     ->         bestellung_position bp2 INNER JOIN artikel a2
17     ->           USING(artikel_id)
18     ->         GROUP BY
19     ->           bp2.bestellung_id
20     ->       ) AS 'bw'
21     ->   )
22     -> ;
23 +-----+
24 | bestellung_id | bwert1      |
25 +-----+
26 |      1 | 206.7000000000000 |
27 |      5 | 488.5500000000000 |
28 +-----+
29
30 mysql> EXPLAIN /wie oben/\G
31 **** 1. row ****
32      id: 1
33      select_type: PRIMARY
34      partitions: NULL
35      table: bp1
36      type: index
37      possible_keys: PRIMARY,artikel_id
38          key: PRIMARY
39          key_len: 8
40          ref: NULL
41          rows: 13
42          filtered: 100.00
43          Extra: NULL
44 **** 2. row ****
45      id: 1
46      select_type: PRIMARY
47      table: a1
48      partitions: NULL
49      type: eq_ref
50      possible_keys: PRIMARY
51          key: PRIMARY
52          key_len: 4
53          ref: oshop_bp1.artikel_id
54          rows: 1
55          filtered: 100.00
56          Extra: NULL
57 **** 3. row ****
58      id: 2
59      select_type: SUBQUERY
60      table: <derived3>
61      partitions: NULL

```

```

62      type: ALL
63  possible_keys: NULL
64      key: NULL
65      key_len: NULL
66      ref: NULL
67      rows: 13
68  filtered: 100.00
69      Extra: NULL
70 **** 4. row ****
71      id: 3
72  select_type: DERIVED
73      table: bp2
74      partitions: NULL
75      type: index
76  possible_keys: PRIMARY,artikel_id
77      key: PRIMARY
78      key_len: 8
79      ref: NULL
80      rows: 13
81  filtered: 100.00
82      Extra: NULL
83 **** 5. row ****
84      id: 3
85  select_type: DERIVED
86      table: a2
87      partitions: NULL
88      type: eq_ref
89  possible_keys: PRIMARY
90      key: PRIMARY
91      key_len: 4
92      ref: oshop_bp2.artikel_id
93      rows: 1
94  filtered: 100.00
95      Extra: NULL

```

Das ist schon etwas beeindruckend, oder? Immer noch eine nicht korrelierende Skalarunterabfrage und trotzdem schon schön umfangreich; so lieben wir es.

Zunächst 1. row und 2. row. Beide haben die id: 1 (siehe Zeilen 32 und 45). Das bedeutet, dass beide Ausgaben zum gleichen SELECT gehören. Schaut man sich an, auf welche Tabellen sich dieser SELECT bezieht (Zeile 35 und 47), so muss es sich um den SELECT handeln, der sich oben zwischen den Zeilen 1 und 4 befindet. Dies wird ebenfalls durch die Angabe PRIMARY in den Zeilen 33 und 46 bestätigt. Die Angaben in 1. row und 2. row beschreiben somit die Art und Weise, wie der INNER JOIN zwischen den beiden Tabellen organisiert wird. Verwendet wird die Spalte artikel_id (Zeilen 37 und 53) und die damit verbundenen Indizes (Primärschlüssel: Zeilen 51, Fremdschlüssel 37).

Jetzt 3. row. Mit der id: 2 wird mir angezeigt, dass es ein neuer SELECT sein muss. Unter type (Zeile 59) wird mir SUBQUERY mitgeteilt, dass dieser SELECT eine Unterabfrage außerhalb einer FROM-Klausel ist. Damit kann nur der SELECT oben ab Zeile 10 gemeint sein, denn der SELECT oben ab Zeile 13 ist innerhalb einer FROM-Klausel. Der Tabellename <derived3> sagt mir aber spontan gar nichts. Die 3 ist aber schon ein Hinweis, denn die nächsten beiden Zeilen haben die id: 3. Fazit: Die 3. row ist eine Unterabfrage auf die Tabelle, die durch den SELECT mit der id: 3 entsteht.

Zum Schluss 4. row und 5. row. Beide gehören zum gleichen SELECT, da sie die gleiche id: 3 haben. Durch den select_type DERIVED (Zeile 72) wird mir mitgeteilt, dass

MySQL die Unterabfrage vorab ausführt und das Ergebnis in eine – unsichtbare – temporäre Tabelle abspeichert. Die restlichen Angaben entsprechen denen von 1. row und 2. row, da hier der gleiche INNER JOIN ausgeführt wird.

Beispiel 4: Auch bei Beispiel 4 lohnt sich ein Blick ins Innere:

```

1 mysql> EXPLAIN
2   -> SELECT rechnung_id
3   ->   FROM rechnung
4   ->   WHERE bestellung_id IN
5   ->   (
6   ->     SELECT bestellung_id FROM bestellung
7   ->   )\G
8 **** 1. row ****
9   id: 1
10  select_type: PRIMARY
11  table: rechnung
12  type: ALL
13  possible_keys: NULL
14  key: NULL
15  key_len: NULL
16  ref: NULL
17  rows: 11
18  Extra: Using where
19 **** 2. row ****
20  id: 2
21  select_type: DEPENDENT SUBQUERY
22  table: bestellung
23  type: unique_subquery
24  possible_keys: PRIMARY
25  key: PRIMARY
26  key_len: 4
27  ref: func
28  rows: 1
29  Extra: Using index

```

Sie erkennen, dass der MySQL-/MariaDB-Optimierer die Abfrage bzgl. interner Optimierstrategien umbaut. Ich möchte hier auf eine nähere Erklärung verzichten, da ich nur eine Ahnung darüber vermitteln wollte, dass Abfragen ggf. umgebaut werden und deshalb das EXPLAIN manchmal Angaben macht, die man auf den ersten Blick nicht versteht.

MySQL oder MariaDB führt den ersten SELECT aus und wendet auf das Ergebnis eine WHERE-Klausel an (siehe Zeile 18). Der zweite SELECT kann einen Primärschlüsselindex verwenden und liefert deshalb eine sortierte Liste zurück. Das ist für die Performance sehr wichtig, da dann innerhalb der Liste im IN() mit binärer Suche gearbeitet werden kann. Bei langen Listen ist dies eine erhebliche Zeitsparnis. Alternativ muss die ganze Liste im IN() sequenziell durchsucht werden.

Warum er eine korrelierende Abfrage (Zeile 21) mit einer Referenz auf eine Funktion (Zeile 27) ausgibt, kann man erst verstehen, wenn man sich das Ergebnis des EXPLAIN EXTENDED anschaut:

```

1 mysql> EXPLAIN EXTENDED
2   -> SELECT rechnung_id
3   ->   FROM rechnung
4   ->   WHERE bestellung_id IN
5   ->   (
6   ->     SELECT bestellung_id FROM bestellung

```

```
7      ->  )\G
8  ***** 1. row *****
9  [...]
10 ***** 2. row *****
11 [...]
12 mysql> SHOW WARNINGS\G
13 ***** 1. row *****
14   Level: Note
15   Code: 1003
16 Message:
17 select 'oshop'.'rechnung'.'rechnung_id' AS 'rechnung_id'
18   from 'oshop'.'rechnung'
19   where
20     <in_optimizer>
21     ('oshop'.'rechnung'.'bestellung_id',<exists>
22      (<primary_index_lookup>
23        (<cache>('oshop'.'rechnung'.'bestellung_id') in bestellung on PRIMARY
24      )
25    )
26  )
```

■ 13.6 Aufgaben



Aufgabe 13.10: Geben Sie den Kunden mit der kleinsten Postleitzahl aus.

Aufgabe 13.11: Geben Sie die jüngste Bestellung aus.

Aufgabe 13.12: Geben Sie den Lieferanten mit den meisten Artikel an.

Aufgabe 13.13: Ermitteln Sie die Bankleitzahlen mit den meisten Banken.

Aufgabe 13.14: Geben Sie das Clearinggebiet (1. Stelle der Bankleitzahl) aus, welches die wenigsten Banken hat.

Aufgabe 13.15: Ermitteln Sie alle Bestellungen, in denen Artikel vorkommen, die ihre Mindestmenge auf dem Lager unterschritten haben.

Aufgabe 13.16: Geben Sie alle Kundennamen aus, die eine Rechnung im Mahn- oder Inkassoverfahren haben.

Aufgabe 13.17: Geben Sie alle Artikel aus, die in der Warengruppe Gartenbedarf und in der Warengruppe Pflanzen sind.

Aufgabe 13.18: Gibt es Kunden, die nur in 2011 Rechnungen hatten?

Aufgabe 13.19: Geben Sie alle Rechnungen aus, die keine Artikel der Warengruppe Gartenbedarf haben.

Aufgabe 13.20: Ermitteln Sie alle Rechnungen, zu denen es keine Bestellungen gibt.

14

Mengenoperationen



Haben Sie die Mengenlehre in der Grundschule auch so gehasst? Ich hoffe, das hat sich gelegt.

- Grundkurs
 - Auswertungen vereinigen mit UNION
 - Schnittmenge mit INTERSECT
 - Differenzmenge mit EXCEPT
- Vertiefendes
 - ALL und DISTINCT
 - Simulation von INTERSECT
 - Simulation von EXCEPT



Die Quelltexte dieses Kapitels stehen in der Datei `listing11.sql` (siehe [Listing 29.11 auf Seite 501](#), [Listing 29.47 auf Seite 624](#) und [Listing 29.32 auf Seite 571](#)).

■ 14.1 Die Vereinigung mit UNION

Wir möchten alle verwendeten Adressen ausgeben. Unser erster Ansatz dürfte wie folgt aussehen:

```
1 mysql> SELECT
2     -> strasse, hnr, lkz, plz, ort
3     -> FROM
4     -> kunde INNER JOIN adresse ON rechnung_adresse_id = adresse_id
5     -> ;
6 +-----+-----+-----+-----+
7 | strasse      | hnr | lkz | plz  | ort   |
8 +-----+-----+-----+-----+
9 | Beutelhaldenweg | 5   | AL  | 67676 | Hobbingen |
10 [...]
11 | Stockstraße    | 1   | AL  | 45689 | Buckelstadt |
12 +-----+-----+-----+-----+
13 21 rows in set (0.00 sec)
```

In der Ausgabe fehlen aber die Adressen der Lieferanten. Ein zweiter INNER JOIN liefert mir zwar diese Adressen,

```

1 mysql> SELECT
2      -> strasse, hnr, lkz, plz, ort
3      -> FROM
4      -> lieferant INNER JOIN adresse USING(adresse_id)
5      -> ;
6 +-----+-----+-----+-----+
7 | strasse | hnr | lkz | plz | ort |
8 +-----+-----+-----+-----+
9 | Hochstrasse | 4a | DE | 44879 | Bochum |
10 | Industriegebiet | 8 | DE | 44878 | Bochum |
11 | Industriegebiet | 8 | DE | 44878 | Bochum |
12 +-----+-----+-----+-----+

```

aber ich möchte *alle* in *einer* Ausgabe haben. Hier hilft ein UNION:

 SQL:2016, T-SQL	MySQL/MariaDB, PostgreSQL
SELECT auswahl [UNION [ALL] SELECT auswahl]* ;	SELECT auswahl [UNION [ALL DISTINCT] SELECT auswahl]* ;

Bei SQL:2016 kann die Option DISTINCT zwar nicht angegeben werden, ist aber – wie bei MySQL und MariaDB – der Default. Mit dieser neuen Möglichkeit versuchen wir es noch einmal:

```

1 mysql> SELECT
2      -> strasse, hnr, lkz, plz, ort
3      -> FROM
4      -> kunde INNER JOIN adresse ON rechnung_adresse_id = adresse_id
5      -> UNION
6      -> SELECT
7      -> strasse, hnr, lkz, plz, ort
8      -> FROM
9      -> lieferant INNER JOIN adresse USING(adresse_id)
10     -> ;
11 +-----+-----+-----+-----+
12 | strasse | hnr | lkz | plz | ort |
13 +-----+-----+-----+-----+
14 | Beutelhaldenweg | 5 | AL | 67676 | Hobbingen |
15 | Beutelhaldenweg | 1 | AL | 67676 | Hobbingen |
16 | Auf der Feste | 1 | GO | 54786 | Minas Tirith |
17 | Letztes Haus | 4 | ER | 87567 | Bruchtal |
18 | Brandyschloss | 1 | BL | 57990 | Bockenburg |
19 | Brandyschloss | 1a | BL | 57990 | Bockenburg |
20 | Bocklandstraße | 4 | BL | 57996 | Krickloch |
21 | Maggots Weg | 2 | BL | 57980 | Maggots Hof |
22 | Stockstraße | 1 | AL | 45689 | Buckelstadt |
23 | Hochstrasse | 4a | DE | 44879 | Bochum |
24 | Industriegebiet | 8 | DE | 44878 | Bochum |
25 +-----+-----+-----+-----+
26 11 rows in set (0.00 sec)

```


Aufgabe 14.1: Mit der Option ALL erhalte ich folgende Ausgabe:

```
+-----+-----+-----+-----+
| strasse      | hnr | lkz | plz   | ort
+-----+-----+-----+-----+
| Beutelhaldenweg | 5  | AL  | 67676 | Hobbingen
| Beutelhaldenweg | 1  | AL  | 67676 | Hobbingen
| Beutelhaldenweg | 1  | AL  | 67676 | Hobbingen
| Auf der Feste  | 1  | GO  | 54786 | Minas Tirith
| Letztes Haus   | 4  | ER  | 87567 | Bruchtal
| Brandyschloss  | 1  | BL  | 57990 | Bockenburg
| Brandyschloss  | 1  | BL  | 57990 | Bockenburg
| Brandyschloss  | 1  | BL  | 57990 | Bockenburg
| Brandyschloss  | 1  | BL  | 57990 | Bockenburg
| Brandyschloss  | 1  | BL  | 57990 | Bockenburg
| Brandyschloss  | 1  | BL  | 57990 | Bockenburg
| Brandyschloss  | 1a | BL  | 57990 | Bockenburg
| Brandyschloss  | 1a | BL  | 57990 | Bockenburg
| Brandyschloss  | 1a | BL  | 57990 | Bockenburg
| Brandyschloss  | 1a | BL  | 57990 | Bockenburg
| Brandyschloss  | 1a | BL  | 57990 | Bockenburg
| Brandyschloss  | 1a | BL  | 57990 | Bockenburg
| Brandyschloss  | 1a | BL  | 57990 | Bockenburg
| Brandyschloss  | 1a | BL  | 57990 | Bockenburg
| Brandyschloss  | 1a | BL  | 57990 | Bockenburg
| Bocklandstraße | 4  | BL  | 57996 | Krickloch
| Maggots Weg    | 2  | BL  | 57980 | Maggots Hof
| Stockstraße    | 1  | AL  | 45689 | Buckelstadt
| Hochstrasse     | 4a | DE  | 44879 | Bochum
| Industriegebiet | 8  | DE  | 44878 | Bochum
| Industriegebiet | 8  | DE  | 44878 | Bochum
+-----+-----+-----+-----+
24 rows in set (0.00 sec)
```

Welche Bedeutung haben die Optionen DISTINCT und ALL?

Es gibt folgende Szenarien, die der UNION hauptsächlich abarbeitet:

1. Zwischen zwei Tabellen bestehen zwei Verknüpfungen. Denken Sie beispielsweise an die Tabellen **verein** und **spielpaarung**. In der Tabelle **spielpaarung** würde mit zwei Fremdschlüsseln auf die Tabelle **verein** verwiesen werden: **heimmannschaft** und **gastmannschaft**. Möchte ich jetzt wissen, wie viele Tore eine Mannschaft geschossen hat, wird ein Teil des UNION über die **heimmannschaft** und der zweite über die **gastmannschaft** sein¹.
2. Eine Tabelle wird über den gleichen Primärschlüssel mit zwei verschiedenen Tabellen verknüpft. Das ist das Szenario, was wir oben beschrieben haben. Die Tabelle **adresse** wird einmal mit **lieferant** und einmal mit **kunde** verknüpft.
3. Tabellen werden aufgeteilt, aber es werden Auswertungen über alle Daten benötigt. Eine Analyse der Datenzugriffe könnte ergeben, dass bestimmte Datensätze über einen Zeitraum von einem Monat oft und danach nur noch selten verwendet werden. Es ist sinnvoll, diese Tabelle – besonders, wenn sie groß und performancerelevant ist – in mehrere Tabellen aufzuteilen. In der einen stehen die Daten mit häufigem Zugriff und in der

¹ Das Aufsummieren der Tore selbst erfolgt über **SUM()**. Mehr dazu in [Abschnitt 12.1 auf Seite 211](#).

anderen die mit seltenem. Für Auswertungen müssen diese Tabellen wieder vereinigt werden. Eine temporäre Tabelle oder ein UNION wäre dann eine gute Wahl.

- Hinzufügen von *dummy*-Zeilen z.B. für Listboxen auf der Oberfläche. Oft werden die Inhalte einer Listbox direkt aus einer Tabelle ermittelt. Denken Sie beispielsweise an unsere Kundenart oder die Bankleitzahl. Ein SELECT DISTINCT mit ORDER BY, und schon hat man alle verwendbaren Alternativen. Die Vorbelegung der Listbox soll aber ein Text wie **Bitte auswählen!** sein. Diesen Text müsste man mit einem UNION dem eigentlichen SELECT anfügen.



Hinweis: Sie müssen darauf achten, dass beide SELECT des UNION die gleichen Spalten in der gleichen Reihenfolge zurückliefern. Wenn dies nicht der Fall ist, erhalten Sie diese Fehlermeldung:

ERROR 1222 (21000):
The used SELECT statements have a different number of columns

■ 14.2 Die Schnittmenge

Manchmal braucht man die Zeilen, die auch in einer anderen Tabelle vorkommen. Man nennt dies die Schnittmenge zweier Tabellen. MySQL und MariaDB kennen keinen entsprechenden Operator, aber man kann das Verhalten mit einer Unterabfrage emulieren.

14.2.1 Mit INTERSECT



SQL:2016

```
SELECT auswahl/  
[INTERSECT [ALL]  
SELECT auswahl]*  
;
```

PostgreSQL

```
SELECT auswahl/  
[INTERSECT [ALL|DISTINCT]  
SELECT auswahl]*  
;
```

T-SQL

```
SELECT auswahl/  
[INTERSECT  
SELECT auswahl]*  
;
```

Ich möchte Folgendes wissen: Welche Kunden haben sowohl 2011 als auch 2012 eine Rechnung erhalten (Beispiel in PostgreSQL ausgeführt)?

```
1 oshop=# SELECT kunde_id FROM rechnung  
2 oshop-# WHERE EXTRACT(ISOYEAR FROM datum) = '2011'  
3 oshop-# INTERSECT  
4 oshop-# SELECT kunde_id FROM rechnung  
5 oshop-# WHERE EXTRACT(ISOYEAR FROM datum) = '2012'  
6 oshop-# ;  
7  
8 kunde_id  
9 -----  
10      1
```

Wie bei UNION werden Dubletten hier unterdrückt. Mit INTERSECT ALL würden ggf. vorhandene Dubletten ausgegeben werden. Die Anforderungen an die einzelnen SELECT sind wie bei einem UNION. Alle SELECT müssen die gleiche Anzahl von Spalten mit den gleichen Reihenfolgen von Datentypen haben.

14.2.2 Mit Unterabfragen

In MySQL und MariaDB muss dies mittels einer Unterabfrage emuliert werden.

Schritt 1 – Unterabfrage ermitteln: Wir brauchen alle Kundennummern, die im Jahr 2011 eine Rechnung hatten:

Natürlich haben Sie sofort und im Kopf folgende Lösung gebaut:

```
1 SELECT kunde_id  
2   FROM rechnung  
3 WHERE YEAR(datum) = '2011' AND YEAR(datum) = '2012';
```



Aufgabe 14.2: Warum ist das keine Lösung, auch nicht mit OR? Verwenden Sie die [Definition 35 auf Seite 142](#).

Nein, wir brauchen Mengenoperationen, daher also Schritt 1:

```
1 SELECT kunde_id FROM rechnung WHERE YEAR(datum) = '2011';
```

Schritt 2 – Bau der Oberanweisung ohne Unterabfrage: Die sieht fast gleich aus. Nur dass hier mit AND das *Enthaltensein* in einer Liste verknüpft wird. Die Werte 1,2,3 sind hier nur willkürlich, um den Befehl ausprobieren zu können.

Ein weiterer Punkt ist das DISTINCT. Es verhindert die Mehrfachausgabe von Kundennummern, schließlich könnte ein Kunde ja mehrfach in beiden Jahren etwas gekauft haben.

```
1 SELECT DISTINCT kunde_id  
2   FROM rechnung  
3 WHERE YEAR(datum) = '2012' AND kunde_id IN (1, 2, 3);
```

Schritt 3 – Einbau der Unterabfrage in die Oberanweisung: In das Klammerpaar von IN() wird jetzt die Unterabfrage eingebaut.

```
1 mysql> SELECT DISTINCT kunde_id  
2      ->   FROM rechnung  
3      ->   WHERE  
4      ->   YEAR(datum) = '2012' AND kunde_id IN  
5      ->   (  
6      ->     SELECT kunde_id FROM rechnung WHERE YEAR(datum) = '2011'  
7      ->   )  
8      -> ;  
9 +-----+  
10 | kunde_id |  
11 +-----+  
12 |       1 |  
13 +-----+
```

In [Bild 14.1 auf der nächsten Seite](#) ist der Umbau grafisch dargestellt.

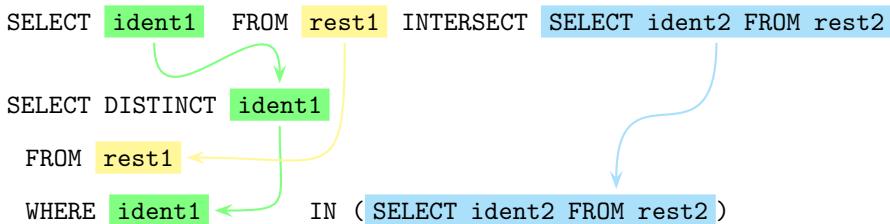


Bild 14.1 Umbau von INTERSECT in MySQL und MariaDB



Aufgabe 14.3: Was muss man hier weglassen, um einen INTERSECT ALL zu emulenzen?

Aufgabe 14.4: Ermitteln Sie alle Adressen, die sowohl Rechnungs- als Lieferadresse sind.

■ 14.3 Die Differenzmenge

Die Zeilen, die in einer, aber nicht in einer anderen Tabelle vorkommen, nennt man die Differenzmenge zweier Tabellen. Wie bei `INTERSECT` kennen MySQL und MariaDB keinen entsprechenden Operator, aber man kann das Verhalten mit einer Unterabfrage emulieren. Anders als bei der Vereinigung oder dem Durchschnitt zweier Tabellen ist hier die Reihenfolge der Tabellen wichtig. Es ist ein Unterschied, ob Sie den Inhalt der Tabelle A von der Tabelle B abziehen oder umgekehrt. Ziehen Sie von allen Hauptstädten der Welt die europäischen ab, erhalten Sie alle außereuropäischen Hauptstädte. Ziehen von allen europäischen Hauptstädten alle Hauptstädte der Welt ab, erhalten Sie eine leere Menge!

14.3.1 Mit EXCEPT

SQL:2016	PostgreSQL	T-SQL
<pre>SELECT auswahl [EXCEPT [ALL] SELECT auswahl]* ;</pre>	<pre>SELECT auswahl [EXCEPT [ALL DISTINCT] SELECT auswahl]* ;</pre>	<pre>SELECT auswahl [EXCEPT SELECT auswahl]* ;</pre>

Ich möchte Folgendes wissen: Welche Adressen sind keine Lieferadressen (Beispiel in PostgreSQL ausgeführt)?

```
1 oshop=# SELECT strasse, hnr, lkz, plz, ort  FROM adresse
2 oshop=# EXCEPT
3 oshop=# SELECT strasse, hnr, lkz, plz, ort
4 oshop=#   FROM kunde INNER JOIN adresse ON liefer_adresse_id = adresse_id
```

```

5 oshop-# ;
6      strasse | hnr | lkz |     plz |     ort
7 -----
8 Hochstrasse | 4a | DE | 44879 | Bochum
9 Stockstraße | 1  | AL | 45689 | Buckelstadt
10 Industriegebiet | 8 | DE | 44878 | Bochum
11 Brandyschloss | 1  | BL | 57990 | Bockenburg
12 Auf der Feste | 1  | GO | 54786 | Minas Tirith
13 Bocklandstraße | 4  | BL | 57996 | Krickloch
14 Baradur | 1  | MO | 62519 | Lugburz
15 Brandyschloss | 1a | BL | 57990 | Bockenburg
16 Beutelhaldenweg | 5 | AL | 67676 | Hobbingen
17 Maggots Weg | 2  | BL | 57980 | Maggots Hof
18 Letztes Haus | 4  | ER | 87567 | Bruchtal
19 (11 Zeilen)

```

Bitte beachten Sie, dass der Beutelhaldenweg 1 fehlt.

Wie bei UNION werden Dubletten hier unterdrückt. Mit EXCEPT ALL würden ggf. vorhandene Dubletten ausgegeben werden. Die Anforderungen an die einzelnen SELECT ist wie bei einem UNION. Alle SELECT müssen die gleiche Anzahl von Spalten mit den gleichen Reihenfolgen von Datentypen haben.

14.3.2 Mit Unterabfragen

Auch den fehlenden EXCEPT kann man in MySQL und MariaDB emulieren. Als Beispiel wollen wir alle Neukunden des Jahres 2012.

Schritt 1 – Unterabfrage ermitteln: Wir brauchen alle Kundennummern, die im Jahr 2011 eine Rechnung hatten:

Natürlich haben Sie wieder sofort und im Kopf folgende Lösung gebaut:

```

1 SELECT kunde_id
2   FROM rechnung
3 WHERE YEAR(datum) <> '2011' AND YEAR(datum) = '2012';

```



Aufgabe 14.5: Warum ist das keine Lösung, auch nicht mit OR? Verwenden Sie die [Definition 35 auf Seite 142](#).

Nein, wir brauchen schon eine Mengenoperationen, daher also Schritt 1:

```
1 SELECT kunde_id FROM rechnung WHERE YEAR(datum) = '2011';
```

Schritt 2 – Bau der Oberanweisung ohne Unterabfrage: Die sieht fast gleich aus wie beim INTERSECT. Der einzige Unterschied ist das NOT vor dem IN:

```

1 SELECT DISTINCT kunde_id
2   FROM rechnung
3 WHERE
4   YEAR(datum) = '2012' AND kunde_id NOT IN (1, 2, 3);

```

Schritt 3 – Einbau der Unterabfrage in die Oberanweisung: Wie beim INTERSECT wird die Unterabfrage in die Klammerung des IN() eingefügt.

```
1 mysql> SELECT DISTINCT kunde_id
```

```

2      -> FROM rechnung
3      -> WHERE
4      -> YEAR(datum) = '2012' AND kunde_id NOT IN
5      -> (
6          ->     SELECT kunde_id FROM rechnung WHERE YEAR(datum) = '2011'
7      -> )
8      -> ;
9 +-----+
10 | kunde_id |
11 +-----+
12 |      2   |
13 |      3   |
14 |      5   |
15 +-----+

```

In Bild 14.2 ist der Umbau grafisch dargestellt.

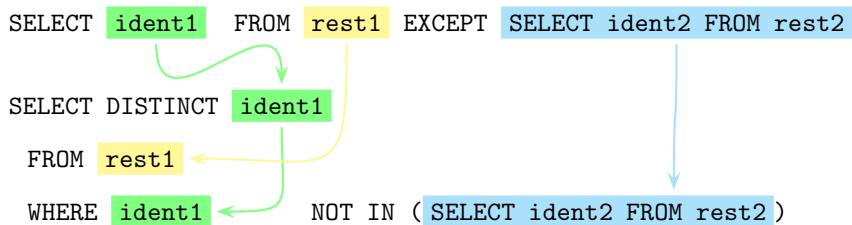


Bild 14.2 Umbau von EXCEPT in MySQL und MariaDB



Aufgabe 14.6: Was muss man hier weglassen, um den Befehl EXCEPT ALL zu emulieren?

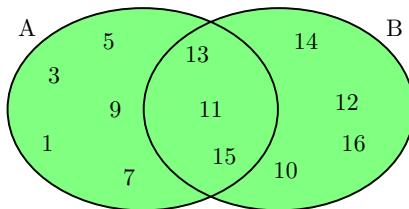
■ 14.4 UNION, INTERSECT und EXCEPT ... versteh' ich nicht!

Ich werde nach einer formalen Definition versuchen, anhand einer Grafik die jeweilige Operation zu verdeutlichen.



Definition 53: UNION

Die Menge aller Zeilen $z \in U$ nennt man *UNION* oder *Vereinigung* von A und B , wenn Folgendes gilt: $U = \{z \mid z \in A \vee z \in B\}$. Kurzschreibweise: $U = A \cup B$



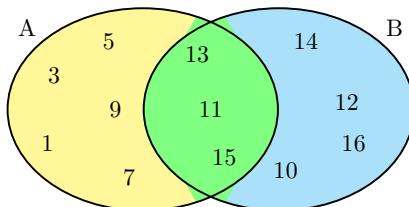
Die Vereinigung liefert mir also alle Zeilen, egal aus welcher Tabelle sie stammen, zurück. In unserem Beispiel wäre $U = \{1, 3, 5, 7, 9, 10, 11, 12, 13, 14, 15, 16\}$. Bei einer Vereinigung kann es vorkommen, dass es Zeilen gibt, die sowohl in der Tabelle A als auch in der Tabelle B enthalten sind. Mit der Option ALL werden diese Zeilen mehrfach ausgegeben, ansonsten nicht.

Wenn wir uns jetzt noch einmal die Ergebnisse von Seite 251 ff. anschauen, dann sind dies immer alle Zeilen, die durch die SELECT vor und nach dem UNION erzeugt wurden.



Definition 54: INTERSECT

Die Menge aller Zeilen $z \in I$ nennt man *INTERSECT* oder *Schnittmenge* von A und B, wenn Folgendes gilt: $I = \{z | z \in A \wedge z \in B\}$. Kurzschreibweise: $I = A \cap B$



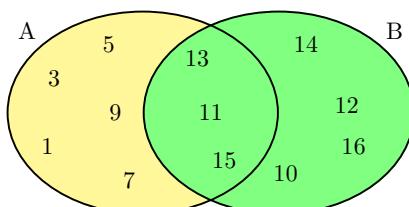
Die Schnittmenge liefert mir nur die Zeilen, die sowohl in der Tabelle A als auch in der Tabelle B vorkommen. In unserem Beispiel wäre $I = \{11, 13, 15\}$. Auch hier können mehrere Zeilen in beiden Tabellen vorkommen. Die Option ALL liefert jede Zeile ggf. mehrfach.

Wenn wir uns das Ergebnis auf Seite 254 ff. anschauen, kommt dort nur die Zeile vor, die sowohl vor dem INTERSECT als auch danach durch die SELECT erzeugt wurde.



Definition 55: EXCEPT

Die Menge aller Zeilen $z \in E$ nennt man *EXCEPT* oder *Differenzmenge* von A ohne B, wenn Folgendes gilt: $E = \{z | z \in A \wedge z \notin B\}$. Kurzschreibweise: $E = A \setminus B$



Die Differenzmenge liefert mir nur die Zeilen, die in der Tabelle A , aber nicht in der Tabelle B vorkommen. In unserem Beispiel wäre $E = \{1, 3, 5, 7, 9\}$. Auch hier können mehrere Zeilen in Tabelle A vorkommen. Die Option ALL liefert jede Zeile ggf. mehrfach.

Auf den Seiten 256ff. werden nur die Zeilen ausgegeben, die ausschließlich durch das SELECT vor dem INTERSECT ermittelt wurden.



Hinweis: Anders als bei UNION und INTERSECT ist die Reihenfolge hier bedeutsam:

$$E_1 = A \setminus B = \{1, 3, 5, 7, 9\}, \text{ aber } E_2 = B \setminus A = \{10, 12, 14, 16\}$$

■

Zur Beruhigung möchte ich anmerken, dass der UNION häufiger vorkommt, aber INTERSECT und EXCEPT relativ selten. Natürlich kommt es auf die Anwendung an, aber für *stink-normale* Geschäftsprozesse und Auswertungen braucht man sie fast nie.

15

Bedingungslogik



Fallunterscheidungen fallen einem nur von Fall zu Fall ein!

- Vertiefendes
 - Problem und Lösung
 - CASE mit Test auf Gleichheit
 - CASE mit beliebigen Bedingungen



Die Quelltexte dieses Kapitels stehen in der Datei `listing12.sql` (siehe [Listing 29.12 auf Seite 502](#), [Listing 29.48 auf Seite 625](#) und [Listing 29.33 auf Seite 572](#)).

■ 15.1 Warum ein CASE?

Wir haben zwei Kundenarten: Geschäftskunden und Privatkunden. Diese Information ist in einem ENUM der Tabelle `kunde` abgelegt:

```
1 mysql> SELECT kunde_id, nachname, vorname, art FROM kunde;
2 +-----+-----+-----+-----+
3 | kunde_id | nachname      | vorname     | art   |
4 +-----+-----+-----+-----+
5 |       1 | Gamdschie    | Samweis     | prv   |
6 |       2 | Beutlin       | Frodo       | prv   |
7 |       3 | Beutlin       | Bilbo       | prv   |
8 |       4 | Telcontar     | Elessar     | prv   |
9 |       5 | Earendillionn | Elrond      | gsch  |
10 |      6 | Eichenschild  | Thorin     | unb   |
11 [...]
12 |     22 | Tuk          | Peregrin   | unb   |
13 +-----+-----+-----+-----+
14 22 rows in set (0.00 sec)
```

Die abkürzende Schreibweise des ENUM ist nicht selbsterklärend oder präsentabel. Wir wollen, dass die Wörter `Privatkunde`, `Geschäftskunde` und `Unbekannt` ausgegeben werden. Eine Möglichkeit wäre die Anlage einer temporären Tabelle und einer Verknüpfung:

```

1 mysql> CREATE TEMPORARY TABLE tmp_art
2     -> (
3         -> art VARCHAR(255),
4         -> k_art VARCHAR(255)
5     );
6
7 mysql> INSERT INTO tmp_art
8     -> VALUES
9     -> ('prv', 'Privatkunde')
10    -> ,('gsch', 'Geschäftskunde')
11    -> ,('unb', 'Unbekannt');
12
13 mysql> SELECT kunde_id, nachname, vorname, k_art
14     -> FROM
15     -> kunde INNER JOIN tmp_art USING(art);
16
17 +-----+-----+-----+
18 | kunde_id | nachname      | vorname      | k_art
19 +-----+-----+-----+
20 |       1 | Gamdschie     | Samweis      | Privatkunde
21 |       2 | Beutlin        | Frodo        | Privatkunde
22 |       3 | Beutlin        | Bilbo        | Privatkunde
23 |       4 | Telcontar      | Elessar      | Privatkunde
24 |       5 | Earendillionn | Elrond       | Geschäftskunde
25 |       6 | Eichenschield  | Thorin      | Unbekannt
26 [...]
27 |      22 | Tuk           | Peregrin    | Unbekannt
28 +-----+-----+-----+
29 22 rows in set (0.00 sec)

```

Klappt ja prima, aber geht das nicht einfacher? Klar – mit vielen UNIONs und einer schönen Unterabfrage:

```

1 mysql> SELECT kunde_id, nachname, vorname, k_art
2     -> FROM
3     -> kunde INNER JOIN
4     -> (
5         -> SELECT 'prv' AS art, 'Privatkunde' AS k_art
6     -> UNION
7         -> SELECT 'gsch' AS art, 'Geschäftskunde' AS k_art
8     -> UNION
9         -> SELECT 'unb' AS art, 'Unbekannt' AS k_art
10    -> ) t
11    -> USING(art)
12    -> ;
13
14 +-----+-----+-----+
15 | kunde_id | nachname      | vorname      | k_art
16 +-----+-----+-----+
17 |       1 | Gamdschie     | Samweis      | Privatkunde
18 |       2 | Beutlin        | Frodo        | Privatkunde
19 |       3 | Beutlin        | Bilbo        | Privatkunde
20 |       4 | Telcontar      | Elessar      | Privatkunde
21 |       5 | Earendillionn | Elrond       | Geschäftskunde
22 |       6 | Eichenschield  | Thorin      | Unbekannt
23 [...]
24 |      22 | Tuk           | Peregrin    | Unbekannt
25 +-----+-----+-----+
26 22 rows in set (0.01 sec)

```

Warum soll das einfacher sein? Es muss keine temporäre Tabelle angelegt werden. Auch der separate `INSERT INTO` entfällt. Die Unterabfrage in den [Zeilen 4 bis 10](#) erzeugt durch `UNION` eine *temporäre Tabelle*, die durch die Vergabe der Spaltenalias auch spaltenweise angesprochen werden kann. Der `INNER JOIN` verwendet hier kein Primär-Fremdschlüsselpaar, sondern die Textgleichheit in der Spalte `art`.

Sie glauben, das war's? Nö:

```

1 mysql> SELECT kunde_id, nachname, vorname,
2       -> CASE art
3       -> WHEN 'prv' THEN 'Privatkunde'
4       -> WHEN 'gsch' THEN 'Geschäftskunde'
5       -> ELSE 'Unbekannt'
6       -> END AS k_art
7       -> FROM
8       -> kunde
9       -> ;
10 +-----+-----+-----+
11 | kunde_id | nachname      | vorname     | k_art      |
12 +-----+-----+-----+
13 |      1 | Gamdschie    | Samweis     | Privatkunde |
14 |      2 | Beutlin       | Frodo       | Privatkunde |
15 |      3 | Beutlin       | Bilbo       | Privatkunde |
16 |      4 | Telcontar     | Elessar     | Privatkunde |
17 |      5 | Earendillionn | Elrond      | Geschäftskunde |
18 |      6 | Eichenschield | Thorin     | Unbekannt   |
19 [...]
20 |     22 | Tuk          | Peregrin    | Unbekannt   |
21 +-----+-----+-----+
22 22 rows in set (0.01 sec)

```

In [Zeile 2](#) beginnt eine Fallunterscheidung. Der Inhalt der Spalte `art` wird untersucht, indem er mit einer Textkonstanten verglichen wird. Trifft der Vergleich zu, wird dieser Teil ausgeführt. Trifft keine der Bedingungen zu, wird der `ELSE` in [Zeile 5](#) ausgeführt. Es ist guter Programmierstil, immer einen `ELSE` zu haben, damit die Abfrage robust gegenüber unbekannten Situationen ist.

■ 15.2 Einfacher CASE



SQL:2016, MySQL/MariaDB, PostgreSQL, T-SQL

```

CASE {spaltenname|ausdruck}
  WHEN {spaltenname|ausdruck} THEN anweisungen
  [WHEN {spaltenname|ausdruck} THEN anweisungen]*
  [ELSE anweisungen]
END

```

Der `{spaltenname|ausdruck}` hinter dem `CASE` wird immer auf Gleichheit mit dem `{spaltenname|ausdruck}` hinter dem `WHEN` getestet. Unser einführendes Beispiel ist somit ein einfacher CASE. Die Spalte `art` wird auf Gleichheit mit der Textkonstanten getestet.

Die Ausgabe der Währungen soll durch das Währungssymbol ersetzt werden:

```

1 mysql> SELECT artikel_id, bezeichnung,
2      -> CASE waehrung
3      -> WHEN 'EUR' THEN CONCAT(FORMAT(einzelpreis, 2, 'de_DE'), ' €')
4      -> WHEN 'USD' THEN CONCAT(FORMAT(einzelpreis, 2, 'de_DE'), ' $')
5      -> ELSE '?????'
6      -> END AS preis
7      -> FROM artikel
8      -> ;
9 +-----+-----+-----+
10 | artikel_id | bezeichnung | preis |
11 +-----+-----+-----+
12 | 3001 | Papier (100) | 2,32 € |
13 | 3005 | Tinte (gold) | 56,26 € |
14 | 3006 | Tinte (rot) | 6,26 € |
15 | 3007 | Tinte (blau) | 4,17 € |
16 | 3010 | Feder | 5,05 € |
17 | 7856 | Silberzwiebel | 0,52 € |
18 | 7863 | Tulpenzwiebel | 3,42 € |
19 | 9010 | Schaufel | 15,10 $ |
20 | 9015 | Spaten | 20,10 € |
21 +-----+-----+-----+

```

Die Funktion `FORMAT()` bastelt mir den Preis ein wenig nach meinem Geschmack um. Der erste Parameter ist die zu formatierende Zahl. Der zweite Parameter gibt die Anzahl der Nachkommastellen an, und der dritte optionale Parameter formatiert Länderspezifisches, und zwar hier das Komma anstelle des Punkts.

Auch bei diesem Beispiel habe ich ein `ELSE` eingebaut. Was würde eigentlich passieren, wenn es kein solches `ELSE` gäbe, aber kein passender Eintrag gefunden wird? Probieren wir es aus:

```

1 mysql> UPDATE artikel SET waehrung = 'XYZ' WHERE artikel_id = 3001;
2
3 mysql> SELECT artikel_id, bezeichnung,
4      -> CASE waehrung
5      -> WHEN 'EUR' THEN CONCAT(FORMAT(einzelpreis, 2, 'de_DE'), ' €')
6      -> WHEN 'USD' THEN CONCAT(FORMAT(einzelpreis, 2, 'de_DE'), ' $')
7      -> END AS preis
8      -> FROM artikel ;
9 +-----+-----+-----+
10 | artikel_id | bezeichnung | preis |
11 +-----+-----+-----+
12 | 3001 | Papier (100) | NULL |
13 | [...] |
14 +-----+-----+-----+

```

Was hätte auch sonst dort stehen sollen? Einzige Alternative wäre eine Fehlermeldung gewesen. Das Problem ist aber nun, dass es nicht mehr unterscheidbar ist, ob hier schon vorher `NULL` stand oder nicht. Es ist deshalb immer besser, über das `ELSE` ein unmissverständliches Signal abzusetzen: *Hier stimmt etwas nicht!*

■ 15.3 SEARCHED CASE



SQL:2016, MySQL/MariaDB, PostgreSQL, T-SQL

CASE

```
WHEN bedingung THEN anweisungen
[WHEN bedingung THEN anweisungen]*
[ELSE anweisungen]
END
```

Der Unterschied in der Syntax ist nicht so groß, aber die dadurch entstehenden Möglichkeiten. Hinter dem CASE steht nichts. Stattdessen ist die gesamte Abfrage in die *bedingung* hinter dem WHEN verlagert worden. Die Bedingung kann nun beliebig kompliziert sein. Sie kann sogar eigene Unterabfragen enthalten.

Aber erst einmal langsam; wir fangen mit einem einfachen Beispiel an. Wir wollen wissen, ob ein Eintrag in der Tabelle *beitrag* eine Antwort ist oder nicht.

```
1 mysql> SELECT beitrag_id,
2       -> CASE
3       -> WHEN bezug_beitrag_id > 1 THEN 'Antwort'
4       -> WHEN bezug_beitrag_id <= 1 THEN 'Keine Antwort'
5       -> ELSE '??????'
6       -> END AS Typ
7       -> FROM beitrag
8       -> ORDER BY beitrag_id
9       -> ;
10 +-----+-----+
11 | beitrag_id | Typ      |
12 +-----+-----+
13 |          1 | Keine Antwort |
14 |          2 | Keine Antwort |
15 |          3 | Antwort      |
16 |          4 | Antwort      |
17 |          5 | Antwort      |
18 |          6 | Keine Antwort |
19 +-----+-----+
```

Hinter dem CASE fehlt ein Spaltenname oder ein Ausdruck ([Zeile 2](#)). Hinter dem WHEN wird als Ersatz für den Test auf Gleichheit eine komplette Bedingung angegeben ([Zeile 3](#)).

Jetzt wollen wir das Ganze etwas erweitern. Wir wollen die Nachrichten sehen, und falls es sich um eine Antwort handelt, soll der referenzierte Beitrag angegeben werden. Eine Sonderstellung hat der Beitrag 1. Er dient als Dummy-Referenz für Beiträge, die keine Antworten sind.

```
1 mysql> SELECT beitrag_id,
2       -> CASE
3       -> WHEN bezug_beitrag_id > 1
4       ->     THEN CONCAT(nachricht, ' Antwort auf ', bezug_beitrag_id, ': >',
5       ->     (
6       ->       SELECT nachricht
7       ->         FROM beitrag b2
8       ->         WHERE b2.beitrag_id = b1.bezugeitrag_id
9       ->     )
10      ->       ) -- Ende CONCAT
11      -> WHEN bezug_beitrag_id <= 1 THEN nachricht
```

```

12      -> ELSE '??????'
13      -> END AS Inhalt
14      -> FROM beitrag b1
15      -> WHERE beitrag_id > 1
16      -> ;
17 +-----+-----+-----+-----+-----+-----+
18 | beitrag_id | Inhalt |           |
19 +-----+-----+-----+-----+-----+-----+
20 |          2 | Der Lieferservice ist super. |           |
21 |          3 | Das finde ich auch. Antwort auf 2: >Der Lieferserv[...] |           |
22 |          4 | Aber ein wenig langsam. Antwort auf 2: >Der Liefer[...] |           |
23 |          5 | Finde ich nicht. Antwort auf 4: >Aber ein wenig la[...] |           |
24 |          6 | Angebot könnte besser sein. |           |
25 +-----+-----+-----+-----+-----+-----+

```

In Zeile 15 wird der Beitrag 1 ausgefiltert. Der erste Fall in Zeile 4 tritt ein, wenn der Datensatz eine Antwort ist. In der entsprechenden Unterabfrage wird die Nachricht ermittelt, auf die geantwortet wurde. Der zweite Fall in Zeile 11 gibt mir nur den Nachrichtentext der aktuellen Nachricht aus, da er keine Antwort auf irgendeinen anderen Beitrag ist.

Dadurch, dass jeder Fall seine eigene Bedingung hat, kann es vorkommen, dass für mehr als ein WHEN die Bedingung wahr wird. Man stelle sich zum Beispiel vor, dass zweimal die gleiche Bedingung da steht. Stellt sich doch die Frage: Werden dann beide Fälle ausgeführt oder nur der erste?

```

1 SELECT beitrag_id,
2 CASE
3   WHEN bezug_beitrag_id >= 0 THEN 'Antwort'
4   WHEN bezug_beitrag_id >= 1 THEN 'Keine Antwort'
5   ELSE '??????'
6 END AS Typ
7 FROM beitrag
8 ORDER BY beitrag_id
9 ;

```



Aufgabe 15.1: Überlegen Sie sich vorher, was für eine Ausgabe Sie erwarten. Wie sieht die Ausgabe aus, wenn nach der ersten Bedingung, die TRUE ist, die Fallunterscheidung abbricht, und wie sieht die Ausgabe aus, wenn jede Bedingung geprüft wird?

Der SELECT erzeugt folgende Ausgabe:

```

1 +-----+-----+
2 | beitrag_id | Typ |           |
3 +-----+-----+-----+
4 |          1 | Antwort |           |
5 |          2 | Antwort |           |
6 |          3 | Antwort |           |
7 |          4 | Antwort |           |
8 |          5 | Antwort |           |
9 |          6 | Antwort |           |
10 +-----+-----+

```

Bitte denken Sie darüber nach, welche Aufgabe der CASE hat. Der Inhalt einer Spaltenausgabe soll festgelegt werden. Würde er jede Bedingung, die wahr ist, ausführen, hätte er

mehr als einen Inhalt für eine Spalte. Das ist nicht nur fachlich bedenklich, sondern auch nicht in Tabellenform darstellbar.



Hinweis: Die Fallunterscheidung bricht mit der ersten Bedingung ab, die das Ergebnis TRUE hat.

■ 15.4 Fallbeispiele

15.4.1 Lagerbestand überprüfen

Für jeden Artikel soll ausgegeben werden, ob noch genügend davon im Lager vorhanden ist. Zuerst ein Blick ins Lager:

```
1 mysql> SELECT * FROM lagerbestand;
2 +-----+-----+-----+-----+
3 | artikel_id | menge_mindest | menge_aktuell | deleted |
4 +-----+-----+-----+-----+
5 |      3001 | 1000.000000 | 5000.000000 |      0 |
6 |      3005 | 200.000000 | 250.000000 |      0 |
7 |      3006 | 200.000000 | 250.000000 |      0 |
8 |      3007 | 200.000000 | 149.000000 |      0 |
9 |      3010 | 100.000000 | 3.000000 |      0 |
10 |     7856 | 100.000000 | 500.000000 |      0 |
11 |     7863 | 100.000000 | 400.000000 |      0 |
12 |     9010 | 10.000000 | 30.000000 |      0 |
13 |     9015 | 10.000000 | 25.000000 |      0 |
14 +-----+-----+-----+-----+
```

Die Artikelbezeichnung werden wir über einen INNER JOIN zur Tabelle artikel ermitteln müssen. Ob der Lagerbestand ausreicht, kann anhand der Mengenspalten ermittelt werden. Ist die aktuelle Menge kleiner oder gleich der Mindestmenge, dann muss der Artikel nachbestellt werden, ansonsten nicht.

```
1 mysql> SELECT a.bezeichnung,
2       -> CASE
3       -> WHEN l.menge_aktuell <= l.menge_mindest
4       -> THEN 'Artikel nachbestellen'
5       -> ELSE 'Bestand ausreichend'
6       -> END AS lagerstand
7       -> FROM lagerbestand l INNER JOIN artikel a USING(artikel_id)
8       -> ORDER BY l.menge_aktuell / l.menge_mindest
9       -> ;
10 +-----+-----+
11 | bezeichnung | lagerstand |
12 +-----+-----+
13 | Feder | Artikel nachbestellen |
14 | Tinte (blau) | Artikel nachbestellen |
15 | Tinte (gold) | Bestand ausreichend |
16 | Tinte (rot) | Bestand ausreichend |
17 | Spaten | Bestand ausreichend |
18 | Schaufel | Bestand ausreichend |
19 | Tulpenzwiebel | Bestand ausreichend |
```

```

20 | Papier (100) | Bestand ausreichend |
21 | Silberzwiebel | Bestand ausreichend |
22 +-----+-----+

```

Die Sortierung hat sich am Bedarf orientiert. Der Artikel, der am meisten die Mindestmenge unterschreitet, steht oben.

15.4.2 Kundengruppen ermitteln

Wir wollen unseren Kunden unterschiedliche Rabatte einräumen. Wer weniger als zwei Rechnungen hat, gilt als *Kleinkunde*, wer drei hat, als *Guter Kunde* und wer mehr als drei hat, als *Premiumkunde*.

```

1 mysql> SELECT DISTINCT k.kunde_id, k.nachname, k.vorname,
2      -> CASE
3      -> WHEN 3 <
4      -> (
5      ->   SELECT COUNT(rechnung_id) FROM rechnung r
6      ->   WHERE r.kunde_id = k.kunde_id
7      -> ) THEN 'Premiumkunde'
8      -> WHEN 2 <
9      -> (
10     ->   SELECT COUNT(rechnung_id) FROM rechnung r
11     ->   WHERE r.kunde_id = k.kunde_id
12     -> ) THEN 'Guter Kunde'
13     -> ELSE 'Kleinkunde'
14     -> END kundenart
15     -> FROM
16     -> rechnung r RIGHT OUTER JOIN kunde k USING(kunde_id)
17     -> ;
18 +-----+-----+-----+-----+
19 | kunde_id | nachname      | vorname    | kundenart |
20 +-----+-----+-----+-----+
21 |       3 | Beutlin        | Bilbo      | Kleinkunde |
22 [...] 
23 |       17 | Brandybock     | Saradoc    | Kleinkunde |
24 |       5 | Earendillionn | Elrond     | Guter Kunde |
25 |       6 | Eichenschield  | Thorin     | Kleinkunde |
26 |       1 | Gamdschie      | Samweis    | Premiumkunde |
27 |       10 | Goldwert        | Hanna      | Kleinkunde |
28 |       16 | Guld           | Menegilda  | Kleinkunde |
29 [...] 
30 |       22 | Tuk            | Peregrin   | Kleinkunde |
31 +-----+-----+-----+-----+

```



Aufgabe 15.2: Sind die beiden Unterabfragen in den Zeilen 4 und 9 korrelierende oder nicht korrelierende Unterabfragen?

Die Unterabfragen ermitteln die Anzahl der Rechnungen, die ein Kunde hat. Da es sich um eine skalare Unterabfrage handelt, kann das Ergebnis direkt mit einer Zahl verglichen werden. Falls keine der beiden Bedingungen zutrifft, muss es sich um einen Kleinkunden handeln.



Aufgabe 15.3: Warum wird hier ein RIGHT OUTER JOIN verwendet? Was will ich damit erreichen?

Wenn man so recht darüber nachdenkt, ist eine Kategorisierung nach den Rechnungsbeträgen sinnvoller, als nach der Anzahl der Rechnungen zu gehen.

```

1 mysql> SELECT k.kunde_id, k.nachname, k.vorname,
2      -> (
3      ->     SELECT SUM(a.einzelpreis * rp.menge)
4      ->       FROM rechnung r INNER JOIN rechnung_position rp
5      ->             USING (rechnung_id)
6      ->           INNER JOIN artikel a
7      ->             USING (artikel_id)
8      -> WHERE r.kunde_id = k.kunde_id
9      -> ) rechnungswert,
10     -> CASE
11     -> WHEN
12     ->     kunde_id NOT IN (SELECT kunde_id FROM rechnung)
13     ->     THEN 'Kleinkunde'
14     -> WHEN
15     -> (
16     ->     SELECT SUM(a.einzelpreis * rp.menge) rechnungswert
17     ->       FROM rechnung r INNER JOIN rechnung_position rp
18     ->             USING (rechnung_id)
19     ->           INNER JOIN artikel a
20     ->             USING (artikel_id)
21     -> WHERE r.kunde_id = k.kunde_id
22     -> ) BETWEEN 0 AND 300 THEN 'Kleinkunde'
23     -> WHEN
24     -> (
25     ->     SELECT SUM(a.einzelpreis * rp.menge) rechnungswert
26     ->       FROM rechnung r INNER JOIN rechnung_position rp
27     ->             USING (rechnung_id)
28     ->           INNER JOIN artikel a
29     ->             USING (artikel_id)
30     -> WHERE r.kunde_id = k.kunde_id
31     -> ) BETWEEN 300 AND 1000 THEN 'Guter Kunde'
32     -> ELSE 'Prämiumkunde'
33     -> END AS kundenart
34     -> FROM kunde k
35     -> ORDER BY
36     -> (
37     ->     SELECT SUM(a.einzelpreis * rp.menge) rechnungswert
38     ->       FROM rechnung r INNER JOIN rechnung_position rp
39     ->             USING (rechnung_id)
40     ->           INNER JOIN artikel a
41     ->             USING (artikel_id)
42     -> WHERE r.kunde_id = k.kunde_id
43     -> ) DESC
44     -> ;
45 +-----+-----+-----+-----+-----+
46 | kunde_id | nachname    | vorname    | rechnungswert | kundenart   |
47 +-----+-----+-----+-----+-----+
48 |      5 | Earendillionn | Elrond     | 1212.550000000000 | Prämiumkunde |
49 |      1 | Gamdschie    | Samweis    | 761.110000000000 | Guter Kunde  |
50 |      3 | Beutlin       | Bilbo      | 115.850000000000 | Kleinkunde   |
51 |      2 | Beutlin       | Frodo      | 80.700000000000 | Kleinkunde   |

```

```

52 |      4 | Telcontar    | Elessar     |           NULL | Kleinkunde   |
53 [...]          |              |             |           NULL | Kleinkunde   |
54 |      22 | Tuk         | Peregrin   |           NULL | Kleinkunde   |
55 +-----+-----+-----+-----+-----+

```

Spätestens jetzt fällt einem auf, dass die Unterabfragen zur Ermittlung der Umsätze immer die gleichen sind ([Zeilen 3, 16, 25 und 37](#)). Das war zwar auch schon bei der ersten Variante so, fiel aber nicht ins Gewicht¹. Eine sinnvolle Maßnahme wäre hier das Vorschalten einer temporären Tabelle.

```

1 mysql> CREATE TEMPORARY TABLE tmp_umsatz
2   -> SELECT kunde_id, SUM(a.einzelpreis * rp.menge) rechnungswert
3   ->   FROM rechnung r INNER JOIN rechnung_position rp USING (rechnung_id)
4   ->           INNER JOIN artikel a USING (artikel_id)
5   -> GROUP BY kunde_id
6   -> ;
7
8 mysql> SELECT k.kunde_id, k.nachname, k.vorname, tmp.rechnungswert,
9   -> CASE
10  -> WHEN
11  ->     tmp.kunde_id IS NULL THEN 'Kleinkunde'
12  -> WHEN
13  ->     rechnungswert BETWEEN 0 AND 300 THEN 'Kleinkunde'
14  -> WHEN
15  ->     rechnungswert BETWEEN 300 AND 1000 THEN 'Guter Kunde'
16  -> ELSE 'Prämiumkunde'
17  -> END AS kundenart
18  -> FROM kunde k LEFT JOIN tmp_umsatz tmp USING(kunde_id)
19  -> ORDER BY rechnungswert DESC
20  -> ;
21 +-----+-----+-----+-----+-----+
22 | kunde_id | nachname | vorname | rechnungswert | kundenart |
23 +-----+-----+-----+-----+-----+
24 |      5 | Earendillion | Elrond | 1212.550000000000 | Prämiumkunde |
25 |      1 | Gamdschie   | Samweis | 761.110000000000 | Guter Kunde  |
26 |      3 | Beutlin      | Bilbo  | 115.850000000000 | Kleinkunde   |
27 |      2 | Beutlin      | Frodo  | 80.700000000000 | Kleinkunde   |
28 |      7 | Brandybock  | Gormadoc |                 NULL | Kleinkunde   |
29 [...]          |              |          |           NULL | Kleinkunde   |
30 |      18 | Tuk         | Esmeralda |           NULL | Kleinkunde   |
31 +-----+-----+-----+-----+-----+

```

In [Zeile 1](#) wird eine temporäre Tabelle aus einem SELECT heraus definiert. Diese Technik haben wir jetzt schon einige Male verwendet, und sie sollte mittlerweile zum Handwerkzeug dazu gehören. Der Inhalt der temporären Tabelle tmp_umsatz entspricht genau den oben mehrfach ausgeführten Unterabfragen; sie wird lediglich um die kunde_id erweitert, damit man die temporäre Tabelle verknüpfen kann.

Jetzt wird die Auswertung durchgeführt, indem zuerst die Kundentabelle mit der temporären Tabelle verknüpft wird ([Zeile 18](#)). Da der Rechnungswert pro Kunde nun schon vorliegt, muss dieser nicht n -fach berechnet werden, sondern kann durch eine Verknüpfung sofort in den Bedingungen ([Zeilen 10 und 12](#)), der Sortieranweisung ([Zeile 19](#)) und der Spaltenausgabe ([Zeile 8](#)) verwendet werden.

¹ Ich meine hier natürlich syntaktisch. Vom Laufzeitverhalten will ich erst gar nicht sprechen, werden doch die korrelierenden Unterabfragen n -fach wiederholt.



Aufgabe 15.4: Warum muss in Zeile 18 ein LEFT OUTER JOIN verwendet werden? Welcher Datensatz würde in der Ausgabe bei einem INNER JOIN fehlen?

Hier hat sich die Anlage einer temporären Tabelle wirklich gelohnt, aber ... können Sie sich noch an CTEs (siehe [Abschnitt 11.5 auf Seite 208](#)) erinnern? Es geht noch einfacher:

```

1 WITH tmp_umsatz (kunde_id, rechnungswert)
2 AS (
3   SELECT kunde_id, SUM(a.einzelpreis * rp.menge) rechnungswert
4   FROM rechnung r INNER JOIN rechnung_position rp USING (rechnung_id)
5   INNER JOIN artikel a USING (artikel_id)
6   GROUP BY kunde_id
7 )
8 SELECT k.kunde_id, k.nachname, k.vorname, tmp.rechnungswert,
9 CASE
10 WHEN tmp.kunde_id IS NULL THEN 'Kleinkunde'
11 WHEN rechnungswert BETWEEN 0 AND 300 THEN 'Kleinkunde'
12 WHEN rechnungswert BETWEEN 300 AND 1000 THEN 'Guter Kunde'
13 ELSE 'Premiumkunde'
14 END AS kundenart
15 FROM kunde k LEFT JOIN tmp_umsatz tmp USING(kunde_id)
16 ORDER BY rechnungswert DESC
17 ;

```

Diese Lösung ist ein gutes Beispiel für den Einsatz von CTEs. Man spart sich das explizite Erstellen einer temporären Tabelle und kann trotzdem auf dynamisch ermittelte Inhalte mehrfach zugreifen.

15.4.3 Aktive Lieferanten ermitteln

Wir wollen wissen, welche Lieferanten noch aktiv sind und welche nicht. Dazu reicht es aus nachzuschauen, ob der Primärschlüssel des Lieferanten in artikel_nm_lieferant vorkommt. Es ist irrelevant, ob der Primärschlüsselwert mehrfach vorkommt, da hier nur zwischen aktiv und inaktiv unterschieden werden soll.

```

1 mysql> SELECT l.firmenname,
2      -> CASE
3      -> WHEN
4      -> l.lieferant_id IN
5      -> (SELECT lieferant_id FROM artikel_nm_lieferant) THEN 'aktiv'
6      -> ELSE 'inaktiv'
7      -> END AS status
8      -> FROM lieferant l
9      -> ORDER BY firmenname;
10 +-----+-----+
11 | firmenname          | status   |
12 +-----+-----+
13 | Bürohengst GmbH    | aktiv    |
14 | Gartenbedarf AllesGrün | aktiv    |
15 | Office International | inaktiv |
16 +-----+-----+

```

15.4.4 Aufgaben



Aufgabe 15.5: Geben Sie für alle Zeilen in der Tabelle account klartextlich an, ob es sich um einen Administrator handelt oder nicht.

Aufgabe 15.6: Geben Sie für alle Zeilen der Tabelle adresse klartextlich an, in welchem Land die Adresse liegt: AL = Auenland, DE = Deutschland, ER = Eriador, GO = Gondor, MO = Mordor.

Aufgabe 15.7: Es sollen drei Zustände ermittelt werden:

1. Die Mindestmenge ist erreicht oder unterschritten: Artikel sofort nachbestellen.
2. Die aktuelle Menge ist 130 % oder weniger als die Mindestmenge: Artikel bald nachbestellen.
3. Sonst: Bestand ausreichend.

Vergleichen Sie Ihr Ergebnis mit meinem:

bezeichnung	lagerstand
Feder	Artikel sofort nachbestellen
Tinte (blau)	Artikel sofort nachbestellen
Tinte (gold)	Artikel bald nachbestellen
Tinte (rot)	Artikel bald nachbestellen
Spaten	Bestand ausreichend
Schaufel	Bestand ausreichend
Tulpenzwiebel	Bestand ausreichend
Papier (100)	Bestand ausreichend
Silberzwiebel	Bestand ausreichend

Aufgabe 15.8: Geben Sie eine Namensliste für Kunden aus, die um die Angabe erweitert wird, ob ein Inkassoverfahren anhängig ist.

Aufgabe 15.9: Geben Sie eine Namensliste für Kunde aus, die angibt, ob er im Forum angemeldet ist.

Aufgabe 15.10: Teilen Sie die Warengruppen danach ein, wie oft diese in Rechnungen vorkommen. Es soll drei Gruppen geben: *oft*, *selten* und *nie*. Ermitteln Sie selbst anhand einer Auswertung mögliche Abgrenzungen.

Aufgabe 15.11: Geben Sie für alle Artikel den Mindestbestand im Lager aus. Falls es diesen Artikel noch nicht im Lager gibt, soll der Text *Keine Lagerdaten vorhanden* ausgegeben werden.

16

Ansichtssache



Frei nach Karl Marx: Die Ansicht bestimmt das Bewusstsein.

- Grundkurs
 - Anlegen einer CREATE VIEW
 - Löschen einer VIEW mit DROP VIEW
 - Ändern mit ALTER VIEW
 - Unterschied zwischen Tabelle und VIEW
- Vertiefendes
 - Verarbeitungsstrategien einer VIEW
 - Ändern mit CREATE OR REPLACE
 - Selektionssicht
 - Verbundsicht



Die Quelltexte dieses Kapitels stehen in der Datei `listing13.sql` (siehe [Listing 29.13 auf Seite 507](#), [Listing 29.49 auf Seite 630](#) und [Listing 29.34 auf Seite 577](#)).

■ 16.1 Was ist eine Ansicht?

Wir haben in den vorangegangenen Kapiteln immer wieder temporäre Tabellen eingesetzt. Manchmal, weil es nicht anders ging, manchmal, um die Komplexität der Auswahl zu verringern, und manchmal, um die Performance zu steigern. Temporäre Tabellen haben die Daten einer Auswahl abgespeichert und anderen Auswertungen zur Verfügung gestellt.

Ein Nachteil temporärer Tabellen ist, dass sie nur für diese Verbindung existieren und danach gelöscht werden. Das ist so, um Namenskonflikte mit anderen Verbindungen zu verhindern. Nun denken Sie sich, dem kann man mit einer Namenskonvention begegnen, zum Beispiel indem der Name der temporären Tabelle die Verbindungsnummer und/oder einen Zeitstempel enthält.

Dann kommt aber der nächste Nachteil: Die Daten veralten im Laufe der Zeit. Eine Analyse muss vorab ergeben, wie lange die Daten einer temporären Tabelle seriöserweise verwendet werden können. Handelt es sich um Bewegungsdaten, ist die Verwendbarkeit einer temporären Tabelle nach einiger Zeit nicht mehr gegeben.¹ Anders ist es bei Stammdaten. Hier kann es sehr wohl sinnvoll sein, langfristige Auswertungen vorrätig zu halten.

Eine Ansicht (VIEW) speichert nicht die Daten einer Auswahl, sondern die Konstruktionsvorschrift einer Auswahl. Ruft man die Ansicht ab, wird die Konstruktionsvorschrift so aufgerufen, als ob diese direkt eingegeben wurde.²

16.1.1 Wie wird eine Ansicht angelegt?

 SQL:2016	MySQL/MariaDB PostgreSQL	T-SQL
CREATE	CREATE [OR REPLACE]	CREATE [OR ALTER]
VIEW	VIEW	VIEW
viewname[(spaltenliste)]	viewname[(spaltenliste)]	viewname[(spaltenliste)]
AS	AS	AS
SELECT auswahl	SELECT auswahl	SELECT auswahl
; ; ;		

Wir wollen beispielsweise eine Adressliste für alle Kunden als Ansicht zur Verfügung stellen.

```

1 mysql> CREATE VIEW
2     -> view_kundenrechnungsadresse (id, nachname, vorname, strasse, ort)
3     -> AS
4     -> SELECT
5     ->     kunde_id, nachname, vorname,
6     ->     CONCAT_WS(' ', strasse, hnr),
7     ->     CONCAT_WS(' ', lkz, plz, ort)
8     -> FROM
9     ->     kunde INNER JOIN adresse
10    ->         ON rechnung_adresse_id = adresse_id
11    -> WHERE kunde.deleted = 0
12    -> ;
13
14 mysql> SELECT * FROM view_kundenrechnungsadresse;
15 +-----+-----+-----+-----+
16 | id | nachname | vorname | strasse | ort |
17 +-----+-----+-----+-----+
18 | 1 | Gamdschie | Samweis | Beutelhaldenweg 5 | AL 67676 Hobbingen |
19 | 2 | Beutlin | Frodo | Beutelhaldenweg 1 | AL 67676 Hobbingen |
20 [...]
21 | 21 | Lochner | Rufus | Maggots Weg 2 | BL 57980 Maggots Hof |
22 | 22 | Tuk | Peregrin | Stockstraße 1 | AL 45689 Buckelstadt |
23 +-----+-----+-----+-----+
24 21 rows in set (0.00 sec)
```

¹ Natürlich hängt dies von der Aktualisierungsfrequenz ab. Aber gehen wir mal davon aus, dass unsere Daten wirklich genutzt werden.

² Na ja, nicht ganz, aber dazu später mehr ...

Ab der [Zeile 3](#) wird angegeben, was für ein SELECT ausgeführt wird, wenn man diese Ansicht abfragt. Der SELECT der Ansicht ist eine eigenständig ausführbare Anweisung³. Der SELECT kann genauso einfach oder kompliziert sein wie jeder andere SELECT, sie kann sogar wiederum auf Ansichten aufbauen. Die Ansicht kapselt den SELECT, um dann für weitere Auswertungen zur Verfügung zu stehen.

In der [Zeile 2](#) steht hinter dem Schlüsselwort VIEW der *viewname* und anschließend eine *spaltenliste*. Diese Spaltenliste ist nur eine Umbenennung der Spalten aus dem SELECT. Natürlich können hier auch Ausdrücke stehen, die die Spalten aufbereiten, aber meistens stehen hier nur Umbenennungen. In unserem Fall ist das nötig, da wir sonst hässliche Spaltennamen wie CONCAT_WS(' ', strasse, hnr) hätten.



Aufgabe 16.1: Legen Sie eine zweite VIEW namens view_wurstbrot an. Lassen Sie dabei einfach die Spaltenliste weg und führen Sie dann SELECT * FROM view_wurstbrot aus.

In [Zeile 11](#) werden alle als gelöscht markierten Zeilen aussortiert. In unserem Fall ist das Ergebnis das gleiche wie ohne die WHERE-Klausel. Wir erkennen aber hier schon eine wesentliche Anwendung der VIEW. Werden für die Tabellen und deren Standardauswertungen Ansichten erstellt, die immer mit einer WHERE-Klausel auf die deleted-Spalten ausgestattet sind, kann man sich in den weiteren Befehlen diese Abfrage sparen.



Hinweis: Der Name einer Ansicht sollte einer Namenskonvention folgend immer mit dem Präfix view_ beginnen.

In MySQL und MariaDB gibt es kein zu SHOW TABLE analoges SHOW VIEWS. Vielmehr werden die Ansichten fast immer wie Tabellen behandelt. Damit man die Ansichten von den Tabellen unterscheiden kann, ist eine solche Namenskonvention sehr sinnvoll. Ein weiterer Grund ist, dass man mit einer Ansicht eben *nicht* alles wie mit einer Tabelle machen kann. Um es erst gar nicht zu versuchen, sollte schon am Namen erkennbar sein, dass es sich um eine Ansicht handelt. Selbst bei Systemen, die ein Analogon zu SHOW VIEWS haben, wie beispielsweise PostgreSQL mit \dv, empfiehlt sich die Namenskonvention, damit man in einem SELECT unterscheiden kann, ob die Datenquelle eine Tabelle oder eine Ansicht ist.



Hinweis: MySQL und MariaDB haben kein SHOW VIEWS, aber man kann die Ansichten einer Datenbank wie folgt ermitteln:

```
SHOW FULL TABLES IN datenbankname WHERE TABLE_TYPE = 'VIEW';
```

```
1 mysql> SHOW FULL TABLES IN oshop WHERE TABLE_TYPE = 'VIEW';
2 +-----+-----+
3 | Tables_in_oshop           | Table_type |
4 +-----+-----+
5 | view_artikel_aktiv        | VIEW       |
6 | view_artikel_verfuegbar   | VIEW       |
```

³ Probieren Sie es aus!

```

7 | view_kundenrechnungsadresse      | VIEW
8 | view_lagerbestand_aktiv        | VIEW
9 | view_lagerbestand_artikelbezeichnung | VIEW
10 | view_tmp                      | VIEW
11 | view_tmp_eins                 | VIEW
12 | view_tmp_zwei                 | VIEW
13 +-----+-----+
14 8 rows in set (0.00 sec)

```

Folgt man der Namenskonvention oben, so stehen diese bei einem `SHOW TABLES` wenigstens untereinander:

```

1 mysql> SHOW TABLES;
2 +-----+
3 | Tables_in_oshop               |
4 +-----+
5 | account                       |
6 | adresse                        |
7 | artikel                        |
8 | [...]                          |
9 | rechnung_position             |
10 | view_kundenrechnungsadresse   |
11 | warengruppe                   |
12 +-----+

```

16.1.2 Wie wird eine Ansicht verarbeitet?

In MySQL kann man die Art und Weise, wie eine Ansicht intern verarbeitet wird, festlegen oder festlegen lassen. Schauen wir uns dazu eine erweiterte Version des Befehls an.

 **MySQL/MariaDB**
`CREATE [OR REPLACE]`
`[ALGORITHM = {UNDEFINED|MERGE|TEMPTABLE}]`
`VIEW viewname[(spaltenliste)]`
`AS`
`SELECT auswahl`
`;`

Die Option `ALGORITHM` bedeutet:

- **MERGED:** Die Auswahl, die hinter der Ansicht steht, wird in den verwendeten Befehl so eingebaut, als ob man ihn direkt dort hinein programmiert hätte. Dies erlaubt es dem internen Optimierer, ggf. den `SELECT` zu berücksichtigen.
- **TEMPTABLE:** Das Ergebnis der Ansicht wird in eine – nicht sichtbare – temporäre Tabelle abgespeichert, und erst dieses Ergebnis wird in der verwendeten Anweisung benutzt.
- **UNDEFINED:** MySQL/MariaDB entscheiden selbst darüber, welche der beiden Varianten verwendet wird. Wird keine Angabe zum Algorithmus gemacht, wird wie bei `UNDEFINED` verfahren.

Als Beispiel nehmen wir eine einfache Ansicht:

```

1 CREATE VIEW view_artikel_aktiv
2 AS
3   SELECT * FROM artikel
4   WHERE deleted = 0;

```

Das entscheidende Kriterium für die Verwendung von MERGED ist, ob zwischen den Ergebnissen der Ansicht und dem Rest der Anweisung eine 1 : 1-Verknüpfung hergestellt werden kann. Bei Gruppenbildungen mit GROUP BY oder anderen Mengenoperationen (wie beispielsweise UNION) geht die direkte Verknüpfung der Datensätze verloren. Mit EXPLAIN kann man sich die Entscheidung von MySQL bzw. MariaDB anzeigen lassen:

```

1 mysql> SELECT FLOOR(artikel_id / 1000) gruppe, AVG(einzelpreis)
2      ->   FROM view_artikel_aktiv
3      ->   GROUP BY gruppe
4      -> ;
5 +-----+-----+
6 | gruppe | AVG(einzelpreis) |
7 +-----+-----+
8 |      3 |     14.8120000000 |
9 |      7 |     1.9700000000 |
10 |      9 |    17.6000000000 |
11 +-----+-----+

```

In diesem Beispiel werden die Durchschnittspreise nach Warengruppen gruppiert ermittelt.

```

1 mysql> EXPLAIN SELECT wie oben\G
2 **** 1. row ****
3      id: 1
4  select_type: SIMPLE
5        table: artikel
6    partitions: NULL
7      type: ALL
8 possible_keys: PRIMARY,idx_artikel_bezeichnung
9       key: NULL
10      key_len: NULL
11        ref: NULL
12      rows: 4
13     filtered: 25.00
14 Extra: Using where; Using temporary
15 1 row in set, 1 warning (0.00 sec)

```

Unter Extra wird angegeben, dass eine temporäre Tabelle verwendet wurde. Als Alternative schauen wir uns jetzt eine Einschränkung der Datensätze auf einen bestimmten Nummernbereich an:

```

1 mysql> SELECT artikel_id, bezeichnung
2      ->   FROM
3      ->   view_artikel_aktiv
4      ->   WHERE
5      ->   artikel_id BETWEEN 7000 AND 9000
6      -> ;
7 +-----+-----+
8 | artikel_id | bezeichnung   |
9 +-----+-----+
10 |      7856 | Silberzwiebel |
11 |      7863 | Tulpenzwiebel |
12 +-----+-----+

```

EXPLAIN teilt mir lediglich mit, dass hier die WHERE-Klausel verwendet wird. Die Ansicht wird somit mit der verwendenden Anweisung verschmolzen.

```

1 mysql> EXPLAIN SELECT wie oben\G
2 **** 1. row ****
3   id: 1
4   select_type: SIMPLE
5     table: artikel
6     partitions: NULL
7       type: range
8   possible_keys: PRIMARY
9     key: PRIMARY
10    key_len: 4
11    ref: NULL
12    rows: 2
13   filtered: 25
14   Extra: Using where

```

Stellt sich doch die Frage, warum man etwas angeben kann, was doch MySQL oder MariaDB eigenständig entscheiden. Na, weil wir den Durchblick haben:



Hinweis: Wird das Ergebnis einer Ansicht mehrfach verwendet, sollte man ALGORITHM=TEMPTABLE setzen, da diese temporäre Tabelle im Cache gehalten wird und daher wieder verwendet werden kann.

Erzeugen wir uns die gleiche Ansicht, allerdings um die Angabe ALGORITHM=TEMPTABLE erweitert:

```

1 CREATE ALGORITHM=TEMPTABLE VIEW view_artikel_aktiv_tmp
2 AS
3   SELECT * FROM artikel
4   WHERE deleted = 0;
5 ;

```

Führen wir nun zwei Mal einen SELECT mit unterschiedlicher WHERE-Klausel, aber der neuen Ansicht aus:

```

1 SELECT artikel_id, bezeichnung
2   FROM view_artikel_aktiv_tmp
3  WHERE artikel_id BETWEEN 7000 AND 8000
4 ;
5
6 SELECT artikel_id, bezeichnung
7   FROM view_artikel_aktiv_tmp
8  WHERE artikel_id BETWEEN 3000 AND 4000
9 ;

```

Jetzt liegt das Ergebnis der Ansicht in einer unsichtbaren temporären Tabelle im Cache. Ein EXPLAIN auf die letzte Abfrage bestätigt uns, dass dieser Cache-Inhalt verwendet wird.

```

1 mysql> EXPLAIN SELECT wie oben WHERE artikel_id BETWEEN 3000 AND 4000\G
2 **** 1. row ****
3   id: 1
4   select_type: PRIMARY
5     table: <derived2>
6     partitions: NULL
7       type: ALL

```

```

8 possible_keys: NULL
9      key: NULL
10     key_len: NULL
11      ref: NULL
12      rows: 2
13   filtered: 50.00
14      Extra: Using where
15 **** 2. row ****
16      id: 2
17 select_type: DERIVED
18      table: artikel
19 partitions: NULL
20      type: ALL
21 possible_keys: NULL
22      key: NULL
23     key_len: NULL
24      ref: NULL
25      rows: 4
26   filtered: 25.00
27      Extra: Using where

```

Die temporäre Tabelle wird als Basis für die neue Auswertung mit der veränderten WHERE-Klausel verwendet. Ohne die Angabe ALGORITHM=TEMPTABLE würde hier der gleiche EXPLAIN ausgegeben werden wie bei der Verwendung von view_artikel_aktiv.

16.1.3 Wie wird eine Ansicht gelöscht?

 SQL:2016	MySQL/MariaDB PostgreSQL	T-SQL
DROP VIEW viewname[, viewname]* ;	DROP [IF EXISTS] VIEW viewname[, viewname]* [CASCADE RESTRICT] ;	DROP [IF EXISTS] VIEW viewname[, viewname]* [CASCADE RESTRICT] ;

```

1 mysql> SHOW TABLES;
2 +-----+
3 | Tables_in_oshop |
4 +-----+
5 | account          |
6 | [...]           |
7 | rechnung_position |
8 | view_artikel_aktiv |
9 | view_artikel_aktiv_tmp |
10 | view_kundenrechnungsadresse |
11 | warengruppe      |
12 +-----+
13 20 rows in set (0.00 sec)
14
15 mysql> DROP VIEW view_artikel_aktiv_tmp;
16
17 mysql> SHOW TABLES;
18 +-----+
19 | Tables_in_oshop |

```

```

20 +-----+
21 | account          |
22 | [...]           |
23 | rechnung_position |
24 | view_artikel_aktiv |
25 | view_kundenrechnungsadresse |
26 | warengruppe      |
27 +-----+
28 19 rows in set (0.00 sec)

```

So weit, so einfach. Was passiert aber, wenn eine Ansicht Teil einer anderen Ansicht ist? Oder was passiert, wenn die Tabelle gelöscht wird, die Teil der Ansicht ist? Fragen über Fragen ...

Bauen wir uns schnell ein Beispiel für die Frage zusammen: *Was passiert, wenn man eine Tabelle löscht, die in einer Ansicht verwendet wird?*

```

1 mysql> CREATE DATABASE tmp1;
2 mysql> USE tmp1;
3 mysql> CREATE TABLE a (i INT);
4 mysql> CREATE TABLE b (i INT);
5 mysql> CREATE VIEW ab
6   -> AS
7   ->   SELECT * FROM a INNER JOIN b USING(i)
8   -> ;
9
10 mysql> SHOW TABLES;
11 +-----+
12 | Tables_in_tmp1 |
13 +-----+
14 | a              |
15 | ab             |
16 | b              |
17 +-----+
18
19 mysql> DROP TABLE a;
20
21 mysql> SHOW TABLES;
22 +-----+
23 | Tables_in_tmp1 |
24 +-----+
25 | ab             |
26 | b              |
27 +-----+

```

Das ist überraschend! Die Ansicht **ab** ist nicht gelöscht worden. Der Versuch, mit dieser Ansicht zu arbeiten, schlägt allerdings fehl:

```

1 mysql> SELECT * FROM ab;
2 ERROR 1356 (HY000): View 'tmp1.ab' references invalid table(s) or column(s) or
   function(s) or definer/invoker of view lack rights to use them

```

Wenn man vorab klären möchte, ob eine Ansicht einsatzfähig ist, verwenden Sie **CHECK TABLE**:

```

1 mysql> CHECK TABLE ab\G
2 **** 1. row ****
3 Table: tmp1.ab
4 Op: check
5 Msg_type: Error

```

```

6 Msg_text: View 'tmp1.ab' references invalid table(s) or column(s) or function(
   s) or definer/invoker of view lack rights to use them
7 **** 2. row ****
8 Table: tmp1.ab
9 Op: check
10 Msg_type: error
11 Msg_text: Corrupt

```

Analog sieht es in T-SQL aus:

```

1 [...]
2 1> SELECT * FROM ab;
3 2> GO
4 ai          bi
5 -----
6
7 (0 rows affected)
8 1> DROP TABLE a;
9 2> GO
10 1> SELECT * FROM ab;
11 2> GO
12 Msg 208, Level 16, State 1, Server 86e3151cc665, Procedure ab, Line 3
13 Invalid object name 'a'.
14 Msg 4413, Level 16, State 1, Server 86e3151cc665, Line 1
15 Could not use view or function 'ab' because of binding errors.

```

In anderen Systemen werden solche Abhängigkeiten anders behandelt. In PostgreSQL beispielsweise erhält man eine Fehlermeldung:

```

1 tmp1=# \dt
2           List of relations
3 Schema | Name | Type  | Owner
4 -----+-----+-----+
5 public | a    | table | postgres
6 public | b    | table | postgres
7
8 tmp1=# \dv
9           List of relations
10 Schema | Name | Type  | Owner
11 -----+-----+-----+
12 public | ab   | view  | postgres
13
14
15 tmp1=# DROP TABLE a;
16 psql:listing13.sql:81: ERROR:  cannot drop table a because other objects
   depend on it
17 DETAIL:  view ab depends on table a
18 HINT:  Use DROP ... CASCADE to drop the dependent objects too.

```

Folgen wir dem Hinweis von PostgreSQL und erweitern den `DROP TABLE`:

```

1 tmp1=# DROP TABLE a CASCADE;
2 HINWEIS:  Löschvorgang löscht ebenfalls die Sicht ab
3
4 tmp1=# \dt
5           Liste der Relationen
6 Schema | Name | Typ   | Eigentümer
7 -----+-----+-----+
8 public | b    | Tabelle | postgres
9 (1 Zeile)
10

```

```

11
12 tmp1=# \dv
13 Keine Relationen gefunden.
```

Die Ansicht verschwindet mit, da diese ein abhängiges Objekt von Tabelle a ist. In MySQL und MariaDB wird die Option CASCADE ignoriert.



Aufgabe 16.2: Und was ist bei einer Ansicht, die auf einer Ansicht basiert? Bauen Sie einen ähnlichen Versuch auf und interpretieren Sie das Ergebnis.

Das Ergebnis lässt sich schon erahnen und führt zu folgendem Hinweis:



Hinweis: Werden in MySQL oder MariaDB Tabelle oder Ansichten, die in anderen Ansichten verwendet werden, gelöscht, erfolgt kein Weiterreichen dieser Löschoperation an die enthaltenden Ansichten.

16.1.4 Wie wird eine Ansicht geändert?

Tabellen können mit ALTER TABLE in fast allen Parametern verändert werden. Analoges gilt auch für die Ansicht. Bis auf den Namen⁴ können alle Parameter/Optionen neu bestimmt werden. Es gibt grundsätzlich zwei Syntaxvarianten.



SQL:2016, MySQL/MariaDB, PostgreSQL, T-SQL

```
ALTER VIEW viewname
view_spezifikation ;
```

In den meisten Fällen wird aber mit dem obigen OR REPLACE bzw. OR ALTER-Syntax des CREATE VIEW gearbeitet.

Somit wird nicht eine vorhandene Ansicht verändert – wie es bei einem ALTER TABLE der Fall ist –, sondern es wird eine neue Ansicht mit den veränderten Parametern geschaffen.

■ 16.2 Anwendungsgebiet: Vereinfachung

Eine wichtige Vereinfachung ist schon erwähnt worden. Wir haben im Rahmen der Diskussion um die Löschweitergabe (siehe Seite 33 und den ersten Hinweis in Kapitel 3 auf Seite 41) die Spalte deleted eingeführt. Jetzt wäre es aber extrem nervig, bei jeder Auswertung in die WHERE-Klausel ein AND deleted = 0 einzubauen⁵. Es ist somit eine gute Idee, für die Tabellen entsprechende Ansichten zu erstellen:

⁴ In PostgreSQL geht auch das mit ALTER TABLE viewname RENAME TO.

⁵ Abgesehen davon muss auch noch die Klammerung oder Auswertungsreihenfolge beachtet werden.

```

1 CREATE OR REPLACE
2 ALGORITHM = TEMPTABLE
3 VIEW view_artikel_aktiv
4 AS
5   SELECT * FROM artikel
6   WHERE deleted = 0;

```

Diese Art der Ansicht hat einen Namen:



Definition 56: Selektionsansicht

Eine Ansicht, deren Aufgabe das Herausfiltern von Zeilen ist, wird *Selektionsansicht* genannt.

Bei der Definition von `view_artikel_aktiv` drängen sich doch sofort Fragen auf:

1. *Warum `ALGORITHM = TEMPTABLE`?*

Die Ansicht ist eine Auswertung der Tabelle `artikel` ohne irgendeine weitere Einschränkung oder Aufbereitung. Mit anderen Worten, es ist recht wahrscheinlich, dass das Ergebnis der Ansicht relativ oft verwendet wird, und zwar jedes Mal dann, wenn auf die bereinigten Artikeldaten zugriffen wird.

2. *Warum `SELECT *` ?*

Unter der Annahme, dass `deleted` nur 0 oder 1 sein kann, haben wir mit `deleted = 0` immer den gleichen Wert in dieser Spalte stehen. Wäre es da nicht besser, diese Spalte gänzlich zu unterdrücken? Dagegen sprechen zwei Argumente: Zum einen müsste jedes Mal, wenn sich die Spalten in der Tabelle `artikel` ändern, die Ansicht angepasst werden. Zum anderen können Ansichten auch in `UPDATE`-Anweisungen verwendet werden. Und eine solche Anwendung kann das *Löschen* sein. Da dabei der Spalte `deleted` der Wert 1 zugewiesen werden muss, sollte diese Spalte sichtbar bleiben.

3. *Ist es wirklich sinnvoll, für alle Tabellen eine solche Ansicht zu erzeugen?*

Ein Frage wie diese lässt sich nicht mit *Ja* oder *Nein* beantworten. Es ist offen, ob diese Ansichten immer gebraucht werden, aber das Erzeugen ist auch nicht teuer, und dann hat man sie schon mal fertig, wenn man sie braucht.



Aufgabe 16.3: Erstellen Sie für die Tabellen `account`, `adresse`, `artikel`, `bank`, `bankverbindung`, `beitrag`, `bestellung`, `bestellung_position`, `kunde`, `lagerbestand`, `lieferant`, `rechnung`, `rechnung_position` und `warengruppe` die entsprechende Selektionsansicht.

Weitere Ansichten ergeben sich in der Regel aus den Anforderungen der Präsentationsschicht. Werden beispielsweise in einer Listbox alle Artikelnummern und -bezeichnungen angezeigt, die noch im Lager ausreichend vorhanden sind, so könnte man aus der Funktionsschicht einen solchen `SELECT` absetzen. Man hätte dann in der Funktionsschicht einen etwas komplizierteren `SELECT`, den man dann auch immer dort warten müsste. Anders, wenn eine `VIEW` vorhanden ist:

```

1 mysql> CREATE VIEW view_artikel_verfügbar
2   -> AS
3   ->   SELECT artikel_id, bezeichnung
4   ->     FROM artikel INNER JOIN lagerbestand USING(artikel_id)

```

```

5      -> WHERE menge_aktuell >= menge_mindest
6      -> ORDER BY artikel_id;
7
8 mysql> SELECT * FROM view_artikel_verfügbar;
9 +-----+-----+
10 | artikel_id | bezeichnung   |
11 +-----+-----+
12 |     3001 | Papier (100)  |
13 |     3005 | Tinte (gold)   |
14 |     3006 | Tinte (rot)    |
15 |    7856 | Silberzwiebel |
16 |    7863 | Tulpenzwiebel|
17 |    9010 | Schaufel       |
18 |    9015 | Spaten         |
19 +-----+-----+

```

Eine weitere, oft gewünschte Vereinfachung ist, dass man die *Standardverknüpfungen* schon vorwegnimmt. Dazu gehören Kunde – Rechnungsadresse, Kunde – Lieferadresse, Kunde – Rechnung, Kunde – Bestellung, Rechnung – Positionen, Bestellung – Positionen, Bankverbindung – Bankname, Lagerbestand – Artikelname.

Fangen wir mit der letzten an:

```

1 mysql> CREATE VIEW view_lagerbestand_artikelbezeichnung
2      -> AS
3      -> SELECT l.*, a.bezeichnung
4      -> FROM
5      ->   view_lagerbestand_aktiv l INNER JOIN view_artikel_aktiv a
6      ->     USING(artikel_id)
7      -> ;
8
9 mysql> SELECT * FROM view_lagerbestand_artikelbezeichnung;
10 +-----+-----+-----+-----+
11 | artikel_id | menge_mindest | menge_aktuell | deleted | bezeichnung   |
12 +-----+-----+-----+-----+
13 |     3001 | 1000.000000 | 5000.000000 |     0 | Papier (100)  |
14 |     3005 | 200.000000  | 250.000000  |     0 | Tinte (gold)   |
15 |     3006 | 200.000000  | 250.000000  |     0 | Tinte (rot)    |
16 |     3007 | 200.000000  | 149.000000  |     0 | Tinte (blau)   |
17 |     3010 | 100.000000  | 3.000000   |     0 | Feder          |
18 |    7856 | 100.000000  | 500.000000  |     0 | Silberzwiebel |
19 |    7863 | 100.000000  | 400.000000  |     0 | Tulpenzwiebel |
20 |    9010 | 10.000000   | 30.000000  |     0 | Schaufel       |
21 |    9015 | 10.000000   | 25.000000  |     0 | Spaten         |
22 +-----+-----+-----+-----+

```

Auch diese Art von Ansicht hat einen Namen:



Definition 57: Verbundansicht

Eine Ansicht, deren Aufgabe die Verknüpfung von Tabellen ist, wird *Verbundansicht* genannt.



Aufgabe 16.4: Erstellen Sie die verbleibenden Verbundansichten, die wir oben aufgelistet haben.

Ich halte es für eine gute Idee, Auswertungen aus der Funktionsschicht der Anwendung herauszunehmen und in einer Ansicht zu kapseln. In der Anwendung müssen dann nur noch Befehle wie `SELECT * FROM view_xyz WHERE ...` abgesetzt werden. Oft sind in Projekten Entwickler mit unterschiedlichen Skills eingesetzt. Ein guter Java-Programmierer muss nicht unbedingt auch ein guter SQL-Programmierer sein.

Ein gewaltiger Vorteil ist auch, dass eine Änderung der Abfrage zu dieser Auswertung – beispielsweise aus Gründen der Performance – keine Änderung im Quelltext der Funktionsschicht zur Folge hätte.

Zum Schluss noch eine kleine Aufgabe zum Selbsttest:



Aufgabe 16.5: Bei der Besprechung darüber, wie man die Ergebnisse einer Verknüpfung wiederverwenden kann, sind wir am Beispiel Bankeinzug (siehe [Abschnitt 11.2.3 auf Seite 192](#)) auf eine Hierarchie von Auswertungen gekommen. Bauen Sie diese Hierarchie mithilfe von Ansichten nach. Überprüfen Sie, ob Sie das gleiche Ergebnis bekommen.

■ 16.3 Anwendungsgebiet: Datenschutz

In der Literatur findet man oft einen Hinweis darauf, dass man mithilfe der Ansicht Daten vor den Anwendern verbergen kann. Soll beispielsweise die Bürokraft die Gehaltsangaben eines Mitarbeiters nicht, aber alle anderen Angaben sehen können, so könne man eine entsprechende Ansicht bilden, die die sensiblen Spalten ausblendet.



Definition 58: Projektionsansicht

Eine Ansicht, deren Aufgabe das Herausfiltern von Spalten ist, wird *Projektionsansicht* genannt.

Das Argument halte ich in der so vorgetragenen Form für Der Anwender arbeitet gar nicht direkt auf Ebene der Datenbank, sondern auf Ebene der Anwendung. Er kann doch nur sehen und verändern, was die Anwendung zulässt.

■ 16.4 Grenzen einer Ansicht

Die MySQL- und MariaDB-Referenz schränkt Ansichten an vielen Stellen ein⁶. Hier einige davon:

- Es dürfen keine Unterabfragen in der `FROM`-Klausel verwendet werden.

⁶ Ich kann nicht sagen, ob diese Einschränkungen vom SQL-Sprachstandard oder den technischen Grenzen von MySQL und MariaDB herrühren.

```

1 mysql> CREATE VIEW v AS SELECT bezeichnung FROM (SELECT * FROM artikel) a;
2 ERROR 1349 (HY000): View's SELECT contains a subquery in the FROM clause

```

- Es dürfen keine System- oder Benutzervariablen verwendet werden.

```

1 mysql> SET @b = 3000;
2 Query OK, 0 rows affected (0.00 sec)
3
4 mysql> CREATE VIEW v AS SELECT * FROM artikel WHERE artikel_id > @b;
5 ERROR 1351 (HY000): View's SELECT contains a variable or parameter

```

- Es dürfen keine temporären Tabellen verwendet werden. Auch kann keine temporäre View erzeugt werden.

```

1 mysql> CREATE TEMPORARY TABLE tmp SELECT * FROM artikel;
2 Query OK, 18 rows affected (0.16 sec)
3 Records: 18 Duplicates: 0 Warnings: 0
4
5 mysql> CREATE VIEW view_tmp AS SELECT artikel_id FROM tmp;
6 ERROR 1352 (HY000): View's SELECT refers to a temporary table 'tmp'

```

- Das ORDER BY der Ansicht wird vom ORDER BY der enthaltenden Anweisung überlagert.

```

1 mysql> CREATE VIEW view_tmp
2      -> AS
3      ->   SELECT nachname FROM kunde ORDER BY nachname DESC;
4
5 mysql> SELECT * FROM view_tmp;
6 +-----+
7 | nachname |
8 +-----+
9 | Tuk      |
10 [...]
11 | Beutlin  |
12 +-----+
13
14 mysql> SELECT * FROM view_tmp ORDER BY nachname;
15 +-----+
16 | nachname |
17 +-----+
18 | Beutlin  |
19 [...]
20 | Tuk      |
21 +-----+

```

- Nicht jede Ansicht ist veränderbar.

Den letzten Punkt müssen wir uns mal genauer anschauen. Zunächst zwei Beispiele:

```

1 CREATE VIEW view_tmp_eins
2   AS
3     SELECT nachname FROM kunde
4 ;
5
6 CREATE VIEW view_tmp_zwei
7   AS
8     SELECT rechnung_id, COUNT(*) anzahl
9       FROM rechnung_position
10      GROUP BY rechnung_id
11 ;

```

Die erste Ansicht liefert mir einfach nur die Nachnamen der Kunden und die zweite zu jeder Rechnung die Anzahl der Positionen. Und jetzt kommt's:

```

1 mysql> UPDATE view_tmp_eins
2      -> SET nachname = 'Streicher'
3      -> WHERE nachname = 'Telcontar'
4 ;
5 Query OK, 1 row affected (0.03 sec)
6 Rows matched: 1  Changed: 1  Warnings: 0
7
8 mysql> UPDATE view_tmp_zwei
9      -> SET anzahl = 2
10     -> WHERE anzahl > 2;
11 ERROR 1288 (HY000): The target table view_tmp_zwei of the UPDATE is not
      updatable

```

Der erste UPDATE hat tadellos funktioniert. Ein SELECT würde mir das bestätigen. Aber der zweite lieferte eine hässliche Fehlermeldung: Die Ansicht ist nicht *updateable*⁷. Wenn ich der Anzahl einen neuen Wert zuweisen möchte, verkenne ich, dass es überhaupt keine Spalte einer Zeile gibt, in den ich diesen neuen Wert hineinschreiben könnte. Der Wert wird ja durch Nachzählen ermittelt. Würde ich jetzt die Anzahl auf 2 setzen, was sollte er dann tun? Alle Positionen bis auf zwei löschen? Ein UPDATE auf diese Zeile ist also sinnlos.

```

mysql> UPDATE view_tmp_zwei SET rechnung_id = 1 WHERE rechnung_id > 1;
ERROR 1288 (HY000): The target table view_tmp_zwei of the UPDATE is not
      updatable

```

Man könnte meinen, dass dieser UPDATE eigentlich klappen müsste, da die `rechnung_id` das Gruppierungselement ist. Aber auch hier erscheint die Fehlermeldung. Warum? Weil es eben keine 1 : 1-Verknüpfung mehr zwischen den *Zeilen der Originaltabelle* und den *Zeilen der Ansicht* mehr gibt. Das erscheint als ein sehr formales Argument, weil man sich hier vorstellen könnte, dass die Aktualisierung hätte klappen können⁸.



Definition 59: Veränderbare Ansicht

Eine Ansicht kann dann durch INSERT, UPDATE oder DELETE verändert werden, wenn zwischen den Werten der Ansicht und den Ausgangsdaten eine 1 : 1-Verknüpfung besteht.

Das heißt, dass bei `view_tmp_eins` auch INSERT- und DELETE-Operationen möglich sind. Zunächst das Einfügen zweier neuer Kundennachnamen:

```

1 mysql> INSERT INTO view_tmp_eins VALUES ('Wurst'), ('Brot');
2 Query OK, 2 rows affected (0.04 sec)
3 Records: 2  Duplicates: 0  Warnings: 0
4
5 mysql> SELECT * FROM view_tmp_eins;
6 +-----+
7 | nachname      |
8 | Beutlin       |
9 | Beutlin       |
10 [...]

```

⁷ engl.: aktualisierbar

⁸ Leider entzieht es sich meiner Kenntnis, ob diese Einschränkung MySQL-/MariaDB-spezifisch ist. In PostgreSQL gibt es auch entsprechende Einschränkungen, die aber anders motiviert sind.

```

11 | Brot      |
12 [...]      |
13 | Wurst     |
14 +-----+
15 24 rows in set (0.00 sec)

```

Die neuen Kunden sind auch tatsächlich in der Tabelle kunde angekommen und nicht in irgendeiner ominösen neuen Tabelle gelandet, wie Sie leicht mit einem SELECT auf kunde bestätigen können. Jetzt wollen wir das Löschen versuchen:

```

1 mysql> DELETE FROM view_tmp_eins WHERE nachname IN ('Wurst', 'Brot');
2 Query OK, 2 rows affected (0.04 sec)

```

Auch hier sind die Zeilen in der Tabelle kunde gelöscht worden.



Hinweis: Bei Änderungen in einer veränderbaren Ansicht gelten die gleichen Regeln, als ob Sie die Daten direkt in den Tabellen einfügen, ändern oder löschen. Besonders die Fremdschlüssel- und die NOT NULL-Constraints sind hier zu beachten.

Wann ist eine Ansicht denn nicht mehr veränderbar, also wann geht die geforderte 1 : 1-Verknüpfung verloren? Diese Frage hat einige einfache und einige kompliziertere Antworten.

- Zunächst ist klar, dass die Ansicht nicht mehr aktualisierbar ist, wenn Aggregatfunktionen (siehe [Abschnitt 26.2.3 auf Seite 411](#)) vorkommen.
- Enthält eine Ansicht eine nichtveränderbare Ansicht, ist diese auch nicht veränderbar.
- Nach einem DISTINCT (siehe [Abschnitt 10.4 auf Seite 173](#)) können einzelne Zeilen nicht mehr der ursprünglichen Tabelle zugeordnet werden; analog das GROUP BY (siehe [Abschnitt 12.2 auf Seite 214](#)).
- Theoretisch wäre ein UNION machbar, aber dazu müsste der Verweis auf die ursprüngliche Tabelle bei einem UNION mit verwaltet werden. Somit ist auch dann keine Veränderung mehr möglich.
- Durch die Verwendung einer Unterabfrage (siehe [Kapitel 13 auf Seite 225](#)).
- Die Spaltenwerte werden aus Konstanten oder Ausdrücken ermittelt. Bei Konstanten gibt es keine Zeilen geschweige denn Tabellen, die man verändern könnte. Bei Ausdrücken müsste die Eindeutigkeit und Existenz des Inversen des Ausdrucks gegeben sein, was natürlich nicht der Fall ist.

Aber: Werden bei einem UPDATE nur die Spalten verändert, die sich nicht aus einem Ausdruck herleiten oder durch Konstanten bestimmt sind, ist eine Aktualisierung möglich.

- ALGORITHM=TEMPTABLE erzeugt von einer Auswertung eine temporäre Tabelle. Diese ist eine Kopie der ursprünglichen Daten. Daher sind Änderungen auf dieser ggf. gefährlich.

Besonders unübersichtlich wird es, wenn man eine Verbundansicht hat. Zunächst muss ein INNER JOIN verwendet werden. Grundsätzlich wird die Tabelle bei der Verwendung von OUTER JOIN oder UNION unveränderbar.

Auf einer Verbundansicht sind INSERT, UPDATE möglich, wenn sich die neuen oder geänderten Spaltenwerte nur auf *eine* Tabelle der Verknüpfung beziehen und keine der obigen Bedingungen verletzt werden und ist DELETE grundsätzlich nicht möglich.

So viel zu Ansichten.

17

Exkurs NoSQL



Die Wiese des Nachbarn ist immer grüner als die eigene.

- Was ist NoSQL?
- JSON-Objekte
- Arbeiten mit Collections
- Arbeiten mit dem SQL-Datentyp JSON



Die Quelltexte dieses Kapitels stehen in den Dateien `listing13NoSQL.sql` (siehe [Listing 29.15 auf Seite 515](#)) und `listing13NoSQL.js` (siehe [Listing 29.14 auf Seite 510](#)). Die Bestellungen liegen in `mysql/lstRechnung01.json` und der Beispielenwarenkorb liegt in `mysql/lstWarenkorb01.json`.

Das Kapitel kann kein NoSQL¹-Buch ersetzen; ich möchte aber dieses für die Datenbank-Praxis so wichtige Thema hier eingeführt haben. Anwendungsgebiete gibt es genug und für die Prüfungen wird es auch immer bedeutender, Grundkenntnisse darin zu besitzen. Leider muss ich viele Themen, die typischerweise mit NoSQL verbunden werden wie beispielsweise *Big Data* oder *räumliche Daten*, völlig außen vor lassen. Für eine tiefere Einführung empfehle ich daher die Bücher [\[EFH\]](#) und [\[Har\]](#).

Aber was ist NoSQL eigentlich? Mithilfe von NoSQL werden Daten in nichttabellarischer Form abgespeichert und mit SQL oder einer Client-Sprache (JavaScript, PHP, .NET usw.) bearbeitet und ausgewertet. Da es noch keine NoSQL-Normierung oder etablierte Konvention gibt, ist NoSQL extrem proprietär und ich kann hier nur auf die im MySQL Server implementierten Features eingehen. Als Client-Sprache verwende ich JavaScript.

¹ Abkürzung für: Not only SQL

■ 17.1 Vorbereitung der MySQL-Shell

Normalerweise werden die NoSQL-Anweisungen von einer Client-Anwendung erteilt. Die entsprechenden Funktionen stehen Ihnen in den passenden Connectoren (Python, JavaScript, .NET etc.) zur Verfügung. Da wir mit der erweiterten interaktiven MySQL-Shell arbeiten wollen, müssen wir auf eine dieser Clientsprachen schalten. Dies ist mit dem X-Plugin von MySQL 8.0 und der erweiterten MySQL-Shell (`mysqlsh`) möglich. Leider ist die erweiterte MySQL-Shell noch nicht in unserer Dockerinstanz (siehe [Abschnitt 4.3.1 auf Seite 58](#)) installiert, also frisch ans Werk.

Zunächst müssen zwei Programmpakete vorbereitend installiert werden. Bei Rückfragen geben Sie einfach ein bestätigendes J ein:

```
1 myqsldocker:~# apt-get update
2 myqsldocker:~# apt-get install wget
3 myqsldocker:~# apt-get install lsb-release
```

Nun erstelle ich ein Unterverzeichnis und lade mithilfe von `wget` eine MySQL-APT-Konfigurationsdatei in das neue Verzeichnis:

```
1 myqsldocker:~# mkdir downloads
2 myqsldocker:~# cd downloads/
3 myqsldocker:~/downloads# wget https://dev.mysql.com/get/mysql-apt-config_0
     .8.13-1_all.deb
4 [...]
5 2019-08-24 11:17:08 (3.12 MB/s) - 'mysql-apt-config_0.8.13-1_all.deb' saved
[35560/35560]
```

Jetzt wird die MySQL-APT-Konfigurationsdatei eingerichtet und das Repository aktualisiert. Ich wähle die Option 4 (siehe [Zeile 11](#)), da die Vorbelegung schon genau so ist, wie wir sie brauchen:

```
1 myqsldocker:~/downloads# dpkg -i mysql-apt-config_0.8.13-1_all.deb
2 [...]
3 MySQL APT Repo features MySQL Server along with a variety of MySQL components.
   You may select the appropriate product to choose the version
4 that you wish to receive.
5
6 Once you are satisfied with the configuration then select last option 'Ok' to
   save the configuration, then run 'apt-get update' to load
7 package list. Advanced users can always change the configurations later,
   depending on their own needs.
8
9 1. MySQL Server & Cluster (Currently selected: mysql-8.0) 3. MySQL Preview
   Packages (Currently selected: Disabled)
10 2. MySQL Tools & Connectors (Currently selected: Enabled) 4. Ok
11 Which MySQL product do you wish to configure? 4
12
13 Warning: apt-key should not be used in scripts (called from postinst
   maintainerscript of the package mysql-apt-config)
14 OK
15
16 myqsldocker:~/downloads# apt-get update
```

Zum Schluss wird die erweiterte MySQL-Shell installiert und gestartet:

```
1 myqsldocker:~/downloads# apt-get install mysql-shell
2 [...]
```

```
3 myqsldocker:~/downloads# mysqlsh
4 Logger: Tried to log to an uninitialized logger.
5 MySQL Shell 8.0.17
6
7 Copyright (c) 2016, 2019, Oracle and/or its affiliates. All rights reserved.
8 Oracle is a registered trademark of Oracle Corporation and/or its affiliates.
9 Other names may be trademarks of their respective owners.
10
11 Type '\help' or '\?' for help; '\quit' to exit.
12 MySQL JS > \q
13 Bye!
14 myqsldocker:~/downloads#
```



Hinweis: Falls Sie den MySQL Server unter Windows installiert haben, müssen Sie nur eine PowerShell öffnen und mysqlsh eintippen.

■ 17.2 Datenmodellierung des Warenkorbs

Ich möchte einen Warenkorb modellieren. Dabei soll der online zusammengestellte Warenkorb nicht in eigenen Tabellen wie beispielsweise bestellung abgelegt werden – dies wäre mit Kanonen auf Spatzen schießen. Vielmehr möchte ich, dass jeder Kunde einen Warenkorb hat, der schnell und einfach verarbeitet werden kann, ohne dass dabei jedes Mal ein JOIN oder ein GROUP BY mit Aggregatfunktionen über die ganzen Tabellen abgesetzt werden muss.

Klären wir also zunächst, in welchem Format wir die Daten speichern werden.

17.2.1 JavaScript Object Notation (JSON)

In [Abschnitt 2.1.1 auf Seite 11](#) haben wir uns verschiedene Datenstrukturen angesehen. Üblicherweise werden bei NoSQL die Daten als JSON-Dokumente² (siehe [[Int13](#)]) abgelegt und man benötigt Funktionen, mit deren Hilfe man diese Dokumente auslesen kann.

Wie auch bei XML handelt es sich um eine einfache Auszeichnungssprache für nicht binäre Daten. Dabei können auch Hierarchien oder Mengen flexibel und bedarfsgerecht genutzt werden. JSON selbst ist eine Teilmenge von JavaScript bzw. ECMA und muss einen durch eval() ausführbaren JavaScript-Quelltext abbilden.

Schauen wir uns das Beispiel aus [Abschnitt 2.1.1](#) mal in JSON an. Grundbaustein von JSON sind Attribute, denen man Werte zuweist; beide werden durch Doppelpunkt getrennt.

```
1 "artikel_id": "7856"
```

Mehrere Attribute können durch Komma trennt zu Objekten zusammengefasst werden. Ganze Objekte werden durch geschweifte Klammern begrenzt. Dabei fungiert der Ob-

² Abkürzung für: JavaScript Object Notation

Attributname wie ein Attributname und wird daher ebenfalls durch einen Doppelpunkt von seiner Attributliste getrennt. Hier ein Artikelobjekt:

```

1 "artikel":{
2   "artikelnummer":"7856",
3   "artikelname":"Silberzwiebel",
4   "einzelpreis":6.15
5 }
```

Objekte enthalten einfache Attribute wie beispielsweise die Artikelnummer oder wiederum Objekte mit Attributlisten. So können diese beliebig tief verschachtelt werden. Die Bestellposition enthält beispielsweise ein Artikelobjekt und eine Mengenangabe:

```

1 "position":{
2   "artikel":{
3     "artikelnummer":"7856",
4     "artikelname":"Silberzwiebel",
5     "einzelpreis":6.15,
6     "währung":"EUR"
7   },
8   "menge":15
9 }
```

Mengen von Objekten können in Arrays³ abgelegt werden; sie sind durch rechteckige Klammern markiert. In unserer Bestellung kommen mehrere Bestellpositionen vor:

```

1 "positionen":[
2   {
3     "position":{
4       "artikel":{
5         "artikelnummer":"7856",
6         "artikelname":"Silberzwiebel",
7         "einzelpreis":6.15,
8         "währung":"EUR"
9       },
10      "menge":15
11    }
12  },
13  {
14    "position":{
15      "artikel":{
16        "artikelnummer":"7863",
17        "artikelname":"Tulpenzwiebel",
18        "einzelpreis":32.90,
19        "währung":"EUR"
20      },
21      "menge":10
22    }
23  },
24  {
25    "position":{
26      "artikel":{
27        "artikelnummer":"9015",
28        "artikelname":"Spaten",
29        "einzelpreis":19.90,
30        "währung":"EUR"
31      },
32      "menge":1
33    }
34  }
35 ]
```

Dies soll für eine minimale Einführung reichen. Eine vertiefende finden Sie unter [Hel19]. Der gesamte JSON-Quelltext ist in der Datei `lstRechnung01.json` abgelegt. Um Ihren JSON-Quelltext vorab validieren zu lassen, finden Sie unter <https://jsonformatter.curiousconcept.com/> einen Quelltext-Validator.

³ Eigentlich sind es Listen. Arrays bestehen aus gleichartigen Objekten, aber in vielen modernen Programmiersprachen – besonders bei nicht typisierten – sind Arrays und Listen Synonyme.

17.2.2 Struktur unseres JSON-Dokuments

Nach kurzer Reflexion kommen wir zu dem Ergebnis, dass der Warenkorb einer Bestellung sehr ähnlich ist. Im Wesentlichen besteht der Warenkorb aus folgenden Inhalten:

- Kundennummer: Da jeder Kunde nur einen Warenkorb haben kann, reicht hier eine Kundennummer und es muss keine Information wie eine Bestellnummer miterfasst werden.
- Warenkorbpositionen: Artikelnummer, Artikelbezeichnung, Artikelpreis, Währung, Anzahl und Gesamtpreis
- Der Warenwert des Warenkorbs soll nach Währung getrennt berechnet werden: Wert und Währung.



Aufgabe 17.1: Transformieren Sie die Inhalte von der ersten Bestellung von Kundennummer 12345 aus `lstRechnung01.json` in ein entsprechendes JSON-Dokument mit dem Namen `lstWarenkorb01.json`.

Das Ergebnis könnte so aussehen:

```

1  {
2      "warenkorb": {
3          "kunde_id": "12345",
4          "positionen": [
5              {
6                  "position": {
7                      "artikel": {
8                          "artikel_id": "7856",
9                          "artikelname": "Silberzwiebel",
10                         "einzelpreis": 6.15
11                     },
12                     "menge": 15,
13                     "gesamtpreis": 92.25,
14                     "währung": "EUR"
15                 }
16             },
17             {
18                 "position": {
19                     "artikel": {
20                         "artikel_id": "7863",
21                         "artikelname": "Tulpenzwiebel",
22                         "einzelpreis": 32.90
23                     },
24                     "menge": 10,
25                     "gesamtpreis": 329.0,
26                     "währung": "EUR"
27                 }
28             }
29         ],
30         "position": {
31             "artikel": {
32                 "artikel_id": "9015",
33                 "artikelname": "Spaten",
34                 "einzelpreis": 19.90
35             },
36             "menge": 1,
37             "gesamtpreis": 19.90,
38             "währung": "EUR"
39         }
40     }
41 ],
42     "warenwerte": [
43         {
44             "EUR": 441.15
45         }
46     ]
47 }
48 }
```



Aufgabe 17.2: Wenn Sie an die Normalisierungen denken (siehe [Abschnitt 2.4 auf Seite 34](#)), was ist bezüglich der Angaben gesamtpreis und warenwerte zu sagen?

Sie sehen hier ein häufiges Phänomen bei NoSQL-Datenbanken: Man verzichtet auf Redundanzfreiheit um einer besseren Performance willen. Die Erhaltung der Datenkonsistenz muss dann vom Programmierer überwacht werden.

■ 17.3 NoSQL: MySQL mit JavaScript-Client

So, jetzt haben wir alles beisammen: die erweiterte MySQL-Shell und das Datenmodell. Jetzt wollen wir mal das Zusammenspiel zwischen MySQL und JavaScript betrachten.

Alle JSON-Dokumente werden in einer *Collection* abgelegt. Dabei bekommt jedes Dokument einen automatisch oder manuell zugewiesenen Schlüssel mit dem festen Namen `_id`. Die Funktion dieses Schlüssels entspricht dem eines Primärschlüssels ([Definition 11](#)) in Tabellen; nur so kann bequem auf einzelne Dokumente in der Collection zugegriffen werden. Für unseren Anwendungsfall bietet es sich an, die Kundennummer als Schlüssel zu verwenden, da jeder Kunde immer nur einen Warenkorb haben kann.

Starten wir also die erweiterte MySQL-Shell mit `mysqlsh` und gehen unsere ersten vorsichtigen Schritte in JavaScript. Zunächst ein paar Vorbereitungen: In [Zeile 1](#) wird ein Objekt erstellt, welches alle Verbindungsparameter enthält; Sie müssten hier ggf. Anpassungen vornehmen. Mit `require()` stellen wir uns in [Zeile 3](#) die MySQL-JavaScript-Umgebung zur Verfügung. Hier – wie auch bei zukünftigen Lösungen – werden die Arbeiten in Funktionen gekapselt. Mithilfe des Schlüsselwortes `function` leiten wir die Funktionsdefinition ab [Zeile 5](#) ein.

Innerhalb der Funktion holen wir uns in [Zeile 6](#) mit `getSchema()` eine Referenz auf die Datenbank und in [Zeile 7](#) versuchen wir eine ggf. vorhandene Collection mit `dropCollection()` zu löschen. Jetzt wird in [Zeile 8](#) die Collection angelegt und per `return` dem Aufrufer zur Verfügung gestellt. Das war die Funktionsdefinition.

Jetzt wollen wir das Ding auch benutzen: [Zeile 11](#) öffnet eine Verbindung/Sitzung zum Server und liefert mir eine Referenz darauf zurück. Diese übergeben ich anschließend der Funktion als Parameter. Mit `getCollections()` lasse ich mir in [Zeile 13](#) alle in der Datenbank angelegten Collections anzeigen: In unserem Fall gibt es nur eine.

Zum Abschluss: *Türe schließen nicht vergessen!* In [Zeile 18](#) wird die Sitzung mit `close()` beendet und dadurch werden serverseitig die Ressourcen wieder freigegeben.

```

1  MySQL  JS > var connect = { host: 'localhost', port: 33060,
2          ->                           user: 'root', password: 'weinschlauchX10' };
3  MySQL  JS > var mysqlx = require('mysqlx');
4  MySQL  JS >
5  MySQL  JS > function warenkorbCollectionAnlegen(mySession) {
6          ->   var db0shop = mySession.getSchema('oshop');
7          ->   db0shop.dropCollection('warenkörbe');
8          ->   return db0shop.createCollection('warenkörbe');
9          -> }
10         ->
11 MySQL  JS > var session = mysqlx.getSession(connect);
12 MySQL  JS > warenkorbCollectionAnlegen(session);
13 MySQL  JS > db0shop.getCollections();
14 [
15     <Collection:warenkörbe>
16 ]
17 MySQL  JS >
18 MySQL  JS > session.close();
```



Hinweis: Um Platz zu sparen, verkürze ich den Prompt von MySQL JS > auf JS >. Ebenso verzichte ich zukünftig auf die Zeilen, mit denen ich die Verbindung zum Server herstelle (Zeilen 3 bis 11), die Datenbank auswähle (Zeile 6) und die Collection ermitte bzw. anlege (Zeilen 13 bis 8). Auch das Schließen der Sitzung möchte ich nicht jedes Mal (Zeile 18) angeben.

17.3.1 Anlegen eines Warenkorbs

Als Erstes wird allen Kunden ein leerer Warenkorb zugewiesen. Dieser sollte folgende Struktur haben:

```

1  {
2    "warenkorb": {
3      "kNr": "12345",
4      "positionen": [],
5      "warenwerte": []
6    }
7 }
```

Mit `getCollection()` besorgen wir uns eine Referenz auf die Collection. Über diese Referenz wiederum ermitteln wir eine weitere Referenz, und zwar auf die Kundentabelle (Zeile 7). In Zeile 4 löscht zunächst `remove()` alle schon vorhandenen Warenkörbe (nicht die Dokumente!). Mit der Methode `select()` wird jetzt die Kundennummer aller Kunden ermittelt und einer Ergebnisvariablen zugewiesen (Zeile 7). Danach kopiert uns `fetchAll()` das Ergebnis in ein Array und wir können mit der `foreach`-Variante von JavaScript `for ... in` alle Arrayelemente durchwandern (Zeile 11). In Zeile 13 wird der leere Warenkorb gebastelt und mit `add()` der Collection hinzugefügt. Das Resultat sind lauter leere Warenkörbe (Zeile 25).

```

1  JS > function warenkorbLeeren(mySession) {
2    ->   var db0shop = mySession.getSchema('oshop');
3    ->   var collWarenkörbe = db0shop.getCollection('warenkörbe');
4    ->   collWarenkörbe.remove({_id:0}).execute();
5    ->
6    ->   // Für alle Kunden einen leeren Warenkorb anlegen.
7    ->   var tabKunde = db0shop.getTable("kunde");
8    ->   var alleKunden = tabKunde.select("kunde_id").execute();
9    ->
10   ->   var zeilen = alleKunden.fetchAll();
11   ->   for (index in zeilen) {
12   ->     var zeile = zeilen[index];
13   ->     collWarenkörbe.add({
14   ->       _id: zeile.kunde_id,
15   ->       warenkorb: {
16   ->         kunde_id: zeile.kunde_id,
17   ->         positionen: new Array(),
18   ->         warenwerte: new Array()
19   ->       }
20   ->     }).execute();
21   ->   }
22   -> }
23   ->
24 JS > warenkorbLeeren(session);
```

```

25 JS > session.getSchema('oshop').getCollection('warenkörbe').find();
26 {
27   "_id": 1,
28   "warenkorb": {
29     "kunde_id": 1,
30     "positionen": [],
31     "warenwerte": []
32   }
33 }
34 [...]
35 22 documents in set (0.0004 sec)

```

17.3.2 Inhalte des Warenkorbs anlegen

Gehen wir davon aus, dass ein Kunde auf der Client-Oberfläche folgende Dinge tut: Der Kunde selektiert einen Artikel, legt eine Anzahl fest und fügt dies dem Warenkorb hinzu.

Um die Warenkorbpositionen einzufügen, müssen wir auf die Artikeltabelle zugreifen. Dazu wird zunächst in einem String der SELECT vorformuliert ([Zeile 3](#)). Haben Sie das Fragezeichen am Ende des Strings bemerkt? Dies ist ein Platzhalter für die *Parameterbindung* (*Parameter Binding*). Sie können den Mechanismus gut in [Zeile 11](#) erkennen. Durch die Methode `bind()` in [Zeile 11](#) wird das Fragezeichen durch den Wert von `myArtikelId` ersetzt. Die eigentlichen Methoden, die den SQL-Befehl ausführen, sind `sql()` und `execute()`.

Mit `fetchOne()` wird die erste Zeile der Ergebnismenge einer Variablen zugewiesen. Bei jedem Aufruf der Methode springt ein interner Zeiger immer eine Zeile in der Ergebnismenge weiter. So kann man – beispielweise in einer `while`-Schleife – die gesamte Ergebnismenge abarbeiten. Da wir nur einen Artikel ausgewählt haben, liefert mir `fetchOne()` die erste und einzige Zeile der Ergebnismenge.

Und wie kann nun auf die Spalten der Ergebnissezeile zugegriffen werden? In JavaScript ist die Ergebnissezeile ein Objekt und daher kann auf die Spalten per Punktnotation zugegriffen werden, wie z.B. in [Zeile 6](#).

Ab [Zeile 13](#) werden die Positionsobjekte gebastelt und mit den Methoden `modify()` und `arrayInsert()` in [Zeile 8](#) zurückgeschrieben. Dabei kommt eine Variante in der Syntax der Parameterbindung vor. In der Anweisung wird dem Parameternamen ein Doppelpunkt vorangestellt und anschließend kommt der Parametername – hier `param`. Durch die Vergabe von Namen könnten so für die Anweisung mehrere Parameterbindungen definiert werden.

In [Zeile 28](#) können Sie sehen, wie mithilfe des Schlüssels `_id` und der Methode `find()` gezielt auf ein bestimmtes Dokument der Collection zugegriffen wird.

```

1 JS > function warenkorbPositionEinfügen(mySession, myKundeId, myArtikelId,
2   myMenge) {
3   ->   var db0shop = mySession.getSchema('oshop');
4   ->   var collWarenkörbe = db0shop.getCollection('warenkörbe');
5   ->   var sql = "SELECT";
6   ->   sql += " artikel_id, bezeichnung, einzelpreis, waehrung";
7   ->   sql += " FROM oshop.artikel";
8   ->   sql += " WHERE artikel_id=?";
9   ->   var artikel=mySession.sql(sql).bind(myArtikelId).execute().fetchOne();

```

```

9    -> var objArtikel = {
10   ->     artikel_id: artikel.artikel_id,
11   ->     artikelname: artikel.bezeichnung,
12   ->     preis: artikel.einzelpreis
13   -> };
14   -> var objPosition = {
15   ->     artikel: objArtikel,
16   ->     menge: myMenge,
17   ->     gesamtpreis: myMenge * artikel.einzelpreis,
18   ->     währung: artikel.waehrung
19   -> };
20   -> collWarenkörbe.modify({_id=:param}).arrayInsert('warenkorb.positionen
[0]', {
21   ->     position: objPosition
22   -> }).bind('param', myKundeId).execute();
23 }
24
25 JS > warenkorbPositionEinfügen(session, 2, 3007, 10);
26 JS > warenkorbPositionEinfügen(session, 2, 9010, 5);
27 JS > warenkorbPositionEinfügen(session, 2, 3010, 2);
28 JS > session.getSchema('oshop').getCollection('warenkörbe').find("_id=2");

```

Das Ergebnis der letzten Anweisung ist:

```

1 {
2   "_id":2,
3   "warenkorb":{
4     "kunde_id":2,
5     "positionen":[
6       {
7         "position":{
8           "menge":2,
9           "artikel":{
10             "preis":"5.050000",
11             "artikel_id":3010,
12             "artikelname":"Feder"
13           },
14           "währung":"EUR",
15           "gesamtpreis":10.1
16         }
17       },
18       {
19         "position":{
20           "menge":5,
21           "artikel":{
22             "preis":"15.100000",
23             "artikel_id":9010,
24             "artikelname":"Schaufel
"           }
25       }
26     ],
27     "warenwerte": [
28       {
29         "position": {
30           "menge":10,
31           "artikel": {
32             "preis": "4.170000",
33             "artikel_id": 3007,
34             "artikelname": "Tinte (blau)"
35           },
36           "währung": "EUR",
37           "gesamtpreis": 41.7
38         }
39       }
40     }
41   }
42 }
43
44
45 ]
46 }
47 }
48 1 document in set (0.0006 sec)

```



Hinweis: Mithilfe von Parameterbindung wird ein Weg der SQL-Injection verhindert. Der Parameter selbst kann dann keinen SQL-Code mehr enthalten, da der Parameter intern entsprechend analysiert und präpariert ist.

Was mich am bisherigen Status stört, ist, dass der Warenwert des Warenkorbs nicht berechnet wird. Bauen wir also eine Funktion, die den Warenwert und den Gesamtpreis neu

berechnet. Wir können diese dann nach dem Einfügen oder Ändern von Daten aufrufen. Spannend sind die beiden Methoden `hasOwnProperty()` ([Zeile 8](#)) und `keys` ([Zeile 15](#)). Beide Methoden entspringen der Technik *reflection* in der objektorientierten Programmierung. Dabei wird ein System so programmiert, dass es zur Laufzeit seine eigenen Eigenschaften kennt oder anderen mitteilt. Genauer an einem Beispiel: Ein Dreieck-Objekt kennt nicht nur seine Seitenlängen, die dann mit Methoden wie `GetSeiteA()` einem anderen mitgeteilt werden können, sondern kann einem anderen auch mitteilen, dass es eine Seite A hat.

`hasOwnProperty()` testet, ob der übergebene Wert ein Attribut der Klasse ist und `keys()` liefert mir alle Schlüssel eines Arrays oder einer Objektliste.

```

1 function aktualisiereGesamtpreisWarenwert(myWarenkorb) {
2     var ergebnis = Object();
3     var posindizes = Object.keys(myWarenkorb.warenkorb.positionen);
4     posindizes.forEach(function(posindex) { // Positionen durchwandern
5         var myPos = myWarenkorb.warenkorb.positionen[posindex].position;
6         // gesamtpreis der Position berechnen
7         myPos.gesamtpreis = myPos.artikel.preis * myPos.menge;
8         if (ergebnis.hasOwnProperty(myPos.währung)) {
9             ergebnis[myPos.währung] += myPos.gesamtpreis;
10        } else {
11            ergebnis[myPos.währung] = myPos.gesamtpreis;
12        }
13    });
14    var warenwert = new Array();
15    for (index in Object.keys(ergebnis)) {
16        var währung = Object.keys(ergebnis)[index];
17        var wert = ergebnis[währung];
18        warenwert.push({ [währung]: wert });
19    }
20
21    myWarenkorb.warenkorb.warenwerte = warenwert;
22    return myWarenkorb;
23 }
```



Aufgabe 17.3: Versuchen Sie in Prosa den Quelltext zu interpretieren. Was wird beispielsweise in [Zeile 8](#) getestet und warum?

Jetzt können wir diese Methode geschickt einbauen (siehe [Zeile 24](#)):

```

1 function warenkorbPositionEinfügen(mySession, myKundeId, myArtikelId, myMenge)
2     {
3         var dbOshop = mySession.getSchema('oshop');
4         var collWarenkörbe = dbOshop.getCollection('warenkörbe');
5         var sql = "SELECT";
6             sql += " artikel_id, bezeichnung, einzelpreis, waehrung";
7             sql += " FROM oshop.artikel";
8             sql += " WHERE artikel_id=?";
9         var artikel = mySession.sql(sql).bind(myArtikelId).execute().fetchOne();
10        var objArtikel = {
11            artikel_id: artikel.artikel_id,
12            artikelname: artikel.bezeichnung,
13            preis: artikel.einzelpreis
14        };
15        var objPosition = {
```

```

15     artikel: objArtikel,
16     menge: myMenge,
17     gesamtpreis: myMenge * artikel.einzelpreis,
18     währung: artikel.waehrung
19   };
20   collWarenkörbe.modify({_id=:param}).arrayInsert('warenkorb.positionen[0]', {
21     position: objPosition
22 }).bind('param', myKundeId).execute();
23 var wk = collWarenkörbe.find({_id=:param}).bind('param', myKundeId).execute
24   ().fetchOne();
25 wk = aktualisiereGesamtpreisWarenwert(wk);
26 collWarenkörbe.modify({_id=:param}).set('$', wk).bind('param', myKundeId).
27   execute();
28
29 }
30 {
31   "position": {
32     "artikel": {
33       "artikel_id": 3007,
34       "artikelname": "Tinte
35         (blau)",
36       "preis": "4.170000"
37     },
38     "gesamtpreis": 41.7,
39     "menge": 10,
40     "währung": "EUR"
41   }
42 },
43 "warenwerte": [
44   {
45     "EUR": 51.800000000000004
46   },
47   {
48     "USD": 75.5
49   }
50 ],
51 }
52 }

```



Aufgabe 17.4: Wie können Sie sich das Ergebnis anschauen? Programmieren Sie es und überprüfen Sie, ob der Gesamtpreis und die Warenwerte stimmen. Das Ergebnis müsste so aussehen:

```

1 {
2   "_id": 2,
3   "warenkorb": {
4     "kunde_id": 2,
5     "positionen": [
6       {
7         "position": {
8           "artikel": {
9             "artikel_id": 3010,
10            "artikelname": "Feder"
11            ,
12            "preis": "5.050000"
13          },
14          "gesamtpreis": 10.1,
15          "menge": 2,
16          "währung": "EUR"
17        }
18      },
19      {
20        "position": {
21          "artikel": {
22            "artikel_id": 9010,
23            "artikelname": "
24              Schaufel",
25            "preis": "15.100000"
26          },
27          "gesamtpreis": 75.5,
28          "menge": 5,
29          "währung": "USD"
30        }
31      }
32    ],
33    "warenwerte": [
34      {
35        "EUR": 51.800000000000004
36      },
37      {
38        "USD": 75.5
39      }
40    ]
41  }
42 }

```

17.3.3 Inhalte des Warenkorbs auswerten

Für die Darstellung auf der WEB-Oberfläche soll der Inhalt des Warenkorbs ausgewertet werden.

Beispiel: Zu den Artikeln sollen die Warengruppen ausgegeben werden

Um zu den Artikeln die Warengruppen ausgeben zu können, bedienen wir uns des guten alten SQL. Der SELECT ermittelt für alle Artikel die Warengruppe, sodass ich das Ergebnis in Zeile 31 als Nachschlage-Tabelle nutzen kann. Aber erst mal zur Ausführung des Befehls. Anders als oben wird der SELECT hier nicht über eine Tabellenreferenz abgesetzt, sondern in Zeile 15 direkt über die Session. Die Session stellt dafür die Methode sql() zur Verfügung. Die Abfrage liefert mir mehr als eine Zeile zurück, die ich auch direkt komplett verarbeiten möchte, sodass ich nicht wie oben fetchOne(), sondern fetchAll() verwende. Diese Ergebnismenge wird ab Zeile 17 in ein Array konvertiert, da damit leichter zu arbeiten ist. Der Rest sollte selbsterklärend sein ;-)

```

1  JS > function printWarenkorbMitWarengruppen(mySession, myKundeId) {
2    -> var dbOshop = mySession.getSchema('oshop');
3    -> var collWarenkörbe = dbOshop.getCollection('warenkörbe');
4    ->
5    -> var select = "SELECT";
6    -> select += " artikel.artikel_id,";
7    -> select += " GROUP_CONCAT(warengruppe.bezeichnung) 'warengruppen'";
8    -> select += " FROM";
9    -> select += " oshop.artikel INNER JOIN oshop.artikel_nm_warengruppe";
10   -> select += " USING(artikel_id)";
11   -> select += " INNER JOIN";
12   -> select += " oshop.warengruppe";
13   -> select += " USING(warengruppe_id)";
14   -> select += " GROUP BY artikel.artikel_id";
15   -> var alleArtikel = mySession.sql(select).execute().fetchAll();
16   ->
17   -> var arrArtikel = new Array();
18   -> for (index in alleArtikel) {
19     -> var zeile = alleArtikel[index];
20     -> arrArtikel.push({
21       -> artikel_id: zeile["artikel_id"],
22       -> warengruppen: zeile["warengruppen"]
23     });
24   }
25   ->
26   -> var wk = collWarenkörbe.find({_id:=param}).bind('param', myKundeId).
27     execute().fetchOne();
28   -> var n = 1;
29   -> var items = Object.keys(wk.warenkorb.positionen);
30   -> items.forEach(function(item) {
31     -> var myPos = wk.warenkorb.positionen[item].position;
32     -> var wg = arrArtikel.find(function(element) {
33       -> return element.artikel_id == myPos.artikel.artikel_id
34     });
35     -> print('Warenkorbposition ' + n++ + ':');
36     -> print(' ' + myPos.artikel.artikel_id);
37     -> print(', ' + myPos.artikel.artikelname);
38     -> print(', ' + wg.warengruppen + "'");
39     -> print(', ' + myPos.artikel.preis);
40     -> print(', ' + myPos.menge);
41     -> print(', ' + myPos.gesamtpreis);
42     -> print(', ' + myPos.währung + "\n");
43   });
44 }
```

```

45 JS > printWarenkorbMitWarengruppen(session, 2);
46 Warenkorbposition 1: 3010, Feder, "Bürobedarf", 5.050000, 2, [...]
47 Warenkorbposition 2: 9010, Schaufel, "Gartenbedarf,Werkzeug", [...]
48 Warenkorbposition 3: 3007, Tinte (blau), "Bürobedarf", 4.1700[...]

```

Beispiel: Anzahl der Artikel pro Warenguppe

Eigentlich wäre es doch ganz nützlich, wenn die Warengruppen direkt bei den Artikeln stehen würden. Auch dies lässt sich nun einfach mit einem Misch von SQL und JavaScript bewerkstelligen:

```

1  function erweiternUmWarengruppen(mySession, myKundeId) {
2    var db0shop = mySession.getSchema('oshop');
3    var collWarenkörbe = db0shop.getCollection('warenkörbe');
4    var wk = collWarenkörbe.find({"_id=:param").bind('param', myKundeId).execute()
5      ().fetchOne();
6
7    var items = Object.keys(wk.warenkorb.positionen);
8    items.forEach(function(item) {
9      var myPos = wk.warenkorb.positionen[item].position;
10
11    var select = "SELECT";
12    select += " warengruppe_id, bezeichnung";
13    select += " FROM";
14    select += " oshop.artikel_nm_warengruppe";
15    select += " INNER JOIN";
16    select += " oshop.warengruppe";
17    select += " USING(warengruppe_id)";
18    select += " WHERE artikel_id = ?";
19
20    var wgs = mySession.sql(select).bind(myPos.artikel.artikel_id).execute().
21      fetchAll();
22    arr = new Array();
23    for (index in wgs) {
24      var zeile = wgs[index];
25      arr.push({
26        warengruppe_id: zeile.warengruppe_id,
27        bezeichnung: zeile.bezeichnung
28      });
29    }
30    myPos.artikel.warengruppen = arr;
31    collWarenkörbe.modify('_id=:param').set('$', wk).bind('param', myKundeId).
32      execute();
33  }

```

Die JSON-Struktur wird damit erweitert und nach dem Aufruf sieht sie dann wie folgt aus:

```

1  [...]
2  {
3    "position": {
4      "artikel": {
5        "artikel_id": 9010,
6        "artikelname": "Schaufel",
7        "preis": "15.100000",
8        "warengruppen": [
9          {

```

```

10          "bezeichnung": "Gartenbedarf",
11          "warengruppe_id": 3
12      },
13      {
14          "bezeichnung": "Werkzeug",
15          "warengruppe_id": 4
16      }
17  ],
18  },
19  "gesamtpreis": 75.5,
20  "menge": 5,
21  "währung": "USD"
22 },
23 [...]
24 [...]
```

Ich habe bereits oben erwähnt, dass in NoSQL-Lösungen oft auf Redundanzfreiheit verzichtet wird. Vielmehr stattet man die Entitäten mit möglichst vielen Informationen aus, sodass bei einer Verarbeitung eben keine Tabellen mehr verknüpft werden müssen; die Information ist ja in unmittelbarer Nähe vorhanden. Wir können die Vorteile hier gut beobachten, indem wir auf Basis der erweiterten JSON-Struktur ermitteln, wie oft eine Warengruppe im Warenkorb vertreten ist:

```

1 JS > function getAnzahlWarenguppe(mySession, myKundeId) {
2   -> var ergebnis = Object();
3   -> var posindizes = Object.keys(wk.warenkorb.positionen);
4   -> posindizes.forEach(function(posindex) {
5     -> var myPos = wk.warenkorb.positionen[posindex].position;
6     -> var wgs = myPos.artikel.warenguppen;
7     -> var wgindizes = Object.keys(wgs);
8     -> wgindizes.forEach(function(wgindex) {
9       -> if (ergebnis.hasOwnProperty(wgs[wgindex].bezeichnung)) {
10         -> ergebnis[wgs[wgindex].bezeichnung]++;
11       } else {
12         -> ergebnis[wgs[wgindex].bezeichnung] = 1;
13       }
14     })
15   });
16   ->
17   -> return ergebnis;
18 }
19
20 JS >
21 JS > getAnzahlWarenguppe(session, 2);
22 {
23   "Bürobedarf": 2,
24   "Gartenbedarf": 1,
25   "Werkzeug": 1
26 }
```

17.3.4 Inhalte des Warenkorbs verändern

Beispiel: Änderung der Mengenangabe

Der Kunde ändert über den Client die Mengenangabe im Warenkorb. Diese Änderung soll nun im JSON-Dokument abgelegt werden. Nichts ist einfacher als das. Im Grunde werden

folgende Schritte unternommen: finde die richtige Warenkorbposition zur Artikelnummer, lege den neuen Wert dort ab und lasse Gesamtpreis und Warenkorbwerte neu berechnen.

```

1 JS > function mengenÄndern(mySession, myKundeId, myArtikelId, myMenge) {
2   -> var db0shop = mySession.getSchema('oshop');
3   -> var collWarenkörbe = db0shop.getCollection('warenkörbe');
4   -> var wk = collWarenkörbe.find({_id=:param}).bind('param', myKundeId).
5     execute().fetchOne();
6   ->
7   -> for (index in Object.keys(wk.warenkorb.positionen)) {
8     -> if (wk.warenkorb.positionen[index].position.artikel.artikel_id ==
9       myArtikelId) {
10      -> wk.warenkorb.positionen[index].position.menge = myMenge;
11    }
12  -> wk = aktualisiereGesamtpreisWarenwert(wk);
13  -> collWarenkörbe.modify('_id=:param').set('$', wk).bind('param',
14    myKundeId).execute();
15  ->
16  -> return wk;
17 }
18 JS > mengenÄndern(session, 2, 3010, 0);
19 JS > mengenÄndern(session, 2, 3007, 1);
20 JS > mengenÄndern(session, 2, 9010, 2);

```

Beispiel: Löschen einer Warenkorbposition

Der Kunde 2 löscht beispielsweise die Warenkorbposition mit der Artikelnummer 3007. Auch hier können wir schon komplett auf SQL verzichten. Die Warenkorbposition wird im JSON-Dokument gesucht und mit `arrayDelete()` in Zeile 9 entfernt.

```

1 JS > function warenkorbPositionLöschen(mySession, myKundeId, myArtikelId) {
2   -> var db0shop = mySession.getSchema('oshop');
3   -> var collWarenkörbe = db0shop.getCollection('warenkörbe');
4   -> var wk = collWarenkörbe.find({_id=:param}).bind('param', myKundeId).
5     execute().fetchOne();
6   ->
7   -> for (index in Object.keys(wk.warenkorb.positionen)) {
8     -> if (wk.warenkorb.positionen[index].position.artikel.artikel_id ==
9       myArtikelId) {
10      -> var str = 'warenkorb.positionen[' + index + ']';
11      -> collWarenkörbe.modify('_id=:param').arrayDelete(str).bind('param',
12        myKundeId).execute();
13      -> wk = collWarenkörbe.find({_id=:param}).bind('param', myKundeId).
14        execute().fetchOne();
15      -> wk = aktualisiereGesamtpreisWarenwert(wk);
16    }
17  ->
18  -> return wk;
19 }
20 JS > warenkorbPositionLöschen(session, 2, 3007);

```

■ 17.4 NoSQL: klassisches SQL mit JSON-Funktionen

In MySQL stehen zwei Möglichkeiten zur Verfügung, JSON-Objekte zu verarbeiten: die Dokumentensammlung, die wir oben verwendet haben, und Tabellenspalten vom Datentyp JSON. Da es für jeden Kunden nur einen Warenkorb geben soll, bietet es sich auch an, diesen als JSON-Spalte in der Tabelle kunde zu speichern. Die Tabelle kunde muss somit um die entsprechende Spalte erweitert werden:

```
1 ALTER TABLE kunde
2   ADD warenkorb JSON AFTER COLUMN art
3 ;
```

17.4.1 Anlegen eines Warenkorbs

Wir haben vorbereitend jedem Kunden die Spalte warenkorb zur Verfügung gestellt. Diese Spalte sollte zu Beginn einen leeren Warenkorb – wie auf Seite [295](#) modelliert – enthalten. Wie bauen wir einen solchen Warenkorb nun auf? Im ersten Schritt sollte nach dem Anlegen eines neuen Kunden in dieser Spalte ein leerer Warenkorb zugewiesen werden⁴.

MySQL stellt Funktionen für JSON-Dokumente zur Verfügung, die ich hier nicht ausführlich besprechen kann. Die meisten ergeben sich aber selbsterklärend aus dem Kontext. Wir wollen erst mal allen vorhandenen Kunden einen leeren Warenkorb zuweisen:

```
1 mysql> UPDATE kunde
2   -> SET warenkorb = JSON_OBJECT(
3   ->           'warenkorb',
4   ->           JSON_OBJECT(
5   ->             'knr', kunde_id,
6   ->             'positionen', JSON_ARRAY(),
7   ->             'warenwerte', JSON_ARRAY()
8   ->           )
9   ->         )
10  -> ;
11 Query OK, 22 rows affected (0.19 sec)
12 Rows matched: 22  Changed: 22  Warnings: 0
13
14 mysql> SELECT kunde_id, warenkorb FROM kunde;
15 +-----+-----+
16 | kunde_id | warenkorb          |
17 +-----+-----+
18 |      1 | {"warenkorb": {"knr": 1, "positionen": [], "warenwerte": []}} |
19 |      2 | {"warenkorb": {"knr": 2, "positionen": [], "warenwerte": []}} |
20 [...]
21 |     21 | {"warenkorb": {"knr": 21, "positionen": [], "warenwerte": []}} |
22 |     22 | {"warenkorb": {"knr": 22, "positionen": [], "warenwerte": []}} |
23 +-----+-----+
```

Das Ergebnis ist recht vielversprechend, aber nun die Funktionen im Einzelnen:

⁴ Das reicht geradezu nach einem Trigger (siehe [Abschnitt 22 auf Seite 363](#)).

- **JSON_OBJECT()**: Diese Funktion bekommt eine beliebig lange, aber gerade Anzahl von Parametern übergeben. Der ungerade Parameter wird dabei immer als *Schlüssel* und der gerade als *Wert* interpretiert, formatiert und als ein gültiges JSON-Objekt verpackt. Eine leere Klammer liefert ein leeres Objekt.
- **JSON_ARRAY()**: Diese Funktion bekommt eine beliebige Anzahl von Parametern übergeben und liefert diese als ein JSON-Array zurück. Eine leere Parameterliste liefert ein leeres Array.

17.4.2 Inhalte des Warenkorbs anlegen

Beispiel: Kunde 2 bestellt zehnmal Artikel 3007.

Ab [Zeile 1](#) werden die Vorgaben in Variablen gespeichert. Dies macht den Quelltext erheblich flexibler. In [Zeile 5](#) wird der vorhandene Warenkorb des Kunden ebenfalls in einer Variablen abgelegt und anschließend wird in [Zeile 6](#) das Array mit den Positionen extrahiert. In [Zeile 8](#) wird die neue Position aus den Tabellen und den VorgabevARIABLEN ermittelt, damit sie in [Zeile 26](#) an das Positionsarray angehängt und in die Kundentabelle zurückgeschrieben werden kann.

```

1 mysql> SELECT 2 INTO @kunde_id;                                -- Vorgaben
2 mysql> SELECT 3007 INTO @artikel_id;
3 mysql> SELECT 10 INTO @menge;
4
5 mysql> SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id INTO @wk;
6 mysql> SELECT JSON_EXTRACT(@wk, "$.warenkorb.positionen");
7
8 mysql> SELECT
9     -> JSON_OBJECT(
10    -> 'position', JSON_OBJECT(
11      -> 'artikel', JSON_OBJECT(
12        -> 'artikel_id', CAST(artikel_id AS CHAR),
13        -> 'bezeichnung', bezeichnung,
14        -> 'einzelpreis', einzelpreis
15        -> ),
16        -> 'menge', @menge,
17        -> 'gesamtpreis', @menge * einzelpreis,
18        -> 'währung', waehrung
19        -> )
20    -> )
21    -> FROM artikel
22    -> WHERE artikel_id = @artikel_id
23    -> INTO @position
24    -> ;
25
26 mysql> UPDATE kunde
27    -> SET warenkorb = JSON_ARRAY_APPEND(
28    ->           @wk,
29    ->           "$.warenkorb.positionen",
30    ->           CAST(@position AS JSON)
31    ->           )
32    -> WHERE kunde_id = @kunde_id
33    -> ;

```

Das Ergebnis lässt sich aufgehübscht leicht interpretieren:

```

1 mysql> SELECT JSON_PRETTY(warenkorb) FROM kunde WHERE kunde_id = @kunde_id\G
2 **** 1. row ****
3 JSON_PRETTY(warenkorb): {
4     "warenkorb": {
5         "kunde_id": 2,
6         "positionen": [
7             {
8                 "position": {
9                     "menge": 10,
10                    "artikel": {
11                        "artikel_id": 3007,
12                        "bezeichnung": "Tinte (blau)",
13                        "einzelpreis": 4.17
14                    },
15                    "gesamtpreis": 41.7,
16                    "währung": "EUR"
17                }
18            }
19        ],
20        "warenwerte": []
21    }
22 }
```

Und was ist neu?

- **JSON_EXTRACT()**: Diese Funktion liefert ein JSON-Objekt oder einen JSON-Wert aus einem Dokument. Dazu muss sie wissen, wo das Objekt oder der Wert im Dokument liegt. Deshalb hat diese Funktion drei Parameter: das JSON-Dokument, den Pfad im Dokument zum Wert und den Schlüssel des Objektes oder des Wertes, der zurückgeliefert werden soll.
- **JSON_ARRAY_APPEND()**: Diese Funktion hängt ein neues Element einem Array an. Dazu muss sie wissen, wo das Array im Dokument abgelegt ist. Deshalb hat diese Funktion drei Parameter: das JSON-Dokument, den Pfad zum Wert im Dokument und das neue Arrayelement.
- **JSON_PRETTY()**: Diese Funktion liefert das JSON-Dokument in einer für Menschen gut lesbaren Form zurück.



Aufgabe 17.5: Erweitern Sie den Quelltext so, dass Kunde 2 zweimal Artikel 3010 und fünfmal Artikel 9010 bestellt. Das Ergebnis müsste so aussehen:

```

mysql> SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id\G
**** 1. row ****
warenkorb: {"warenkorb": {"kunde_id": 2, "positionen": [{"position": {"menge": 10, "artikel": {"artikel_id": 3007, "bezeichnung": "Tinte (blau)", "einzelpreis": 4.17}, "gesamtpreis": 41.7, "währung": "EUR"}}, {"position": {"menge": 2, "artikel": {"artikel_id": 3010, "bezeichnung": "Feder", "einzelpreis": 5.05}, "gesamtpreis": 10.1, "währung": "EUR"}}, {"position": {"menge": 5, "artikel": {"artikel_id": 9010, "bezeichnung": "Schaufel", "einzelpreis": 15.1}, "gesamtpreis": 75.5, "währung": "USD"}}], "warenwerte": []}}
```

Was wir bisher schön vergessen haben, ist die Berechnung der Warenwerte des Warenkorbs. Das möchte ich auf später verschieben.

17.4.3 Inhalte des Warenkorbs auswerten

Für die Darstellung auf der WEB-Oberfläche soll der Inhalt des Warenkorbs ausgewertet werden. Da wir hier nur mit SQL-Befehlen arbeiten wollen, müssen wir das JSON-Dokument immer wieder in Tabellenform bringen, um dieses auswerten zu können. Hier hilft die Funktion `JSON_TABLE()`:

```

1 mysql> SELECT 2 INTO @kunde_id;
2 mysql> SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id INTO @wk;
3 mysql>
4 mysql> SELECT @kunde_id AS 'kunde_id', wk.* 
5     ->   FROM
6     ->   JSON_TABLE(
7     ->     @wk,
8     ->     '$.warenkorb.positionen[*]'
9     ->     COLUMNS (
10    ->       posnr      FOR ORDINALITY,
11    ->       artikel_id INT          PATH '$.position.artikel.artikel_id',
12    ->       einzelpreis DECIMAL(12,2) PATH '$.position.artikel.einzelpreis',
13    ->       menge      INT          PATH '$.position.menge',
14    ->       währung    CHAR(3)      PATH '$.position.währung'
15    ->     )
16    ->   ) AS wk
17    -> ;
18 +-----+-----+-----+-----+-----+
19 | kunde_id | posnr | artikel_id | einzelpreis | menge | währung |
20 +-----+-----+-----+-----+-----+
21 | 2 | 1 | 3007 | 4.17 | 1 | EUR |
22 | 2 | 2 | 3010 | 5.05 | 0 | EUR |
23 | 2 | 3 | 9010 | 15.10 | 10 | USD |
24 +-----+-----+-----+-----+-----+

```

Man kann hier gut sehen, wie das JSON-Dokument durchwandert wird und die Inhalte auf Tabellenspalten verteilt werden. Die Angabe `FOR ORDINALITY` in [Zeile 10](#) ist so eine Art `MINI-AUTO_INCREMENT` und liefert mir eine eindeutige Zeilennummer.

Beispiel: Anzahl der Artikel pro Warengruppe

Wenn man einmal verstanden hat, wie das mit `JSON_TABLE()` funktioniert, ist eigentlich alles wie gehabt. In [Zeile 16](#) wird das JSON-Dokument in eine Tabelle überführt. Danach wird diese in der [Zeile 17](#) und [Zeile 18](#) mit den entsprechenden Tabellen der *n:m*-Verknüpfung verknüpft.

```

1 mysql> SELECT 2 INTO @kunde_id;
2 mysql> SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id INTO @wk;
3 mysql>
4 mysql> SELECT bezeichnung AS 'warengruppe', COUNT(*) AS 'anzahl' 
5     ->   FROM
6     ->   JSON_TABLE(
7     ->     @wk,
8     ->     '$.warenkorb.positionen[*]'
9     ->     COLUMNS (
10    ->       posnr      FOR ORDINALITY,
11    ->       artikel_id INT          PATH '$.position.artikel.artikel_id',
12    ->       einzelpreis DECIMAL(12,2) PATH '$.position.artikel.einzelpreis',
13    ->       menge      INT          PATH '$.position.menge',
14    ->       währung    CHAR(3)      PATH '$.position.währung'

```

```

15      ->      )
16      ->      ) AS wk
17      ->      INNER JOIN artikel_nm_warengruppe USING(artikel_id)
18      ->      INNER JOIN warengruppe           USING(warengruppe_id)
19      ->      GROUP BY bezeichnung
20      -> ;
21 +-----+-----+
22 | warengruppe | anzahl |
23 +-----+-----+
24 | Bürobedarf   |    2 |
25 | Gartenbedarf  |    1 |
26 | Werkzeug     |    1 |
27 +-----+-----+

```

Richtig schön wird das erst, wenn das Erstellen der Tabelle mithilfe von CTEs ([Abschnitt 11.5 auf Seite 208](#)) vorangestellt wird. Man erkennt deutlich, dass die eigentliche Abfrage leichter zu lesen und zu interpretieren ist:

```

1 mysql> WITH wk AS (
2       -> SELECT * FROM JSON_TABLE(
3       ->      @wk,
4       ->      '$.warenkorb.positionen[*]')
5       ->      COLUMNS (
6       ->          posnr      FOR ORDINALITY,
7       ->          artikel_id  INT          PATH '$.position.artikel.artikel_id',
8       ->          einzelpreis DECIMAL(12,2) PATH '$.position.artikel.einzelpreis',
9       ->          menge      INT          PATH '$.position.menge',
10      ->          währung    CHAR(3)      PATH '$.position.währung'
11      ->      )
12      ) AS t
13      )
14      -> SELECT bezeichnung AS 'warengruppe', COUNT(*) AS 'anzahl'
15      -> FROM
16      ->      wk INNER JOIN artikel_nm_warengruppe USING(artikel_id)
17      ->      INNER JOIN warengruppe           USING(warengruppe_id)
18      ->      GROUP BY bezeichnung
19      -> ;
20 +-----+-----+
21 | warengruppe | anzahl |
22 +-----+-----+
23 | Bürobedarf   |    2 |
24 | Gartenbedarf  |    1 |
25 | Werkzeug     |    1 |
26 +-----+-----+

```

17.4.4 Inhalte des Warenkorbs verändern

Wie werden Werte verändert? Grundsätzlich ist die größte Schwierigkeit – nach dem Berechnen des neuen Wertes –, die Stelle im JSON-Dokument zu finden, wo der Wert geändert werden soll. Schließlich können wir nicht einfach wie bei Tabellen über Spaltennamen direkt darauf zugreifen, sondern wir müssen im Dokumentenbaum den Pfad finden, der zur richtigen Position führt.

Beispiel: Kunde 2 ändert bei Artikel 3007 die Anzahl auf 1.

In Zeile 6 wird die Artikelnummer im Warenkorb gesucht. Da diese nur einmal im Warenkorb vorkommen kann, liefert sie uns die richtige Position. Danach wird in Zeile 8 innerhalb der zur Artikelnummer passenden Position die Mengenangabe zu finden⁵. Dazu wird die Position im String gesucht, wo die Artikelnummer angegeben wird. In der Ebene darüber ist dann die Mengenangabe. Zum Schluss wird die neue Mengenangabe mit `JSON_REPLACE()` an die richtige Position geschrieben (siehe Zeile 31).

```

1 mysql> SELECT 2 INTO @kunde_id;
2 mysql> SELECT 3007 INTO @artikel_id;
3 mysql> SELECT 1 INTO @menge;
4 mysql> SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id INTO @wk;
5
6 mysql> SELECT JSON_SEARCH(@wk, 'one', @artikel_id) INTO @path;
7
8 mysql> SELECT CONCAT(                                     -- Bastelarbeiten
9      ->      SUBSTRING(
10     ->      @path,
11     ->      2,
12     ->      22+LOCATE('.', LEFT(@path, 13))
13     ->      ),
14     ->      '.position.menge'
15     ->      )
16     ->      INTO @path_to_pos;
17
18 mysql> UPDATE kunde
19   -> SET warenkorb = JSON_REPLACE(@wk, @path_to_pos, @menge)
20   -> WHERE kunde_id = @kunde_id
21   -> ;
22
23 mysql> SELECT JSON_PRETTY(warenkorb) FROM kunde WHERE kunde_id = @kunde_id\G
24 **** 1. row ****
25 JSON_PRETTY(warenkorb): {
26   "warenkorb": {
27     "kunde_id": 2,
28     "positionen": [
29       {
30         "position": {
31           "menge": 1,           ←Hier
32           "artikel": {
33             "artikel_id": "3007",
34             "bezeichnung": "Tinte (blau)",
35             "einzelpreis": 4.17
36           },
37           "währung": "EUR",
38           "gesamtpreis": 41.7
39         }
40       },
41     [...]

```

Und was ist neu?

- `JSON_SEARCH()`: Diese Funktion liefert den Pfad zu einem Suchstring zurück und hat drei Parameter: das JSON-Dokument, die Angabe `'one'` oder `'all'` und den gesuchten

⁵ In einem Python- oder JavaScript- Client gestaltet sich dies viel eleganter. Aber was soll's, es klappt.

String. Bei 'one' wird der erste Fund zurückgeliefert und bei 'all' alle gefundenen Stellen.

- **JSON_REPLACE()**: Diese Funktion ersetzt einen vorhandenen Wert durch einen neuen. Auch diese Funktion hat drei Parameter: das JSON-Dokument, den Pfad im Dokument und den neuen Wert.



Hinweis: Böses Foul: Die Funktion `JSON_SEARCH()` findet nur Strings; numerische Werte werden nicht gefunden. Ein Workaround ist mir nicht bekannt!



Aufgabe 17.6: Erweitern Sie den Quelltext so, dass auch der Artikel 3010 0-mal und 9010 zehnmal bestellt wird. Das Ergebnis müsste so aussehen:

```
mysql> SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id\G
***** 1. row *****
warenkorb: {"warenkorb": {"kunde_id": 2, "positionen": [{"position": {"menge": 1, "artikel": {"artikel_id": "3007", "bezeichnung": "Tinte (blau)", "einzelpreis": 4.17}, "gesamtpreis": 41.7, "währung": "EUR"}}, {"position": {"menge": 0, "artikel": {"artikel_id": "3010", "bezeichnung": "Feder", "einzelpreis": 5.05}, "gesamtpreis": 10.1, "währung": "EUR"}}, {"position": {"menge": 10, "artikel": {"artikel_id": "9010", "bezeichnung": "Schaufel", "einzelpreis": 15.1}, "gesamtpreis": 75.5, "währung": "USD"}]}, "warenwerte": []}}
```

Aktualisierung von Gesamtpreis und Warenwert

Und jetzt sollte es auch kein Problem mehr sein, die oben ausgesparten Aktualisierungen (Gesamtpreis und Warenwert) zu ermitteln.

```
1 mysql> SELECT 2 INTO @kunde_id;
2 mysql> SELECT 3007 INTO @artikel_id;
3 mysql> SELECT 1 INTO @menge;
4
5 mysql> SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id INTO @wk;
6 mysql> SELECT JSON_SEARCH(@wk, 'one', @artikel_id) INTO @path;
7 mysql> SELECT CONCAT(
8     SUBSTRING(
9         @path,
10        2,
11        22+LOCATE('.', LEFT(@path, 13))
12    ),
13    '.position.menge'
14 )
15    INTO @path_to_pos;
16 mysql> SELECT CONCAT(
17     SUBSTRING(
18         @path,
19        2,
20        22+LOCATE('.', LEFT(@path, 13))
21    ),
22    '.position.gesamtpreis'
23 )
24    INTO @path_to_gesamtpreis;
25 mysql> SELECT '$.warenkorb.warenwerte' INTO @path_to_warenwerte;
```

```

26
27 mysql> CREATE TEMPORARY TABLE wk
28     -> SELECT * FROM JSON_TABLE(
29     ->     @wk,
30     ->     '$.warenkorb.positionen[*]')
31     ->     COLUMNS (
32     ->         posnr      FOR ORDINALITY,
33     ->         artikel_id INT          PATH '$.position.artikel.artikel_id',
34     ->         einzelpreis DECIMAL(12,2) PATH '$.position.artikel.einzelpreis',
35     ->         menge      INT          PATH '$.position.menge',
36     ->         währung    CHAR(3)      PATH '$.position.währung'
37     ->     )
38     -> ) AS t
39     -> ;
40
41 mysql> SELECT einzelpreis*menge
42     -> FROM wk
43     -> WHERE artikel_id = @artikel_id
44     -> INTO @gesamtpreis
45     -> ;
46
47 mysql> SELECT JSON_ARRAYAGG(warenwert)
48     -> FROM
49     -> (SELECT JSON_OBJECT(währung, SUM(einzelpreis*menge)) warenwert
50     ->     FROM wk
51     ->     GROUP BY währung) AS a
52     -> INTO @warenwerte
53     -> ;
54
55 mysql> UPDATE kunde
56     -> SET warenkorb = JSON_REPLACE(
57     ->             @wk,
58     ->             @path_to_pos, @menge,
59     ->             @path_to_gesamtpreis, @gesamtpreis,
60     ->             @path_to_warenwerte, CAST(@warenwerte AS JSON)
61     ->         )
62     -> WHERE kunde_id = @kunde_id
63     -> ;

```



Aufgabe 17.7: Überprüfen Sie mit `JSON_PRETTY()` das Ergebnis und versuchen Sie den Quelltext in Prosa zu formulieren. Machen Sie dabei keinen auf *SQL→Deutsch-Übersetzer*, sondern formulieren Sie die Aufgabe und den Zweck der jeweiligen Anweisung. Warum – beispielsweise – verwende ich hier eine temporäre Tabelle?

17.4.5 Inhalte des Warenkorbs löschen

Gehen wir davon aus, dass ein Kunde auf der Client-Oberfläche folgende Dinge tut: Der Kunde löscht eine Warenkorbposition. Gelöscht wird mit den Funktionen `JSON_REMOVE()`, `JSON_REPLACE()` oder `JSON_SET()`.

In Zeile 6 wird wieder die Stelle im JSON-Dokument gesucht, wo die Information steht. Dabei suche ich den ersten Punkt hinter dem Arrayelement und schneide den Rest ab. An-

schließend ersetze ich die Mengenangabe und speichere das Ergebnis zurück in die Tabelle.

```

1 mysql> SELECT 2 INTO @kunde_id;
2 mysql> SELECT 9010 INTO @artikel_id;
3 mysql> SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id INTO @wk;
4 mysql> SELECT JSON_SEARCH(@wk, 'one', @artikel_id) INTO @path;
5
6 mysql> SELECT SUBSTRING(@path, 2, 22+LOCATE('.', LEFT(@path, 13)))
7     -> INTO @path_to_pos;
8
9 mysql> UPDATE kunde
10    -> SET warenkorb = JSON_REMOVE(@wk, @path_to_pos)
11    -> WHERE kunde_id = @kunde_id
12    -> ;

```

Die Warenkorbposition ist verschwunden:

```

1 JSON_PRETTY(warenkorb): {                                17      {
2   "warenkorb": {                                         18      "position": {
3     "kunde_id": 2,                                         19      "menge": 2,
4     "positionen": [                                       20      "artikel": {
5       {                                                 21      "artikel_id": "3010",
6         "position": {                                     22      "bezeichnung": "Feder",
7           "menge": 1,                                     23      "einzelpreis": 5.05
8           "artikel": {                                 24      },
9             "artikel_id": "3007",                         25      "währung": "EUR",
10            "bezeichnung": "Tinte (               26      "gesamtpreis": 10.1
11              blau)",                                27      }
12            "einzelpreis": 4.17                         28      },
13          },                                              29      ],
14          "währung": "EUR",                            30      "warenwerthe": []
15          "gesamtpreis": 41.7                           31      }
16        }                                              32      }
}

```



Aufgabe 17.8: Programmieren Sie, dass der gesamte Warenkorb des Kunden geleert wird.

Was ist neu?

- **JSON_REPLACE()**: Diese Funktion ersetzt einen vorhandenen Inhalt durch einen neuen. Sie benötigt drei Parameter: das JSON-Dokument, den Pfad zum Wert und den neuen Wert.
- **JSON_SET()**: Diese Funktion erstellt einen neuen Inhalt oder ersetzt einen vorhandenen durch einen neuen. Sie ist damit sehr mit **JSON_REPLACE()** verwandt. Im Gegensatz zu dieser wird entweder ein vorhandener Wert ersetzt oder es wird ein neues Wert angelegt. Sie benötigt drei Parameter: das JSON-Dokument, den Pfad zum Wert und den neuen Wert.
- **JSON_REMOVE()**: Diese Funktion löscht das ganze Element aus dem Dokument. Es wird also nicht der Inhalt eines Elements geleert, sondern das ganze Element wird entfernt. Sie hat zwei Parameter: das JSON-Dokument und den Pfad zum Element.

TEIL V

Anweisungen kapseln

18

Locking



Bitte nicht alle gleichzeitig; immer der Reihe nach.

- Grundkurs
 - Zeilen-, Seiten- und Tabellensperre
 - Deadlock



Die Quelltexte des Kapitels stehen in der Datei `listing14.sql` (siehe [Listing 29.16 auf Seite 521](#) und [Listing 29.35 auf Seite 580](#)).

Eine Datenbank wird in vielen Fällen für Anwendungen, die auf mehreren Clients laufen, entwickelt. Denken Sie an ein Online-Forum, einen Internetshop, ein Zeiterfassungssystem etc. Dabei kommt es oft vor, dass zwei Anwendungen gleichzeitig auf eine Datenbank, eine Tabelle, eine Zeile oder einen Spalteninhalt zugreifen. Tun beide dies nur lesend, hat man kein Problem. Tun es beide oder auch nur eine von ihnen auch schreibend, hat man ein Problem. Konkret: Über eine offene Sitzung unseres Online-Shops werden fünf Spaten bestellt und über eine andere drei Spaten. Kein Problem, oder?

```
1 mysql> SELECT * FROM lagerbestand WHERE artikel_id = 9015;
2 +-----+-----+-----+
3 | artikel_id | menge_mindest | menge_aktuell | deleted |
4 +-----+-----+-----+
5 |      9015 |     10.000000 |    25.000000 |      0 |
6 +-----+-----+-----+
7
8 mysql> UPDATE lagerbestand
9      -> SET menge_aktuell = menge_aktuell - 5 WHERE artikel_id = 9015;
10
11 mysql> UPDATE lagerbestand
12      -> SET menge_aktuell = menge_aktuell - 3 WHERE artikel_id = 9015;
13
14 mysql> SELECT * FROM lagerbestand WHERE artikel_id = 9015;
15 +-----+-----+-----+
16 | artikel_id | menge_mindest | menge_aktuell | deleted |
17 +-----+-----+-----+
18 |      9015 |     10.000000 |    17.000000 |      0 |
19 +-----+-----+-----+
```

Dieser Selbstbetrug geht aber davon aus, dass diese beiden Aktualisierungen nacheinander ausgeführt werden. Aber warum sollten die beiden Sitzungen nicht fast gleichzeitig diese Aktualisierungen durchführen?

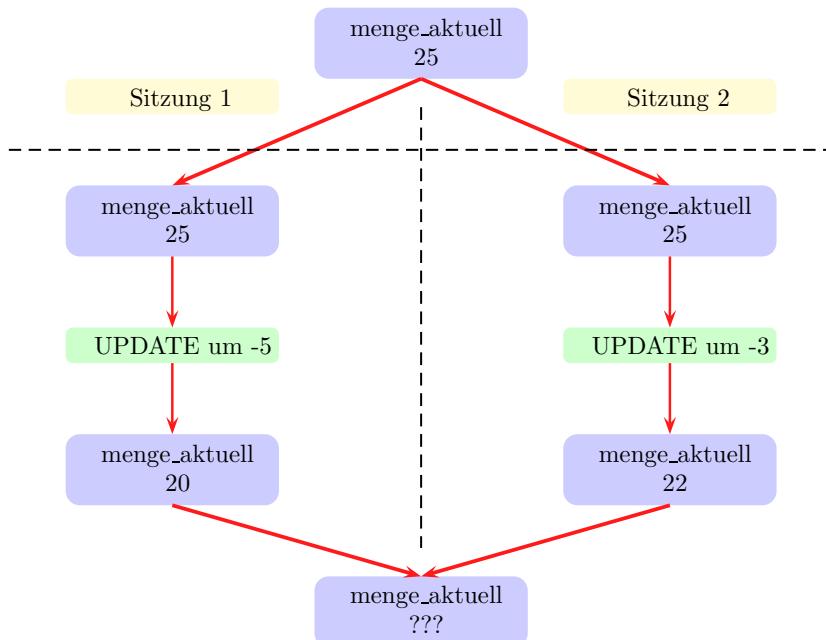


Bild 18.1 Was gilt jetzt?

In Bild 18.1 wird das Phänomen *lost update* verdeutlicht. Beide Sitzungen melden fast gleichzeitig, dass sie eine Aktualisierung des Lagerbestands machen wollen. Beide erhalten den aktuellen Lagerbestand zwecks Veränderung. Beide verändern ihre Bestände und wollen das Ergebnis in die Tabelle zurückschreiben. Egal, wer zuerst wegschreibt, das Ergebnis ist falsch!

Wir brauchen einen Mechanismus, der einer Sitzung sagt, dass sie einen Moment auf die andere warten soll. Diesen Mechanismus nennt man *Sperren*¹. Derzeit werden drei verschiedene Sperrstrategien verwendet:

- *Tabellensperre (table lock)*: Die gesamte Tabelle ist für die Dauer einer Änderung für andere gesperrt.
- *Seitensperre (page lock)*: Falls nicht die gesamte Tabelle betroffen ist, werden die Daten einer Tabelle in der Regel *seitenweise* eingelesen. Eine Seite hat meist eine Größe von 2 bis 16 KBytes. Alle Zeilen der Tabelle, die in der entsprechenden Seite sind, werden dabei für andere gesperrt. Die Zeilen dieser Seite müssen nicht in der Tabelle hintereinander liegen.

¹ engl.: *locking* oder *lock*

- **Zeilensperre (row lock):** Die einzelne oder die betreffenden Zeilen werden für andere gesperrt.



Hinweis: In MySQL und MariaDB unterstützt die InnoDB bzw. XtraDB die Zeilen- und MyISAM die Tabellensperre.

Normalerweise sollte man das Sperren der Engine bzw. dem Server überlassen. Dort sind Anwendungsregeln definiert, wann welche Sperre einzusetzen ist. Sie können in MySQL oder MariaDB aber auch eine Tabelle explizit mit `LOCK TABLES` sperren²:



MySQL/MariaDB

```
LOCK TABLES
    tabellenname [[AS] alias] sperrtyp[, tabellenname [[AS] alias] sperrtyp]*
;
sperrtyp:
    READ [LOCAL] | [LOW_PRIORITY] WRITE
```

Der `sperrtyp READ` sagt aus, dass ich aus dieser Tabelle etwas lesen möchte. Alle anderen Sitzungen können diese Tabelle ebenfalls noch lesen, ja sogar auch eine READ-Sperre auslösen. Keiner der Sitzungen, auch nicht die, welche diese Sperrung zuerst ausgelöst hat, kann jetzt in die Tabelle schreiben. Wird die Option LOCAL angegeben, können andere Sitzungen neue Datensätze einfügen, sofern diese nicht zu einem Konflikt mit der Leseoperation führen. In MySQL setzt die InnoDB READ mit READ LOCAL gleich.

Der zweite `sperrtyp WRITE` erlaubt es, dass die Sitzung, die die Sperre durchführt, auch in die Tabelle schreiben darf. Alle anderen Sitzungen müssen warten, bis die Sperre wieder aufgehoben wird. Selbst lesende Zugriffe sind jetzt nicht mehr erlaubt. Die Option LOW_PRIORITY kann verwendet werden, wenn der Schreibzugriff lange dauert und die Serverressourcen für Zugriffe auf andere Tabellen gebraucht werden.

Die Sperrung wird mit `UNLOCK TABLES` aufgehoben.



MySQL/MariaDB

```
UNLOCK TABLES
;
```



Hinweis: Wenn in einer Sitzung eine Tabelle gesperrt wurde, kann nur auf diese zugegriffen werden.

Äh, wie bitte? Das wollen wir uns mal genauer anschauen. Zuerst wird die Tabelle `artikel` gesperrt ([Zeile 1](#)). Eine Auswertung der Tabelle ist problemlos möglich ([Zeile 4](#)). Der Ver-

² Da es keinen SQL:2016-Standard zum Locking gibt, sind zwar die Strategien vergleichbar, aber die einzelnen Befehle und ihre Parameter oft sehr unterschiedlich. In den PostgreSQL-Varianten der Quelltextlistings habe ich – so weit sinnvoll – die MySQL/MariaDB-Beispiele überführt, um einen syntaktischen Vergleich zu ermöglichen.

such, auf eine andere Tabelle zuzugreifen, schlägt dann aber fehl ([Zeile 10](#)). Entweder sperrt man gleich mehrere Tabellen in [Zeile 1](#) oder man gibt die Tabelle wieder frei ([Zeile 13](#)).

```
1 mysql> LOCK TABLES artikel WRITE;
2 Query OK, 0 rows affected (0.00 sec)
3
4 mysql> SELECT COUNT(*) FROM artikel;
5 +-----+
6 | COUNT(*) |
7 +-----+
8 [...]
9
10 mysql> SELECT COUNT(*) FROM warengruppe;
11 ERROR 1100 (HY000): Table 'warengruppe' was not locked with LOCK TABLES
12
13 mysql> UNLOCK TABLES;
14 Query OK, 0 rows affected (0.00 sec)
15
16 mysql> SELECT COUNT(*) FROM warengruppe;
17 +-----+
18 | COUNT(*) |
19 +-----+
20 |      5   |
21 +-----+
```



Hinweis: Zum Locking gehört auch das Problem des *Deadlocks*. Mehr dazu in [Abschnitt 19.5 auf Seite 331](#).

19

Transaktion



Wo zwei oder drei in meinem Server versammelt sind, da ist die Transaktion mitten unter ihnen.

- Grundkurs
 - Was ist eine Transaktion?
 - Dead Lock
 - Start einer Transaktion mit START TRANSACTION
 - Impliziter Start einer Transaktion
 - COMMIT
 - ROLLBACK
- Vertiefendes
 - ACID-Test
 - Isolationsebene READ UNCOMMITTED
 - Isolationsebene READ COMMITTED
 - Isolationsebene REPEATABLE READ
 - Isolationsebene SERIALIZABLE
 - Transaktion aus einem C#-Client heraus



Die Quelltexte des Kapitels stehen in der Datei `Listing15.sql` (siehe [Listing 29.17 auf Seite 522](#) und [Listing 29.36 auf Seite 580](#)).

■ 19.1 Das Problem

Die Auswertungen und Änderungen, die wir bisher durchgeführt haben, sind mehr oder weniger atomar gewesen. Sie standen für sich alleine und hatte kaum Abhängigkeiten mit anderen Aufgaben. Dies ist aber nur der einfache Fall. Oft sind SQL-Anweisungen inhaltlich zusammenhängend und müssen zwingend *ganz oder gar nicht* ausgeführt werden.

Wir legen über eine Oberfläche den neuen Artikel Säge an. Gleichzeitig können wir über Checkboxen angeben, dass er zu den Warengruppen Gartenbedarf und Werkzeug gehört. Wenn wir jetzt auf OK drücken, müssen zwei Dinge passieren: In der Tabelle artikel muss der neue Artikel angelegt und in der Tabelle artikel_nm_warengruppe müssen die beiden Verknüpfungen erstellt werden.

Beide Anweisungen sollten kein Problem darstellen, oder?

```

1 mysql> INSERT INTO artikel
2      -> (bezeichnung, einzelpreis, waehrung)
3      -> VALUES
4      -> ('Säge', 17.85, 'EUR')
5      -> ;
6
7 mysql> INSERT INTO artikel_nm_warengruppe
8      -> VALUES
9      -> (3, ???)
10     -> ,(4, ???)
```

Da ist auf einmal eine Frage aufgetaucht. Da wir im INSERT auf die Tabelle artikel keine artikel_id vorgegeben haben, greift hier der AUTO_INCREMENT. Wir wissen aber nicht, welche Nummer dabei vergeben wurde. Wie soll man dann den zweiten INSERT ausführen?

Da hilft die MySQL-/MariaDB-Funktion LAST_INSERT_ID(). Sie liefert mir den Wert, der zuletzt durch ein AUTO_INCREMENT erzeugt wurde.

```

1 mysql> INSERT INTO artikel
2      -> (bezeichnung, einzelpreis, waehrung)
3      -> VALUES
4      -> ('Säge', 17.85, 'EUR')
5      -> ;
6
7 mysql> SET @id = LAST_INSERT_ID();
8
9 mysql> INSERT INTO artikel_nm_warengruppe
10     -> VALUES
11     -> (3, @id)
12     -> ,(4, @id)
13     -> ;
```

Man stelle sich aber mal vor, dass diese Anweisungen nicht *alle* ausgeführt werden. Ursachen kann es viele geben. Wenn wir uns im ersten Befehl vertippt haben, könnte das Ergebnis so aussehen:

```

1 mysql> INSERT ITO artikel[label={lst15_003}]
2      -> (bezeichnung, einzelpreis, waehrung)
3      -> VALUES
4      -> ('Säge', 17.85, 'EUR')
5      -> ;
6 ERROR 1064 (42000): You have an error in your SQL syntax; check the manual
       that corresponds to your MySQL server version for the right syntax to use
       near 'artikel'
7
8 mysql> SET @id = LAST_INSERT_ID();
9
10 mysql>
11 mysql> INSERT INTO artikel_nm_warengruppe
12     -> VALUES
```

```

13      ->      (3, @id)
14      ->      ,(4, @id)
15      -> ;
16 ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint
    fails ('oshop`.`artikel_nm_warengruppe', CONSTRAINT '
    artikel_nm_warengruppe_ibfk_2' FOREIGN KEY ('artikel_id') REFERENCES '
    artikel' ('artikel_id'))

```

Ein Haufen Fehlermeldungen, und glücklicherweise verhindert der Constraint, dass unsinnige Werte in die Hilfstabelle eingefügt werden.

Nehmen wir ein anderes Beispiel. Sam Gamdschie hat einen Spaten bestellt, und dieser wird jetzt versendet. Dann müssen zwei Dinge passieren: Erstens muss ein Spaten aus dem Lagerbestand entfernt und zweitens der Status der Bestellung auf `versendet` gesetzt werden.

```

1 UPDATE lagerbestand
2 SET menge_aktuell = menge_aktuell - 1
3 WHERE artikel_id = 9015
4 ;
5
6 UPDATE bestellung SET status = 'versendet' WHERE bestellung_id = 1;

```

Was würde passieren, wenn eine der beiden Anweisungen fehlschlägt und die andere trotzdem ausgeführt wird? In einem Fall hätten wir einen Fehlbestand im Lager, und im anderen Fall würden wir vermutlich die Bestellung noch einmal versenden. Beides bedeutet einen wirtschaftlichen Schaden.

■ 19.2 Was ist eine Transaktion?

Bevor ich auf verschiedene Aspekte der Theorie eingehe, möchte ich den grundsätzlichen Ablauf einer Transaktion verdeutlichen (siehe [Bild 19.1 auf der nächsten Seite](#)):

Der Ablauf sieht einigermaßen kompliziert aus, ist er aber nicht. Zuerst muss klargestellt werden, dass eine Transaktion zwar auf dem Server ausgeführt wird, aber durch einen Client¹ gesteuert wird.

- **Phase Transaktionsbeginn:** Der Client baut eine Verbindung auf und sendet den Befehl `START TRANSACTION` an den Server. Dieser richtet daraufhin eine Transaktionsumgebung ein. Was das genau ist, erkläre ich später. Im Prinzip läuft es aber darauf hinaus, dass man zu dem Datenbankzustand zurückkehren kann, der bei `START TRANSACTION` bestand.
- **Phase Transaktionsausführung:** Der Client sendet die erste Anweisung an den Server. Diese ist meist eine datenverändernde, denn bei Transaktionen kommt es in der Regel auf datenverändernde Anweisungen an. Der Server antwortet mit einer Exception oder einem `ERROR CODE (EC)`. Der Client entscheidet anhand der Fehlermeldung², ob er weitermachen möchte oder die Verarbeitung abgebrochen werden soll.

¹ PHP-Anwendung, .NET-Anwendung etc.

² Ich habe hier vereinfachend $EC > 0$ angenommen. Das ist nicht immer sinnvoll.

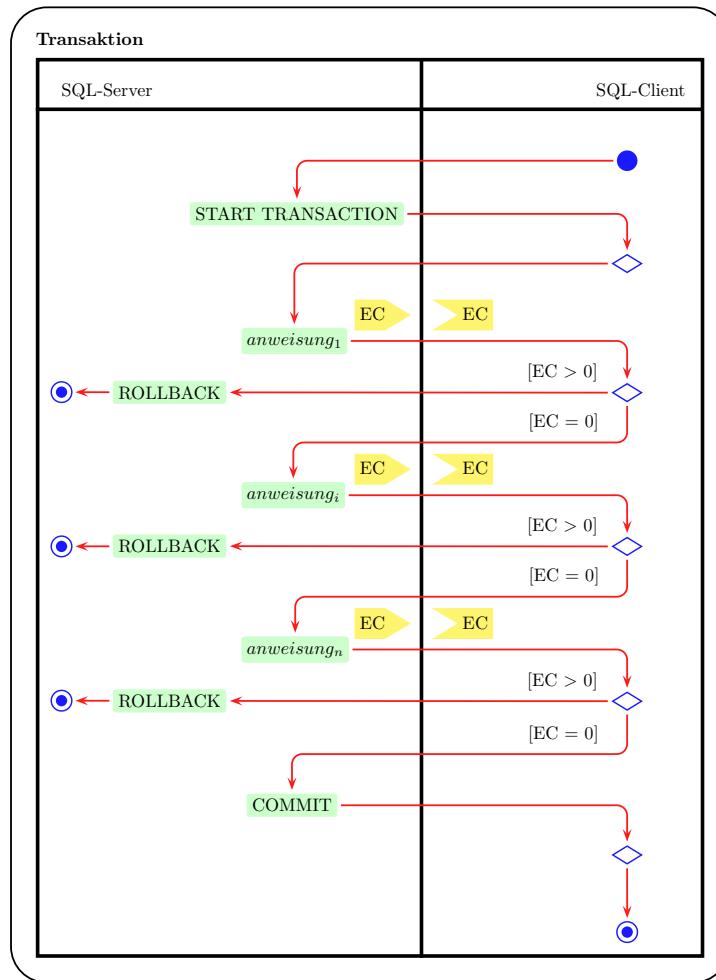


Bild 19.1 Ablauf einer Transaktion

- Will er abbrechen, so sendet er einen ROLLBACK an den Server. Der Server stellt den ursprünglichen Zustand wieder her, und die Transaktion ist beendet.
- Will er nicht abbrechen, so sendet er die nächste Anweisung an den Server.
- **Phase Transaktionsabschluss:** Alle Anweisungen sind wunschgemäß ausgeführt worden. Der Client hat dies überprüft und deshalb ein COMMIT an den Server. Erst jetzt werden die Änderungen tatsächlich in die Tabellen überführt.

Was soll das heißen, erst jetzt werden die Änderungen in die Tabellen überführt? Wo waren sie denn vorher? Das lässt sich nun nicht so einfach beantworten. Hier brauchen wir ein wenig Theorie.



Definition 60: Transaktion

Einen Anweisungsblock, der den ACID-Anforderungen entspricht, nennt man *Transaktion*.

ACID ist die Abkürzung für *Atomicity (Atomarität)*, *Consistency (Konsistenz)*, *Isolation* und *Durability (Dauerhaftigkeit)*.

- **Atomicity (Atomarität):** Der altgriechische Ursprung des Wortes soll ausdrücken, dass etwas nicht weiter aufteilbar ist³. Bzgl. der Transaktionen ist damit gemeint, dass man die Anweisungen der Transaktion nicht aufteilen kann. Entweder werden alle vollständig ausgeführt und umgesetzt oder keine⁴. Durch die Atomarität erscheint eine Transaktion nach außen wie eine Anweisung.
- **Consistency (Konsistenz):** Die Datenbank ist immer in einem konsistenten Zustand. Wird die Verarbeitung der Transaktion unter- oder abgebrochen, sind die Inhalte der Tabellen immer noch fachlich und technisch integer. Wie wird das hier erreicht? Durch das COMMIT. Würde die Verarbeitung der Transaktion nicht bis zum Ende durchlaufen, erfolgt kein COMMIT. Wenn die Datenbank vor der Transaktion in einem konsistenten Zustand war, so wird sie das nach einen ROLLBACK auch wieder sein. Nicht verhindert werden kann, dass innerhalb der Transaktion durch falsche Programmierung ein nicht konsistenter Zustand erreicht wird.
- **Isolation:** Da die Konsistenz der Datenbank während der Ausführung einer Transaktion nicht garantiert wird, sollten andere auch nicht auf diesem inkonsistenten Zustand arbeiten. Aus diesem Grund sind die Ergebnisse der Anweisungen innerhalb der Transaktion bis zum COMMIT für andere unsichtbar. Wie unsichtbar hier unsichtbar ist, wird mit den sogenannten *Isolationsebenen* (s.u.) festgelegt.
- **Durability (Dauerhaftigkeit):** Während die Ergebnisse der Anweisungen innerhalb der Transaktion *in der Schwebe sind*, sollten nach einem COMMIT die Änderungen wirklich und tatsächlich abgespeichert sein. Ein Systemabsturz oder Ähnliches sollte das Vorhandensein der Daten nicht mehr gefährden. Auch gibt es verschiedene Grade von Dauerhaftigkeit.



Hinweis: Unterschiede MySQL/MariaDB und SQL:2016:

- Damit man in MySQL oder MariaDB Transaktionen verwendet kann, muss die Engine transaktionsfähig sein. Die InnoDB bzw. XtraDB ist transaktionsfähig.
- In SQL:2016 wird eine neue Transaktion implizit nach Abschluss einer Transaktion gestartet.
- In MySQL oder MariaDB wird eine neue Transaktion implizit nach Abschluss einer Transaktion gestartet, wenn der AUTOCOMMIT-Modus ausgeschaltet ist.
- Ist in MySQL oder MariaDB der AUTOCOMMIT-Modus eingeschaltet, wird eine Transaktion mit START TRANSACTION begonnen. Während der Transaktion bleibt der AUTOCOMMIT-Modus bis zu einem COMMIT oder einem ROLLBACK ausgeschaltet.

³ altgr. ἀτομός: das Unzerschneidbare, von ἀ- *un-* und τέμνειν *schnieden*

⁴ Mithilfe von *snapshots* lässt sich diese strenge Forderung etwas aufweichen.

■ 19.3 Isolationsebenen

Eine Forderung im ACID war die Isolation. Dabei werden die Zwischenstände einer Transaktion vor anderen Transaktionen bzw. Sessions verborgen, um zu gewährleisten, dass diese immer nur einen konsistenten Datenzustand vorfinden.

Wenn Sie nichts eingestellt haben, wird Ihr Server vermutlich im Modus AUTOCOMMIT laufen. Dass bedeutet, dass jede Anweisung implizit von einem COMMIT abgeschlossen wird. Wir haben es dann mit lauter Transaktionen mit nur einer Anweisung zu tun. Um herauszufinden, ob meine Session im AUTOCOMMIT-Modus läuft, müssen wir den Inhalt der Variable AUTOCOMMIT auslesen:

```
1 mysql> SHOW VARIABLES LIKE 'AUTOCOMMIT';
2 +-----+-----+
3 | Variable_name | Value |
4 +-----+-----+
5 | autocommit     | ON   |
6 +-----+-----+
```

Die Werte ON und 1 haben die gleiche Bedeutung. Um mit den Transaktionen starten zu können, sollte der erste Befehl der Session den AUTOCOMMIT-Modus ausschalten:

```
1 mysql> SET AUTOCOMMIT = OFF;
2 Query OK, 0 rows affected (0.00 sec)
3
4 mysql> SHOW VARIABLES LIKE 'AUTOCOMMIT';
5 +-----+-----+
6 | Variable_name | Value |
7 +-----+-----+
8 | autocommit     | OFF  |
9 +-----+-----+
```

Jetzt gilt, dass wir immer in einer Transaktion sind, bis wir COMMIT oder ROLLBACK anwisen. Allerdings wird dann sofort eine neue Transaktion gestartet. Bei nicht transaktionsfähigen Engines wird der Wert der Variable AUTOCOMMIT ignoriert.



Hinweis: Es gibt Anweisungen, die implizit eine Transaktion beenden. Dies sind meist Befehle der DDL (siehe [Abschnitt 26.4.1 auf Seite 419](#)) wie ALTER TABLE.

Die Isolationsebene wird mit dem Befehl

```
1 SET SESSION TRANSACTION ISOLATION LEVEL isolationsebene;
```

festgelegt. Derzeit sind 4 Isolationsebenen in SQL spezifiziert: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ und SERIALIZABLE.

19.3.1 READ UNCOMMITTED

Es handelt sich eigentlich nicht um eine Isolationsebene, da die anderen Transaktionen oder Sessions den Zwischenstand lesen können. Da dabei auch inkonsistente Zustände auftreten können, wird dieser auch *dirty read* genannt.

SESSION 1

```

1 SET AUTOCOMMIT = 0;
2
3
4 SELECT artikel_id , menge_aktuell
5 FROM lagerbestand
6 WHERE artikel_id = 9015;
7 +-----+
8 | artikel_id | menge_aktuell |
9 +-----+
10 | 9015 | 16.000000 |
11 +-----+
12
13 UPDATE lagerbestand
14 SET menge_aktuell = menge_aktuell - 1
15 WHERE artikel_id = 9015;
16
17 SELECT artikel_id , menge_aktuell
18 FROM lagerbestand
19 WHERE artikel_id = 9015;
20 +-----+
21 | artikel_id | menge_aktuell |
22 +-----+
23 | 9015 | 15.000000 |
24 +-----+
25
26 ROLLBACK;
27
28 SELECT artikel_id , menge_aktuell
29 FROM lagerbestand
30 WHERE artikel_id = 9015;
31 +-----+
32 | artikel_id | menge_aktuell |
33 +-----+
34 | 9015 | 16.000000 |
35 +-----+

```

SESSION 2

```

1 SET SESSION TRANSACTION ISOLATION LEVEL
2 READ UNCOMMITTED;
3
4 SELECT artikel_id , menge_aktuell
5 FROM lagerbestand
6 WHERE artikel_id = 9015;
7 +-----+
8 | artikel_id | menge_aktuell |
9 +-----+
10 | 9015 | 16.000000 |
11 +-----+
12
13
14
15
16
17 SELECT artikel_id , menge_aktuell
18 FROM lagerbestand
19 WHERE artikel_id = 9015;
20 +-----+
21 | artikel_id | menge_aktuell |
22 +-----+
23 | 9015 | 15.000000 |
24 +-----+
25
26
27
28 SELECT artikel_id , menge_aktuell
29 FROM lagerbestand
30 WHERE artikel_id = 9015;
31 +-----+
32 | artikel_id | menge_aktuell |
33 +-----+
34 | 9015 | 16.000000 |
35 +-----+

```

Um dieses Beispiel nachzubauen, sollten Sie zwei Instanzen des MySQL oder MariaDB Clients gleichzeitig vor sich haben. In Session 1 wird in der Zeile 1 der AUTOCOMMIT-Modus so eingestellt, dass ich mich automatisch immer in einer Transaktion befinde.

In der gleichen Zeile wird in Session 2 dafür gesorgt, dass diese einen *dirty read* auf die Datenbank durchführen kann. Danach wird überprüft, dass auch in beiden Sessions die gleichen Daten vorliegen.

In Session 1 wird ab Zeile 13 der Lagerbestand der Spaten aktualisiert. Bitte beachten Sie, dass dies in der zweiten Session nicht passiert. In Zeile 17 wird jetzt in beiden Sessions wieder der Lagerbestand überprüft. Beide geben nun das gleiche Ergebnis aus, obwohl in Session 1 noch kein COMMIT erfolgt ist. Session 2 arbeitet also mit inkonsistenten Daten.

An diesem Beispiel können Sie auch den Effekt von ROLLBACK beobachten. In Session 1 Zeile 26 wird der ROLLBACK angewiesen. Eine anschließende Überprüfung zeigt, dass in beiden Sessions wieder der ursprüngliche Zustand hergestellt wurde.

Es gibt eigentlich keinen guten Grund, das READ UNCOMMITTED zu verwenden. In [SZT⁺09], Seite 9, steht, dass ein *dirty read* keinen nennenswerten Performancevorteil bietet und deshalb in der Praxis selten eingesetzt wird.

19.3.2 READ COMMITTED

Bei dieser Isolationsebene werden nur die Daten in einer Transaktion berücksichtigt, die durch andere mit einem COMMIT bestätigt wurden.

SESSION 1

```

1 SET AUTOCOMMIT = 0;
2
3
4 SELECT artikel_id , menge_aktuell
5 FROM lagerbestand
6 WHERE artikel_id = 9015;
7 +-----+-----+
8 | artikel_id | menge_aktuell |
9 +-----+-----+
10 | 9015 | 16.000000 |
11 +-----+
12
13 UPDATE lagerbestand
14 SET menge_aktuell = menge_aktuell - 1
15 WHERE artikel_id = 9015;
16
17 SELECT artikel_id , menge_aktuell
18 FROM lagerbestand
19 WHERE artikel_id = 9015;
20 +-----+-----+
21 | artikel_id | menge_aktuell |
22 +-----+-----+
23 | 9015 | 15.000000 |
24 +-----+
25
26 COMMIT;
27
28 SELECT artikel_id , menge_aktuell
29 FROM lagerbestand
30 WHERE artikel_id = 9015;
31 +-----+-----+
32 | artikel_id | menge_aktuell |
33 +-----+-----+
34 | 9015 | 15.000000 |
35 +-----+

```

SESSION 2

```

1 SET SESSION TRANSACTION
2 ISOLATION LEVEL READ COMMITTED;
3
4 SELECT artikel_id , menge_aktuell
5 FROM lagerbestand
6 WHERE artikel_id = 9015;
7 +-----+-----+
8 | artikel_id | menge_aktuell |
9 +-----+-----+
10 | 9015 | 16.000000 |
11 +-----+
12
13
14
15
16
17 SELECT artikel_id , menge_aktuell
18 FROM lagerbestand
19 WHERE artikel_id = 9015;
20 +-----+-----+
21 | artikel_id | menge_aktuell |
22 +-----+-----+
23 | 9015 | 16.000000 |
24 +-----+
25
26
27
28 SELECT artikel_id , menge_aktuell
29 FROM lagerbestand
30 WHERE artikel_id = 9015;
31 +-----+-----+
32 | artikel_id | menge_aktuell |
33 +-----+-----+
34 | 9015 | 15.000000 |
35 +-----+

```

In Session 2, Zeile 1 wird die Isolationsebene auf READ COMMITTED gesetzt. Dadurch erscheinen in Zeile 23 in Session 1 und Session 2 unterschiedliche Werte. In Session 1 wird mit dem veränderten und in Session 2 mit dem ursprünglichen Lagerbestand gearbeitet.

Der Nachteil dieser Isolationsebene ist aber auch zu erkennen. Die gleiche Anweisung führt zu unterschiedlichen Ergebnissen. Die Ergebnisse sind somit nicht wiederholbar (*repeatable*), was zu unerwünschten Effekten innerhalb einer Transaktion führen kann.



Hinweis: Erinnern Sie sich noch an die Fußnote bei COUNT()? Hier können Sie sehen, dass in unterschiedlichen Sitzungen unterschiedliche Ergebnisse ermittelt werden, obwohl diese auf der gleichen Tabelle arbeiten.



Aufgabe 19.1: Versuchen Sie, unterschiedliche COUNT()-Ergebnisse in zwei verschiedenen Sitzungen zu erzielen.

19.3.3 REPEATABLE READ

Das Ziel dieser Isolationsebene ist, dass in einer Transaktion, die selbst die Daten nicht verändert, die gleiche Auswertung das gleiche Ergebnis liefert. In MySQL und MariaDB ist diese Isolationsebene der Standard, wenn Sie eine Transaktion starten.

SESSION 1

```

1 SET AUTOCOMMIT = 0;
2
3
4
5
6 SELECT artikel_id , menge_aktuell
7 FROM lagerbestand
8 WHERE artikel_id = 9015;
9 +-----+-----+
10 | artikel_id | menge_aktuell |
11 +-----+-----+
12 | 9015 | 15.000000 |
13 +-----+-----+
14
15 UPDATE lagerbestand
16 SET menge_aktuell = menge_aktuell - 1
17 WHERE artikel_id = 9015;
18
19 COMMIT;
20
21 UPDATE lagerbestand
22 SET menge_aktuell = menge_aktuell - 1
23 WHERE artikel_id = 9015;
24
25 COMMIT;
26
27 SELECT artikel_id , menge_aktuell
28 FROM lagerbestand
29 WHERE artikel_id = 9015;
30 +-----+-----+
31 | artikel_id | menge_aktuell |
32 +-----+-----+
33 | 9015 | 13.000000 |
34 +-----+-----+

```

SESSION 2

```

1 SET SESSION TRANSACTION
2 ISOLATION LEVEL REPEATABLE READ;
3
4 START TRANSACTION;
5
6 SELECT artikel_id , menge_aktuell
7 FROM lagerbestand
8 WHERE artikel_id = 9015;
9 +-----+-----+
10 | artikel_id | menge_aktuell |
11 +-----+-----+
12 | 9015 | 15.000000 |
13 +-----+-----+
14
15
16
17
18
19
20
21
22
23
24
25
26
27 SELECT artikel_id , menge_aktuell
28 FROM lagerbestand
29 WHERE artikel_id = 9015;
30 +-----+-----+
31 | artikel_id | menge_aktuell |
32 +-----+-----+
33 | 9015 | 15.000000 |
34 +-----+-----+
35
36 COMMIT;
37
38 SELECT artikel_id , menge_aktuell
39 FROM lagerbestand
40 WHERE artikel_id = 9015;
41 +-----+-----+
42 | artikel_id | menge_aktuell |
43 +-----+-----+
44 | 9015 | 13.000000 |
45 +-----+-----+

```

In Session 2 wird in [Zeile 1](#) die Isolationsebene auf REPEATABLE READ eingestellt. Dadurch werden die Daten ab [Zeile 4](#) für diese Transaktion eingefroren. Selbst die COMMIT-Anweisungen in Session 1 ([Zeilen 19](#) und [25](#)) führen nicht dazu, dass in beiden Transaktionen die gleichen Werte gelten. Erst der COMMIT in Session 2 ([Zeile 36](#)) macht die Veränderungen von Session 1 in Session 2 sichtbar.

Somit ist gewährleistet, dass beide Sessions/Transaktionen mit in sich konsistenten Datenbeständen arbeiten.

19.3.4 SERIALIZABLE

Das Ziel der Isolationsebene REPEATABLE READ ist, dass innerhalb einer Transaktion die Daten nicht von außen verändert werden können. Trotzdem sind sogenannte *phantom reads* möglich. Nehmen wir mal an, dass in Session 1 neue Zeilen mit COMMIT in die Tabelle `artikel` hinzugefügt werden und in Session 2 Auswertungen auf der Tabelle stattfinden. Dann werden in Session 2 diese neuen Zeilen berücksichtigt.

Um dieses zu verhindern, sperrt die Isolationsebene SERIALIZABLE alles, was in einer Transaktion verwendet wird, für andere Transaktionen.

SESSION 1

```

1 SET AUTOCOMMIT = 0;
2
3
4
5
6 SELECT COUNT(*) FROM artikel;
7 +-----+
8 | COUNT(*) |
9 +-----+
10 |      10 |
11 +-----+
12
13 INSERT INTO artikel
14   VALUES
15     (1, 'X', 0.0, 'EUR', 0)
16 ;
17
18
19
20
21 COMMIT;
22
23
24
25
26
27
28
29 SELECT COUNT(*) FROM artikel;
30 +-----+
31 | COUNT(*) |
32 +-----+
33 |      11 |
34 +-----+
35
36 SELECT COUNT(*) FROM artikel;
37 +-----+
38 | COUNT(*) |
39 +-----+
40 |      10 |
41 +-----+

```

SESSION 2

```

1 SET SESSION TRANSACTION
2   ISOLATION LEVEL SERIALIZABLE;
3 START TRANSACTION;
4
5
6 SELECT COUNT(*) FROM artikel;
7 +-----+
8 | COUNT(*) |
9 +-----+
10 |      10 |
11 +-----+
12
13
14
15
16
17 DELETE FROM artikel WHERE artikel_id = 1;
18 ERROR 1205 (HY000):
19 Lock wait timeout exceeded;
20 try restarting transaction
21
22 DELETE FROM artikel WHERE artikel_id = 1;
23 SELECT COUNT(*) FROM artikel;
24 +-----+
25 | COUNT(*) |
26 +-----+
27 |      10 |
28 +-----+
29
30
31
32
33
34
35 COMMIT;
36 SELECT COUNT(*) FROM artikel;
37 +-----+
38 | COUNT(*) |
39 +-----+
40 |      10 |
41 +-----+

```

In Session 1 wird in [Zeile 13](#) ein neuer Datensatz in die Tabelle `artikel` eingefügt. In Session 2 wird nun versucht, diesen Artikel wieder zu löschen ([Zeile 17](#)). Dieser Datensatz ist aber von Session 1 immer noch blockiert. Session 2 wartet auf eine Freigabe. Erfolgt dies nicht in angemessener Frist, ist ein Timeout-Fehler die Folge (*locking read*).

Jetzt wird in Session 1 mit COMMIT der Datensatz freigegeben. Wird jetzt in Session 2 versucht, den Artikel zu löschen, gelingt dies ([Zeile 22](#)); wir haben nur noch zehn Zeilen in der Tabelle `artikel`.

Bitte beachten Sie, dass die Anzahl der Zeilen in Session 1 immer noch 11 ist. Warum? Weil der Löschvorgang in Session 2 noch nicht durch ein COMMIT bestätigt wurde. Erfolgt dies, ist in beiden Sessions der gleiche Datenbestand.



Hinweis: Wegen dieser doch sehr restriktiven Sperrvorgänge ist diese Isolationsebene sehr anfällig für Performanceverluste und Timeout-Fehler und sollte nur sehr überlegt eingesetzt werden.

In MySQL, MariaDB und anderen Datensystemen wie Oracle und PostgreSQL wird diese Isolationsebene auch nicht zur Vermeidung von *phantom read* benötigt, da das Multi-Version Concurrency Control (MVCC) eingesetzt wird. Dabei wird – vereinfacht gesagt – für jede Transaktion ein *snapshot* der Tabellen erstellt. Jede Transaktion arbeitet dann auf ihrem *snapshot*. MVCC steht nur bei READ COMMITTED und REPEATABLE READ und nicht bei READ UNCOMMITTED und SERIALIZABLE zur Verfügung.

Tabelle 19.1 SQL-Isolationsebenen

Isolationsebene	dirty read	nonrepeatable read	phantom read	locking read
READ UNCOMMITTED	möglich	möglich	möglich	unmöglich
READ COMMITTED	unmöglich	möglich	möglich	unmöglich
READ COMMITTED (MVCC)	unmöglich	möglich	unmöglich	unmöglich
REPEATABLE READ	unmöglich	unmöglich	möglich	unmöglich
REPEATABLE READ (MVCC)	unmöglich	unmöglich	unmöglich	unmöglich
SERIALIZABLE	unmöglich	unmöglich	unmöglich	möglich

■ 19.4 Fallbeispiel in C#

Das obige Beispiel mit dem neuen Artikel Säge wollen wir aus einer C#-Anwendung heraus durchführen. Ich verzichte hier auf eine Beschreibung der MySQL-API und vertraue darauf, dass die meisten Befehle selbsterklärend sind. Zunächst brauche ich ein Feld, in dem ich die derzeit offene Verbindung zum Server ablege ([Zeile 10](#)). In `Main()` werden dann nur drei Methoden aufgerufen, die die Details erledigen.

Listing 19.1 Transaktion, Teil 1

```

6  using MySql.Data.MySqlClient;
7
8  namespace TransaktionBsp {
9      class Program {
10         static MySqlConnection mysqlConnection = new MySqlConnection();//
11
12         static void Main(string[] args) {
13             Start();
14             InsertArtikel("Sge", 14.85, "EUR");
15             Stopp();
16         }
}

```

Die Methode `Start()` öffnet eine Verbindung zum Server und stellt diese Verbindung anderen Aktionen über das Feld `mysqlConnection` zur Verfügung. Die Variable `strConnectionString` enthält die Verbindungsparameter zum Server. Da ich kein Passwort gesetzt habe, bleibt dieser Parameter leer.

Listing 19.2 Transaktion, Teil 2

```

18     static void Start() {
19         String strConnectionString = "";
20         strConnectionString += "server=127.0.0.1;uid=root";
21         strConnectionString += ";password=;database=oShop;charSet=utf8";
22         try {
23             mysqlConnection.ConnectionString = strConnectionString;
24             mysqlConnection.Open();
25         }
26         catch (Exception ex) {
27             Console.WriteLine(ex.Message);
28         }
29     }

```

Das Analogon ist die Methode `Stopp()`. Sie ist nur eine Wrapper-Methode für die Methode `Close()`:

Listing 19.3 Transaktion, Teil 3

```

31     static void Stopp() {
32         mysqlConnection.Close();
33     }

```

Und jetzt die Transaktion: Im `try`-Block werden die SQL-Anweisungen aufgebaut und an den Server geschickt. Geht dabei etwas schief (ein SQL-Befehl ist falsch geschrieben, ein Constraint meldet sich etc.), wird vom Server eine `Exception` ausgelöst, die im `catch`-Block verarbeitet wird. Dort wird ein `ROLLBACK` zum Server gesendet ([Zeile 72](#)), damit keine inkonsistenten oder halbfertigen Umbauten zurückbleiben. Wird vom Server kein Fehler gemeldet, werden die Ergebnisse der Anweisungen mit `COMMIT` ([Zeile 65](#)) bestätigt.

Listing 19.4 Transaktion, Teil 4

```

35     static void InsertArtikel(string strArtikel, double dPreis, string
36         strWaehrung) {
37         MySqlCommand command = new MySqlCommand();
38         command.Connection = mysqlConnection;
39         long artikel_id = 0;
40         try {
41             // Befehl 1
42             command.CommandText = "START TRANSACTION";
43             command.Prepare();
44             command.ExecuteNonQuery();
45             // Befehl 2
46             command.CommandText = "INSERT INTO artikel (bezeichnung, einzelpreis,
47                 waehrung) VALUES (@bezeichnung, @einzelpreis, @waehrung)";
48             command.Prepare();
49             command.Parameters.AddWithValue("@bezeichnung", strArtikel);
50             command.Parameters.AddWithValue("@einzelpreis", dPreis);
51             command.Parameters.AddWithValue("@waehrung", strWaehrung);
52             command.ExecuteNonQuery();
53             artikel_id = command.LastInsertedId;

```

```

52     // Befehl 3
53     command.CommandText = "INSERT INTO artikel_nm_warengruppe VALUES (
54         @warengruppe_id, @artikel_id)";
55     command.Prepare();
56     command.Parameters.Clear();
57     command.Parameters.AddWithValue("@warengruppe_id", 3);
58     command.Parameters.AddWithValue("@artikel_id", artikel_id);
59     command.ExecuteNonQuery();
60     // Befehl 4
61     command.Parameters["@warengruppe_id"].Value = 4;
62     command.Parameters["@artikel_id"].Value = artikel_id;
63     command.ExecuteNonQuery();
64     // Befehl 5
65     command.Parameters.Clear();
66     command.CommandText = "COMMIT"; // Alles in Ordnung
67     command.Prepare();
68     command.ExecuteNonQuery();
69 }
70 catch (Exception ex) {
71     Console.WriteLine(ex.Message);
72     command.Parameters.Clear();
73     command.CommandText = "ROLLBACK"; // Alles wieder auf Anfang
74     command.Prepare();
75     command.ExecuteNonQuery();
76 }
```

■ 19.5 Deadlock

Ein besonders Problem tritt immer dann auf, wenn sich Sperren/Transaktionen gegenseitig blockieren.

SESSION 1

```

1 START TRANSACTION;
2
3 UPDATE artikel
4 SET einzelpreis = 2.32
5 WHERE artikel_id = 3001;
6
7 UPDATE artikel
8 SET einzelpreis = 56.26
9 WHERE artikel_id = 3005;
```

SESSION 2

```

1 START TRANSACTION;
2
3 UPDATE artikel
4 SET einzelpreis = 56.26
5 WHERE artikel_id = 3005;
6
7
8
9
10
11 UPDATE artikel
12 SET einzelpreis = 2.32
13 WHERE artikel_id = 3001;
```

Ab **Zeile 5** werden zwei Zeilen gesperrt. Session 1 sperrt den Artikel 3001 und Session 2 Artikel 3005. Session 1 versucht, in **Zeile 9** anschließend auch auf Artikel 3005 zuzugreifen. Da dieser noch gesperrt ist, wartet die Session auf Freigabe durch Session 2. Wartet man lange genug, erhält man folgende Fehlermeldung:

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

Wir wollen diese Fehlermeldung aber gar nicht und führen in Session 2 in [Zeile 13](#) einen UPDATE auf Artikel 3001 aus. Dieser ist aber von Session 1 gesperrt. Somit warten beide auf die Freigabe durch einen anderen. Und das täten sie bis in alle Ewigkeit, wenn nicht seitens des Servers versucht wird, Deadlocks zu identifizieren und unterbrechen. MySQL liefert jedenfalls unmittelbar nach [Zeile 13](#) die Fehlermeldung:

```
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction
```



Definition 61: Deadlock

Ist die Freigabe einer Sperrung A von einer Sperrung B abhängig, die von der Freigabe der Sperrung A abhängt, spricht man von einem Deadlock.



Endlich wieder zu Hause: schöne prozedurale, imperative Programmierung.

- Grundkurs
 - Anlegen mit CREATE PROCEDURE
 - Das DELIMITER-Problem
 - Löschen mit DROP PROCEDURE
 - Variablen: lokale Variable, Sitzungsvariable, globale Variable
 - Parameter: IN, OUT, INOUT
 - Verzweigung mit IF
 - Fallunterscheidung mit CASE
 - WHILE-Schleife
 - Aufruf mit CALL
- Vertiefendes
 - LOOP-Schleife
 - REPEAT-Schleife
 - Transaktion innerhalb einer Prozedur
 - CURSOR



Die Quelltexte des Kapitels stehen in der Datei `listing16.sql` (siehe [Listing 29.18 auf Seite 525](#)). Sie brauchen auch die beiden Dateien `blz_20120305.csv` und `lstBestellungen.xml`.



Hinweis: Anders als in den vorherigen Kapiteln wird hier auf eine Gegenüberstellung von SQL:2016 und MySQL oder MariaDB verzichtet. Prozeduren werden so unterschiedlich von den DBS-Anbietern spezifiziert, dass ich hier sinnvollerweise nur eine Sprachimplementierung vorstellen werde: die von MySQL. Die Grundelemente Aufruf, Parameter, Verzweigung, Schleife und Cursor kommen aber überall vor, sodass sich das erworbene Verständnis übertragen lässt.

Um einen syntaktischen Vergleich zu haben, habe ich die entsprechenden Beispiele auch in PostgreSQL erstellt und Ihnen zum Download zur Verfügung gestellt.

■ 20.1 Einstieg und Variablen

Manche Aufgaben lassen sich nicht in einem SQL-Befehl unterbringen. Das kann inhaltliche Gründe haben, wie wir sie beispielsweise bei den Transaktionen kennengelernt haben (siehe [Kapitel 19 auf Seite 319](#)), aber auch technische.

In vielen Fällen wird dazu auf der Client-Seite eine passende PHP-, Java- oder C#-Methode erstellt, die die Aufgabe als Ganzes löst. Man kann aber auch wollen, dass diese Aufgabe auf dem SQL Server gelöst werden soll. Ein Grund könnte ein hohes Datenvolumen und der damit verbundene Datenverkehr sein.

Wir bleiben bei dem Beispiel, einen Artikel mit seiner Warengruppen einzufügen. Dazu brauchen wir im ersten Schritt nur zwei Informationen: den Artikelnamen und die Warengruppe.



MySQL/MariaDB

```
CREATE PROCEDURE name ([parameter][, parameter]*)
    anweisungsblock
;
parameter:
    [IN | OUT | INOUT] name typ
typ:
    jeder gültige MySQL Datentyp (siehe Abschnitt 26.1 auf Seite 397)
```

Tabelle 20.1 Parameterart

Art	Präfix	Beschreibung
IN	i	Der Parameter ist eine lokale Variable. Jede Wertänderungen innerhalb der Prozedur bleiben außerhalb der Prozedur unbekannt. Wird kein Parameter angegeben, wird IN angenommen.
OUT	o	Der Parameter wird innerhalb der Prozedur mit einem neuen Wert versehen. Dieser neue Wert kann anschließend außerhalb der Prozedur weiter verarbeitet werden.
INOUT	io	Der Wert des Parameters wird innerhalb der Prozedur verwendet und abgeändert wieder in diesen Parameter abgelegt.

Ein Parameter wird mit einer Art (siehe [Tabelle 20.1](#)), einem Typ (siehe [Abschnitt 26.1 auf Seite 397](#)) und einem Namen festgelegt. Der Name sollte selbsterklärend sein. Um die Übersicht nicht zu verlieren, verwende ich immer einen Präfix.



Hinweis: Parameter sollten vor ihrer Verwendung immer formal und fachlich plausibilisiert werden.

Formal bedeutet hier, dass er einen passenden Datentyp hat; ggf. müssen diese konvertiert werden. Fachlich meint, dass die Übergabe innerhalb des Wertebereichs liegt.

Der Anweisungsblock beginnt immer mit dem Schlüsselwort BEGIN und schließt mit END ab. Jetzt haben wir schon genügend Informationen gesammelt, um uns an das erste Procedurfragment zu wagen:

```

1 CREATE PROCEDURE insert_artikel
2 (
3   IN iArtikelname VARCHAR(255),
4   IN iWarengruppe VARCHAR(255)
5 )
6 BEGIN
7   INSERT INTO artikel (bezeichnung) VALUES (iArtikelname);

```

Und schon ist Schluss :-(. Wir haben hier ein Problem, das wir zuerst lösen müssen, bevor es weitergehen kann. Die Anweisung in Zeile 7 endet mit einem Semikolon, mit was auch sonst? Aber ein Semikolon bedeutet doch, dass die Anweisung abgeschlossen/fertig ist. Wir erhalten also eine unschöne Fehlermeldung und sind erst einmal ratlos.

Das Problem ist, dass die eigentliche Anweisung, die SQL ausführen soll, CREATE PROCEDURE ist. Innerhalb dieser Anweisung tauchen nun weitere Anweisungen auf, die ebenfalls mit einem Semikolon enden. Um diese Mehrdeutigkeit zu umgehen, kann man das Zeichen zum Abschluss einen Befehls mit DELIMITER festlegen.



Hinweis: Achten Sie bei der Wahl des DELIMITERS darauf, dass dieses Zeichen nicht anderweitig innerhalb der Prozedur verwendet wird.

Würden wir beispielsweise die öffnende Klammer oder ein Komma wählen, würde die Verwirrung sich nur steigern. Beliebte DELIMITER sind ; oder // oder \$\$ oder ^.

Bevor wir die Prozedur weiter programmieren, fällt uns ein, dass wir den durch AUTO_INCREMENT neu erzeugten Primärschlüsselwert erfahren müssen. Hierzu haben Sie schon die Funktion LAST_INSERT_ID() kennengelernt. Diesen Wert wollen wir in einer lokalen Variablen ablegen, um ihn später weiterzuverwenden.

Lokale Variablen werden mit dem Schlüsselwort DECLARE bekannt gemacht. Wie bei Übergabeparametern können diese auch die bekannten Datentypen haben. Mit dem Zusatz DEFAULT kann man ihnen einen Startwert mitgeben. Wie bei den Übergabeparametern verwende ich auch hier immer einen Präfix: v.



MySQL/MariaDB
`DECLARE variablenname datentyp[DEFAULT vorbelegung];`

Wird ein DEFAULT angegeben, ist dies der Startwert der Variable. Oben ist erwähnt, dass es sich um eine lokale Variable handelt, aber was ist das eigentlich?



Definition 62: Lokale Variable

Ist ein Variablenname nur innerhalb des Anweisungsblocks bekannt, in dem er deklariert wurde, nennt man diese eine *lokale Variable*.

Dadurch, dass eine Variable lokal ist, kommt man sich nicht mit den lokalen Variablen anderer Prozeduren ins Gehege. Die meisten Variablen in Prozeduren sind entweder Übergabeparameter oder lokale Variablen. Aber sorry, was sind noch mal Übergabeparameter?



Definition 63: Übergabeparameter

Übergabeparameter vom Typ IN sind lokale Variablen, deren Wert von außerhalb der Prozedur beim Aufruf festgelegt werden. Sind sie vom Typ OUT, sind es ebenfalls lokale Variablen, aber der Wert der Variablen bleibt nach dem Verlassen der Prozedur erhalten. INOUT kann beides.

Der Wert einer Variable wird entweder durch SET oder ein SELECT . . . INTO festgelegt.



MySQL/MariaDB

```
SET variablenname = wert;
SELECT selectausdruck INTO variablenname;
```

Wenn es lokale Variablen gibt, gibt es dann auch was anderes? Klar, die globalen Variablen und die Sitzungsvariablen.



Definition 64: Sitzungsvariablen

Ist eine Variable allen Prozeduren, Funktionen und Eingaben einer Sitzung bekannt, so wird diese *Sitzungsvariable* oder *session variable* genannt. Sie wird mit SESSION oder @@session. gekennzeichnet.

Der Wert kann innerhalb einer Sitzung¹ verändert werden. Diese Veränderung ist nur innerhalb dieser Sitzung bekannt. Andere Sitzungen kennen die Variable nicht oder haben immer noch den alten Wert. Viele Sitzungsvariablen sind schon vom Server eingerichtet worden. Eine davon haben wir bei den Transaktionen (siehe [Kapitel 19 auf Seite 319](#)) kennengelernt: AUTOCOMMIT.



Definition 65: Globale Variable

Ist eine Variable allen Prozeduren, Funktionen und Eingaben aller Sitzungen bekannt, so wird diese *globale Variable* genannt. Sie wird mit GLOBAL oder @@global. gekennzeichnet.

Eine Änderung ist allen anderen Sitzungen bekannt. Auch hier stellt der Server schon viele solcher Variablen zur Verfügung. Als Beispiel sei hier CONNECT_TIMEOUT genannt.



Hinweis: Variablen, die durch Wertzuweisung deklariert werden, haben den Präfix @.

Zurück zu unserem Beispiel: Zu Beginn unserer Programmierung wird als Erstes der neue Begrenzer mit DELIMITER definiert. Ich bevorzuge das //. Bitte vergessen Sie nicht: Wenn

¹ engl: session

irgendetwas schiefläuft und Sie mit dem Client weiter arbeiten wollen, setzen Sie den Begrenzer wieder auf das Semikolon.

```

1  DELIMITER //
2  CREATE PROCEDURE insert_artikel
3  (
4    IN iArtikelname VARCHAR(255),
5    IN iWarengruppe VARCHAR(255)
6  )
7
8  BEGIN
9
10  DECLARE v_artikel_id      INT DEFAULT 0;
11  DECLARE v_warengruppe_id INT DEFAULT 0;
12
13
14  INSERT INTO artikel
15    (bezeichnung)
16    VALUES (iArtikelname);
17
18  SET v_artikel_id = LAST_INSERT_ID();           -- Variable setzen
19
20  SELECT warengruppe_id
21    FROM warengruppe
22   WHERE bezeichnung = iWarengruppe
23   INTO v_warengruppe_id;                      -- Oder so
24
25  INSERT INTO artikel_nm_warengruppe          -- Eintrag Hilfstabelle
26    (warengruppe_id, artikel_id)
27    VALUES
28    (v_warengruppe_id, v_artikel_id);
29
30 END//                                         -- Anderer DELIMITER
31 DELIMITER ;

```

Mit der Anweisung `SET` wird in [Zeile 18](#) der Inhalt einer Variable mit einem neuen Wert versehen. Dabei können rechts vom Gleichheitszeichen Konstanten, Ausdrücke oder Funktionsaufrufe stehen, sofern diese skalar sind und einen Wert zurückliefern, der zum Datentyp der Variablen passt.

Das Ergebnis einer Auswahl mit `SELECT` kann ebenfalls in einer Variablen abgelegt werden. Nach dem `SELECT` wird durch das Schlüsselwort `INTO` in [Zeile 23](#) die Zielvariable festgelegt.

In [Zeile 30](#) kommt unser neuer Begrenzer zum Einsatz. Die Semikolons des Anweisungsblocks sind von SQL-Interpreter bis jetzt ignoriert worden. Das `//` schließt den `CREATE` ab, und die Prozedur wird übernommen. Jetzt ist es wichtig, den Begrenzer wieder auf das Semikolon zu setzen, damit er für nachfolgende Befehle wieder gilt.

Sicherlich haben Sie auch bei `LAST_INSERT_ID()` die Stirn gerunzelt, oder? Immerhin könnten mehrere Sessions gleichzeitig in die Tabelle `artikel` neue Zeilen einfügen. Stellen Sie sich vor, ein anderer Anwender würde in der Zeitspanne zwischen dem Einfügen und dem Aufruf von `LAST_INSERT_ID()` ebenfalls einen Datensatz einfügen. Stellt sich doch die Frage, welchen Wert liefert jetzt diese Funktion?

Ein Blick ins Handbuch verschafft Klarheit. Jede Session merkt sich nur die ID, die in dieser erstellt wurde. Werden in anderen Sessions andere IDs erstellt, sind die in meiner nicht durch `LAST_INSERT_ID()` ermittelbar. Oder positiv ausgedrückt: Es ist garantiert, dass diese Funktion den zuletzt erzeugten `AUTO_INCREMENT`-Wert meiner Session liefert.

Rufen wir die Prozedur mal auf:

```

1 mysql> CALL insert_artikel ('Schlauch', 'Gartenbedarf');
2
3 mysql> SELECT artikel_id, bezeichnung FROM artikel ORDER BY artikel_id;
4 +-----+-----+
5 | artikel_id | bezeichnung |
6 +-----+-----+
7 [...]
8 | 9017 | Schlauch |
9 +-----+-----+
10
11 mysql> SELECT warengruppe_id, bezeichnung FROM warengruppe;
12 +-----+-----+
13 | warengruppe_id | bezeichnung |
14 +-----+-----+
15 [...]
16 | 3 | Gartenbedarf |
17 [...]
18 +-----+-----+
19
20 mysql> SELECT * FROM artikel_nm_warengruppe;
21 +-----+-----+
22 | warengruppe_id | artikel_id |
23 +-----+-----+
24 [...]
25 | 3 | 9017 |
26 +-----+-----+

```

Eine Prozedur kann mit `DROP PROCEDURE` gelöscht werden.



MySQL/MariaDB

```
DROP PROCEDURE [IF EXISTS] name
;
```

Mit `SHOW PROCEDURE STATUS` kann eine Liste der verfügbaren Prozeduren eingesehen werden:

```

1 mysql> SHOW PROCEDURE STATUS\G
2 **** 1. row ****
3           Db: oshop
4           Name: insert_artikel
5           Type: PROCEDURE
6           Definer: root@localhost
7           Modified: 2012-06-15 18:02:48
8           Created: 2012-06-15 18:02:48
9           Security_type: DEFINER
10          Comment:
11 character_set_client: utf8
12 collation_connection: utf8_general_ci
13 Database Collation: utf8_unicode_ci

```

■ 20.2 Verzweigung

20.2.1 Einfache Verzweigung mit IF

Da sind wir mal mutig:

```
1 mysql> CALL insert_artikel ('Fleischwurst', 'Lebensmittel');
2 ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint
      fails ('oshop'.'artikel_nm_warengruppe', CONSTRAINT 'artikel_nm_warengruppe_ibfk_1' FOREIGN KEY ('warengruppe_id') REFERENCES
      'warengruppe' ('warengruppe_id'))
```

Das ist hässlich. Der Artikel **Fleischwurst** ist zwar angelegt worden, aber der Eintrag in der Tabelle **artikel_nm_warengruppe** ist fehlgeschlagen.



Aufgabe 20.1: Finden Sie anhand der Fehlermeldung die Fehlerursache heraus.

Um diesen Fehler in Zukunft zu vermeiden, wollen wir unsere Prozedur um eine Plausibilitätsüberprüfung der Übergabeparameter erweitern.



Definition 66: Plausibilitätsüberprüfung

Eine *Plausibilitätsüberprüfung* ist der Test eines Übergabeparameters oder einer Anwendereingabe auf fachliche Richtigkeit.

Wie bei jeder guten Analyse sollte man die fachlichen Regeln und die daraus abgeleiteten Handlungen vor der Programmierung notieren:

- Ist die Warenguppenbezeichnung nicht vorhanden, soll eine neue Warengruppe angelegt werden.
- Ist die Artikelbezeichnung nicht vorhanden, soll ein neuer Artikel angelegt werden.
- Ist die Artikelbezeichnung schon vorhanden, soll überprüft werden, ob es schon eine Zuordnung des Artikels zu dieser Warengruppe gibt.
 - Es gibt keine Zuordnung, dann soll eine neue Zuordnung in die Hilfstabelle eingetragen werden.
 - Es gibt eine Zuordnung, dann brauchen wir nichts tun.

Wir brauchen ein Sprachkonstrukt, welches uns festzulegen erlaubt, ob ein Anweisungsblock überhaupt ausgeführt wird.



Definition 67: Verzweigung

Wenn ein Anweisungsblock nur abhängig von einer Bedingung ausgeführt wird, wird er *Verzweigung* genannt.

In MySQL und MariaDB wird für eine Verzweigung das **IF** verwendet. Der Aufbau eines **IF** ist wie folgt:



MySQL/MariaDB

```
IF bedingung THEN
    anweisungsblock 1
[ELSE
    anweisungsblock 2]
END IF;
```

■

Der *anweisungsblock 1* wird nur ausgeführt, wenn die Bedingung TRUE oder einen Wert $\neq 0$ liefert. Der *anweisungsblock 2* wird nur ausgeführt, wenn *anweisungsblock 1* nicht ausgeführt wird. Der ELSE-Teil ist optional.

Zurück zu unserem oben beschriebenen Plan. Es ist nicht möglich, eine Prozedur im Anweisungsblock nur zu ändern. Sie muss mit `DROP PROCEDURE` gelöscht und mit `CREATE PROCEDURE` wieder angelegt werden:

```
1  DROP PROCEDURE IF EXISTS insert_artikel;
2
3  DELIMITER //
4  CREATE PROCEDURE insert_artikel
5  (
6      IN iArtikelname VARCHAR(255),
7      IN iWarengruppe VARCHAR(255)
8  )
```

Die Variable `v_artikel_id` wird den neuen oder ggf. schon vorhandenen Primärschlüsselwert des Artikels enthalten. Das Gleiche gilt für `v_warengruppe_id`. In `v_count_zuordnung` wird die Anzahl der schon vorhandenen Einträge in der Hilfstabelle bzgl. des Artikels und der Warengruppe abgelegt. Erwartet werden die Werte 1 oder 0.

```
10 BEGIN
11  DECLARE v_artikel_id      INT DEFAULT -1;
12  DECLARE v_warengruppe_id  INT DEFAULT -1;
13  DECLARE v_count_zuordnung INT DEFAULT 0;
```

In `iWarengruppe` steht die Warengruppe, die als Parameter beim Aufruf übergeben wird. Diese wird nun innerhalb der `WHERE`-Klausel dazu verwendet, den passenden Primärschlüsselwert zu ermitteln. Das Ergebnis wird in `v_warengruppe_id` abgespeichert. In Zeile 23 wird mit IF überprüft, ob ein Datensatz vorhanden war. War er nicht vorhanden, wird er neu angelegt und der neue Primärschlüsselwert mithilfe von `LAST_INSERT_ID()` ermittelt. In beiden Fällen steht der Primärschlüsselwert der Warengruppe in der lokalen Variablen `v_warengruppe_id`: entweder der schon vorhandene oder der neue.

```
16  -- Teste, ob Warengruppe vorhanden ist
17  SELECT warengruppe_id
18  FROM warengruppe
19  WHERE bezeichnung = iWarengruppe
20  INTO v_warengruppe_id;
21
22  -- Wenn nein, füge eine ein
23  IF v_warengruppe_id < 0 THEN          -- Einfache Verzweigung
24      INSERT INTO warengruppe (bezeichnung) VALUES (iWarengruppe);
25      SET v_warengruppe_id = LAST_INSERT_ID();
26 END IF;
```

Analog wird bei dem Artikel vorgegangen. Zuerst wird versucht, den ggf. schon vorhandenen Primärschlüsselwert zu ermitteln. Dieser wird auf < 0 getestet und falls die Bedingung TRUE liefert, wird ein neuer Artikel mit der übergebenen Bezeichnung angelegt.

```

28 -- Teste, ob artikel vorhanden ist
29 SELECT artikel_id
30   FROM artikel
31 WHERE bezeichnung = iArtikelname
32   INTO v_artikel_id;
33
34 -- Wenn nein, teste auf fuege ein
35 IF v_artikel_id < 0 THEN
36   INSERT INTO artikel (bezeichnung) VALUES (iArtikelname);
37   SET v_artikel_id = LAST_INSERT_ID();
38 END IF;
```

Jetzt wird in der Hilfstabelle anhand der ermittelten Primärschlüsselwerte ein Eintrag gesucht, genauer: die Anzahl der passenden Einträge. Liefert die Bedingungen `v_count_zuordnung <= 0` den Wert TRUE, muss eine Zeile in die Hilfstabelle eingefügt werden. Der Hinweis in [Zeile 53](#) ist nur für eine Warnung da.

```

40 -- Test, ob Zuordnung schon vorhanden
41 SELECT COUNT(*)
42   FROM artikel_nm_warengruppe
43 WHERE artikel_id = v_artikel_id AND warengruppe_id = v_warengruppe_id
44   INTO v_count_zuordnung;
45
46 -- Wenn nein, fuege eine hinzu
47 IF v_count_zuordnung <= 0 THEN
48   INSERT INTO artikel_nm_warengruppe
49     (warengruppe_id, artikel_id)
50   VALUES
51     (v_warengruppe_id, v_artikel_id);
52 ELSE
53   SELECT 'Zuordnung schon vorhanden';      -- Hinweis
54 END IF;
55 END//
```

DELIMITER ;

Ein neuer Versuch:

```

40 mysql> CALL insert_artikel ('Fleischwurst', 'Lebensmittel');
41 mysql> CALL insert_artikel ('Fleischwurst', 'Lebensmittel');
42 +-----+
43 | Zuordnung schon vorhanden |
44 +-----+
45 | Zuordnung schon vorhanden |
46 +-----+
```



Aufgabe 20.2: Testen Sie alle Fälle mit eigenen Beispielen durch und überprüfen Sie die Ergebnisse in den Tabellen.

20.2.2 Mehrfache Verzweigung mit CASE

Für die Fallunterscheidung gibt es zwei Varianten: eine, die eine Wertübereinstimmung ermittelt, und eine, die Bedingungen prüft.

Hier die Syntax für die Wertübereinstimmung. Hinter dem CASE steht ein irgendwie gearteter Container für Werte, z.B. ein Variablenname. Hinter dem WHEN wird ein konkreter Wert angegeben. Stimmen die Werte überein, wird der hinter dem THEN stehende Anweisungsblock ausgeführt.



MySQL/MariaDB

```
CASE spaltenname|ausdruck|variablenname
    WHEN wert THEN anweisungsblock
    [WHEN wert THEN anweisungsblock]*
    [ELSE anweisungsblock]
END CASE;
```

Hier die Syntax für die Bedingung. Hinter jedem WHEN steht eine Bedingung. Ist diese TRUE, wird der Anweisungsblock hinter dem THEN ausgeführt. Die Formulierung der Bedingung erfolgt wie bei der Verzweigung mit IF (siehe [Abschnitt 20.2.1 auf Seite 339](#)).



MySQL/MariaDB

```
CASE
    WHEN bedingung THEN anweisungsblock
    [WHEN bedingung THEN anweisungsblock]*
    [ELSE anweisungsblock]
END CASE;
```

Ich möchte durch eine Prozedur abhängig vom Übergabeparameter unterschiedliche Umsatzberichte erstellen lassen. Ist der Übergabeparameter eine 1, soll der totale Umsatz anhand der Tabelle rechnung ermittelt werden. Ist er 2, wird der Umsatz pro Jahr, bei 3 pro Monat berechnet. Das Ergebnis wird in der Tabelle umsatzzahlen abgelegt.

```
1  DELIMITER //
2  CREATE PROCEDURE umsatzreport
3  (
4      IN iZeitraum INT
5  )
6  BEGIN
7      DROP TABLE IF EXISTS umsatzzahlen;
8
9      CASE iZeitraum
10         WHEN 1 THEN      -- Total
11             CREATE TABLE umsatzzahlen
12                 SELECT 'Total', SUM(einzelpreis * menge) 'Umsatz'
13                 FROM rechnung_position INNER JOIN artikel USING (artikel_id)
14                               INNER JOIN rechnung USING (rechnung_id)
15             ;
16         WHEN 2 THEN      -- Pro Jahr
17             CREATE TABLE umsatzzahlen
18                 SELECT YEAR(datum) 'Jahr', SUM(einzelpreis * menge) 'Umsatz'
```

```

20      FROM
21          rechnung_position INNER JOIN artikel USING (artikel_id)
22                      INNER JOIN rechnung USING (rechnung_id)
23      GROUP BY 'Jahr'
24      ORDER BY 'Jahr' DESC
25      ;
26 WHEN 3 THEN    -- Pro Monat
27 CREATE TABLE umsatzzahlen
28     SELECT YEAR(datum) 'Jahr', MONTH(datum) 'Monat', SUM(einzelpreis * menge)
29             'Umsatz'
30     FROM
31         rechnung_position INNER JOIN artikel USING (artikel_id)
32                     INNER JOIN rechnung USING (rechnung_id)
33     GROUP BY 'Jahr', 'Monat'
34     ORDER BY 'Jahr' DESC, 'Monat' DESC
35     ;
36 ELSE           -- Sowas sollte es immer geben
37 CREATE TABLE umsatzzahlen
38     SELECT 'Unbekannte Berichtsart', iZeitraum;
39 END CASE;
40 END//          --
41 DELIMITER ;

```

Schauen wir uns die Ausgaben an:

```

1 mysql> CALL umsatzreport(1); SELECT * FROM umsatzzahlen;
2 +-----+-----+
3 | Total | Umsatz      |
4 +-----+-----+
5 | Total | 2170.210000000000 |
6 +-----+-----+

```

Der Übergabeparameter ist 1. Beim ersten **WHEN** in [Zeile 10](#) wird der Inhalt von **iZeitraum** genau auf diesen Wert überprüft. Die anschließende Anweisung erzeugt den Bericht. Würden mehrere Anweisungen auszuführen sein, muss dieser mit **BEGIN ... END;** umschlossen sein.

```

1 mysql> CALL umsatzreport(2); SELECT * FROM umsatzzahlen;
2 +-----+-----+
3 | Jahr | Umsatz      |
4 +-----+-----+
5 | 2012 | 1615.800000000000 |
6 | 2011 | 554.410000000000 |
7 +-----+-----+

```

Jetzt wird der Fall in [Zeile 17](#) ausgeführt. Dabei werden die Umsatzdaten nach der Jahresangabe gruppiert, summiert und sortiert. Analoges geschieht in [Zeile 26](#) bei 3:

```

1 mysql> CALL umsatzreport(3); SELECT * FROM umsatzzahlen;
2 +-----+-----+-----+
3 | Jahr | Monat | Umsatz      |
4 +-----+-----+-----+
5 | 2012 |     4 | 1328.400000000000 |
6 | 2012 |     3 | 287.400000000000 |
7 | 2011 |     1 | 554.410000000000 |
8 +-----+-----+-----+

```

Wird ein Parameterwert der Prozedur übergeben, für den keine Wertübereinstimmung gefunden wird, wird der **ELSE**-Teil ab [Zeile 35](#) ausgeführt:

```
1 mysql> CALL umsatzreport(0); SELECT * FROM umsatzzahlen;
2
3 +-----+-----+
4 | Unbekannte Berichtsart | iZeitraum |
5 +-----+-----+
6 | Unbekannte Berichtsart |          0 |
7 +-----+-----+
```



Hinweis: Die erste Wertübereinstimmung oder die erste zutreffende Bedingung wird ausgeführt. Anschließend wird die Fallunterscheidung verlassen.

Das bedeutet, dass die nachfolgenden Bedingungen nicht mehr überprüft werden.

```
1 DELIMITER //
2 CREATE PROCEDURE b()
3 BEGIN
4   DECLARE vZeitraum INT ;
5   SET vZeitraum = 2;
6   CASE
7     WHEN vZeitraum = 2 THEN SELECT 'A';
8     WHEN vZeitraum > 0 THEN SELECT 'B';
9     WHEN vZeitraum < 2 THEN SELECT 'C';
10    END CASE;
11  END//
12 DELIMITER ;
13
14 mysql> CALL b();
15 +---+
16 | A |
17 +---+
18 | A |
19 +---+
```



Aufgabe 20.3: Hier hätte man vielleicht auch folgende Ausgabe erwarten können.
Warum?

```
+---+
| A |
+---+
| A |
+---+

+---+
| B |
+---+
| B |
+---+
```

■ 20.3 Schleifen

Anweisungen werden oft mehrfach oder besser gesagt wiederholt ausgeführt. Denken Sie an eine Performancemessung, die bei wachsender Datenmenge eigentlich immer das Gleiche tut.



Definition 68: Schleife

Ein Anweisungsblock, der sich abhängig von einer Bedingung wiederholt, wird *Schleife* genannt.

In der Programmierung kennt man viele Arten von Schleifen: kopf- oder fußgesteuerte, annehmende oder abweisende, offene oder geschlossene Schleifen.



Definition 69: Kopf-/Fußgesteuerte Schleife

Wird bei einer Schleife vor der Ausführung des Anweisungsblocks die Bedingung überprüft, so spricht man von einer *kopfgesteuerten* Schleife. Wird der Anweisungsblock ausgeführt und danach die Bedingung überprüft, so spricht man von einer *fußgesteuerten* Schleife.

Aus der Definition folgt, dass bei einer fußgesteuerten Schleife der Anweisungsblock mindestens einmal ausgeführt wird; bei einer kopfgesteuerten ist das nicht garantiert. In Sprachen wie C entspricht die kopfgesteuerte einer `while`-Schleife und die fußgesteuerte einer `do/while`-Schleife.



Definition 70: Annehmende/Abweisende Schleife

Wird bei einer Schleife der Anweisungsblock ausgeführt, bis eine Bedingung wahr ist, wird dies *abweisende* Schleife genannt. Wird der Anweisungsblock wiederholt, solange eine Bedingung wahr ist, wird sie *annehmende* Schleife genannt.

Abweisende Schleifen sind aus der Mode gekommen. Heute werden fast nur noch annehmende verwendet. Die `repeat/until`-Schleife in Delphi ist eine abweisende Schleife.



Definition 71: Offene/Geschlossene Schleife

Liegt die Anzahl der Wiederholungen im Prinzip fest, so spricht man von einer *geschlossenen* Schleife. Ist die Anzahl der Wiederholungen unbestimmt, von einer *offenen*.

Alle Arten von Zählschleifen wie die `for`-Schleife in C# sind geschlossene Schleifen. Offene Schleifen werden meist als `while`-Schleife spezifiziert.

20.3.1 LOOP-Schleife



MySQL/MariaDB
 [labelname :]LOOP
anweisungsblock
 END LOOP[labelname];

Die LOOP-Schleife ist eine offene Schleife, eigentlich eine Endlosschleife. Da es keine Bedingung gibt, die den Abbruch oder die Fortsetzung der Schleife steuert, läuft sie im Prinzip ohne Ende durch. In der Anwendungsentwicklung haben Endlosschleifen durchaus ihre Berechtigung, aber nicht bei SQL-Prozeduren. Schließlich soll der Server nicht durch eine Endlosschleife endlos viele Ressourcen verbrauchen.

Innerhalb der LOOP-Schleife – aber nicht nur dort – kommen daher folgende Schlüsselwörter zum Einsatz:

- **LEAVE:** Der Anweisungsblock oder die Schleife werden verlassen. Dieses Schlüsselwort kann innerhalb eines BEGIN...END-Anweisungsblocks oder in einem Schleifenkonstrukt wie LOOP, REPEAT oder WHILE verwendet werden. Die Schleife wird durch LEAVE beendet, und die erste Anweisung – falls vorhanden – nach der Schleife wird ausgeführt. Gibt es keine nachfolgende Anweisung oder keinen Anweisungsblock, in den man zurückkehren kann, ist die Prozedur beendet. Handelt es sich um eine verschachtelte Schleife², wird die darüberliegende Schleife fortgesetzt³. Der Anweisungsblock oder die Schleife muss einen Label haben.
- **ITERATE:** Die Schleife beginnt mit einem neuen Durchlauf (Iteration). Dieses Schlüsselwort kann innerhalb eines Schleifenkonstrukt wie LOOP, REPEAT oder WHILE verwendet werden. Egal an welcher Stelle ich mich im *anweisungsblock* der Schleife befinde, die Prozedur springt zur ersten Anweisung der Schleife oder bei kopfgesteuerten Schleifen zur Überprüfung der Schleifenbedingung⁴. Die Schleife muss einen Label haben.

Da mir keine sinnvollen Beispiele für eine Endlosschleife eingefallen sind, hier ein sinnloses. Wobei so sinnlos vielleicht doch nicht, wird hier doch das Verhalten und der Wertebereich von einer ganzzahligen Variablen visualisiert.

```

1  DELIMITER //
2  CREATE PROCEDURE warte_auf_ueberlauf()
3  BEGIN
4    DECLARE vInt TINYINT;
5
6    SET vInt = 0;
7    LOOP                      -- Schleifenstart
8      SELECT vInt;
9      SET vInt = vInt + 1;
10     END LOOP;              -- Schleifenende
11   END;
12  //
13  DELIMITER ;
```

² Das ist einer Schleife innerhalb einer Schleife.

³ Für alle, die in der C-Sprachfamilie zu Hause sind, dürfte sich ein Vergleich mit dem break; anbieten.

⁴ Für alle, die in der C-Sprachfamilie zu Hause sind, dürfte sich ein Vergleich mit dem continue; anbieten.

```

14
15 CALL warte_auf_ueberlauf();

```

Und schon kann man sich einen Kaffee (Tee, Apfelsaft ...) holen gehen. Eine wichtige Beobachtung ist, dass der Zähler bei 127 aufhört, sich weiter zu erhöhen! Es gibt also keinen klassischen Überlauf oder eine Exception wie bei C# oder Java.



Hinweis: Die Ausführung kann auf dem MySQL oder MariaDB Client durch STRG+C unterbrochen werden.

Ich möchte auf diese Art und Weise herausfinden, was der höchste und was der kleinste Wert einer INT-Variablen ist. In [Zeile 9](#) wird die Schleife mit einem Aufkleber⁵ versehen. In [Zeile 13](#) kann daher der LEAVE genau sagen, *was* er verlassen möchte. Analog arbeitet die zweite Schleife. Der Abbruch einer Schleife mit LEAVE wird immer dann ausgelöst, wenn sich die Werte der Variablen nicht mehr erhöhen oder erniedrigen.

```

1  DELIMITER //
2  CREATE PROCEDURE min_max()
3  BEGIN
4      DECLARE vInt TINYINT;
5      DECLARE vMin INT;
6      DECLARE vMax INT;
7
8      SET vInt = 0;
9      maxsuche: LOOP          -- Jetzt mit Label
10     SET vMax = vInt;
11     SET vInt = vInt + 1;
12     IF vMax = vInt THEN
13         LEAVE maxsuche;    -- Verlasse Label
14     END IF;
15     END LOOP maxsuche;
16
17     SET vInt = 0;
18     minsuche: LOOP
19     SET vMin = vInt;
20     SET vInt = vInt - 1;
21     IF vMin = vInt THEN
22         LEAVE minsuche;
23     END IF;
24     END LOOP minsuche;
25
26     SELECT 'MIN = ', vMin, 'MAX = ', vMax;
27 END;
28 //
29
30 DELIMITER ;

```



Aufgabe 20.4: Führen Sie `CALL min_max();` aus. Warum dürfen `vMin` und `vMax` nicht den gleichen Datentyp verwenden wie `vInt`?

⁵ engl.: label

20.3.2 WHILE-Schleife



MySQL/MariaDB

```
[labelname:]WHILE schleifenbedingung DO
    anweisungsblock
END WHILE[ labelname];
```

Die WHILE-Schleife ist eine offene, kopfgesteuerte und annehmende Schleife. Und jetzt kommt's:

- Jede fußgesteuerte Schleife kann durch eine kopfgesteuerte simuliert werden.
- Jede abweisende Schleife kann durch eine annehmende simuliert werden.
- Jede geschlossene Schleife kann durch eine offene simuliert werden.

Mit anderen Worten, ich kann jede Art von Schleife bauen, wenn ich eine kopfgesteuerte, annehmende und offene Schleife in einer Programmiersprache zur Verfügung stelle. Und genau das ist eine WHILE-Schleife. In den meisten Fällen werden daher bei Prozeduren WHILE-Schleifen verwendet. Zuerst ein kleines numerisches Beispiel: *das Sieb des Eratosthenes*⁶.

```

1  DELIMITER //
2  CREATE PROCEDURE sieb
3  (
4      iGrenze BIGINT UNSIGNED
5  )
6  BEGIN
7      DECLARE vZahl BIGINT UNSIGNED DEFAULT 2;          -- Start mit 2
8      DECLARE vPosition BIGINT UNSIGNED DEFAULT 2;
9
10     -- Aufbau der Tabelle
11     DROP TABLE IF EXISTS zahlenstrahl;
12     CREATE TABLE zahlenstrahl
13     (
14         position BIGINT UNSIGNED,
15         zahl BIGINT UNSIGNED,
16         PRIMARY KEY(position)
17     );
18
19     WHILE vZahl <= iGrenze DO
20         INSERT INTO zahlenstrahl VALUES (vZahl, vZahl);
21         SET vZahl = vZahl + 1;
22     END WHILE;
23
24     -- Sieb
25     WHILE vPosition < CEILING(SQRT(iGrenze)) DO
26         SELECT zahl FROM zahlenstrahl WHERE position = vPosition INTO vZahl;
27         IF vZahl > 0 THEN
28             UPDATE zahlenstrahl SET zahl = 0
29                 WHERE (position > vPosition) AND (zahl % vZahl = 0);
30         END IF;
31         SET vPosition = vPosition + 1;
32     END WHILE;
```

⁶ griechischer Gelehrter: * zwischen 276 und 273 v. Chr. in Kyrene; † um 194 v. Chr. in Alexandria

```

33
34 -- Bereinigen der Tabelle
35 DELETE FROM zahlenstrahl WHERE zahl = 0;
36 END//'
37 DELIMITER ;

```

Jetzt kann das Sieb aufgerufen werden. Wenn Sie die Idee des Algorithmus nachlesen wollen, sei Ihnen [MOP06] empfohlen.

```

1 mysql> CALL sieb(20);
2
3 mysql> SELECT zahl FROM zahlenstrahl ORDER BY zahl;
4 +-----+
5 | zahl |
6 +-----+
7 |   2 |
8 |   3 |
9 |   5 |
10 |   7 |
11 |  11 |
12 |  13 |
13 |  17 |
14 |  19 |
15 +-----+

```

Ein besseres Beispiel: Ein häufiges Problem ist, dass eine umfangreiche Aktion eine Tabelle für andere Aktionen blockiert. Wenn wir beispielsweise die Bankleitzahlen importieren, ist die Tabelle bank für alle anderen Schreib- und vielleicht sogar Leseoperationen nicht verfügbar. Ist diese Nichtverfügbarkeit eine Frage von einer halben bis ganzen Sekunde, könnte man ein solches Locking noch hinnehmen. Vergehen mehrere Sekunden, riskiert man einen Timeout-Fehler.

Eine Lösungsstrategie ist, die umfangreiche Aktion in Teilaktionen zu zerlegen und so anderen Aktionen Zeit für Auswertungen zu lassen. Bauen wir uns eine passende Beispielumgebung:

```

1 DROP DATABASE IF EXISTS tmpSchleife;
2 CREATE DATABASE tmpSchleife CHARACTER SET utf8;
3
4 USE tmpSchleife;
5
6 CREATE TABLE bank_quelle
7 (
8   blz      CHAR(8),  merkmal      CHAR(1),      bezeichnung  VARCHAR(255),
9   plz      CHAR(5),   ort        VARCHAR(255),  kurz        VARCHAR(255),
10  pan      CHAR(5),   bic        VARCHAR(255),  sm          CHAR(2),
11  ds       CHAR(6),   kz         CHAR(1),      blzl        CHAR(1),
12  blzn     CHAR(8)
13 )
14 ;
15
16 LOAD DATA LOCAL INFILE 'blz_20120305.csv'
17 INTO TABLE bank_quelle
18 FIELDS TERMINATED BY ','
19 LINES TERMINATED BY '\n'
20 ;

```

Wir haben jetzt eine Tabelle, die ca. 19600 Zeilen umfasst, und diese will ich in eine andere Tabelle kopieren. Dabei sollen die Adressdaten in ein eigenes Feld zusammengefasst werden.

```

1 CREATE TABLE bank_ziel
2 (
3   blz          CHAR(8),
4   bezeichnung  VARCHAR(255),
5   adresse      TEXT
6 )
7 ;

```

Der erste Lösungsansatz ist ein normaler, uns bekannter **INSERT**:

```

1 INSERT INTO bank_ziel (blz, bezeichnung, adresse)
2   SELECT blz, bezeichnung, CONCAT(plz, ' ', ort) FROM bank_quelle;

```

Wie oben schon erwähnt, ist der Nachteil dieser Lösung, dass die Tabelle **bank_ziel** für die Dauer der Ausführung keiner anderen Aktion zur Verfügung steht. Hier kann eine Prozedur Abhilfe schaffen:

```

1 DELIMITER //
2 CREATE PROCEDURE kopiere_blz
3 (
4   iAnzahl INT UNSIGNED           -- Anzahl der Zeilen pro Kopievorgang
5 )
6 BEGIN
7   DECLARE vOffset INT UNSIGNED;    -- Aktueller Startpunkt
8   DECLARE vCount  INT UNSIGNED;    -- Anzahl der zu kopierenden Zeilen
9
10  SELECT COUNT(*) FROM bank_quelle INTO vCount;
11
12  SET vOffset = 0;
13  WHILE vOffset < vCount DO        -- Start der Schleife
14    SELECT 'vOffset', vOffset;     -- Nur zum Gucken
15
16    INSERT INTO bank_ziel (blz, bezeichnung, adresse)
17      SELECT blz, bezeichnung, CONCAT(plz, ' ', ort)
18        FROM bank_quelle
19        LIMIT vOffset, iAnzahl;
20    SET vOffset = vOffset + iAnzahl; -- Neuer Startwert
21  END WHILE;                      -- Ende der Schleife
22 END///
23 DELIMITER ;
24
25 CALL kopiere_blz(1000);

```

In [Zeile 13](#) beginnt die Schleife. Hinter dem Schlüsselwort **WHILE** wird die Schleifenbedingung formuliert. Das Schlüsselwort **DO** startet den Anweisungsblock der Schleife. In [Zeile 21](#) endet der Anweisungsblock der Schleife, und die Prozedur springt wieder zur [Zeile 13](#). Erst, wenn die Schleifenbedingung nicht TRUE ergibt, wird mit der Zeile nach [Zeile 21](#) weitergemacht.

Der Übergabeparameter **iAnzahl** der Prozedur legt fest, wie viele Zeilen pro Schleifendurchlauf von der Tabelle **bank_quelle** zur Tabelle **bank_ziel** kopiert werden. Die Variable **vOffset** enthält die Information darüber, mit welcher Zeile der Kopievorgang beginnen soll. In **vCount** wird die Anzahl der Zeilen von **bank_quelle** abgelegt, damit die Schleife weiß, wann sie fertig ist.

Zuerst wird `vOffset` auf 0 gesetzt, damit der Kopiervorgang mit der ersten Zeile beginnt. In **Zeile 20** wird `vOffset` um die Anzahl der kopierten Zeilen erhöht. Die Schleifenbedingung überprüft, ob dieser Wert kleiner als die Gesamtanzahl der zu kopierenden Zeilen ist. Falls ja, muss die Schleife ausgeführt werden.

Die beiden Variablen `vOffset` und `iAnzahl` werden in einer `LIMIT`-Klausel dazu verwendet, genau die Zeilen aus der Quelltabelle auszuschneiden, die gerade kopiert werden sollen.



Aufgabe 20.5: Überlegen Sie sich eine geeignete Überprüfung des Kopiervorgangs und bauen Sie die Prozedur zu einer Transaktion um. Es soll ein ROLLBACK erfolgen, wenn der Kopiervorgang nicht erfolgreich war.

20.3.3 REPEAT-Schleife



MySQL/MariaDB

```
[labelname:]REPEAT
  anweisungsblock
UNTIL bedingung END REPEAT[ labelname];
```

Die REPEAT-Schleife ist eine abweisende, fußgesteuerte Schleife. Wie oben erwähnt, braucht man diese Schleife nicht unbedingt, da man sie durch eine WHILE-Schleife simulieren kann. *Hier ein numerisches Beispiel:* Es wird der größte gemeinsame Teiler nach Euklid⁷ berechnet⁸.

```

1  DELIMITER //
2  CREATE PROCEDURE ggt
3  (
4    iZahl1 INT UNSIGNED,
5    iZahl2 INT UNSIGNED
6  )
7  BEGIN
8    SET @m = iZahl1;      -- Deklaration durch
9    SET @n = iZahl2;      -- eine Zuweisung
10
11   REPEAT                  -- Schleifenstart
12     SET @t = @m % @n;
13     SET @m = @n;
14     SET @n = @t;
15   UNTIL @n = 0 END REPEAT; -- Schleifenende
16
17   SELECT CONCAT('GGT(',iZahl1,', ',iZahl2,') = ') 'GgT()', @m;
18 END//
19 DELIMITER ;
```

⁷ Euklid von Alexandria; ca. 360 v. Chr. bis ca. 280 v. Chr.; griechischer Mathematiker.

⁸ Bitte, das ist keine numerisch stabile und plausibilisierte Funktion, sondern nur ein Beispiel!

Der Anweisungsblock zwischen [Zeile 11](#) und [15](#) wird mindestens einmal durchgeführt. Dann erfolgt die Überprüfung der Schleifenbedingung. Ergibt diese TRUE, wird die Schleife beendet, ansonsten startet ein neuer Schleifendurchlauf.

Ich persönlich finde abweisende Schleifen in der Analyse und bei der Fehlersuche verwirrend. Wenn es Ihnen genauso geht, vermeiden Sie diese.

Als Beispiel wollen wir den größten gemeinsamen Teiler von 20 und 15 wissen.

```

1 CALL ggt(20, 15);
2 +-----+-----+
3 | GGT()      | @m   |
4 +-----+-----+
5 | GGT(20, 15) = | 5  |
6 +-----+-----+

```

■ 20.4 Transaktion innerhalb einer Prozedur

Erinnern Sie sich noch an unser erstes Beispiel auf [Seite 334](#)? Es ging dabei um das Einfügen eines Artikels und einer Warengruppe. Was dabei noch fehlte, war der Einbau der Transaktion, denn schließlich soll unsere Prozedur nicht weniger können als die in [Kapitel 19 auf Seite 319](#) beschriebene Transaktion. Dazu muss man in dem Quelltext angeben, was passieren soll, falls ein Fehler aufgetreten ist.

```

1 DROP PROCEDURE insert_artikel;
2
3 DELIMITER //
4 CREATE PROCEDURE insert_artikel
5 (
6     IN iArtikelname VARCHAR(255),
7     IN iWarengruppe VARCHAR(255)
8 )
9
10 BEGIN
11     DECLARE v_artikel_id      INT DEFAULT -1;
12     DECLARE v_warengruppe_id  INT DEFAULT -1;
13     DECLARE v_count_zuordnung INT DEFAULT 0;
14
15     DECLARE EXIT HANDLER FOR SQLEXCEPTION    -- Handle fuer Fehler
16         BEGIN
17             SELECT 'Wegen Fehler ROLLBACK';
18             ROLLBACK;
19         END;
20
21     START TRANSACTION;
22
23     [...]
24
25     COMMIT;
26
27 END///
28 DELIMITER ;

```

Wird in unserer Prozedur ein Fehler erzeugt, dann bricht die Verarbeitung ab und springt in den Anweisungsblock des Handle ab [Zeile 15](#). Anschließend wird die Prozedur verlassen,

da der Handle mit `DECLARE EXIT` definiert wurde. Alternativ hätte man auch `DECLARE CONTINUE` angeben können. Dann wird die Prozedur nicht verlassen, sondern die Verarbeitung nach Ausführung des Handle fortgesetzt.

MySQL und MariaDB kennen drei solcher Handles: `NOT FOUND`, `SQLWARNING` und `SQLEXCEPTION` (siehe Tabelle 20.2). Alternativ kann der Zustand von `SQLSTATE` abgefragt werden:

```
1 [...]  
2 DECLARE CONTINUE HANDLER FOR SQLSTATE '23000'  
3 [...]
```

Tabelle 20.2 SQL-HANDLER

Art des Handlers	Erklärung
SQLEXCEPTION	Wird ausgelöst, wenn ein Fehler eintritt. Dies sind alle SQLSTATE, die nicht mit 00, 01 oder 02 beginnen.
SQLWARNING	Wird ausgelöst, wenn eine Warnung erfolgt. Dies sind alle SQLSTATE, die mit 01 beginnen.
NOT FOUND	Wird ausgelöst, wenn ein <code>SELECT ... INTO</code> kein Ergebnis findet oder im Zusammenhang mit einem CURSOR. Die entsprechenden SQLSTATE beginnen mit 02.

■ 20.5 CURSOR

Die Verarbeitung innerhalb der Prozeduren war bisher so, dass abhängig von irgendwelchen Bedingungen Teilmengen und Auswertungen von Tabellen verarbeitet wurden. Gerade in Prozeduren ergibt sich aber oft die Situation, dass man sich jeden einzelnen Datensatz anschauen muss.

Mithilfe eines *Cursors*⁹ kann man in Zusammenarbeit mit einer Schleife für jede einzelne Datenzeile eine Entscheidung treffen. Für einen Cursor braucht man vier Befehle: `DECLARE ... FOR CURSOR`, `OPEN`, `FETCH` und `CLOSE`.



MySQL/MariaDB

```
DECLARE cursorname CURSOR FOR
    SELECT auswahl ;
```

Die *auswahl* darf kein `INTO` enthalten. Schließlich soll das Ergebnis der Auswahl erst durch den `FETCH` in einer Variablen landen.



MySQL/MariaDB

```
OPEN cursorname;
```

⁹ Abkürzung für: current set of records

Mit dem OPEN wird die im DECLARE formulierte *auswahl* ausgeführt.



MySQL/MariaDB

```
FETCH cursorname
      INTO variablename[, variablename]*
;
```

Der erste FETCH liest die erste Zeile der Ergebnismenge der *auswahl*. Der Inhalt der ersten Zeile wird in die Variablen kopiert. Daher müssen genauso viele Variablen hinter dem INTO stehen, wie die Ergebnismenge Spalten hat. Anschließend wird ein interner Zeiger auf die nächste Zeile der Ergebnismenge verschoben.

Wird ein FETCH auf eine leere Ergebnismenge ausgeführt oder sind bereits alle Zeilen *gefetcgt*, so wird die NOT FOUND-Exception ausgelöst (siehe [Tabelle 20.2 auf der vorherigen Seite](#)).



MySQL/MariaDB

```
CLOSE cursorname
;
```

Der Cursor wird geschlossen, die Ergebnismenge verworfen.



Hinweis: In MySQL und MariaDB sind CURSOR schreibgeschützt, vorwärtsgerichtet und serverseitig.

- Schreibgeschützt: Sie arbeiten auf temporären Tabellen, sodass Änderungen nicht in den Tabellen übernommen werden.
- Vorwärtsgerichtet: Man kann immer nur zum nächsten Datensatz gehen und nicht zurück. Auch können keine Datensätze übersprungen werden.
- Serverseitig: Sie können nur innerhalb von Prozeduren verwendet werden. Man kann sie nicht aus Client-Anwendungen heraus erstellen und verwenden.

Als Beispiel soll hier die Aufbereitung eines Datenimports dienen. Unten sehen Sie eine XML-Datei, die Bestelldaten mit ihren Positionen enthält. Unglücklicherweise haben die Positionen keine Nummern, sodass wir den Import nicht direkt in die Tabellen **bestellung** und **bestellung_position** durchführen können. Durch unsere Prozedur soll jede Position eine Nummer zugewiesen bekommen. Erst, wenn das alles passiert ist, werden die Daten den beiden Zieltabellen hinzugefügt.

```

1  <?xml version="1.0"?>
2  <bestellungen>
3  <bestellung>
4    <bestellung_id>1</bestellung_id>
5    <kunde_id>1</kunde_id>
6    <adresse_id>1</adresse_id>
7    <datum>2012-05-10 10:01</datum>
8    <positionen>
9      <position>
10       <artikel_id>3001</artikel_id>
11       <menge>1</menge>
12     </position>
13   <position>
14     <artikel_id>3005</artikel_id>
15     <menge>12</menge>
16   </position>
17 </position>
18   <artikel_id>3007</artikel_id>

```

```

19      <menge>123</menge>          36      </position>
20    </position>                  37      </positionen>
21  </positionen>                38      </bestellung>
22 </bestellung>                39      <bestellung>
23 <bestellung>                40      <bestellung_id>3</bestellung_id>
24   <bestellung_id>2</bestellung_id> 41      <kunde_id>2</kunde_id>
25   <kunde_id>2</kunde_id>        42      <adresse_id>1</adresse_id>
26   <adresse_id>1</adresse_id>    43      <datum>2012-05-12 10:03</datum>
27   <datum>2012-05-11 10:02</datum> 44      <positionen>
28 <positionen>                45      <position>
29   <position>                  46      <artikel_id>9015</artikel_id>
30     <artikel_id>9010</artikel_id> 47      <menge>3</menge>
31     <menge>2</menge>            48      </position>
32   </position>                  49      </positionen>
33   <position>                  50      </bestellung>
34     <artikel_id>7856</artikel_id> 51      </bestellungen>
35     <menge>234</menge>

```



Laden Sie die Prozedur mit SOURCE importMitCursor01.sql. Achten Sie darauf, dass die Datei lstBestellungen.xml im gleichen Verzeichnis liegt, oder passen Sie die Pfadangaben vorher an.

Bevor wir die Prozedur ausführen können, müssen wir Importtabellen anlegen und mit dem Inhalt der XML-Dateien füllen. Ich verwende ein LOAD XML. Die Daten der Bestellungen sind im Tag `<bestellung>` und die der Positionen in `<position>` abgelegt. Mit dem ROWS IDENTIFIED BY kann so der Import auf bestimmte Tags eingegrenzt werden. Da kein Tag `<status>` in der XML-Datei vorkommt, wird der pauschal mit SET auf offen gesetzt.

```

1 USE oshop;
2 -- Anlegen und Befuellen der Zwischentabelle
3 -- fuer die Bestellungen
4 DROP TABLE IF EXISTS tmp_bestellung;
5 CREATE TEMPORARY TABLE tmp_bestellung LIKE bestellung;
6 LOAD XML
7 LOCAL INFILE 'lstBestellungen.xml'
8 INTO TABLE tmp_bestellung
9 ROWS IDENTIFIED BY '<bestellung>';
10 SET status = 'offen';
11
12 -- Anlegen und Befuellen der Zwischentabelle
13 -- fuer die Bestellungen
14 DROP TABLE IF EXISTS tmp_position;
15 CREATE TEMPORARY TABLE tmp_position LIKE bestellung_position;
16 ALTER TABLE tmp_position DROP PRIMARY KEY; -- Sonst Probleme!
17 ALTER TABLE tmp_position ADD position_id INT AUTO_INCREMENT PRIMARY KEY;
18 LOAD XML
19 LOCAL INFILE 'lstBestellungen.xml'
20 INTO TABLE tmp_position
21 ROWS IDENTIFIED BY '<position>';

```

In Zeile 16 wird der Primärschlüssel ausgeschaltet. In der Zeile darauf wird ein neuer angelegt, der mit einer laufenden Nummer die Positionen eindeutig unterscheidbar macht.



Aufgabe 20.6: Was würde passieren, wenn man die Anweisung in [Zeile 16](#) weglassen würde und warum?

Schauen wir uns mal an, wie die Daten in den temporären Tabellen nach dem Import aussehen:

```

1 mysql> SELECT bestellung_id, kunde_id, adresse_id, datum, status
2      -> FROM tmp_bestellung;
3 +-----+-----+-----+-----+-----+
4 | bestellung_id | kunde_id | adresse_id | datum           | status |
5 +-----+-----+-----+-----+-----+
6 |          1 |        1 |          1 | 2012-05-10 19:45:00 | offen  |
7 |          2 |        2 |          1 | 2012-05-08 17:42:00 | offen  |
8 |          3 |        2 |          1 | 2012-05-02 16:00:01 | offen  |
9 +-----+-----+-----+-----+-----+
10 3 rows in set (0.00 sec)
11
12 mysql> SELECT bestellung_id, position_nr, artikel_id, menge, position_id
13      -> FROM tmp_position;
14 +-----+-----+-----+-----+-----+
15 | bestellung_id | position_nr | artikel_id | menge    | position_id |
16 +-----+-----+-----+-----+-----+
17 |          1 |        0 |      3001 | 5.000000 |       1 |
18 |          1 |        0 |      3005 | 10.000000 |      2 |
19 |          1 |        0 |      3008 | 100.000000 |      3 |
20 |          2 |        0 |      9010 | 1.000000  |      4 |
21 |          2 |        0 |      7856 | 100.000000 |      5 |
22 |          3 |        0 |      9015 | 1.000000  |      6 |
23 +-----+-----+-----+-----+-----+

```

Die Bestelldaten sind genauso importiert worden, wie wir uns das vorgestellt haben. Nur sehen wir sofort ein Problem: Die Bestellnummer kann so nicht übernommen werden. Grundsätzlich ist auch davon auszugehen, dass importierte Primärschlüssel zu den schon bestehenden nicht einfach so hinzugefügt werden können.

In der Tabelle `tmp_position` ist die Positionsnummer immer 0, da wir keine Informationen über die Nummer der Position innerhalb der Bestellung haben. Um später einfach eine bestimmte Position referenzieren zu können, hatten wir ja einen neuen Primärschlüssel angelegt. Das hat auch funktioniert, wie wir am Inhalt der Spalte `position_id` sehen können.

Eine ggf. vorhandene Prozedur wird gelöscht und der DELIMITER auf // gesetzt. Die Prozedur wird angelegt und die Variablen werden deklariert.

- `vBId`: Der Cursor `curBestellung` durchwandert mit `FETCH` alle Zeilen der Tabelle `tmp_bestellung` und speichert den aktuellen Wert von `bestellung_id` ab.
- `vPId`: Der Cursor `curPosition` durchwandert mit `FETCH` alle Zeilen der temporären Tabelle `tmp_position` und speichert den aktuellen Wert von `position_id` ab.
- `vBestellungId`: Beim Einfügen der Bestellung in die Tabelle `bestellung` wird durch `AUTO_INCREMENT` eine neue Bestellnummer erzeugt. Diese wird per `LAST_INSERT_ID()` ermittelt und zugewiesen.
- `vPosNr`: Ein Zähler, der dazu verwendet wird, die Positionen pro Bestellung zu nummerieren.

- **vEndeHaupt:** Die Schleifenvariable der Hauptschleife. Bei 0 läuft die Schleife, bei 1 soll sie sich beenden.
- **vEndeInnere:** Die Schleifenvariable der inneren Schleife. Bei 0 läuft die Schleife, bei 1 soll sie sich beenden.
- **vWoBinIch:** Mithilfe dieser Variablen unterscheide ich im Handler, welcher Cursor die NOT FOUND-Exception ausgelöst hat (siehe [Tabelle 20.2 auf Seite 353](#)). Der Wert 1 kennzeichnet die Hauptschleife, 2 die innere Schleife.

```

24  DROP PROCEDURE IF EXISTS importBestellung;
25  DELIMITER //
26  CREATE PROCEDURE importBestellung()
27  BEGIN
28  DECLARE vBId  INT;
29  DECLARE vPId  INT;
30  DECLARE vBestellungId INT;
31  DECLARE vPosNr INT ;
32  DECLARE vEndeHaupt  INT DEFAULT 0;
33  DECLARE vEndeInnere INT DEFAULT 0;
34  DECLARE vWoBinIch INT;
```

Ein Cursor wird mit **DECLARE** eingeleitet. Es folgen der Name des Cursors und die Schlüsselwörter **CURSOR FOR**. Jetzt kommt ein **SELECT**, der festlegt, was eigentlich ausgeführt wird, wenn ein Cursor mit **OPEN** geöffnet wird. Der Cursor **curPosition** enthält eine **WHERE-Klausel**, die nur die Positionen zulässt, die zur gerade aktiven Bestellung gehören.

```

36  DECLARE curBestellung CURSOR FOR
37    SELECT bestellung_id FROM tmp_bestellung;
38
39  DECLARE curPosition CURSOR FOR
40    SELECT position_id FROM tmp_position WHERE bestellung_id = vBId;
```

Anders als im Beispiel oben (siehe [Abschnitt 20.4 auf Seite 352](#)) wird hier etwas mehr im Handler zu tun sein. Daher brauchen wir einen Anweisungsblock, der mit **BEGIN ... END**; umklammert wird. Innerhalb des Handlers wird ermittelt, ob die Exception in der Hauptschleife durch ein **FETCH** auf **curBestellung** oder in der inneren Schleife durch eine Exception wegen eines **FETCH** auf **curPosition** ausgelöst wurde.

```

42  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'
43  BEGIN
44    CASE vWoBinIch
45      WHEN 1 THEN SET vEndeHaupt = 1;
46      WHEN 2 THEN SET vEndeInnere = 1;
47    END CASE;
48  END;
```

In [Zeile 50](#) wird der Cursor **curBestellung** geöffnet. Dabei wird der oben deklarierte **SELECT** ausgeführt. Anschließend wird die Schleifenvariable so gesetzt, dass die Schleife läuft. Die Hauptschleife geht von [Zeile 52 bis 86](#). In der ersten Zeile der Hauptschleife wird erst einmal in **vWoBinIch** markiert, dass wir in der Hauptschleife sind. Als Nächstes wird ein **FETCH** auf den Cursor ausgeführt. Dabei wird mit dem ersten Datensatz begonnen, der durch den **SELECT** entstanden ist. Bei jedem **FETCH** geht ein interner Zeiger zum nächsten Datensatz. Ist der **FETCH** nicht am Ende angekommen, steht in **vBId** die Bestellnummer, auf die der interne Zeiger gerade verweist. Ansonsten wird die **NOT FOUND**-Exception ausgelöst und der Handler ausgeführt. Da dieser von Typ **CONTINUE** ist, springt die Prozedur

zur nächsten Zeile, die nach dem FETCH steht. Dort wird dann ermittelt, ob die Hauptschleife sich beenden soll oder nicht.

Ab [Zeile 59](#) wird die Bestellung in die Tabelle bestellung eingefügt, auf die der Cursor gerade zeigt. Dabei wird eine neue Bestellnummer vergeben. Diese müssen wir uns merken, da wir den Wert als Fremdschlüsselwert noch brauchen. Wichtig ist, dass wir hier den Zähler für die Position innerhalb der Bestellung wieder auf 1 setzen. Jetzt wird mit OPEN der zweite Cursor geöffnet. Dieser enthält alle Positionen, die zur aktuellen Bestellung gehören.

Die innere Schleife geht von [Zeile 69](#) bis [83](#). Als Erstes wird wieder in vWoBinIch vermerkt, dass wir in der inneren Schleife sind. Wie oben wird mit FETCH die nächste Position ermittelt oder die Exception ausgelöst. Wurde ein position_id ermittelt, werden die entsprechenden Daten ab [Zeile 76](#) in die Tabelle bestellung_position eingefügt. Bitte beachten Sie, dass hierbei die neu ermittelte bestellung_id und die mitgezählte vPosNr verwendet werden.

Ist die innere Schleife zu Ende, wird der Cursor curPosition geschlossen, und die Hauptschleife startet eine neue Iteration. Wird die Hauptschleife beendet, wird der Cursor curBestellung geschlossen, und alles ist gut ;-).

```

50  OPEN curBestellung;           -- Jetzt wird der SELECT ausgefuehrt
51  SET vEndeHaupt = 0;
52  hauptschleife: LOOP          -- Start Hauptschleife
53  SET vWoBinIch = 1;           -- Ich bin in der Hauptschleife
54  FETCH curBestellung INTO vBId; -- Entweder Wert oder CONTINUE HANDLER
55  IF vEndeHaupt = 1 THEN
56    LEAVE hauptschleife;
57  END IF;
58
59  INSERT INTO bestellung      -- Fuege die neue Bestellung ein
60  SELECT NULL, kunde_id, adresse_id, datum, status, deleted
61  FROM tmp_bestellung
62  WHERE bestellung_id = vBId; -- Und zwar nur die eine!
63
64  SET vBestellungId = LAST_INSERT_ID(); -- Neue Bestellnummer
65  SET vPosNr = 1;                  -- Position bei 1 starten
66  SET vEndeInnere = 0;            -- 
67  OPEN curPosition;              -- Jetzt wird der SELECT ausgefuehrt
68
69  innereschleife: LOOP          -- Start inneren Schleife
70  SET vWoBinIch = 2;           -- Ich bin in der inneren Schleife
71  FETCH curPosition INTO vPId; -- Entweder Wert oder CONTINUE HANDLER
72  IF vEndeInnere = 1 THEN
73    LEAVE innereschleife;
74  END IF;
75
76  INSERT INTO bestellung_position -- Position mit neuen Bestell-/Posnr
77  SELECT vBestellungId, vPosNr, artikel_id, menge, deleted
78  FROM tmp_position
79  WHERE position_id = vPId;
80
81  SET vPosNr = 1 + vPosNr;       -- Erhoehe Positionsnummer
82  ITERATE innereschleife;
83  END LOOP;                   -- Ende inneren Schleife
84  CLOSE curPosition;           -- Schliesse den Cursor
85  ITERATE hauptschleife;
```

```

86 END LOOP;          -- Ende Hauptschleife
87
88 CLOSE curBestellung;    -- Schliesse den Cursor
89 END///
90 DELIMITER ;

```

Führen wir die Prozedur nun aus und betrachten das Ergebnis:

```

1 mysql> CALL importBestellung();
2
3 mysql> SELECT bestellung_id, kunde_id, adresse_id, datum, status
4     -> FROM bestellung;
5 +-----+-----+-----+-----+-----+
6 | bestellung_id | kunde_id | adresse_id | datum           | status   |
7 +-----+-----+-----+-----+-----+
8 [...]
9 |      7 |      1 |          1 | 2012-05-10 10:01:00 | offen    |
10 |     8 |      2 |          1 | 2012-05-11 10:02:00 | offen    |
11 |     9 |      2 |          1 | 2012-05-12 10:03:01 | offen    |
12 +-----+-----+-----+-----+-----+
13 9 rows in set (0.00 sec)
14
15 mysql> SELECT * from bestellung_position;
16 +-----+-----+-----+-----+-----+
17 | bestellung_id | position_nr | artikel_id | menge      | deleted |
18 +-----+-----+-----+-----+-----+
19 [...]
20 |      7 |      1 |    3001 | 1.000000 |      0 |
21 |      7 |      2 |    3005 | 12.000000 |      0 |
22 |      7 |      3 |    3007 | 123.000000 |      0 |
23 |     8 |      1 |    9010 | 2.000000 |      0 |
24 |     8 |      2 |    7856 | 234.000000 |      0 |
25 |     9 |      1 |    9015 | 3.000000 |      0 |
26 +-----+-----+-----+-----+-----+
27 19 rows in set (0.01 sec)

```



Hinweis: Cursor sind nicht sehr schnell. Falls es irgendwie möglich ist, sollte man die üblichen INSERT, UPDATE, DELETE und SELECT mit sehr schlau gebauten WHERE-Klauseln verwenden.

■ 20.6 Aufgaben



Aufgabe 20.7: Machen Sie eine Performancemessung: Ein SELECT * auf die Tabelle bank versus Durchwandern mit einem Cursor.

Aufgabe 20.8: Erstellen Sie eine Prozedur, die einer Bestellung eine neue Position hinzufügt. Die Übergabeparameter müssen plausibilisiert werden.

Aufgabe 20.9: Erweitern Sie diese Prozedur als Transaktion und berücksichtigen Sie auch den Lagerbestand! Einmal bei der Plausibilitätsüberprüfung und wobei noch?

Aufgabe 20.10: Erstellen Sie eine Prozedur, die aus einer Bestellung eine Position löscht. Die Positionsnummern sollen wieder richtig durchnummiert sein. Die Übergabeparameter müssen plausibilisiert werden.

Aufgabe 20.11: Erstellen Sie eine Prozedur, die zwei Positionsnummern vertauscht. Die Übergabeparameter müssen plausibilisiert werden.

Aufgabe 20.12: Erstellen Sie eine Prozedur, die zu einer Bestellnummer eine passende Rechnung erstellt. Dabei soll überprüft werden, ob es schon eine Rechnung zu der Bestellung gibt. Vergessen Sie mir nicht die Positionen!

Aufgabe 20.13: Erstellen Sie eine Prozedur, die in einer neuen Tabelle `revision` folgende Informationen ablegt: Liste der Warengruppen ohne Artikel, Liste der Artikel ohne Warengruppe, Liste der Adressen ohne Kunden oder Lieferanten, Liste der Lieferanten ohne Artikel und Liste der Artikel ohne Lieferanten. Die Tabelle soll nur zwei Spalten haben: ein Zeitstempel und ein Berichtstext.



Hinweis: Weitere Beispiele für Prozeduren finden Sie in [Kapitel 27 auf Seite 439](#) über die Experimente.

21

Funktion



Eigene Ergebnisse mit selbsterstellten Funktionen erzeugen.

- Vertiefendes
 - Anlegen mit CREATE FUNCTION
 - Werterückgabe mit RETURN
 - Funktionen mit DROP FUNCTION löschen



Die Quelltexte des Kapitels stehen in der Datei `listing17.sql` (siehe [Listing 29.19 auf Seite 532](#)).



MySQL/MariaDB

```
CREATE FUNCTION name ([parameter][, parameter]*)  
    RETURNS typ  
    [option]*  
    anweisungsblock  
;  
    parameter:  
        [IN|OUT|INOUT] name typ  
    typ:  
        jeder gültige MySQL-Datentyp (siehe Abschnitt 26.1 auf Seite 397)  
option:  
    COMMENT 'kommentar'  
    | LANGUAGE SQL  
    | [NOT] DETERMINISTIC  
    | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }  
    | SQL SECURITY { DEFINER | INVOKER }
```

Funktionen ähneln in vielen Aspekten den Prozeduren. Wie diese kennen Funktionen folgende Komponenten: Namen, Übergabeparameter, lokale Variablen, Anweisungen, Ver-

zweigungen, Schleifen und Cursor. Der wesentliche Unterschied ist, dass Funktionen *immer* einen Wert zurückliefern müssen.

Auch Prozeduren können Rückgaben liefern, aber die Betonung liegt auf *können*. Funktionen müssen Werte liefern. Im Falle eines Fehlers oder eines unbestimmten Ergebnisses wird üblicherweise NULL verwendet.

Ein einfaches Beispiel: Wir wollen mithilfe einer Funktion Preise prozentual verändern. Das Ergebnis soll auf zwei Stellen nach dem Komma gerundet werden.

```

1  DROP FUNCTION IF EXISTS preis_anpassen;
2  DELIMITER //
3  CREATE FUNCTION preis_anpassen
4  (
5      iPreisOriginal DECIMAL(14,6),          -- Originalpreis
6      iProzentsatz  DECIMAL(7,3)           -- Anpassung in Prozent (0-100)
7  )
8  RETURNS DECIMAL(14,6) DETERMINISTIC
9  BEGIN
10    DECLARE vPreisNeu DECIMAL(14,6);      -- Neuer Preis
11
12    IF iPreisOriginal IS NULL THEN        -- Test auf NULL
13        RETURN NULL;
14    END IF;
15    IF iProzentsatz IS NULL THEN         -- Test auf NULL
16        RETURN NULL;
17    END IF;
18    IF iProzentsatz NOT BETWEEN 0 AND 100 THEN -- Wertebereich Prozent
19        RETURN NULL;
20    END IF;
21    SET vPreisNeu = iPreisOriginal + ((iPreisOriginal * iProzentsatz) / 100.0);
22    SET vPreisNeu = ROUND(vPreisNeu, 2);
23    RETURN vPreisNeu;
24  END //
25  DELIMITER ;

```



Hinweis: Eine weitere Möglichkeit ist das Erstellen *user defined functions*, die als DLL-Dateien dem Server zur Verfügung gestellt werden. Das ist cool und interessant und kann im Detail in <http://dev.mysql.com/doc/refman/5.7/en/adding-functions.html> nachgelesen werden.

Sie können aus Funktionen keine Ausgaben beispielsweise mit SELECT erzeugen. Dies verhindert leider auch die Ausgabe von Warnungen oder Fehlermeldungen. Sie werden dazu *exception handle* verwenden müssen (siehe [Kapitel 20.4 auf Seite 352](#)).

Eine Prozedur kann mit `DROP FUNCTION` gelöscht werden.



MySQL/MariaDB

```

DROP FUNCTION [IF EXISTS] name
;
```

22

TRIGGER



Domino Day für Fortgeschrittene: Wie eins zum anderen führt.

- Vertiefendes
 - Anlegen mit CREATE TRIGGER
 - Verfügbarkeit von OLD und NEW
 - Trigger mit DROP TRIGGER löschen



Die Quelltexte des Kapitels stehen in der Datei `listing18.sql` (siehe [Listing 29.20 auf Seite 533](#)).

■ 22.1 Was ist das?



Definition 72: Trigger

Ein *Trigger* ist ein benannter Anweisungsblock mit folgenden Eigenschaften:

- Der Triggername ist in der Datenbank einmalig.
- Der Trigger hat keine Übergabeparameter.
- Der Trigger liefert nichts zurück.
- Der Trigger ist mit einer Tabelle verbunden.
- Sie müssen festlegen, wann der Trigger automatisch aufgerufen wird: bei einem INSERT, UPDATE oder DELETE auf der verbundenen Tabelle.
- Sie müssen festlegen, ob der Trigger vor oder nach Ausführung von INSERT, UPDATE oder DELETE automatisch aufgerufen wird.

Trigger erinnern stark an Prozeduren (siehe [Kapitel 20 auf Seite 333](#)), da es benannte Anweisungsblöcke sind. Der Unterschied zu einer Prozedur ist, dass ein Trigger *automatisch* aufgerufen wird, wenn ein festgelegtes Ereignis bei einer Tabelle eintritt.

Man kann Trigger immer dann einsetzen, wenn das Ereignis auf der Tabelle zwingend mit einer Aktion verbunden werden muss. Natürlich kann dies auch durch die Programmierung im Client oder durch passende Prozeduren erreicht werden. Trigger bieten sich hier aber als Alternativen an. Der Trigger wird mit CREATE TRIGGER angelegt.



MySQL/MariaDB

```
CREATE TRIGGER triggername
  {BEFORE | AFTER}
  {INSERT | UPDATE | DELETE}
  ON tabellename
  FOR EACH ROW anweisungsblock
;
```

Mit DROP TRIGGER wird er wieder gelöscht.



MySQL/MariaDB

```
DROP TRIGGER [IF EXISTS] triggername
;
```

Trigger sind aber auch nicht ohne. Sie werden automatisch aufgerufen, was zwangsläufig dazu führt, dass man vergisst, dass sie aufgerufen werden. Stellen Sie sich vor, wir machen einen UPDATE auf einer Tabelle mit 100.000 Zeilen. Dann wird für jede Zeile, die verändert wird, der Anweisungsblock des Triggers aufgerufen. Das kann die Performance einer Anweisung dramatisch verschlechtern.



Hinweis: Achten Sie darauf, dass es keine Zirkelaufrufe gibt.

Was bedeutet das? In Tabelle 1 wird eine Zeile eingefügt. Dies löst einen Trigger aus, der in Tabelle 2 eine Zeile einfügt. Dies löst ein Trigger aus, der in Tabelle 3 etwas löscht. Dies löst einen Trigger aus, der in Tabelle 1 eine Zeile einfügt :-(.

Innerhalb eines Triggers stehen bestimmte Werte zur Verfügung, die man oft bei der Trigerverarbeitung braucht. Bei einem INSERT kann über NEW der neu eingefügte Datensatz erreicht werden. Bei UPDATE steht NEW für den neuen Wert und OLD für den alten Wert einer Spalte. Bei DELETE gibt es nur ein OLD.

Tabelle 22.1 Verfügbarkeit von NEW und OLD in einem Trigger

Anweisung	NEW	OLD
INSERT	ja	nein
UPDATE	ja	ja
DELETE	nein	ja

■ 22.2 Ein Beispiel für einen INSERT-Trigger

Wenn wir eine Bestellung anlegen, soll auch eine passende Rechnung erzeugt werden. Dadurch muss der Anwender die Bestelldaten nicht noch einmal manuell als Rechnungsdaten erfassen. Fangen wir mit den Bestelldaten an und machen anschließend die Positionen. In [Zeile 2](#) wird der Triggername festgelegt. Ich verwende dazu folgende Namenskonvention:



Hinweis: Namenskonvention bei einem Trigger:

`tri_tabellenname_zeitpunkt_anweisung`

In [Zeile 3](#) wird festgelegt, dass es ein Trigger ist, der *nach* einem `INSERT` in Tabelle `bestellung` ([Zeile 4](#)) erfolgt.

```

1  DELIMITER //
2  CREATE TRIGGER tri_bestellung_after_insert -- ein schöner Name
3  AFTER INSERT                      -- wann?
4  ON bestellung                     -- bei wem?
5  FOR EACH ROW
6  BEGIN
7    INSERT INTO rechnung            -- was?
8    SET
9      kunde_id      = NEW.kunde_id,
10     bestellung_id = NEW.bestellung_id,
11     adresse_id    = NEW.adresse_id,
12     datum         = NEW.datum;
13 END//
14 DELIMITER ;

```

In [Zeile 7](#) steht der neue `INSERT`, der die Daten von den Bestellungen zu den Rechnungen kopiert. Dabei muss der Trigger die gerade neu eingefügten Daten kennen. Dazu steht das Schlüsselwort `NEW` zur Verfügung. Durch einen Punkt abgetrennt gibt man den Spaltennamen an, in dem der neue Wert steht. Das ist schwieriger zu formulieren als zu verwenden. Schauen Sie in den Quelltext ;-).

Jetzt sind wir doch alle gespannt, ob alles klappt:

```

1  mysql> INSERT INTO
2      -> bestellung (kunde_id, adresse_id, datum)
3      -> VALUES (1, 1, NOW())
4      -> ;
5
6  mysql> SELECT bestellung_id, kunde_id, adresse_id, datum
7      -> FROM bestellung
8      -> ;
9 +-----+-----+-----+
10 | bestellung_id | kunde_id | adresse_id | datum          |
11 +-----+-----+-----+
12 [...]
13 |           10 |        1 |           1 | 2012-06-16 16:30:21 |
14 +-----+-----+-----+

```

Das war klar, in der Tabelle `bestellung` ist der `INSERT` angekommen. Jetzt stellt sich die spannende Frage, was in der Tabelle `rechnung` steht:

```

1 mysql> SELECT rechnung_id, kunde_id, bestellung_id, adresse_id, datum
2      ->   FROM rechnung\G
3 [...]
4 **** 12. row ****
5 rechnung_id: 12
6 kunde_id: 1
7 bestellung_id: 10
8 adresse_id: 1
9 datum: 2012-06-16 16:30:21
10 12 rows in set (0.00 sec)

```

Genau wie wir es wollten: Zur Bestellung ist eine passende Rechnung angelegt worden.



Aufgabe 22.1: Testen Sie das Ganze noch einmal mit einem INSERT, der gleich mehrere Bestellungen einfügt.

Aufgabe 22.2: Programmieren Sie den passenden Trigger zum Kopieren der Bestellpositionen zu Rechnungspositionen.

■ 22.3 Ein Beispiel für einen UPDATE-Trigger

Werden in einer Bestellung die Daten oder Positionen geändert, muss dies auch in der entsprechenden Rechnung nachgehalten werden. Zuerst merken wir uns in [Zeile 36](#) den Primärschlüsselwert der Bestellung, damit wir die passende Rechnung wiederfinden.

Anschließend wird für die entscheidenden Spalten überprüft, ob dort eine Änderung erfolgt ist. Dazu wird der alte Wert mit dem neuen verglichen ([Zeile 38](#))¹.

```

1 DELIMITER //
2 CREATE TRIGGER tri_bestellung_after_update
3 AFTER UPDATE
4 ON bestellung
5 FOR EACH ROW
6 BEGIN
7     DECLARE vBestellungId INT;
8
9     SET vBestellungId = OLD.bestellung_id;    -- Damit wir es wiederfinden
10
11    IF OLD.kunde_id != NEW.kunde_id THEN      -- Nur, wenn nötig
12        UPDATE rechnung
13            SET kunde_id = NEW.kunde_id
14            WHERE bestellung_id = vBestellungId;
15    END IF;
16
17    IF OLD.adresse_id != NEW.adresse_id THEN
18        UPDATE rechnung

```

¹ Das ist natürlich aus Sicht der Performance Quatsch. Es wäre viel besser, alle relevanten Spalten in einer Anweisung zu aktualisieren, egal ob sich die Werte verändert haben. Ich wollte hier nur den Effekt von OLD und NEW darstellen.

```

19     SET adresse_id = NEW.adresse_id
20     WHERE bestellung_id = vBestellungId;
21 END IF;
22
23 IF OLD.datum != NEW.datum THEN
24     UPDATE rechnung
25         SET datum = NEW.datum
26         WHERE bestellung_id = vBestellungId;
27 END IF;
28
29 END//
30 DELIMITER ;

```

Durch einen UPDATE lösen wir jetzt nicht nur einen Trigger aus, sondern mehrere. Alle Bestellungen des Kunden 1 hatten die `adresse_id` 1, bevor sie auf 2 geändert wurden.

```

1 mysql> UPDATE bestellung SET adresse_id = 2 WHERE kunde_id = 1;
2
3 mysql> SELECT bestellung_id, kunde_id, adresse_id, datum
4      -> FROM bestellung
5      -> WHERE kunde_id=1;
6 +-----+-----+-----+-----+
7 | bestellung_id | kunde_id | adresse_id | datum
8 +-----+-----+-----+-----+
9 |           1 |       1 |         2 | 2012-03-24 17:41:00 |
10 |          3 |       1 |         2 | 2011-01-15 16:43:00 |
11 |          4 |       1 |         2 | 2011-01-16 09:15:00 |
12 |          5 |       1 |         2 | 2011-01-16 09:16:00 |
13 |          7 |       1 |         2 | 2012-05-10 10:01:00 |
14 |         10 |       1 |         2 | 2012-06-16 16:30:21 |
15 +-----+-----+-----+-----+

```

Der UPDATE in der Tabelle `bestellung` war erfolgreich, und wie sieht es nun in der Tabelle `rechnung` aus?

```

1 mysql> SELECT rechnung_id, kunde_id, bestellung_id, adresse_id
2      -> FROM rechnung
3      -> WHERE kunde_id = 1;
4 +-----+-----+-----+-----+
5 | rechnung_id | kunde_id | bestellung_id | adresse_id |
6 +-----+-----+-----+-----+
7 |           1 |       1 |           1 |         2 |
8 |           3 |       1 |           3 |         2 |
9 |           4 |       1 |           4 |         2 |
10 |          5 |       1 |           5 |         2 |
11 |          12 |       1 |          10 |         2 |
12 +-----+-----+-----+-----+

```

Sie können jetzt sehen, dass nicht nur eine Zeile geändert wurde, sondern alle betroffenen Zeilen.



Aufgabe 22.3: Programmieren Sie den analogen Trigger für die Aktualisierung der Rechnungspositionen.

■ 22.4 Ein Beispiel für einen DELETE-Trigger

Wird eine Bestellung gelöscht, werden vorher alle seine Positionen gelöscht².

```

1  DELIMITER //
2  CREATE TRIGGER tri_bestellung_before_delete
3  BEFORE DELETE
4  ON bestellung
5  FOR EACH ROW
6  BEGIN
7      DECLARE vBestellungId INT;
8      DECLARE vRechnungId INT;
9
10     SET vBestellungId = OLD.bestellung_id;    -- Damit wir es wiederfinden
11     SELECT rechnung_id
12         FROM rechnung
13        WHERE bestellung_id = vBestellungId
14        INTO vRechnungId;
15
16     DELETE FROM rechnung_position WHERE rechnung_id = vRechnungId;
17     DELETE FROM rechnung          WHERE rechnung_id = vRechnungId;
18     DELETE FROM bestellung_position WHERE bestellung_id = vBestellungId;
19 END//
20 DELIMITER ;

```

Zuerst will ich den Ist-Zustand ermitteln:

```

1  mysql> SELECT
2      -> (
3      ->     SELECT COUNT(*) FROM bestellung_position WHERE bestellung_id = 5
4      -> ) AS AnzPosBestellung,
5      -> (
6      ->     SELECT COUNT(*)
7      ->     FROM
8      ->         rechnung_position INNER JOIN rechnung USING (rechnung_id)
9      ->     WHERE bestellung_id = 5
10     -> ) AS AnzPosRechnung
11     -> ;
12 +-----+-----+
13 | AnzPosBestellung | AnzPosRechnung |
14 +-----+-----+
15 |                 4 |                 4 |
16 +-----+-----+

```

Beide Tabellen haben vier Positionen zur Bestellung mit der Nummer 5. Wir können blind annehmen, dass es jeweils einen passenden Datensatz in den Tabellen `bestellung` und `position` gibt.



Aufgabe 22.4: Warum können wir das einfach so annehmen?

Tipp: Siehe [Definition 22 auf Seite 33](#).

² An welche *böse* Praxis erinnert Sie dieses Vorgehen?

Auf zum munteren Löschen:

```

1 mysql> DELETE FROM bestellung WHERE bestellung_id = 5;
2
3 mysql> SELECT
4     -> (
5     ->     SELECT COUNT(*) FROM bestellung WHERE bestellung_id = 5
6     -> ) AS AnzBestellung,
7     -> (
8     ->     SELECT COUNT(*) FROM bestellung_position WHERE bestellung_id = 5
9     -> ) AS AnzPosBestellung,
10    -> (
11    ->     SELECT COUNT(*) FROM rechnung WHERE bestellung_id = 5
12    -> ) AS AnzRechnung,
13    -> (
14    ->     SELECT COUNT(*)
15    ->     FROM
16    ->         rechnung_position INNER JOIN rechnung USING (rechnung_id)
17    ->         WHERE bestellung_id = 5
18    -> ) AS AnzPosRechnung
19    -> ;
20
21 +-----+-----+-----+-----+
22 | AnzBestellung | AnzPosBestellung | AnzRechnung | AnzPosRechnung |
23 +-----+-----+-----+-----+
24 |          0 |              0 |          0 |          0 |
25 +-----+-----+-----+-----+

```

Alles leergeräumt und weggeputzt.



Aufgabe 22.5: Bauen Sie einen Trigger, der lediglich eine gelöschte Bestellposition auch in rechnung_position löscht.



Hinweis: Mit SHOW TRIGGERS kann eine Liste der aktuell gültigen Trigger eingesehen werden.

```

1 mysql> SHOW TRIGGERS\G
2 **** 1. row ****
3     Trigger: tri_bestellung_after_insert
4         Event: INSERT
5         Table: bestellung
6     Statement: BEGIN
7     INSERT INTO rechnung
8 [...]
9     END
10        Timing: AFTER
11        Created: NULL
12        sql_mode:
13            Definer: root@localhost
14 character_set_client: utf8
15 collation_connection: utf8_general_ci
16 Database Collation: utf8_unicode_ci
17 **** 2. row ****
18     Trigger: tri_bestellung_after_update
19         Event: UPDATE

```

```
20          Table: bestellung
21          Statement: BEGIN
22      DECLARE vBestellungId INT;
23  [...]
24  END
25          Timing: AFTER
26          Created: NULL
27          sql_mode:
28          Definer: root@localhost
29 character_set_client: utf8
30 collation_connection: utf8_general_ci
31 Database Collation: utf8_unicode_ci
32 **** 3. row ****
33          Trigger: tri_bestellung_before_delete
34          Event: DELETE
35          Table: bestellung
36          Statement: BEGIN
37      DECLARE vBestellungId INT;
38  [...]
39  END
40          Timing: BEFORE
41          Created: NULL
42          sql_mode:
43          Definer: root@localhost
44 character_set_client: utf8
45 collation_connection: utf8_general_ci
46 Database Collation: utf8_unicode_ci
47 3 rows in set (0.00 sec)
```

23

EVENT



Einen Terminkalender für den SQL Server erstellen.

- Vertiefendes
 - Anlegen mit CREATE EVENT
 - Anzeigen von Terminen mit SHOW EVENTS
 - Konfigurieren des Terminkalenders
 - Termine mit ALTER EVENT ... ENABLE aktivieren
 - Termine mit ALTER EVENT ... DISABLE deaktivieren
 - Termine mit DROP EVENT löschen



Die Quelltexte des Kapitels stehen in der Datei `listing19.sql` (siehe [Listing 29.21](#)).

■ 23.1 Wie legt man ein Ereignis an?



Definition 73: Event

Ein *Event* oder *Ereignis* ist ein benannter Anweisungsblock, der aufgrund eines zeitlichen Ereignisses aufgerufen wird.

Durch ein Ereignis kann man in periodischen Abständen oder einmalig zu einem bestimmten Zeitpunkt einen Anweisungsblock ausführen. Erinnern Sie sich noch daran, dass wir regelmäßig statistische Daten über unsere Bestellungen in Tabellen abgelegt haben (siehe Abschnitt [20.2.2 auf Seite 342](#))? Wir haben dort eine Prozedur erstellt, die für einen bestimmten Zeitraum den Umsatz anhand der Rechnungsdaten ermittelte.

Den Aufruf dazu wollen wir jetzt automatisieren, indem wir ihn an ein bestimmtes zeitliches Ereignis binden. Dazu verwenden wir den `CREATE EVENT`-Befehl. Dieser sieht zwar im ersten Moment etwas kompliziert aus, ist aber im Prinzip sehr einfach und selbsterklärend.



MySQL/MariaDB

```

CREATE
    EVENT [IF NOT EXISTS] name
    ON SCHEDULE termin
    DO
        anweisungsblock
    ;
termin:
    AT zeitpunkt [+ INTERVAL intervall]**
    |EVERY intervall
        [STARTS zeitpunkt [+ INTERVAL intervall]*]
        [ENDS zeitpunkt [+ INTERVAL intervall]*]

intervall:
    zahl {YEAR|QUARTER|MONTH|WEEK|DAY|HOUR|MINUTE|SECOND
    |YEAR_MONTH|DAY_HOUR|DAY_MINUTE|
    DAY_SECOND|HOUR_MINUTE|HOUR_SECOND|MINUTE_SECOND}

```

Man gibt an, zu welchem *termin* ein bestimmter *anweisungsblock* ausgeführt werden soll. Nehmen wir als erstes Beispiel, dass wir den totalen Umsatz wöchentlich ermitteln wollen.



Hinweis: Namenskonvention bei einem Ereignis:

event_aufgabe_termin

```

1 CREATE EVENT event_umssatz_woche
2   ON SCHEDULE EVERY 1 WEEK
3     DO CALL umssatzreport(1);

```

Jetzt wird jede Woche der Totalumsatz ermittelt.



Hinweis: Mit SHOW EVENTS kann eine Liste der aktuell gültigen Ereignisse eingesehen werden.

```

1 mysql> SHOW EVENTS\G
2 *************************** 1. row ****
3           Db: oshop
4           Name: event_umssatz_woche
5           Definer: root@localhost
6           Time zone: SYSTEM
7           Type: RECURRING
8           Execute at: NULL
9           Interval value: 1
10          Interval field: WEEK
11          Starts: 2012-06-16 17:30:00
12          Ends: NULL
13          Status: ENABLED
14          Originator: 1
15      character_set_client: utf8
16      collation_connection: utf8_general_ci
17      Database Collation: utf8_unicode_ci

```



Aufgabe 23.1: Erstellen Sie solche Ereignisse, dass der Jahresumsatz vierteljährlich und der Monatsumsatz monatlich (bezogen auf ein Jahr) erstellt wird.

Damit ein Ereignis überhaupt ausgeführt wird, muss der *event scheduler* aktiviert werden. Den aktuellen Status ermittelt man mit:

```
1 mysql> SHOW VARIABLES LIKE 'event%';
2 +-----+-----+
3 | Variable_name      | Value   |
4 +-----+-----+
5 | event_scheduler    | OFF     |
6 +-----+-----+
```

Tabelle 23.1 Bedeutung von event_scheduler

Wert	Bedeutung
0	OFF
1	ON
2	DISABLED

Der *event scheduler* wird entweder global über die Konfigurationsdatei my.ini oder my.cnf gesteuert oder zur Laufzeit über SET GLOBAL:

```
1 SET GLOBAL event_scheduler = ON;
```

Machen wir doch mal einen kleinen Test:

```
1 DROP TABLE IF EXISTS a;
2 CREATE TABLE a
3 (
4     zeit DATETIME
5 );
6 DROP EVENT IF EXISTS event_zeit_sekunde;
7 CREATE EVENT event_zeit_sekunde
8     ON SCHEDULE EVERY 2 SECOND
9     DO INSERT INTO a VALUES (NOW());
10
11 -- Warte 20 Sekunden
12 SELECT * FROM a;
13 +-----+
14 | zeit          |
15 +-----+
16 | 2012-06-17 08:14:07 |
17 | 2012-06-17 08:14:09 |
18 | 2012-06-17 08:14:11 |
19 | 2012-06-17 08:14:13 |
20 | 2012-06-17 08:14:15 |
21 | 2012-06-17 08:14:17 |
22 | 2012-06-17 08:14:19 |
23 | 2012-06-17 08:14:21 |
24 | 2012-06-17 08:14:23 |
25 +-----+
```

■ 23.2 Wie wird man ein Ereignis wieder los?

Dazu muss man die Art des Ereignisses betrachten:

- *Einmalig*: Wird ein Ereignis nur einmal zu einem Zeitpunkt ausgeführt, wird das Ereignis nach diesem Zeitpunkt automatisch entfernt.
- *Periodisch mit Endeangabe*: Ist der Endetermin erreicht, wird das Ereignis automatisch entfernt.
- *Periodisch ohne Endeangabe*: Wollen Sie das Ereignis nur deaktivieren und nicht löschen, können Sie ALTER EVENT verwenden, ansonsten DROP EVENT.

Man kann ein Ereignis mit ALTER EVENT deaktivieren:



MySQL/MariaDB

```
ALTER
    EVENT eventname
    [ENABLE | DISABLE]
;
```

```
1 ALTER EVENT event_umsetz_woche DISABLE;
```

Damit kann das Ereignis jederzeit wieder aktiviert werden, wenn der Grund für die Pause entfällt.



Hinweis: Bitte beachten Sie, dass alle Ereignisse, die Sie jetzt im Rahmen Ihrer Ausbildung anlegen, automatisch im Hintergrund auf dem Server laufen. Weisen Sie deshalb Übungsereignissen einen zeitnahen Endetermin zu.

Ein Ereignis kann jederzeit komplett mit DROP EVENT gelöscht werden.



MySQL/MariaDB

```
DROP EVENT [IF EXISTS] name ;
```

Also räume ich besser hier noch auf, bevor ich für heute Schluss mache:

```
1 DROP EVENT event_umsetz_woche ;
```



Aufgabe 23.2: Sie müssen noch das Ereignis event_zeit_sekunde löschen und ggf. den event scheduler ausschalten.

TEIL VI

Anhänge

■ 24.1 Backup und Restore

24.1.1 Backup mit mysqldump

Datensicherungen sind eine wichtige Sache. Sie ermöglichen nicht nur die Wiederherstellung von Daten nach einem Hardware-Problem. Auch, wenn Sie versehentlich Daten gelöscht oder verändert haben, sind Sie über jede Sicherheitskopie froh.

Aus eigener Erfahrung kann ich Ihnen berichten, dass nach zerstörter referenzieller Integrität ohne alte Datensicherungen eine Reparatur mit vertretbarem Aufwand kaum möglich ist. Ihnen bleibt nur die manuelle Datenerhebung aus den Papierunterlagen.

Soviel zur Motivation. Möglichkeiten zur Datensicherung gibt es viele. Eine ist das Sichern der Dateien im Datenverzeichnis des Servers¹:

```
1 ralf@localhost:~> cp /var/lib/mysql/oshop/*.frm ./save/
2 ralf@localhost:~> cp /var/lib/mysql/oshop/db.opt ./save/
```



Hinweis: Das Sichern auf Dateiebene kann nur bei MyISAM durchgeführt werden, da bei InnoDB-Tabellen die Dateien nicht zwingend den aktuellen Stand beinhalten.

Ich empfehle Ihnen die Sicherung mit mysqldump. Dieses Tool sichert die Daten als SQL-Skript, wobei binäre Daten – wie Bilder etc. – in hexadezimalen Zahlen abgelegt werden können. Empfohlen wird²:

```
mysqldump --opt --databases datenbankname > dateiname.sql
```

Schauen wir uns an, was --opt bedeutet. Diese Option ist ein Synonym für folgende Optionen:

- --add-drop-table: Vor jedem CREATE TABLE wird die ggf. vorhandene Tabelle gelöscht.

¹ Sie müssen dazu Leserechte auf dem Verzeichnis besitzen.

² Beachten Sie, dass mysqldump im Suchpfad liegen muss. Ggf. im bin-Verzeichnis der Installation nachschauen!

- **--add-locks:** Die Tabellen werden vor dem Einfügen der Daten mit `LOCK TABLES` gesperrt und danach wieder mit `UNLOCK TABLES` freigegeben. Das Einfügen wird dadurch beschleunigt.
- **--create-options:** Alle MySQL-proprietären Tabellenoptionen bleiben erhalten. Alternativ könnte man `--compatible=` diverse SQL-Dialekte auswählen.
- **--disable-keys:** Vor dem Einfügen werden die Schlüssel deaktiviert. Nach dem Einfügen der Datensätze werden diese wieder aktiviert. Dadurch, dass die Schlüssel während des Einfügens nicht aktiv sind, geschieht das Einfügen schneller. Anschließend wird der entsprechende Index wieder aufgebaut. Wie schon auf Seite 104 erwähnt, ist diese Option nur bei MyISAM wirksam.
- **--extended-insert:** Es wird nur ein `INSERT` mit vielen Zeilen anstelle von vielen `INSERTs` mit nur einer Zeile verwendet. Auch diese Option beschleunigt das Einfügen.
- **--lock-tables:** Die Tabellen werden bei MyISAM mit `READ LOCAL` gesperrt. Für InnoDB wird stattdessen die Option `--single-transaction` empfohlen.
- **--quick:** Bei großen Datenmengen werden die Datensätze einzeln vom Server geholt. Ansonsten werden erst alle Datensätze gelesen und im Speicher gehalten.
- **--set-charset:** Der verwendete Zeichensatz wird durch `SET NAMES` am Anfang des Skripts angegeben.

Aus diesen Kommentaren und der Tatsache, dass wir binäre Daten haben (Tabelle `bild`), ergibt sich folgender Aufruf:

```
1 ralf@localhost:~> mysqldump -uroot -p --add-drop-database --add-locks --create
   -options --extended-insert --single-transaction --quick --set-charset --
   hex-blob --databases oshop > ./save/oshop.sql
```

Die Option `--hex-blob` wandelt die binären Daten – das sind `BINARY`, `VARBINARY`, `BLOB` und `BIT` – in eine hexadezimale Schreibweise um.

Die Optionen können der besseren Übersicht wegen in eine Datei ausgelagert werden:

```
1 ralf@localhost:~> mysqldump -uroot -p < ./oshop.opt > ./save/oshop.sql
```

Wenn Sie unter Linux/Unix einen CRON-Job erstellen möchten, der täglich eine Sicherung macht, empfehle ich folgende Zeile³:

```
1 ralf@localhost:~> crontab -e
2 # Jeden Tag um 23:30 Uhr:
3 30 23 * * * /pfadzumskript/oshop.backup.sh
```

Das Shell-Skript⁴ sieht dann so aus:

```
1#!/bin/sh
2## Einige Variablen
3echo on
4MyUSER="root"
5MyPASS=""
6MyDB="oshop"
7MyOption="--add-drop-table --add-locks --create-options --extended-insert --
   single-transaction --quick --set-charset --hex-blob --databases"
8MySQLDUMP="$(which mysqldump)"
```

³ Vergessen Sie nicht, für die Skripte das executable-Flag zu setzen.

⁴ Quelle: <http://serversupportforum.de/forum/sql/40542-mysql-dump-ueber-cronjob-2-von-3-laufen.html>

```

9 MyZIP=$(which gzip)
10
11 ## Verzeichnisname anhand des Datums ermitteln
12 ## und ggf. anlegen
13 MyNOW=$(date +"%Y-%m-%d")
14 MyDEST='./save'
15 MyMBD="$MyDEST/$MyNOW"
16 [ ! -d $MyMBD ] && mkdir -p $MyMBD || :
17
18 ## Los geht's
19 MyFILE="$MyMBD/$MyDB.$MyNOW.sql.gz"
20 $MySQLDUMP -u $MyUSER -p$MyPASS $MyOption $MyDB | $MyZIP -9 > $MyFILE

```

Das Skript ist so aufgebaut, dass Sie eigentlich nur die `My*`-Variablen anpassen müssen, um es verwenden zu können.

Wenn Sie tabellenweise sichern wollen:

```
1 mysqldump --opt --tables -uroot -p oshop artikel > artikel.sql
```

24.1.2 Restore mit mysqldump

Wenn Sie das Backup mit dem obigen Shell-Skript erstellt haben, wird Ihnen beim Restore die gesamte Datenbank wiederhergestellt. Dabei werden alle gerade vorhandenen Tabellenstände etc. gelöscht.

Man kann diesen Dump auf zwei Arten wieder einspielen. In beiden wird der MySQL/MariaDB Client verwendet:

```
1 ralf@localhost:~> zcat oshop.18-06-2012.sql.gz | mysql -uroot -p oshop
```

Mit `zcat` wird die Datei entpackt und auf die Standardausgabe umgeleitet. Das Pipe-Symbol `|` macht daraus einen Eingabestrom für den MySQL Client.

Eine andere Methode liefert der Aufrufparameter `-e`. Bitte beachten Sie dabei, dass die Datei entpackt sein muss.

```
1 ralf@localhost:~> mysql -e "source ./oshop.18-06-2012.sql" -uroot -p oshop
```

Falls Sie die Tabellendaten im CSV-Format gesichert haben, können Sie für die Wiederherstellung auch `mysqlimport` verwenden. Er arbeitet im Prinzip wie ein `LOAD DATA INFILE`. Dieser erwartet, dass der Name der Sicherungsdatei vor der Endung der Tabellennamen ist.

```
1 ralf@localhost:~> mysqlimport -uroot -p --local oshop artikel.csv
```

■ 24.2 Benutzerrechte

24.2.1 Benutzerrechte und Privilegien

Das Anmelden auf dem Server erfolgt in diesem Buch mit dem Benutzer `root`. Dieser darf alles machen und ist ideal für die SQL-Einführung, da keine Einschränkungen bzgl. der Befehle vorliegen.

Das ist aber nicht der Normalfall. Datenbankbenutzer sollten nur das tun dürfen, was sie für die Anwendung brauchen, und auch nur auf den DBMS-Objekten, die zur Anwendung gehören. Soll doch die Anwendung, die den Online-Shop betreibt, nicht auch auf anderen Datenbanken wie einer Vereinsverwaltung oder einem CMS wie Typo3 Datenmanipulationen vornehmen dürfen.

Für unseren Online-Shop wollen wir einen Benutzer haben, der eine Art *Administrator* für die Datenbanken darstellen soll, und einen Benutzer für die alltäglichen Arbeiten: `oshop_root` und `oshop_user`.

Stellt sich zunächst die Frage, was für Rechte es eigentlich gibt, die man vergeben oder entziehen kann. Bei Dateisystemen sind dies in der Regel die Rechte Lesen, Anlegen, Ändern und Löschen in verschiedenen Varianten. So ähnlich ist es auch in SQL, wie Sie der [Tabelle 24.1 auf der nächsten Seite](#) entnehmen können.

Die Rechte bestimmen, welchen SQL-Befehl der Benutzer auszuführen darf. Man könnte jetzt annehmen, dass es somit für jeden Befehl ein solches Recht gibt, aber manche Rechte sind summarisch wie beispielsweise `CREATE USER`, der für mehrere Befehle steht.

SQL:2016 und MySQL/MariaDB unterscheiden sich in vielen Punkten. Zum einen kennen MySQL und MariaDB erheblich mehr Rechte als SQL:2016, zum anderen werden einige Befehle im Detail anders interpretiert. Die wichtigsten Unterschiede zwischen MySQL/-MariaDB und SQL:2016 sind:

- In MySQL/MariaDB wird ein Benutzer über seinen Namen *und* seinen Hostnamen identifiziert: `'abc'@'192.168.41.%'`, `'abc'@'localhost'` und `'abc'@'%'` sind drei verschiedene Benutzer. SQL:2016 nimmt nur den Benutzernamen, hier also `'abc'`.
- SQL:2016 kennt keine Benutzerrechte auf Ebene einer oder aller Datenbanken (global).
- In SQL:2016 sind die Benutzerrechte hierarchisch bzgl. des Rechte vergebenden Benutzers angeordnet. Wird ein Benutzer gelöscht, werden auch alle Rechte entzogen, die dieser anderen per GRANT zugewiesen hat. In MySQL/MariaDB wird das nur bei `DROP USER` gemacht.
- Wenn Sie in SQL:2016 eine Tabelle löschen, werden alle damit verbundenen Rechte entzogen.
- Wenn Sie in SQL:2016 ein Recht entziehen, werden alle Rechte, die aufgrund dieses Rechts erteilt wurden, ebenfalls entzogen.
- In MySQL und MariaDB werden Rechte nur durch `DROP USER` und `REVOKE` entzogen. Alternativ kann man die Systemtabellen mit den Benutzerrechten direkt manipulieren.
- In MySQL und MariaDB kann man das Recht `INSERT` auf bestimmte Spalten einer Tabelle einschränken, sodass ein Benutzer eine neue Zeile anlegen kann, wenn er nur Angaben zu diesen Spalten macht. Die anderen Spalten werden mit ihren Standardwerten aufgefüllt. Ist der MySQL bzw. MariaDB Server im `strict`-Modus gestartet, wird der `INSERT` abgewiesen.

Tabelle 24.1 Von MySQL unterstützte Benutzerrechte
(S= * MySQL proprietär, S= ° von MySQL nicht unterstützt)

S	Benutzerrecht	Erlaubt oder entzieht ...
	ALL [PRIVILEGES]	... alle Rechte außer GRANT OPTION.
	ALTER	... das Recht auf ALTER TABLE.
*	ALTER ROUTINE	... das Recht, Prozeduren zu verändern oder zu löschen.
	CREATE	... das Recht, Schemata oder Tabellen zu erstellen.
*	CREATE ROUTINE	... das Recht, Prozeduren zu erstellen.
*	CREATE TABLESPACE	... alle Rechte bzgl. tablespaces und Logdateigruppen.
*	CREATE TEMPORARY TABLES	... das Recht, temporäre Tabellen anzulegen.
*	CREATE USER	... das Recht auf {CREATE DROP RENAME} USER und REVOKE ALL PRIVILEGES.
*	CREATE VIEW	... das Recht, Ansichten anzulegen und zu verändern.
	DELETE	... das Recht auf DELETE.
	DROP	... das Recht, Schemata, Tabellen und Ansichten zu löschen.
*	EVENT	... alle Rechte bzgl. Ereignisse.
*	EXECUTE	... das Recht, Prozeduren auszuführen.
*	FILE	... das Recht, Dateioperationen auf dem Server auszuführen.
*	GRANT OPTION	... das Recht, Benutzerrechte zuzuweisen oder zu entziehen.
*	INDEX	... das Recht, Indizes anzulegen oder zu löschen.
	INSERT	... das Recht auf INSERT.
*	LOCK TABLES	... das Recht auf LOCK TABLES für Tabellen, auf denen man das SELECT-Recht besitzt.
*	PROCESS	... das Recht auf SHOW PROCESSLIST.
*	PROXY	... das Recht, einen Proxy-Benutzer einzurichten.
°	REFERENCES	... das Recht einen Fremdschlüssel-Constraint einzurichten.
*	RELLOAD	... das Recht auf alle FLUSH-Anweisungen.
*	REPLICATION CLIENT	... das Recht, Master- oder Slave-Server zu erfragen.
*	REPLICATION SLAVE	... das Recht, die Replikationsdaten vom Master auszulesen.
*	SELECT	... das Recht auf SELECT.
*	SHOW DATABASES	... das Recht, alle Datenbanken anzuzeigen.
*	SHOW VIEW	... das Recht auf SHOW CREATE VIEW.
*	SHUTDOWN	... das Recht, mysqladmin shutdown auszuführen.
*	SUPER	... das Recht, administrative Anweisungen wie CHANGE MASTER TO, KILL, PURGE BINARY LOGS, SET GLOBAL und mysqladmin debug auszuführen.
*	TRIGGER	... das Recht auf alle Trigger-Anweisungen.
	UPDATE	... das Recht auf UPDATE.
	USAGE	Synonym für <i>keine Rechte</i> .

24.2.2 Benutzer anlegen/Recht zuweisen

24.2.2.1 CREATE USER

Um einem Benutzer Rechte zuzuweisen, muss es erst mal einen geben:



MySQL/MariaDB

```
CREATE USER
    benutzerspezifikation
    [,benutzerspezifikation]*
;
benutzerspezifikation:
    benutzername
    [IDENTIFIED BY [PASSWORD] 'passwort'
    |IDENTIFIED WIDTH auth_plugin [AS 'auth_string']]
```

Um diesen Befehl verwenden zu können, muss man das CREATE USER- oder das INSERT-Recht auf der Datenbank `mysql` haben, sodass für jeden Benutzer eine Zeile in der Tabelle `mysql.user` eingefügt wird.

Wie oben schon erwähnt, setzt sich in MySQL der `benutzername` aus zwei Teilen zusammen: `name@host`. Der `host` kann ein Hostname wie `www.myhost.de` oder eine IP-Adresse sein. Beide Varianten können durch den Platzhalter `%` Hostfamilien festlegen. Möchten Sie beispielsweise allen Benutzern innerhalb eines Subnetzwerks den Zugriff erlauben: `'192.168.41.%'`.

Dem Benutzernamen kann ein Passwort mitgegeben werden. Dieses wird klartextlich in der Tabelle `mysql.user` abgelegt. Um zu vermeiden, dass dieses einfach ausgespäht werden kann, können Sie die Funktion `PASSWORD()` verwenden:

```
1 mysql> SELECT PASSWORD('86(2,Lphg')\G
2 **** 1. row ****
3 PASSWORD('86(2,Lphg'): *A31AAF3968A1272A06DAC82B3D33646B1BD43A79
```

Wenn Sie ein Plugin zur Authentifizierung verwenden, können Sie anstelle des Passworts dieses angeben.



Hinweis: Falls Sie den Benutzer über einen Client anlegen, ist das klartextlich lesbare Passwort in der History abgelegt (z.B. `~/.mysql_history`).

Legen wir jetzt unsere beiden Benutzer an:

```
1 mysql> CREATE USER
2     -> 'oshop_root'@'%' IDENTIFIED BY 'root'
3     -> , 'oshop_user'@'%' IDENTIFIED BY 'user'
4     -> ;
5 mysql> SHOW GRANTS FOR 'oshop_root'@'%'\G
6 **** 1. row ****
7 Grants for oshop_root@%: GRANT USAGE ON *.* TO 'oshop_root'@'%' IDENTIFIED BY
    PASSWORD '*81F5E21E35407D884A6CD4A731AEBFB6AF209E1B'
```



Aufgabe 24.1: Hinter dem Wort GRANT steht das Recht USAGE. Was bedeutet das noch mal?

Der Platzhalter im Hostnamen ist so gewählt, dass der Benutzer sich von allen Rechnern aus anmelden kann. Ein Anmeldeversuch sorgt für Ernüchterung:

```
1 ralf@localhost:~> mysql -uoshop_root -proot
2 ERROR 1045 (28000): Access denied for user 'oshop_root'@'localhost' (using
password: YES)
```

Ein Anmeldeversuch von einem entfernten Rechner aus wäre übrigens erfolgreich gewesen. Hier kommt eine nicht offensichtliche Eigenschaft von MySQL und MariaDB zum Tragen. Der Hostname `localhost` wird in der Zugriffskontrolle besonders behandelt. Will man dem Benutzer wirklich jeden Zugriffsweg erlauben, muss man tatsächlich noch einmal den gleichen Benutzer mit dem `localhost` anlegen:

```
1 mysql> CREATE USER
2      -> 'oshop_root'@'localhost' IDENTIFIED BY 'root'
3      -> , 'oshop_user'@'localhost' IDENTIFIED BY 'user'
4      -> ;
5 Query OK, 0 rows affected (0.05 sec)
6
7 mysql> exit
8 Bye
9 ralf@localhost:~> mysql -uoshop_root -proot
10 [...]
11 mysql>
```

Jetzt klappt's auch mit dem `localhost`.



Hinweis: Ich habe hier die Passwörter `root` und `user` gewählt. Davon ist in der Praxis dringend abzuraten. Frei nach den Empfehlungen des BSI (siehe [[Inf19](#)]) sollten folgende Regeln gelten:

- Keine Wörter oder Wortkombinationen
- Mindestens 10 Zeichen lang
- Mindestens ein Sonderzeichen
- Kein Trivialpasswörter wie `aa11..bb22` etc.

Mit `DROP USER` kann ein Benutzer wieder gelöscht werden.



MySQL/MariaDB

```
DROP USER [IF EXISTS] benutzerspezifikation[, benutzerspezifikation]*  
;
```

24.2.2.2 GRANT

Wir haben jetzt zwar zwei Benutzer, aber sie dürfen nichts. Die Ausgabe von `SHOW GRANTS FOR` hatte genau dies mit `USAGE` mitgeteilt. Mit `GRANT` können einem Benutzer neue Rechte zugewiesen werden. Hier eine gekürzte Version des Befehls:



SQL:2016, MySQL/MariaDB

```
GRANT benutzerrecht[, benutzerrecht]*  
    ON [objekttyp] privilegtiefe  
    TO benutzerspezifikation[, benutzerspezifikation]*  
    [WITH GRANT OPTION]  
;
```

SQL:2016 kennt die Objekttypen TABLE, DOMAIN, COLLATION, CHARACTER SET und TRANSLATION. MySQL und MariaDB kennen TABLE, FUNCTION und PROCEDURE.

SQL:2016 hat keine *privilegtiefe*; hier steht dann der Objektname wie beispielsweise der Tabellenname. MySQL und MariaDB haben die Privilegiefäden Global (* und *.*), Datenbank (*datenbankname*.*), Tabelle (*datenbankname*.*tabellename*, *tabellename*) und Routine (*datenbankname*.*routinenname*).

Soll der Benutzer anderen Benutzern Rechte vergeben können – natürlich nur solche, die er selber inne hat –, so wird WITH GRANT OPTION ans Ende gesetzt.

Für den Benutzer oshop_root sind folgende Angaben nötig:

- *benutzerrecht*: Unser oshop_root soll alles dürfen, dazu können wir der [Tabelle 24.1 auf Seite 381](#) das ALL PRIVILEGES entnehmen.
- *objekttyp*: Ist optional, daher stellt sich die Frage, brauchen wir die Angabe? Nö. Wir wollen ja, dass er auf allen Objekten der Datenbank alles darf.
- *benutzerspezifikation*: Diese kann genauso umfänglich sein wie bei CREATE TABLE. Wäre der Benutzer noch nicht vorhanden, würde er mit dieser Spezifikation angelegt werden. Da wir ihn schon angelegt haben, reichen hier die Benutzernamen 'oshop_root'@'%' und 'oshop_root'@'localhost'.
- **WITH GRANT OPTION**: Da er Rechte vergeben können soll, muss diese Option verwendet werden.

Das alles führt zu folgender Anweisung:

```
1 mysql> GRANT ALL PRIVILEGES  
2     -> ON oshop.*  
3     -> TO 'oshop_root'@'%', 'oshop_root'@'localhost'  
4     -> WITH GRANT OPTION  
5     -> ;  
6  
7 mysql> SHOW GRANTS FOR 'oshop_root'@'%'\G  
8 *************************** 1. row ***************************  
9 Grants for oshop_root@%: GRANT USAGE ON *.* TO 'oshop_root'@'%' IDENTIFIED BY  
    PASSWORD '*81F5E21E35407D884A6CD4A731AEFB6AF209E1B'  
10 ***** 2. row *****  
11 Grants for oshop_root@%: GRANT ALL PRIVILEGES ON 'oshop'.* TO 'oshop_root'@'%'  
    WITH GRANT OPTION  
12 2 rows in set (0.00 sec)
```

Probieren wir doch mal aus, ob er auch genau das darf oder nicht darf, was wir wollten:

```
1 ralf@localhost:~> mysql -uoshop_root -proot  
2 mysql> CREATE SCHEMA wurst;  
3 ERROR 1044 (42000): Access denied for user 'oshop_root'@'localhost' to  
    database 'wurst'  
4
```

```

5 mysql> CREATE TABLE oshop.wurst (a INT);
6 Query OK, 0 rows affected (0.16 sec)
7
8 mysql> DROP TABLE oshop.wurst;
9 Query OK, 0 rows affected (0.09 sec)

```

In [Zeile 2](#) wird versucht, eine neue Datenbank anzulegen. Dies scheitert, da der Benutzer `oshop_root` keine entsprechenden globalen Rechte besitzt. Das Anlegen einer Tabelle in [Zeile 5](#) und das anschließende Löschen derselben hingegen klappt prima.

Legen wir jetzt den Benutzer `ohop_user` an:

- *benutzerrecht*: Unser `oshop_user` soll die normalen Arbeiten auf den Tabellen durchführen dürfen. Dazu gehören das Einfügen neuer Daten mit `INSERT`, das Verändern und Löschen bestehender Daten mit `UPDATE` bzw. `DELETE`. Außerdem muss er Auswertungen mit `SELECT` durchführen dürfen.
- *objekttyp*: Ist optional, könnte aber `TABLE` lauten.
- *benutzerspezifikation*: Hier gilt das Gleiche wie beim Benutzer `oshop_root`.
- `WITH GRANT OPTION`: Der `oshop_user` soll keine weiteren Benutzer anlegen können, daher fehlt diese Angabe.

Zusammengefasst kommt folgende Anweisung heraus:

```

1 mysql> GRANT INSERT, UPDATE, DELETE, SELECT
2     -> ON TABLE oshop./*
3     -> TO 'oshop_user'@'%', 'oshop_user'@'localhost'
4     -> ;
5 Query OK, 0 rows affected (0.00 sec)
6
7 mysql> SHOW GRANTS FOR 'oshop_user'@'%'\G
8 **** 1. row ****
9 Grants for oshop_user@%: GRANT USAGE ON *.* TO 'oshop_user'@'%' IDENTIFIED BY
    PASSWORD '*D5D9F81F5542DE067FFF5FF7A4CA4BDD322C578F'
10 **** 2. row ****
11 Grants for oshop_user@%: GRANT SELECT, INSERT, UPDATE, DELETE ON 'oshop'.* TO
    'oshop_user'@'%'
12 2 rows in set (0.00 sec)

```



Aufgabe 24.2: Testen Sie die Privilegien des neuen Benutzers: Darf er alles, was wir wollen, und werden ihm unerlaubte Aktionen verweigert?

24.2.2.3 REVOKE

Will man einem Benutzer ein oder mehrere Rechte entziehen: `REVOKE`.



SQL:2016, MySQL/MariaDB

```

REVOKE benutzerrecht[, benutzerrecht]*  

    ON [objekttyp] privilegtiefe  

    FROM benutzernamen[, benutzernamen]*  

    [WITH GRANT OPTION]  

;

```

oder



SQL:2016, MySQL/MariaDB

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM
    FROM benutzername[, benutzername]*
;
```

Es ist letztlich eine Frage, was bequemer ist: eine Positivliste oder eine Negativliste. Gebe ich an, was dem Benutzer erlaubt ist, oder eher, was ihm verboten ist?

In unserem Fall soll der Benutzer `oshop_user` nicht löschen dürfen, da wir ja eigentlich das Löschkennzeichen verwenden wollen (siehe [Seite 33](#)).

```
1 mysql> REVOKE DELETE
2      -> ON TABLE oshop.*
3      -> FROM 'oshop_user'@'%', 'oshop_user'@'localhost'
4      -> ;
```



Aufgabe 24.3: Erlauben Sie dem Benutzer, in den Hilfstabellen einer *n:m*-Verknüpfung zu löschen.

■ 24.3 MySQL und MariaDB Engines

Eine Liste (Auszug) von Engines für MySQL und MariaDB. MariaDB-Engines sind kursiv und der Default ist jeweils fett gedruckt.

- **ARCHIVE:** Wie der Name schon sagt, dient diese Engine dazu, Daten effizient zur Aufbewahrung abzuspeichern. Auswertungen und Datenänderungen werden nur sehr teuer unterstützt. Besonders bei großen Datenmengen ist diese Engine sehr vorteilhaft.
- **Aria:** Diese Engine ist eine Weiterentwicklung von MyISAM unter MariaDB.
- **Cassandra:** Eine NoSQL Engine.
- **BLACKHOLE:** Diese Engine ist ein *WRITE ONLY*-Baustein. Sie können zwar Daten in die Engine reinschreiben, aber nicht mehr auslesen :-(. Tatsächlich verwendet man diese Engine nur zu (Performance-)Testzwecken.
- **CSV:** Die Daten werden als CSV-Datei verwaltet. Man kann also direkt auf CSV-Dateien `SELECT` etc. ausführen. Man erspart sich damit lästiges Importieren oder Exportieren von CSV-Dateien. Der Nachteil ist eine geringe Performance und das Fehlen wichtiger Features.
- **CONNECT:** Zugriff auf Textdateien und Tabellen anderer RDBMS.
- **EXAMPLE:** Keine eigentliche Engine, sondern eine Einstiegshilfe für Programmierer neuer Engines.
- **FEDERATED:** Mithilfe dieser Engine greifen Sie auf Tabellen anderer MySQL Server zu.
- **FederatedX:** MariaDB-spezifische Erweiterung von FEDERATED.
- **InnoDB:** Das Arbeitspferd für die meisten nicht trivialen Anwendungen. Sie liefert fast alles, was man für die übliche Verarbeitung braucht. Wird bei MySQL im `CREATE TABLE` keine andere Engine angegeben, wird diese verwendet.

- **MEMORY:** Die Daten werden im Arbeitsspeicher abgelegt. Dies macht die Verarbeitung der Daten sehr schnell, was besonders für temporäre Tabellen von Vorteil sein kann.
Wichtig: Da die Daten nur im Arbeitsspeicher vorliegen, stehen sie beispielsweise nach einem Serverneustart nicht mehr zur Verfügung!
- **MERGE:** Falls Sie Ihre Daten aus irgendeinem Grund nicht in einer Tabelle, sondern strukturgeleich auf mehrere Tabellen verteilt haben, können Sie mit einer MERGE-Tabelle so tun, als arbeiteten Sie nur mit einer.
- **MyISAM:** Zwar ist diese Engine nicht transaktionsfähig und auch andere Features fehlen, aber sie eignet sich sehr gut für die Auswertung großer, nicht zu verschachtelter Datens Mengen.
- **OQGRAPH:** Mit dieser Engine können baumartige Strukturen und komplexe Graphen abgebildet werden.
- **XtraDB:** Eine MariaDB-spezifische Erweiterung/Verbesserung der InnoDB. Wird bei MariaDB im CREATE TABLE keine andere Engine angegeben, wird diese verwendet.
Wichtig: Geben Sie bei MariaDB als Engine InnoDB an. Es wird automatisch eine XtraDB angelegt, obwohl immer InnoDB angezeigt wird.

25

Rund um den MySQL Client

■ 25.1 Aufruf(parameter)

Ich möchte hier die Aufrufparameter vorstellen, die ich tatsächlich schon mal gebraucht habe. Der Client hat aber noch eine Vielzahl weiterer Parameter. Besonders die zur SSL-Verschlüsselung möchte ich hier erwähnen.

Normalerweise wird der MySQL/MariaDB Client in einer Shell-Oberfläche aufgerufen. Bei einer Installation sollte man darauf achten, dass der Pfad, in dem die ausführbare Clientdatei liegt, im Suchpfad enthalten ist. Falls das nicht so ist¹, muss der Pfad mit angegeben werden:

```
c:\xampp\mysql\bin\mysql Aufrufparameter
```

Beendet wird der Client mit `exit` oder `quit`.

Langform	Kurzform
<code>--help</code>	<code>-?</code>

Der Client gibt den Hilfetext aus und beendet sich selbst.

Langform	Kurzform
<code>--compress</code>	<code>-c</code>

Der Datenverkehr zwischen Client und Server wird komprimiert (nicht verschlüsselt!). Dies geschieht in beide Richtungen. Werden größere Datenmengen verschoben, lohnt sich der Aufwand; ansonsten ist die Rechenzeit zu beachten, die das Komprimieren kostet.

Langform	Kurzform
<code>--database=DBName</code>	<code>-D DBName</code>

Legt die in der Sitzung verwendete Datenbank fest. Alle nachfolgenden Befehle beziehen sich nun auf diese Datenbank. Man muss dann nicht bei Befehlen wie `SELECT` den Datenbanknamen mit Punkt getrennt vor den Tabellennamen schreiben. Die Datenbank muss natürlich vorhanden sein. Wollen Sie während der Sitzung die verwendete Datenbank wechseln, benutzen Sie `USE datenbankname`.

¹ Bei der WAMPP-Installation beispielsweise wird der Pfad nicht gesetzt.

Langform	Kurzform
--default-character-set=Zeichensatz	

Der Standardzeichensatz des MySQL Clients ist `latin1`. Verwenden Sie diesen für Ihre Tabellen, brauchen Sie ihn hier auch nicht anzugeben. Verwenden Sie aber beispielsweise `utf8`, so müssen Sie diesen hier eingeben, damit die Daten korrekt am Server ankommen und dargestellt werden. Die Windows COMMAND-Box kann kein Unicode. Wundern Sie sich also nicht, wenn unter Windows die Umlaute nicht korrekt angezeigt werden. Falls Sie einen Unicode verwenden, ist die Verwendung eines Query-Browsers zu empfehlen.

Langform	Kurzform
- --execute=Anweisung	-e Anweisung

Die Anweisung wird ausgeführt und der MySQL Client wieder verlassen. Diese Option lässt sich hervorragend für das Umleiten von Ergebnissen in eine Datei verwenden:

```
mysql --execute="SELECT * FROM oshop.kunde" > erg.txt
```

Auf den Abschluss des Befehls durch ein Semikolon ; kann verzichtet werden. Werden allerdings mehrere Anweisungen angegeben, so müssen diese wieder getrennt werden.

Langform	Kurzform
- -host=Hostname	-h Hostname

Der Client wird mit dem angegebenen Host verbunden. Der Hostname kann eine IP-Adresse oder – falls eine Namensauflösung erreichbar ist – ein textlicher Name sein. Wird diese Option nicht verwendet, versucht der Client, eine Verbindung mit `127.0.0.1` oder `localhost` aufzubauen.

Typische Gründe, warum der Aufbau scheitert, sind:

- Die Firewall des Zielrechners blockiert.
- Der Hostname wird nicht oder falsch aufgelöst.
- Der Zielrechner verwendet eine andere Portnummer (siehe `--port`).
- Der SQL-User hat nicht das Recht, sich über diese IP-Adresse anzumelden.
- Der Server lässt nur lokale Anmeldungen zu.
- Der Zielrechner ist nicht erreichbar.
- Der MySQL Server ist nicht gestartet.

Langform	Kurzform
--html	-H

Erzeugt eine HTML-Ausgabe. Das Ergebnis einer SQL-Anweisung wird dann als HTML-Tabelle ausgegeben. Durch Umleiten der Ausgabe – beispielsweise in eine Datei – kann dieses dann weiterverarbeitet werden.

```
mysql --execute="SELECT * FROM oshop.kunde" --html > erg.txt
```

Ich möchte allerdings anmerken, dass der erzeugte HTML-Quelltext sehr zu wünschen übrig lässt. Für eine Weiterverarbeitung eignen sich die XML-Ausgaben besser.

Langform	Kurzform
--local-infile	

Aus Gründen der Sicherheit kann es dem Client verboten sein, lokale Dateien für den Import zu verwenden. Durch diesen Parameter kann das Problem behoben werden (siehe Seite 107).

Langform	Kurzform
--no-beep	-b

Schaltet den Beep bei Fehlern aus. Das ist besonders in Schulungsumgebungen sehr sinnvoll. Obwohl ... ich bekomme dadurch genau mit, wann ich im Unterricht weiter machen kann. Wenn keiner mehr beept, sind alle fertig ;-)

Langform	Kurzform
--password[=Passwort]	-p[Passwort]

Dieses Passwort wird für den Verbindungsaufbau verwendet und muss zum angegebenen Benutzer (siehe --user) passen. Wird das Passwort nicht im Parameter mitgegeben, muss dieses über den Prompt eingegeben werden, was aus Sicherheitsgründen sinnvoll sein kann. Wenn Sie das Passwort mit angeben, achten Sie darauf, dass zwischen dem Gleichheitszeichen oder dem p und dem Passwort kein Leerzeichen stehen darf.

Langform	Kurzform
--port=Portnummer	-P Portnummer

Der TCP/IP-Port für die Verbindung (siehe -host). Die Vorbelegung ist 3306. Bitte beachten Sie, dass dieser Port auch durch die Firewall freigegeben sein muss.

Langform	Kurzform
--show-warnings	

Zeigt nach jeder Ausführung Warnungen an, sofern welche auftreten.

Langform	Kurzform
--user=Benutzername	-uBenutzername

Dieser Benutzername wird für den Verbindungsaufbau verwendet. In der Regel erfolgt dies in Kombination mit einer Passworteingabe (siehe -password). Bitte beachten Sie, dass zwischen dem Gleichheitszeichen oder dem u und dem Benutzernamen kein Leerzeichen stehen darf.

Langform	Kurzform
--version	-V

Gibt die Versionsnummer des Clients aus und beendet den Client.

Langform	Kurzform
--xml	-X

Erzeugt eine XML-Ausgabe. Durch Umleiten der Ausgabe – beispielsweise in eine Datei – kann dieses dann weiter verarbeitet werden.

■ 25.2 Befehle

Hier sind die speziellen Befehle des Clients und nicht die SQL-Befehle oder die des Servers gemeint.

Langform	Kurzform
?	\?

Synonym für help.

Langform	Kurzform
clear	\c

Löscht die aktuelle Eingabe. Es kommt vor, dass man sich bei mehrzeiligen Befehlen verfranst. Man kann versuchen, ihn mit einer Fehlermeldung zu beenden oder einfach \c tippen.

Langform	Kurzform
connect	\r

Die Verbindung zum Server wird wieder neu aufgebaut. Falls man durch einen Timeout die Verbindung verloren hat, läuft der Client noch weiter. Für den Wiederaufbau der Verbindung werden die gleichen Parameter (Host, Port, User etc.) verwendet wie beim letzten Aufbau. Optional kann man einen anderen Host und Datenbanknamen mitgeben.

Langform	Kurzform
delimiter	\d

Das normale Trennzeichen für Befehle ist das Semikolon ;. Es kann manchmal nötig sein, das Trennzeichen zu ändern – beispielsweise bei der Definition einer STORED PROCEDURE. Beliebte Alternativen sind: //, # und \$. Vergessen Sie nicht, das Trennzeichen wieder zu ändern; Sie wundern sich sonst, warum Ihre Befehle nicht ausgeführt werden.

Langform	Kurzform
ego	\G

Der Befehl wird zum Server gesendet und das Ergebnis als Liste dargestellt.

```

1 mysql> SELECT nachname, vorname FROM KUNDE\G
2 **** 1. row ****
3 nachname: Beutlin
4 vorname: Bilbo
5 **** 2. row ****
6 nachname: Beutlin
7 vorname: Frodo
8 **** 3. row ****
9 nachname: Elrond
10 vorname:
11 **** 4. row ****
12 nachname: Gamdschie
13 vorname: Samweis
14 **** 5. row ****
15 nachname: Telcontar
16 vorname: Elessar

```

Langform	Kurzform
exit	\q

Synonym für quit

Langform	Kurzform
go	\g

Der Befehl wird zum Server gesendet und das Ergebnis als Tabelle dargestellt.

```

1 mysql> SELECT nachname, vorname FROM KUNDE\g
2 +-----+-----+
3 | nachname | vorname |
4 +-----+-----+
5 | Beutlin   | Bilbo    |
6 | Beutlin   | Frodo    |
7 | Elrond    |          |
8 | Gamdschie | Samweis  |
9 | Telcontar | Elessar  |
10 +-----+-----+

```

Langform	Kurzform
help	\h

Anzeige einer etwas dünnen Hilfe. Diese Hilfe ist nicht mit der Serverhilfe zu verwechseln:

```

1 mysql> help contents
2 You asked for help about help category: "Contents"
3 For more information, type 'help <item>', where <item> is one of the following
      categories:
4   Account Management
5   Administration
6   Compound Statements
7   Data Definition
8   Data Manipulation
9   Data Types
10  Functions
11  Functions and Modifiers for Use with GROUP BY
12  Geographic Features
13  Language Structure
14  Plugins
15  Storage Engines
16  Table Maintenance
17  Transactions
18  User-Defined Functions
19  Utility

```

Diese ist wiederum sehr hilfreich, aber gut versteckt.

Langform	Kurzform
notee	\t

Die Ausgabe kann in eine Datei umgelenkt werden (siehe tee). Mit diesem Befehl schaltet man das Feature wieder ab.

Langform	Kurzform
print	\p

Der aktuelle Befehl wird noch einmal wiederholt:

```

1 mysql> SELECT nachname, vorname FROM KUNDE\p\g
2 -----
3 SELECT nachname, vorname FROM KUNDE
4 -----
5
6 +-----+-----+
7 | nachname | vorname |
8 +-----+-----+
9 | Beutlin  | Bilbo   |
10 | Beutlin  | Frodo   |
11 | Elrond   |          |
12 | Gamdschie| Samweis |
13 | Telcontar| Elessar  |
14 +-----+-----+

```

Langform	Kurzform
prompt	\R

Der MySQL-Prompt wird geändert. Dieses Feature ist genau das richtige für Individualisten oder Witzbolde:

```

1 mysql> prompt :-)
2 PROMPT set to ':-)'
3 :-) SELECT nachname, vorname FROM KUNDE LIMIT 1;
4 +-----+-----+
5 | nachname | vorname |
6 +-----+-----+
7 | Beutlin  | Bilbo   |
8 +-----+-----+
9 1 row in set (0.00 sec)
10
11 :-)

```

Langform	Kurzform
quit	\q

Der Client wird verlassen. Aus gegebenem Anlass: Das ist nicht gleichbedeutend mit dem Herunterfahren des Servers.

Langform	Kurzform
rehash	\#

Die Speicher für die Autovervollständigung wird neu aufgebaut.

Langform	Kurzform
source	\.

Mithilfe dieses Befehls können SQL-Skripte ausgeführt werden. Der Dateiname (optional mit Pfadangabe) wird nach dem Befehl angegeben. Unter Windows haben beide Pfadtrennzeichen \ und / funktioniert; unter Linux nur der Slash /.

Langform	Kurzform
status	\s

Für meine WAMPP-Installation erhalte ich folgende Auskunft:

```

1 mysql> status
2 -----
3 mysql Ver 14.14 Distrib 5.6.28, for Linux (x86_64) using EditLine wrapper
4
5 Connection id:      3
6 Current database:  oshop
7 Current user:       root@localhost
8 SSL:                Not in use
9 Current pager:     less
10 Using outfile:    ''
11 Using delimiter:   ;
12 Server version:  5.6.28 openSUSE package
13 Protocol version: 10
14 Connection:        localhost via UNIX socket
15 Server characterset: utf8
16 Db      characterset: utf8
17 Client characterset: utf8
18 Conn.  characterset: utf8
19 TCP socket:        /var/run/mysql/mysql.sock
20 Uptime:            23 min 59 sec
21
22 Threads: 1 Questions: 9 Slow queries: 0 Opens: 70 Flush tables: 1 Open
          tables: 63 Queries per second avg: 0.06
23 -----

```

Langform	Kurzform
tee	\T

Sämtliche Befehle und Bildschirmausgaben werden in eine Datei geloggt. Der Name und optional der Pfad werden hinter dem Befehl angegeben. Das Logging wird durch den Befehl `notee` wieder ausgeschaltet.

Langform	Kurzform
use	\u

Es wird eine Datenbank ausgewählt. Der Effekt ist folgender:

Ohne `use oshop`: `SELECT * FROM oshop.kunde;`

Mit `use oshop`: `SELECT * FROM kunde;`

Langform	Kurzform
charset	\C

Umstellen auf einen anderen Zeichensatz. Bei Zeichensätzen, die in mehreren Bytes kodiert sind, kann dies für das *binary logging* notwendig sein.

Langform	Kurzform
warnings	\W

Nach jedem Befehl werden ggf. vorhandene Warnungen ausgegeben. Normalerweise wird nur darauf hingewiesen, dass es eine gewisse Anzahl von Warnungen gibt.

Langform	Kurzform
nowarning	\w

Das Gegenteil von `warnings`.

■ 26.1 Datentypen

26.1.1 Numerische Datentypen

26.1.1.1 Ganze Zahlen

Tabelle 26.1 Ganzzahlige Datentypen (* = MySQL proprietär)

Typ	#Bytes	Minimum	Maximum
TINYINT(<i>l</i>)*	1	$-2^7 = -128$	$2^7 - 1 = 127$
TINYINT(<i>l</i>) UNSIGNED*	1	$2^0 - 1 = 0$	$2^8 - 1 = 255$
SMALLINT(<i>l</i>)	2	$-2^{15} = -32.768$	$2^{15} - 1 = 32.767$
SMALLINT(<i>l</i>) UNSIGNED*	2	$2^0 - 1 = 0$	$2^{16} - 1 = 65.535$
MEDIUMINT(<i>l</i>)*	3	$-2^{23} = -8.388.608$	$2^{23} - 1 = 8.388.607$
MEDIUMINT(<i>l</i>) UNSIGNED*	3	$2^0 - 1 = 0$	$2^{24} - 1 = 16.777.215$
INT(<i>l</i>)	4	$-2^{31} = -2.147.483.648$	$2^{31} - 1 = 2.147.483.647$
INT(<i>l</i>) UNSIGNED*	4	$2^0 - 1 = 0$	$2^{32} - 1 = 4.294.967.295$
BIGINT(<i>l</i>)*	8	$-2^{63} \approx -9.223 \times 10^{18}$	$2^{63} - 1 \approx 9.223 \times 10^{18}$
BIGINT(<i>l</i>) UNSIGNED*	8	$2^0 - 1 = 0$	$2^{64} - 1 \approx 18.446 \times 10^{18}$
BOOL*	1	0 (=FALSE)	1 (=TRUE)
ENUM*	2	0	65.535
SET*	1 – 8	-	-
BIT(<i>l</i>)*	1 – 8	0	$2^l - 1$

Bei den Datentypen TINYINT, SMALLINT, MEDIUMINT, INT und BIGINT steht das *l* für die Länge der Ausgabe. Diese Angabe ist optional und kann von der Anwendung verwendet oder ignoriert werden. Sie hat keine Auswirkung auf den Wertebereich.

Der Datentyp ENUM speichert eine Zahl ab. In den internen Infos über eine Tabelle wird eine Liste vorgehalten, die jeder Zahl eine passende Zeichenkette (String) zuordnet. Da ihm 2 Bytes zur Verfügung steht, können maximal 65.535 Zeichenketten verwendet werden. Weil Zeichenketten verwendet werden, kann man diesen einen Zeichensatz und eine Sortierung zuweisen.

Der Datentyp SET weist jedem Element einer Menge eine Zweierpotenz als Wert zu. Der Wert der Spalte in einer Zeile ergibt sich dadurch, welche Elemente der Menge zugewiesen wurden. Als konkrete Werte kann man NULL, einzelne Werte oder eine kommaseparierte Liste von Werten aus der Menge zuweisen. Somit können pro Spalte eine Menge von maximal 64 unterschiedlichen Elementen ($8\text{Bit} \times 8$) definiert werden. In jeder Zeile können einer SET-Spalte eine Werteliste mit maximal 255 Werten zugewiesen werden. Ein Beispiel:

```
mysql> CREATE TEMPORARY TABLE bla
->   (spalte SET('milch', 'kaffee', 'tee', 'wurst', 'marmelade'));

mysql> INSERT INTO bla
->   VALUES
->     ('kaffee,milch')
->     ,('tee,wurst,marmelade') ;

mysql> SELECT * FROM bla;
+-----+
| spalte      |
+-----+
| milch,kaffee |
| tee,wurst,marmelade |
+-----+
```

Der Datentyp BIT wird für das Abspeichern einzelner Bit-Werte verwendet. Dabei gibt l die Anzahl der Bits an. Mit BIT(4) können also vier Bitwerte abgespeichert werden. Maximal ist dabei eine Länge von 64 möglich. Bei der Wertzuweisung wird ein String von 0en und 1en verwendet, denen ein kleines b vorangestellt wird: b'01011'.

26.1.1.2 Gebrochene Zahlen

Tabelle 26.2 Gebrochenzahlige Datentypen (* = MySQL proprietär)

Typ	#Bytes	Minimum	Maximum
FLOAT(l, d)	4	-3.402823466^{+38} $+1.175494351^{-38}$	-1.175494351^{-38} $+3.402823466^{+38}$
REAL	4 8	wie FLOAT	wie FLOAT
REAL(l, d)*	4 8	wie FLOAT	wie FLOAT
DOUBLE(l, d)*	8	-1.7976931348623^{+308} $+2.2250738585072^{-308}$	-2.2250738585072^{-308} $+1.7976931348623^{+308}$
DOUBLE PRECISION	8	wie DOUBLE	wie DOUBLE
DOUBLE PRECISION(l, d)*	8	wie DOUBLE	wie DOUBLE
DECIMAL(l, d)	l	variabel	variabel
NUMERIC(l, d)	l	wie DECIMAL	wie DECIMAL

Alle gebrochenen Zahlen können zusätzlich den Wert 0 annehmen.

Die Angaben l und d sind bei FLOAT, DOUBLE und REAL optional und können von der Anwendung verwendet oder ignoriert werden. Sie haben dort keine Auswirkung auf den Wertebereich.

Der Datentyp REAL ist in MySQL ein Synonym für DOUBLE. Aktivieren Sie den SQL-Modus REAL_AS_FLOAT, so ist er ein Synonym für ein FLOAT. In der SQL:2016-Spezifikation wird

verlangt, dass der REAL einen Wertebereich kleiner als DOUBLE PRECISION hat. Auch in anderen Programmiersprachen (Pascal-Familie) wird ein REAL als ein FLOAT verstanden. Vermeiden Sie den Datentyp FLOAT in MySQL/MariaDB, da die Werte bei Berechnungen in DOUBLE umgewandelt werden. Verwenden Sie lieber direkt DOUBLE oder DECIMAL.

Die Datentypen DECIMAL und NUMERIC sind Festkommazahlen, die l viele Stellen insgesamt umfasst. Der Wertebereich von l liegt zwischen 1 und 65. Die Angabe d legt fest, wie viele Stellen hinter dem Komma gespeichert werden. Der Wertebereich von d ist 0 bis 30. Es muss gelten $l > d$. Anders als bei FLOAT oder DOUBLE bestimmen die Angaben den Wertebereich der Spalte. Innerhalb dieses Wertebereichs sind die Daten exakt und ohne Rundungsfehler. Tatsächlich werden die Daten als Zeichenketten abgespeichert. Es ist allerdings darauf zu achten, dass die Werte, die der Spalte zugewiesen werden, innerhalb der Festlegung liegen, ansonsten werden hier bei der Zuweisung Rundungsfehler erzeugt.

In SQL:2016 besteht ein Unterschied zwischen DECIMAL und NUMERIC. Bei NUMERIC werden exakt d Stellen hinter dem Komma abgespeichert. Bei DECIMAL werden mindestens d Stellen hinter dem Komma abgespeichert. Die Anzahl wird bei Bedarf verlängert und ist nur durch technische Grenzen eingeschränkt. In MySQL/MariaDB sind diese Datentypen Synonyme.

Der wesentliche Unterschied zwischen einem DECIMAL und einem DOUBLE ist, dass der DOUBLE einen großen Wertebereich relativ ungenau abdeckt (Breitenabdeckung) und der DECIMAL innerhalb seines kleineren Wertebereichs exakt ist (Tiefenabdeckung).

26.1.2 Zeichen-Datentypen

Tabelle 26.3 Zeichen-Datentypen (* = MySQL proprietär)

Typ	#Bytes	Maximale Länge
CHAR(l)	l	255
BINARY(l)	l	255
VARCHAR(l)	$l + 1 \mid l + 2$	255 65.535
VARBINARY(l)	$l + 1 \mid l + 2$	255 65.535
TINYTEXT*	$länge + 1$	255
TEXT[/ l]*	$länge + 2$	65.535
MEDIUMTEXT*	$länge + 3$	16.777.215
LONGTEXT*	$länge + 4$	4.294.967.295

Jeder Spalte mit einem Zeichen-Datentyp können ein Zeichensatz (siehe [Abschnitt 28.1 auf Seite 457](#)) und eine Sortierung (siehe [Abschnitt 28.1 auf Seite 457](#)) zugewiesen werden. Ohne Zusätze oder Zuweisung entsprechender Sortierungen wird nicht zwischen Groß- und Kleinschreibung unterschieden.

Bitte beachten Sie, dass bei Zeichensätzen, die mehr als ein Byte pro Zeichen verwenden, die effektive Zeichellänge geringer ist. Ein CHAR(21) kann bei einer utf8-Kodierung im schlechtesten Fall nur sieben Zeichen aufnehmen.

Der Unterschied zwischen CHAR und VARCHAR ist, dass CHAR immer eine gleich lange Zeichenkette abspeichert und VARCHAR nur so viele, wie er braucht. Das Wort *wurstbrot*

würde bei einem VARCHAR(15) nur 10 Zeichen (länge + 1) verbrauchen, bei einem CHAR(15) genau 15.

Dem Datentyp TEXT kann eine Länge mitgegeben werden. Anhand dieser Länge ermittelt MySQL, welcher der Datentypen TINYTEXT, TEXT, MEDIUMTEXT oder LONGTEXT verwendet werden soll.

Will man einen Index auf eine der TEXT-Datentypspalten erstellen, muss eine Präfixlänge angegeben werden. Ansonsten erhält man die Fehlermeldung:

```
ERROR 1170 (42000): BLOB/TEXT column 'spaltenname' used in key specification
without a key length
```

Der Grund ist die Tatsache, dass der Inhalt einer Textspalte ja auch 16 KB groß sein kann. Ein Index über einen solchen Schlüssel ist einfach *Schwachsinn* (siehe Experiment zur Indexselektivität in [Abschnitt 6.2.3 auf Seite 102](#))¹.

Die Datentypen BINARY und VARBINARY unterscheiden sich von CHAR bzw. VARCHAR dadurch, dass die Texte mal als *Zeichenfolge* und mal als *Bytefolge* interpretiert werden. Bei Zeichenfolgen werden die einzelnen Werte anhand ihrer Kodierung und Sortierung mit Zahlenwerten versehen und verarbeitet. Bei Bytefolgen wird jedes Byte als die Zahl interpretiert, die es in seiner Binärdarstellung hat. Daher wird anders sortiert – die *_bin*-Sortierungen der Zeichenkodierungen – und zwischen Groß- und Kleinschreibung unterschieden.

26.1.3 Datums- und Zeit-Datentypen

Datums- und Uhrzeitwerte sind für den Programmierer mühsam und fehlerträchtig. Zum einen gibt es viele verschiedene Kalender: julianisch, gregorianisch, muslimisch, jüdisch, chinesisch etc. Zum anderen gibt es innerhalb der Kalender die Zeitzonen und darüber hinaus noch unterschiedliche Darstellungsformate.

MySQL/MariaDB verwenden den proleptischen gregorianischen Kalender: Beim Wechsel vom julianischen zum gregorianischen Kalender werden einige Tage (z.B. 11) übersprungen. Alle Datumswerte vor diesem Sprung sind julianisch, alle danach gregorianisch. Der proleptische gregorianische Kalender geht vereinfachend davon aus, dass es einen solchen Sprung nie gegeben hat und schon immer der gregorianische Kalender verwendet wurde.

Tabelle 26.4 Datum- und Uhrzeitdatentypen (Auszug, * = MySQL proprietär)

Typ	#Bytes	Minimum	Maximum	Inhalt
YEAR*	1	1901	2155	%Y
YEAR*	1	00	99	%y
DATE	3	1000-01-01	9999-12-31	%Y-%m-%d
TIME	3	00:00:00	23:59:59	%H-%i-%S
TIME	3	-838:59:59	838:59:59	hh - %i - %S
DATETIME*	8	1000-01-01 00:00:00	9999-12-31 23:59:59	%Y-%m-%d %H-%i-%S
TIMESTAMP	8	1970-01-01 00:00:01	2038-01-19 03:14:07	%Y-%m-%d %H-%i-%S

¹ Ich möchte diese Aussage durch den Hinweis auf die Volltextsuche relativieren.

Die Bedeutung der Datumsformatierungszeichen entnehmen Sie der [Tabelle 26.5](#) und die der Uhrzeitformatierungszeichen [Tabelle 26.6 auf der nächsten Seite](#).

- **YEAR:** Die Werte können entweder im Format %y oder %Y eingegeben werden. Bei zweistelligen Jahreszahlen liegt der Wertebereich zwischen (19)70 und (20)69. Bei vierstelligen von 1901 bis 2155.
- **DATE:** Die Werte werden im Format %Y-%m-%d ausgegeben.
- **TIME:** Entweder wird eine Uhrzeit oder eine Zeitdifferenz abgespeichert. Die Zeitdifferenz kann eine dreistellige Stundenangabe haben und auch negative Werte enthalten.
- **DATETIME:** Eine Kombination aus Datum und Uhrzeit. Dieser Datentyp kann auch noch Sekundenbruchteile abbilden. Dazu wird hinter den Sekundenangaben durch einen Punkt getrennt der Sekundenbruchteil angegeben: %Y-%m-%d %H:%i:%S.%f. Die kleinste Angabe ist 10^{-6} sek. Der Speicherplatz wird dabei dynamisch angepasst.
- **TIMESTAMP:** Die Zuweisung eines NULL-Werts setzt das Feld auf die aktuelle **Zeit des Servers**. Ansonsten arbeitet dieser Datentyp wie DATETIME.

Tabelle 26.5 Formatierungszeichen (FZ) für Datumswerte

%FZ	Bedeutung
%a	abgekürzter Wochentag (Son, ..., Sat)
%b	abgekürzter Monat (Jan, ..., Dec)
%c	Monatszahl (0, ..., 12)
%D	Tageszahl des Monats mit englischem Suffix (0th, 1st, 2nd, 3rd, ...)
%d	zweistellige Tageszahl des Monats (00, ..., 31)
%e	Tageszahl des Monats (0, ..., 31)
%j	dreistellige Tageszahl des Jahres (001, ..., 366)
%M	Monatsname (January, ..., December)
%m	zweistellige Monatszahl (00, ..., 12)
%U	zweistellige Wochenzahl (00, ..., 53), wobei Sonntag der erste Wochentag ist
%u	zweistellige Wochenzahl (00, ..., 53), wobei Montag der erste Wochentag ist
%V	zweistellige Wochenzahl (01, ..., 53), wobei Sonntag der erste Wochentag ist; siehe %V
%v	zweistellige Wochenzahl (01, ..., 53), wobei Montag der erste Wochentag ist; siehe %x
%W	Wochentag (Sunday, ..., Saturday)
%w	Tageszahl der Woche (0=Sonntag, ..., 6=Samstag)
%X	vierstellige Jahreszahl einer Woche, wobei Sonntag der erste Wochentag ist; siehe %V
%x	vierstellige Jahreszahl einer Woche, wobei Montag der erste Wochentag ist; siehe %v
%Y	vierstellige Jahreszahl
%y	zweistellige Jahreszahl
%%	das % Zeichen

Tabelle 26.6 Formatierungszeichen (FZ) für Uhrzeitwerte

%FZ	Bedeutung
%f	Mikrosekunden (000000, ..., 999999)
%H	zweistellige Stunde in 24-Stunden-Anzeige (00, ..., 23)
%h	zweistellige Stunde in 12-Stunden-Anzeige (01, ..., 12)
%l	wie %h
%i	zweistellige Minuten (00, ..., 59)
%k	Stunde in 24-Stunden-Anzeige (0, ..., 23)
%l	Stunde in 12-Stunden-Anzeige (1, ..., 12)
%p	AM oder PM
%r	Uhrzeit in 12-Stunden-Anzeige (%h:%m:%S AM oder PM)
%S	zweistellige Sekunden (00, ..., 59)
%s	wie %S
%T	Uhrzeit in 24-Stunden-Anzeige (%H:%m:%S)

MySQL/MariaDB stellen die Funktion `DATE_FORMAT(datum, format)` zur Verfügung. Diese wandelt den Wert `datum` in ein frei definierbares Format um. Das Format wird mithilfe von Formatierungszeichen in `format` festgelegt. Ein Liste der verfügbaren Formatierungszeichen für Datumswerte finden Sie in [Tabelle 26.5 auf der vorherigen Seite](#).

Für die Uhrzeit gibt es die Funktion `TIME_FORMAT(zeit, format)`, welche analog wie `DATE_FORMAT()` arbeitet. Ein Liste der verfügbaren Formatierungszeichen für Uhrzeitwerte finden Sie in [Tabelle 26.6](#).

Einige Formatierungen sind schon passend vordefiniert. Dies betrifft länderübliche Einstellungen oder die ISO 9075. Die vordefinierten Formate können mit `GET_FORMAT(wert, format)` ermittelt werden (siehe [Tabelle 26.7 auf der nächsten Seite](#)).

Ein Beispiel:

```

1 mysql> SELECT
2     -> DATE_FORMAT(datum, GET_FORMAT(DATE, 'ISO')) ISO,
3     -> DATE_FORMAT(datum, GET_FORMAT(DATE, 'EUR')) EUR
4     -> FROM bestellung
5     -> ORDER BY EUR
6     -> LIMIT 6;
7 +-----+-----+
8 | ISO      | EUR      |
9 +-----+-----+
10 | 2012-04-01 | 01.04.2012 |
11 | 2012-05-10 | 10.05.2012 |
12 | 2012-05-11 | 11.05.2012 |
13 | 2012-05-12 | 12.05.2012 |
14 | 2011-01-15 | 15.01.2011 |
15 | 2011-01-16 | 16.01.2011 |
16 +-----+-----+

```

Tabelle 26.7 Vordefinierte Datums- und Uhrzeitformatierungen

Ausrufe	Ergebnis
GET_FORMAT(DATE, 'USA')	'%m.%d.%Y'
GET_FORMAT(DATE, 'JIS')	'%Y-%m-%d'
GET_FORMAT(DATE, 'ISO')	'%Y-%m-%d'
GET_FORMAT(DATE, 'EUR')	'%d.%m.%Y'
GET_FORMAT(DATE, 'INTERNAL')	'%Y%m%d'
GET_FORMAT(DATETIME, 'USA')	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME, 'JIS')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME, 'ISO')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME, 'EUR')	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME, 'INTERNAL')	'%Y%m%d%H%i%s'
GET_FORMAT(TIME, 'USA')	'%h:%i:%s %p'
GET_FORMAT(TIME, 'JIS')	'%H:%i:%s'
GET_FORMAT(TIME, 'ISO')	'%H.%i.%s'
GET_FORMAT(TIME, 'EUR')	'%H.%i.%s'
GET_FORMAT(TIME, 'INTERNAL')	'%H%i%s'

26.1.4 Binäre Datentypen

Binäre Datentypen werden für nicht zeichenbasierte Daten verwendet: Multimedia, pdf-Dateien, ausführbare Dateien, Signaturen etc. Sie können nicht so einfach verarbeitet werden wie die anderen Datentypen. Das Indizieren, Suchen und Ändern ist mit Besonderheiten verbunden. Diese sind für jeden DBMS-Anbieter proprietär, und Sie müssen in den Spezifikationen des Anbieters nachschauen, wie diese verarbeitet werden.

Tabelle 26.8 Binärdatentypen (Auszug, * = MySQL proprietär)

Typ	#Bytes	Kommentar
TINYBLOB*	länge+1	Ein Binärfeld mit einer maximalen Länge von 255 B.
BLOB*	länge+2	Ein Binärfeld mit einer maximalen Länge von 64 KB.
MEDIUMBLOB*	länge+3	Ein Binärfeld mit einer maximalen Länge von 16 MB.
LONGBLOB*	länge+4	Ein Binärfeld mit einer maximalen Länge von 4 GB.

Der Unterschied zwischen TEXT und BLOB ist analog dem von CHAR und BINARY.

In [Abschnitt 7.2.4 auf Seite 116](#) wird gezeigt, wie Sie beispielsweise über C#-API ein Bild einfügen können, und in [Abschnitt 10.6.3 auf Seite 181](#) dann, wie Sie die Bilddaten auslesen und wieder in eine Datei umwandeln.

26.1.5 JSON

Was JSON ist und wie man es verwendet, habe ich schon in [Abschnitt 17.2.1 auf Seite 291](#) beschrieben. Eigentlich könnte man JSON-Dokumente beispielsweise in eine VARCHAR-Spalte speichern. Wird eine Spalte aber mit dem Datentyp JSON deklariert, stehen zwei wichtige Features zur Verfügung²:

- Beim Speichern wird automatisch überprüft, ob es sich um ein gültiges JSON-Dokument handelt. Wenn nicht, wird eine Fehlermeldung erzeugt.

```

1 mysql> CREATE TABLE t (a JSON);
2 Query OK, 0 rows affected (0.47 sec)
3
4 mysql> INSERT INTO t
5 -> SET a = '{"spalte":"wert"}'
6 -> ;
7 Query OK, 1 row affected (0.07 sec)
8
9 mysql> SELECT * FROM t;
10 +-----+
11 | a      |
12 +-----+
13 | {"spalte": "wert"} |
14 +-----+
15 1 row in set (0.01 sec)
16
17 mysql> INSERT INTO t
18 -> SET a = '{"spalte"- "wert"}'
19 -> ;
20 ERROR 3140 (22032): Invalid JSON text: "Missing a colon after a name of object
member." at position 9 in value for column 't.a'.
```

- Intern wird das JSON-Dokument strukturiert abgespeichert, sodass Verarbeitungs- und Auswertungsfunktionen erheblich beschleunigt verarbeitet werden können.

Eine ausführliche Anleitung bzgl. der JSON-Funktionen finden Sie hier: <https://dev.mysql.com/doc/refman/8.0/en/json.html>.

26.1.6 Räumliche Datentypen

Grundlage der räumlichen Datentypen sind die vom *Open Geospatial Consortium (OGC)* in OpenGIS festgelegten Regeln und Normierungen (siehe [[The10](#)]). Unter <https://itnext.io/playing-with-geometry-spatial-data-type-in-mysql-645b83880331> finden Sie eine nette Einführung. Eine tiefergehende Einführung hier verbietet sich leider aus Platzgründen :-(

- **POINT:** Repräsentiert einen zweidimensionalen Punkt. Auf einer Karte kann ein Punkt ein Gebäude oder ein Unternehmen repräsentieren. Zwischen den Werten steht kein Komma: $(x \ y)$.
- **MULTIPOINT:** Eine Folge von Punkten. Auf einer Karte könnten dies die Filialen einer Sparkasse oder alle Haltestellen einer Buslinie sein. Die Punkte werden durch Kommata getrennt: $(x_1 \ y_1, \dots, x_n \ y_n)$.

² In MariaDB ist JSON ab 10.2.7 als Alias für LONGTEXT eingeführt worden, um mit MySQL weiterhin einen Datenaustausch beispielsweise über mysqldump durchführen zu können.

- **LINestring:** Eine durch Punkte definierte interpolierte Kurve. Auf einer Karte könnte dies der Weg einer Buslinie oder eine Stromtrasse sein: $(x_1 \ y_1, \dots, x_n \ y_n)$.
- **MULTILINESTRING:** Eine Sammlung von LINestring-Kurven. Auf einer Karte könnten dies die Routen der Müllabfuhr oder ein Flusssystem sein. Die LINestring-Kurven werden durch runde Klammern definiert: $((x_{11} \ y_{11}, \dots, x_{1n} \ y_{1n}), \dots, (x_{m1} \ y_{m1}, \dots, x_{mn} \ y_{mn}))$
- **POLYGON:** Ein Polygon ist eine geschlossene Fläche, die durch LINestring-Kurven definiert wird. Die Kurven dürfen sich nicht schneiden. In einer Karte können dies die Stadtgrenzen oder die Ufer eines Sees sein.
- **MULTIPOLYGON:** Eine Sammlung von POLYGON-Flächen. Diese müssen schnittmengenfrei sein. In einer Karte könnten dies die landwirtschaftlich genutzten Flächen einer Stadt oder die Fußgängerzonen sein.
- **GEOMETRY:** Dieser Datentyp kann alle Informationen einzelner räumlicher Datenobjekte beinhalten. Der Programmierer muss selbst durch geeignete Funktionen die Informationen für sich aufbereiten.
- **GEOMETRYCOLLECTION:** Dieser Datentyp kann alle Sammlungen von Informationen einzelner räumlicher Datenobjekte beinhalten. Der Programmierer muss selbst durch geeignete Funktionen die Informationen für sich aufbereiten.

26.1.7 Standardwerte

Tabelle 26.9 Datentypen (Auszug): Standardwerte

Typ	Wert	Kommentar
<i>datentyp</i> NULL	NULL	Wenn die Spalte den Wert NULL enthalten darf, wird unabhängig vom Datentyp NULL eingetragen.
<i>datentyp</i> DEFAULT <i>wert</i>	<i>wert</i>	Wenn die Spalte einen Standardwert zugewiesen bekommen hat, wird dieser <i>wert</i> eingetragen.
numerisch	0	Alle numerischen Datentypen werden auf 0 oder 0.0 gesetzt.
Zeichenketten	''	Alle zeichenbasierten Datentypen werden mit der leeren Zeichenkette initialisiert.
Datum	0000-00-00	Dies entspricht der Sekunde 0 am 01.01.1970.
YEAR	0000	Dies entspricht nicht dem Jahr 0.
TIME	00:00:00	Dies entspricht der ersten Sekunde eines Tages.

Wird eine neue Zeile eingefügt und zu einer Spalte keine Angabe gemacht, werden die Standardwerte eingetragen. Wurde ein DEFAULT – beispielsweise beim CREATE TABLE – angegeben, wird dieser verwendet. Ist kein eigener DEFAULT angegeben worden und ist in der Spalte der Wert NULL erlaubt, wird immer NULL verwendet. Ist NULL nicht erlaubt, werden die in [Tabelle 26.9](#) angegebenen Werte eingetragen.

26.1.8 Zusätze für Datentypen

Tabelle 26.10 Zusätze (Auszug, * = MySQL proprietär)

Typ	Kommentar
AUTO_INCREMENT*	Dieser Zusatz ermöglicht einem ganzzahligen Feld die automatische Wertfindung. Dies eignet sich zur Generierung von Schlüsseln. Die Daten können in dieses Feld gelesen und auch geschrieben werden. Wenn aber ein Wert 0 oder NULL zugewiesen wird, wird der nächste Zahlenwert zugewiesen. Dieser Zusatz kann nur einmal pro Tabelle und nur dem Primärschlüssel zugewiesen werden.
BINARY*	Dieser Zusatz kann bei CHAR- und VARCHAR-Typen verwendet werden, um die Beachtung der Groß- und Kleinschreibung zu erzwingen.
CHECK <i>bedingung</i>	Mithilfe der <i>bedingung</i> kann die Domäne (siehe Definition 5 auf Seite 15) festgelegt werden.
DEFAULT <i>wert</i> *	Mit diesem Zusatz können Spalten Vorbelegungen zugewiesen werden. Falls einer Spalte kein Wert zugewiesen wird, wird automatisch der Wert <i>wert</i> zugewiesen.
FOREIGN KEY	Markiert, dass diese Spalte ein Fremdschlüssel ist. Wird in MySQL von MyISAM nicht verarbeitet. InnoDB wertet den Constraint aus.
GENERATED <i>art</i> AS IDENTITY	Dieser Zusatz ermöglicht einem ganzzahligen Feld die automatische Wertfindung. Dies eignet sich zur Generierung von Schlüsseln. Es gibt zwei Varianten: <i>art</i> = ALWAYS erzeugt immer den neuen Wert und akzeptiert keine Werte von außen. <i>art</i> = BY DEFAULT erzeugt nur dann einen Wert, wenn er den Wert 0 oder NULL von außen bekommt (vergleichbar mit AUTO_INCREMENT). Dieser Zusatz kann nur einmal pro Tabelle und nur dem Primärschlüssel zugewiesen werden. Dieser Zusatz wird in dieser Form von MySQL nicht verarbeitet (s.u.).
GENERATED ALWAYS AS PRIMARY KEY*	Dieser Zusatz ermöglicht einem ganzzahligen Feld die automatische Wertfindung. Er erzeugt immer den neuen Wert und akzeptiert keine Werte von außen.
NULL*	Die Spalte darf auch keinen Attributwert haben. Dies ist nicht gleichbedeutend mit der Zahl 0 oder dem Leer-String!
NOT NULL	Spalten mit diesem Zusatz dürfen nicht leer sein.
PRIMARY KEY	Markiert, dass diese Spalte der Primärschlüssel ist.
UNIQUE	Erzwingt die Schlüsseleigenschaft (siehe Definition 8 auf Seite 17) der Spalte.
UNSIGNED*	Bei ganzzahligen Zahlentypen steuert dieser Zusatz, ob nur positive oder auch negative Zahlen in der Spalte abgelegt werden können.

Die Zusätze sind datentypspezifisch.

Bei MySQL/MariaDB gelten folgende Einschränkungen: AUTO_INCREMENT nur bei ganzzahligen Datentypen, BINARY nur bei CHAR oder VARCHAR, DEFAULT nicht bei BLOB oder TEXT und UNSIGNED nur bei numerischen Datentypen. Wann welcher Zusatz möglich ist, entnehmen Sie bitte der Dokumentation Ihres DBMS.

Noch etwas zu GENERATED: Mithilfe dieses Zusatzes können Spalten einer Tabelle definiert werden, die sich aus Auswertungen und Berechnungen ergeben. Natürlich sind diese dann *read only*³. Mithilfe dieser generierten Spalten könnte man häufig gebrauchte Varianten oder Auswertungen schon in die Tabelle programmieren und damit ggf. notwendige An-sichten sparen oder Abfragen vereinfachen:

```

1 MariaDB [test]> CREATE TABLE person (
2             -> vorname VARCHAR(255),
3             -> nachname VARCHAR(255),
4             -> listenname VARCHAR(255) AS (CONCAT(nachname, ', ', vorname))
5             -> );
6 Query OK, 0 rows affected (0.04 sec)
7
8 MariaDB [test]> INSERT INTO person
9             -> SET
10            -> vorname = 'Heinz',
11            -> nachname = 'Bode'
12            -> ;
13 Query OK, 1 row affected (0.00 sec)
14
15 MariaDB [test]> SELECT * FROM person;
16 +-----+-----+
17 | vorname | nachname | listenname |
18 +-----+-----+-----+
19 | Heinz  | Bode    | Bode, Heinz |
20 +-----+-----+-----+
21 1 row in set (0.00 sec)
22
23 MariaDB [test]> UPDATE person
24             -> SET
25             -> listenname = 'abc'
26             -> ;
27 Query OK, 0 rows affected, 1 warning (0.00 sec)
28 Rows matched: 1  Changed: 0  Warnings: 1
29
30 MariaDB [test]> SHOW WARNINGS\G
31 **** 1. row ****
32 Level: Warning
33 Code: 1906
34 Message: The value specified for computed column 'listenname' in table 'person'
            ' ignored
35 1 row in set (0.00 sec)
```

Generierte Spalten lassen gut dafür nutzen, eigentlich etwas sperrige Inhalte – wie beispielsweise TEXT oder JSON – auszuwerten oder zu filtern. Besonders, da Sie diese Spalten indizieren können, ergeben sich viele Möglichkeiten der Performanceverbesserung.

³ Diese Tabelle würde dann nicht mehr den Normalformen entsprechen.

■ 26.2 Operatoren und Funktionen

26.2.1 Mathematische Operatoren

Operator	Name	Beispiel		
-	Unäres Minus	SELECT	-5;	→ -5
*	Multiplikation	SELECT	5*2;	→ 10
/	Division	SELECT	5/2;	→ 2.5
%	Modulo	SELECT	5%2;	→ 1
MOD	Modulo	SELECT	5 MOD 2;	→ 1
DIV	Ganzzahlige Division	SELECT	5 DIV 2;	→ 2
+	Addition	SELECT	-5+2;	→ -3
-	Subtraktion	SELECT	-5-2;	→ -7



Hinweis: Eine Division durch 0 ergibt NULL. Ist einer der Operanden NULL, ist das Ergebnis auch NULL.

26.2.2 Mathematische Funktionen

Funktionen, die mit einem Sternchen (*) versehen sind, sind MySQL-proprietär.

ABS(x)	Absolutwert von x
mysql> SELECT ABS(-5), ABS(0), ABS(5);	+-----+-----+ ABS(-5) ABS(0) ABS(5) +-----+-----+ 5 0 5 +-----+-----+

ACOS(x)	Arcuscosinus von x
mysql> SELECT ACOS(1)\G ACOS(1): 0	

ASIN(x)	Arcussinus von x
mysql> SELECT ASIN(0.5)\G ASIN(0.5): 0.5235987755982989	

ATAN(x)	Arcustangens von x
mysql> SELECT ATAN(1)\G ATAN(1): 0.7853981633974483	

ATAN2(x, y)	Arcustangens von $\frac{x}{y}$
mysql> SELECT ATAN2(1, 0.5)\G	

ATAN2(1, 0.5): 1.1071487177940904

CEILING(x)	Kleinste ganze Zahl $\geq x$
--------------------------------	------------------------------

```
mysql> SELECT CEILING(-3.1), CEILING(-3.9), CEILING(3.1), CEILING(3.9);
+-----+-----+-----+-----+
| CEILING(-3.1) | CEILING(-3.9) | CEILING(3.1) | CEILING(3.9) |
+-----+-----+-----+-----+
|          -3 |         -3 |          4 |          4 |
+-----+-----+-----+-----+
```

*CONV(x,y,z)	Konvertiert x zur Basis y zur Basis z
----------------------------------	---

```
mysql> SELECT CONV(255, 10, 16)\G
CONV(255, 10, 16): FF
```

COS(x)	Cosinus von x
----------------------------	-----------------

```
mysql> SELECT COS(0)\G
COS(0): 1
```

COT(x)	Cotangens von x
----------------------------	-------------------

```
mysql> SELECT COT(1)\G
COT(1): 0.6420926159343308
```

CRC32(<i>ausdruck</i>)	Zyklische Redundanzprüfung von <i>ausdruck</i>
-------------------------------	--

```
mysql> SELECT CRC32('wurstbrot')\G
CRC32('wurstbrot'): 524262623
```

DEGREES(x)	Konvertiert x von Bogenmaß nach Grad
--------------------------------	--

```
mysql> SELECT DEGREES(PI())\G
DEGREES(PI()): 180
```

EXP(x)	e^x
----------------------------	-------

```
mysql> SELECT EXP(1)\G
EXP(1): 2.718281828459045
```

FLOOR(x)	Größte ganze Zahl $\leq x$
------------------------------	----------------------------

```
mysql> SELECT FLOOR(-3.1), FLOOR(-3.9), FLOOR(3.1), FLOOR(3.9);
+-----+-----+-----+-----+
| FLOOR(-3.1) | FLOOR(-3.9) | FLOOR(3.1) | FLOOR(3.9) |
+-----+-----+-----+-----+
|          -4 |         -4 |          3 |          3 |
+-----+-----+-----+-----+
```

FORMAT(x,y)	Formatiert x anhand der Formatangaben in y
---------------------------------	--

```
mysql> SELECT FORMAT(6235217.02562, 2, 'de_DE')\G
FORMAT(6235217.02562, 2, 'de_DE'): 6.235.217,03
```

HEX(*x*)**Liefert die hexadezimale Schreibweise von *x***

```
mysql> SELECT HEX(255)\G
HEX(255): FF
```

LN(*x*)*Logarithmus naturalis von *x***

```
mysql> SELECT LN(2.718281828459045)\G
LN(2.718281828459045): 1
```

LOG(*x,y*)**Logarithmus von *y* zur Basis *x***

```
mysql> SELECT LOG(16, 65536)\G
LOG(16, 65536): 4
```

LOG10(*x*)**Logarithmus zur Basis 10 von *x***

```
mysql> SELECT LOG10(1000)\G
LOG10(1000): 3
```

LOG2(*x*)*Logarithmus zur Basis 2 von *x***

```
mysql> SELECT LOG2(256)\G
LOG2(256): 8
```

MOD(*x,y*)***x* Modulo *y***

```
mysql> SELECT MOD(27, 8)\G
MOD(27, 8): 3
```

PI()**Die Konstante π**

```
mysql> SELECT PI()\G
PI(): 3.141593
```

POWER(*x,y*) **x^y**

```
mysql> SELECT POWER(2,8)\G
POWER(2,8): 256
```

RADIANS(*x*)**Konvertiert *x* von Grad nach Bogenmaß**

```
mysql> SELECT RADIANS(180)\G
RADIANS(180): 3.141592653589793
```

RAND()**Zufallszahl $z \in [0.0, 1.0[$**

```
mysql> SELECT RAND()\G
RAND(): 0.43608484178416884
```

RAND(*x*)**Wie RAND(), aber mit seed *x***

```
mysql> SELECT RAND(1)\G
RAND(1): 0.40540353712197724
```

ROUND(*x,y*)**Rundet *x* auf *y* Stellen**

```
mysql> SELECT ROUND(1.49, 1)\G
ROUND(1.49, 1): 1.5
```

SIGN(*x*)**Vorzeichen von *x***

```
mysql> SELECT SIGN(-5), SIGN(0), SIGN(+5);
+-----+-----+-----+
| SIGN(-5) | SIGN(0) | SIGN(+5) |
+-----+-----+-----+
|      -1 |      0 |      1 |
+-----+-----+-----+
```

SIN(*x*)**Sinus von *x***

```
mysql> SELECT SIN(PI()/2)\G
SIN(PI()/2): 1
```

SQRT(*x*) **\sqrt{x}**

```
mysql> SELECT SQRT(16)\G
SQRT(16): 4
```

TAN(*x*)**Tangens von *x***

```
mysql> SELECT TAN(PI()/4)\G
TAN(PI()/4): 0.999999999999999
```

Das Ergebnis müsste eigentlich 1 lauten. Rundungsfehler ist hier das Problem.

TRUNCATE(*x,y*)**Schneidet *x* nach der *y*ten Stelle ab**

```
mysql> SELECT TRUNCATE(2.367, 2)\G
TRUNCATE(2.367, 2): 2.36
```

26.2.3 Aggregatfunktionen

Die mit einem Sternchen (*) versehenen Funktionen sind MySQL-proprietär.

AVG(*ausdruck*)**Arithmetische Mittel der Werte in *ausdruck***

```
mysql> SELECT AVG(einzelpreis) FROM artikel\G
AVG(einzelpreis): 11.6210000000
```

AVG(DISTINCT *ausdruck*)*Arithmetische Mittel der unterschiedlichen Werte in *ausdruck***

```
mysql> SELECT AVG(DISTINCT einzelpreis) FROM artikel\G
AVG(DISTINCT einzelpreis): 12.4533333333
```

BIT_AND(*ausdruck*)*Liefert die binäre UND-Verknüpfung aller Werte in *ausdruck***

```
mysql> SELECT BIT_AND(artikel_id) FROM artikel WHERE artikel_id % 3 = 0\G
```

```
BIT_AND(artikel_id): 566
```

Was bedeutet das? Alle Werte werden als 64Bit-Zahlen kodiert. Von jeder Zahl wird nun beispielsweise die 5. Position mit allen anderen 5. Positionen der anderen Zahlen per UND verknüpft. Das Ergebnis ist wiederum eine 64Bit-Zahl.

***BIT_OR(ausdruck)**

Liefert die binäre ODER-Verknüpfung aller Werte in *ausdruck*

```
mysql> SELECT BIT_OR(artikel_id) FROM artikel WHERE artikel_id % 3 = 0\G
BIT_OR(artikel_id): 16319
```

***BIT_XOR(ausdruck)**

Liefert die binäre exklusive ODER-Verknüpfung aller Werte in *ausdruck*

```
mysql> SELECT BIT_XOR(artikel_id) FROM artikel WHERE artikel_id % 3 = 0\G
BIT_XOR(artikel_id): 13886
```

COUNT(*)

Anzahl der Zeilen unabhängig vom Inhalt

```
mysql> SELECT COUNT(*) FROM kunde\G
COUNT(*): 5
```

COUNT(ausdruck)

Anzahl der Zeilen, deren *ausdruck* nicht NULL ist

```
mysql> SELECT COUNT(liefer_adresse_id) FROM kunde\G
COUNT(liefer_adresse_id): 2
```

***COUNT(DISTINCT ausdruck)**

Anzahl unterschiedlicher Inhalte in *ausdruck*

```
mysql> SELECT COUNT(DISTINCT rechnung_adresse_id) FROM kunde\G
COUNT(DISTINCT rechnung_adresse_id): 4
```

***GROUP_CONCAT(ausdruck)**

String, der sich aus dem Gruppierungsergebnis von *ausdruck* ergibt

```
mysql> SELECT kunde_id, GROUP_CONCAT(rechnung_id SEPARATOR ',')
      ->   FROM rechnung
      ->   GROUP BY kunde_id
      -> ;
+-----+-----+
| kunde_id | GROUP_CONCAT(rechnung_id SEPARATOR ',') |
+-----+-----+
| 1 | 1;3;4;12 |
| 2 | 2;6 |
| 3 | 7;8 |
| 5 | 9;10;11 |
+-----+-----+
```

***JSON_ARRAYAGG(ausdruck)**

Liefert das Ergebnis der Gruppierung als JSON-Array.

```
mysql> SELECT kunde_id, JSON_ARRAYAGG(rechnung_id)
      ->   FROM rechnung
      ->   GROUP BY kunde_id
      -> ;
+-----+-----+
```

```
+-----+
| kunde_id | JSON_ARRAYAGG(rechnung_id) |
+-----+
| 1 | [1, 3, 4, 12] |
| 2 | [2, 6] |
| 3 | [7, 8] |
| 5 | [9, 10, 11] |
+-----+
```

***JSON_OBJECTAGG(ausdruck)** | Liefert das Ergebnis der Gruppierung als JSON-Objekt.

Dabei werden zwei Spalten oder Ausdrücke verwendet: Der erste ist der Objektname und der zweite liefert die Werte.

```
mysql> SELECT kunde_id, JSON_OBJECTAGG(rechnung_id, kunde_id)
->   FROM rechnung
->   GROUP BY kunde_id
-> ;
+-----+
| kunde_id | JSON_OBJECTAGG(rechnung_id, kunde_id) |
+-----+
| 1 | {"1": 1, "3": 1, "4": 1, "12": 1} |
| 2 | {"2": 2, "6": 2} |
| 3 | {"7": 3, "8": 3} |
| 5 | {"9": 5, "10": 5, "11": 5} |
+-----+
```

MAX(ausdruck) | Liefert den größten Wert in ausdruck

```
mysql> SELECT MAX(einzelpreis) FROM artikel\G
MAX(einzelpreis): 55.700000
```

MIN(ausdruck) | Liefert den kleinsten Wert in ausdruck

```
mysql> SELECT MIN(einzelpreis) FROM artikel\G
MIN(einzelpreis): 0.510000
```

STD(ausdruck)** | Ein Synonym für STDDEV_POP()STDDEV(ausdruck)** | Ein Synonym für STDDEV_POP()**STDDEV_POP(ausdruck)** | $STDDEV_POP(ausdruck) = \sqrt{VAR_POP(ausdruck)}$

```
mysql> SELECT STDDEV_POP(einzelpreis) FROM artikel\G
STDDEV_POP(einzelpreis): 15.7674414221
```

STDDEV_SAMP(ausdruck) | $STDDEV_SAMP(ausdruck) = \sqrt{VAR_SAMP(ausdruck)}$

```
mysql> SELECT STDDEV_SAMP(einzelpreis) FROM artikel\G
STDDEV_SAMP(einzelpreis): 16.6203425891
```

SUM(ausdruck) | Liefert die Summe von ausdruck

```
mysql> SELECT SUM(einzelpreis) FROM artikel\G
SUM(einzelpreis): 116.210000
```

SUM(DISTINCT ausdruck)*Summe unterschiedlicher Werte von ausdruck**

```
mysql> SELECT SUM(DISTINCT einzelpreis) FROM artikel\G
SUM(DISTINCT einzelpreis): 112.080000
```

VAR_POP(ausdruck)

Varianz unter der Annahme, dass alle Elemente der Menge in ausdruck einfließen. Mit anderen Worten: Die Stichprobe enthält die gesamte Menge.

```
mysql> SELECT VAR_POP(einzelpreis) FROM artikel\G
VAR_POP(einzelpreis): 248.6122090000
```

VAR_SAMP(ausdruck)

Varianz unter der Annahme, dass nur eine Teilmenge in ausdruck einfließt. Mit anderen Worten: Die Stichprobe enthält die nicht gesamte Menge.

```
mysql> SELECT VAR_SAMP(einzelpreis) FROM artikel\G
VAR_SAMP(einzelpreis): 276.2357877778
```

VARIANCE(ausdruck)*Synonym für VAR_POP()**

■ 26.3 Bedingungen

26.3.1 Vergleichsoperatoren

Das Ergebnis einer Bedingung ist laut [Definition 34 auf Seite 141](#) TRUE oder FALSE. Kann eine Bedingung nicht ermittelt werden oder ist eine der Teilbedingungen UNKNOWN oder kommt der Wert UNKNOWN in der Bedingung vor, ist das Ergebnis auch UNKNOWN.



Hinweis: Bei Kommazahlen – FLOAT, DOUBLE, DECIMAL – sind Rundungsfehler zu beachten.

Operator	Beispiele
=	wert ₁ = wert ₂ deleted = 0 wert = artikel_id

Liefert TRUE, wenn die rechts und links vom Gleichheitszeichen stehenden Werte gleich sind, sonst FALSE. Haben wert₁ und wert₂ den Wert NULL, liefert der Vergleich ebenfalls NULL. Die Datentypen dürfen unterschiedlich sein. Vorsicht: Anders als bei C/C++ und verwandten Sprachen steht hier nur **ein** Gleichheitszeichen.

Operator	Beispiele
<=>	wert ₁ <=> wert ₂ deleted <=> 0 wert <=> MAX(menge)

Liefert TRUE, wenn die rechts und links von <=> stehenden Werte gleich sind, sonst FALSE. Haben wert₁ und wert₂ den Wert NULL, liefert der Vergleich anders als = den Wert TRUE. Die Datentypen dürfen unterschiedlich sein.

Operator	Beispiele
<code><></code>	<code>wert₁ <> wert₂</code> <code>deleted != 0</code>
<code>!=</code>	

Liefert TRUE, wenn die rechts und links vom Vergleichsoperator stehenden Werte ungleich sind, sonst FALSE. Die Datentypen dürfen unterschiedlich sein.

Operator	Beispiele
<code><</code>	<code>wert₁ < wert₂</code> <code>menge < 10</code>

Liefert TRUE, wenn der linke Wert kleiner als der rechte Wert ist. Bei numerischen Datentypen wird der Zahlenwert als Ordnungskriterium verwendet, bei Texten die festgelegte lexikalische Reihenfolge (siehe Sortierung auf [Seite 70](#)) und nicht die Länge.

Operator	Beispiele
<code><=</code>	<code>wert₁ <= wert₂</code> <code>menge <= 10</code>

Kurzschreibweise von: $((wert_1 < wert_2) \text{ OR } (wert_1 = wert_2))$.

Operator	Beispiele
<code>></code>	<code>wert₁ > wert₂</code> <code>menge > lagerbestand</code>

Liefert TRUE, wenn der linke Wert größer als der rechte Wert ist. Bei numerischen Datentypen wird der Zahlenwert als Ordnungskriterium verwendet, bei Texten die festgelegte lexikalische Reihenfolge (siehe Sortierung auf [Seite 70](#)) und nicht die Länge.

Operator	Beispiele
<code>>=</code>	<code>wert₁ >= wert₂</code> <code>menge >= lagerbestand</code>

Kurzschreibweise von: $((wert_1 > wert_2) \text{ OR } (wert_1 = wert_2))$.

Operator	Beispiele
<code>BETWEEN</code>	<code>wert BETWEEN wert₁ AND wert₂</code> <code>menge BETWEEN 1 AND 100</code>

Liefert TRUE, wenn *wert* in dem Intervall von $(wert_1, wert_2)$ liegt. Kurzschreibweise von: $((wert \geq wert_1) \text{ AND } (wert \leq wert_2))$.

Operator	Beispiele
<code>NOT BETWEEN</code>	<code>wert NOT BETWEEN wert₁ AND wert₂</code> <code>menge NOT BETWEEN 1 AND 100</code>

Liefert TRUE, wenn BETWEEN den Wert FALSE liefert, und umgekehrt.

Operator	Beispiele
<code>IN</code>	<code>wert IN (wert₁, wert₂, ...)</code> <code>plz IN (44879, 44877, 44878)</code>

Liefert TRUE, wenn *wert* in der Werteliste $(wert_1, wert_2, \dots)$ liegt. Kurzschreibweise von: $((wert = wert_1) \text{ OR } (wert = wert_2) \text{ usw.})$.

Operator	Beispiele
<code>NOT IN</code>	<code>wert NOT IN (wert₁, wert₂, ...)</code>

Liefert TRUE, wenn IN den Wert FALSE liefert, und umgekehrt.

Operator	Beispiele
IS TRUE	<i>wert</i> IS TRUE bezahlt IS FALSE
IS FALSE	
IS UNKNOWN	

Ermöglicht den Vergleich mit booleschen Werten. Ist der boolesche Wert nicht ermittelbar, z.B. weil *wert* NULL ist, kann dies über den IS UNKNOWN erkannt werden.

Operator	Beispiele
LIKE	' <i>wert</i> ₁ ' LIKE ' <i>wert</i> ₂ '

Liefert TRUE, wenn der Text *wert*₁ dem Text von *wert*₂ ähnlich ist. Die Ähnlichkeit wird mit *Wildcards* bestimmt. Die Wildcard % steht für jedes Zeichen beliebig (auch 0 mal) oft. Die Wildcard _ steht für jedes beliebige Zeichen genau einmal. Für komplexere Abfragen sollten reguläre Ausdrücke verwendet werden.

Operator	Beispiele
NOT LIKE	' <i>wert</i> ₁ ' NOT LIKE ' <i>wert</i> ₂ '

Liefert TRUE, wenn LIKE den Wert FALSE liefert, und umgekehrt.

Operator	Beispiele
IS NULL	<i>wert</i> IS NULL

Wenn der Inhalt einer Spalte auf NULL überprüft werden soll, können Sie nicht *wert* = NULL schreiben, sondern müssen diese Notation verwenden.

Operator	Beispiele
IS NOT NULL	<i>wert</i> IS NOT NULL

Liefert TRUE, wenn IS NULL den Wert FALSE liefert, und umgekehrt.

26.3.2 Logikoperatoren

26.3.2.1 NOT, Negation, \neg

<i>wert</i>	NOT <i>wert</i> ! <i>wert</i>
UNKNOWN	UNKNOWN
FALSE	TRUE
TRUE	FALSE

Die logische Funktion ist ein *unärer* Operator, d.h., er wird nur mit einem Operanden verwendet. Wann immer eine Bedingung logische Werte liefert, kann mit NOT der logische Ausdruck ins Gegenteil umgewandelt werden. Ausnahme ist der Wert UNKNOWN.

So können beispielsweise die ganzen Tests wie IN, LIKE, BETWEEN etc. leicht erweitert werden: NOT IN, NOT LIKE, NOT BETWEEN.



Hinweis: Im Allgemeinen wird in SQL wegen der besseren Lesbarkeit das NOT dem Ausrufezeichen ! vorgezogen.

26.3.2.2 AND, Konjunktion, \wedge

<i>wert₁</i>	<i>wert₂</i>	<i>wert₁ AND wert₂</i> <i>wert₁ && wert₂</i>
UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	FALSE
UNKNOWN	TRUE	UNKNOWN
FALSE	UNKNOWN	FALSE
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	UNKNOWN	UNKNOWN
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

Mit AND wird überprüft, ob beide Teilbedingungen TRUE sind. Durch Verkettung mit weiteren AND-Teilbedingungen können Sachverhalte abgebildet werden, die mehrere zwingende Voraussetzungen erfüllen müssen.



Hinweis: Im Allgemeinen wird in SQL wegen der besseren Lesbarkeit das AND dem verdoppelten Kaufmannsund && vorgezogen.

Auf [MyS18a] wird ein Versuchsaufbau beschrieben, der nachweist, dass unter MySQL Sprunglogik implementiert ist, durch die man u.U. Rechenzeit einsparen kann.



Definition 74: Sprunglogik

Kann der Wert einer Gesamtbedingung nicht mehr durch die weitere Auswertung seiner Teilbedingungen verändert werden, wird die Auswertung abgebrochen und der aktuelle Wert für den Gesamtausdruck verwendet. Ein solches Vorgehen wird *Sprunglogik* genannt.

Beispiel: Der erste Wert der Teilbedingung einer AND-Verknüpfung ist FALSE, dann können andere Teilbedingungen den Gesamtausdruck nicht zu TRUE werden lassen.



Hinweis: Bei Sprunglogik kann es zu nicht gewollten Nebeneffekten kommen, wenn in einer nicht mehr berücksichtigten Teilbedingung relevante Operationen nicht mehr ausgeführt werden.

26.3.2.3 OR, Disjunktion, \vee

<i>wert₁</i>	<i>wert₂</i>	<i>wert₁ OR wert₂</i> <i>wert₁ wert₂</i>
UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	UNKNOWN
UNKNOWN	TRUE	TRUE
FALSE	UNKNOWN	UNKNOWN
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	UNKNOWN	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

Wie oben muss auch hier Sprunglogik⁴ beachtet werden.

Die OR-Verknüpfung wird immer dann verwendet, wenn schon das Erfüllen einer Teilbedingung für die Gesamtbedingung ausreicht.

26.3.2.4 XOR, Antivalenz, \otimes

<i>wert₁</i>	<i>wert₂</i>	<i>wert₁ XOR wert₂</i>
UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN
FALSE	UNKNOWN	UNKNOWN
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	UNKNOWN	UNKNOWN
TRUE	FALSE	TRUE
TRUE	TRUE	FALSE

Für das XOR gilt folgende Gleichung:

$$wert_1 \otimes wert_2 = (wert_1 \wedge (\neg wert_2)) \vee ((\neg wert_1) \wedge wert_2)$$

Umgangssprachlich kann man ein XOR mit einer *Entweder-oder*-Aussage umschreiben. Nur eine der beiden Teilbedingungen darf TRUE sein, um den Gesamtausdruck TRUE werden zu lassen.

⁴ Siehe [Definition 74 auf der vorherigen Seite](#)

■ 26.4 Befehle

Die hier vorgestellte Referenz orientiert sich am aktuellen MySQL-Sprachstandard (Version 8.0). Es ist mir nicht gelungen, Ihnen hier eine lesbare und effiziente Gegenüberstellung der Sprachstandards zur Verfügung zu stellen. In den vorherigen Kapiteln ist aber bei vielen Befehlen der Unterschied dargestellt worden, sodass ich mich hier mit Verweisen begnügen.

Laut der Dokumentation von MariaDB (siehe [Mar16]) sind die Befehle zwischen MySQL 5.6 und MariaDB 10.0 alle kompatibel und können problemlos zwischen MySQL und MariaDB ausgetauscht werden; ein entsprechender Abgleich mit MySQL 8.0 ist derzeit noch nicht verfügbar. Eine Darstellung entsprechender Unterschiede entfällt dadurch.

26.4.1 Data Definition Language

Die Data Definition Language (DDL) hat die Aufgabe, die Struktur eines Servers, einer Datenbank, einer Tabelle etc. festzulegen, nicht den Inhalt.

ALTER DATABASE

Eigenschaften einer Datenbank ändern

```
ALTER {DATABASE|SCHEMA} datenbankname
  [{DEFAULT] CHARACTER SET [=] zeichensatz]
  [{DEFAULT] COLLATE [=] sortierung]
  [{DEFAULT] ENCRYPTION [=]{'Y'|'N'}}]
;
```

Ändert die Spezifikation einer Datenbank. Den Datenbanknamen kann man nicht ändern, dies erreicht man nur durch Löschen und Neuanlage. Wird die Verschlüsselung geändert, werden nur neu angelegte Tabellen mit dem geänderten Verfahren verschlüsselt. Vorhandene Daten werden nicht migriert. Weitere Hinweise: siehe CREATE SCHEMA auf Seite 425 und Abschnitt 8.1 auf Seite 125.

ALTER EVENT

Eigenschaften eines Events ändern

```
ALTER
  [DEFINER = user]
  EVENT eventname
  [ON SCHEDULE termin]
  [ON COMPLETION {NOT] PRESERVE]
  [RENAME TO eventname_neu]
  [ENABLE|DISABLE|DISABLE ON SLAVE]
  [COMMENT 'kommentar']
  [DO anweisungsblock]
;
```

Ändert die Spezifikation eines Ereignisses. Ebenfalls kann der Anweisungsblock verändert werden. Weitere Hinweise siehe CREATE EVENT auf Seite 422.

ALTER FUNCTION

Eigenschaften einer Funktion ändern

```
ALTER FUNCTION funktionsname
  COMMENT 'kommentar'
  |LANGUAGE SQL
  |{CONTAINS SQL|NO SQL|READS SQL DATA|MODIFIES SQL DATA}
  |SQL SECURITY {DEFINER|INVOKER}
;
```

Hinweise siehe CREATE FUNCTION auf [Seite 423](#).

ALTER INSTANCE

Eigenschaften der MySQL-Instanz ändern

```
ALTER INSTANCE
  ROTATE INNODB MASTER KEY
  |ROTATE BINLOG MASTER KEY
  |RELOAD TLS [NO ROLLBACK ON ERROR]
;
```

Die Option ALTER INSTANCE ROTATE INNODB MASTER KEY löst aus, dass der Verschlüsselungsschlüssel der InnoDB-Tablespaces verändert wird.

Die Option ALTER INSTANCE ROTATE BINLOG MASTER KEY löst aus, dass der Verschlüsselungsschlüssel des Binärlogs verändert wird.

Beide Optionen verwenden einen Schlüsselring als Basis der Veränderung.

Mit ALTER INSTANCE RELOAD TLS werden Systemvariablen der SSL-Umgebung neu eingelesen. Mit der NO ROLLBACK ON ERROR-Option wird gesteuert, ob bei einem Fehler die Änderungen rückgängig gemacht werden oder nicht.

ALTER LOGFILE GROUP

Eigenschaften einer Logdateigruppe ändern

```
ALTER LOGFILE GROUP logdateigruppe
  ADD UNDOFILE 'dateiname'
  |INITIAL_SIZE [=] größe
  |WAIT
  ENGINE [=] engine
;
```

Hinweise siehe CREATE LOGFILE GROUP auf [Seite 424](#).

ALTER PROCEDURE

Eigenschaften einer Prozedur ändern

```
ALTER PROCEDURE prozedurname
  COMMENT 'kommentar'
  |LANGUAGE SQL
  |{CONTAINS SQL|NO SQL|READS SQL DATA|MODIFIES SQL DATA}
  |SQL SECURITY {DEFINER|INVOKER}
;
```

Ändert die Spezifikation einer Prozedur, aber nicht den Anweisungsblock. Dieser muss mit CREATE OR REPLACE verändert werden. Weitere Hinweise siehe CREATE PROCEDURE auf [Seite 424](#).

ALTER SERVER

Eigenschaften eines Servers ändern

```
ALTER SERVER servername
  OPTIONS (option [, option]*)
;
```

Hinweise siehe CREATE SERVER auf [Seite 425](#).

ALTER TABLE

Eigenschaften einer Tabelle ändern

```
ALTER [ONLINE|OFFLINE] [IGNORE] TABLE tabellenname
  [tabellenspezifikation [, tabellenspezifikation]*] [partitionsoption]
;
ALTER [ONLINE|OFFLINE] [IGNORE] TABLE tabellenname
  partitionsoption
;
```

tabellenspezifikation:

tabellenoption

```

ADD [COLUMN] spaltenname spaltendefinition [FIRST|AFTER spaltenname]
|ADD {INDEX|KEY} [indexname][indextyp]
  (indexspaltenname[, indexspaltenname]*) [indexoption]*
|ADD {FULLTEXT|SPATIAL} [INDEX|KEY] [indexname]
  (indexspaltenname[, indexspaltenname]*) [indexoption]*
|ADD [CONSTRAINT [name]] PRIMARY KEY [indextyp]
  (indexspaltenname[, indexspaltenname]*) [indexoption]*
|ADD [CONSTRAINT [name]] UNIQUE [INDEX|KEY] [indexname] [indextyp]
  (spaltenname[, spaltenname]*) [indexoption]*
|ADD FULLTEXT [INDEX|KEY] [indexname]
  (indexspaltenname[, indexspaltenname]*) [indexoption]*
|ADD [CONSTRAINT [name]] FOREIGN KEY [indexname]
  (indexspaltenname[, indexspaltenname]*) referenz
|DROP CHECK checkname
|ALTER CHECK checkname [NOT] ENFORCED
|ALGORITHM [=] {DEFAULT|INSTANT|INPLACE|COPY}
|ALTER [COLUMN] spaltenname {SET DEFAULT literal|DROP DEFAULT}
|ALTER INDEX indexname {VISIBLE|INVISIBLE}
|LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
|CHANGE [COLUMN] spaltenname_alt
  spaltenname_neu spaltendefinition [FIRST|AFTER spaltenname]
|MODIFY [COLUMN] spaltenname spaltendefinition [FIRST|AFTER spaltenname]
|DROP [COLUMN] spaltenname
|DROP PRIMARY KEY
|DROP {INDEX|KEY} indexname
|DROP FOREIGN KEY fk_name
|{DISABLE|ENABLE} KEYS
|MAX_ROWS = anzahl
|RENAME [TO|AS] tabellenname_neu
|RENAME {COLUMN|INDEX|KEY} name_alt TO name_neu
|ORDER BY spaltenname[, spaltenname]*
|CONVERT TO CHARACTER SET zeichensatz [COLLATE sortierung]
|{DEFAULT} CHARACTER SET [=] zeichensatz [COLLATE [=] sortierung]
|{IMPORT|DISCARD} TABLESPACE
|FORCE
|{WITHOUT|WITH} VALIDATION
|ADD PARTITION (partitionsdefinition)
|DROP PARTITION partitionsname
|DISCARD PARTITION {partitionsnamen|ALL} TABLESPACE
|IMPORT PARTITION {partitionsnamen|ALL} TABLESPACE
|TRUNCATE PARTITION {partitionsname|ALL}
|COALESCE PARTITION nummer
|REORGANIZE PARTITION partitionsname INTO (partitionsdefinition)
|ANALYZE PARTITION {partitionsname|ALL}
|CHECK PARTITION {partitionsname|ALL}
|OPTIMIZE PARTITION {partitionsname|ALL}
|REBUILD PARTITION {partitionsname|ALL}
|REPAIR PARTITION {partitionsname|ALL}
|PARTITION BY partitionierungsausdruck
|REMOVE PARTITIONING
|UPGRADE PARTITIONING

```

indexspaltenname:

{spaltenname[(länge)|(ausdruck)]} [ASC|DESC]

indextyp:

USING {BTREE|HASH}

indexoption:

`KEY_BLOCK_SIZE [=] wert
|indextyp
|WITH PARSER parsername
|COMMENT 'kommentar'`

tabellenoption:
siehe CREATE TABLE

partitionsdefinition:
siehe CREATE TABLE

Ändert fast alle Einstellungen, die man bei einer Tabelle machen kann. Weitere Hinweise siehe CREATE TABLE auf [Seite 426](#) und [Abschnitt 8.3 auf Seite 129](#).

ALTER TABLESPACE

Eigenschaften eines Tablespace ändern

```
ALTER [UNDO] TABLESPACE name
  nur bei NDB:
    [ADD|DROP] DATAFILE 'dateiname'
    [INITIAL_SIZE [=] größe]
    [WAIT]
  bei InnoDB und NDB:
    [RENAME TO name]
  nur bei InnoDB:
    [SET {ACTIVE|INACTIVE}]
    [ENCRYPTION [=] {'Y'|'N'}]
  bei InnoDB und NDB:
    [ENGINE [=] engine]
;
```

Hinweise siehe CREATE TABLESPACE auf [Seite 428](#).

ALTER VIEW

Eigenschaften einer Ansicht ändern

```
ALTER
  [ALGORITHM = {UNDEFINED|MERGE|TEMPTABLE}]
  [DEFINER = user]
  [SQL SECURITY {DEFINER|INVOKER}]
  VIEW name [(spaltenname, spaltenname)...]
  AS
    SELECT auswahl
    [WITH [CASCDED|LOCAL] CHECK OPTION]
;
```

Ändert die Spezifikation und die Auswahl der Ansicht. Weitere Hinweise siehe CREATE VIEW auf [Seite 429](#).

CREATE DATABASE

Anlegen einer Datenbank

```
CREATE {DATABASE|SCHEMA} [IF NOT EXISTS] datenbankname
  [[DEFAULT] CHARACTER SET [=] zeichensatz]
  [[DEFAULT] COLLATE [=] sortierung]
  [[DEFAULT] ENCRYPTION [=] {'Y'|'N'}]
;
```

Legt eine neue Datenbank an. Weitere Hinweise siehe [Abschnitt 5.2.2 auf Seite 66](#), zum Löschen DROP SCHEMA auf [Seite 430](#) und zum Ändern ALTER DATABASE auf [Seite 419](#).

CREATE EVENT

Anlegen eines zeitlichen Ereignisses

```
CREATE
  [DEFINER = user]
  EVENT [IF NOT EXISTS] name
```

```

ON SCHEDULE termin
[ON COMPLETION [NOT] PRESERVE]
[ENABLE|DISABLE|DISABLE ON SLAVE]
[COMMENT 'kommentar']
DO
    anweisungsblock
;

termin:
AT zeitpunkt [+ INTERVAL intervall]*  

|EVERY intervall  

[STARTS zeitpunkt [+ INTERVAL intervall]*]  

[ENDS zeitpunkt [+ INTERVAL intervall]*]

intervall:
{YEAR|QUARTER|MONTH|WEEK|DAY|HOUR|MINUTE|SECOND  

|YEAR_MONTH|DAY_HOUR|DAY_MINUTE  

|DAY_SECOND|HOUR_MINUTE|HOUR_SECOND|MINUTE_SECOND}

```

Legt ein zeitgesteuertes Ereignis an. Weitere Hinweise: siehe [Abschnitt 23.1 auf Seite 371](#), zum Löschen `DROP EVENT` auf [Seite 429](#) und zum Ändern `ALTER EVENT` auf [Seite 419](#).

CREATE FUNCTION

Anlegen einer Funktion

```

CREATE
[DEFINER = user]
FUNCTION name ([parameter] [, parameter]*)
RETURNS typ
[option]* anweisungsblock
;

parameter:
[IN|OUT|INOUT] name typ

typ:
jeder gültige MySQL Datentyp

option:
COMMENT 'kommentar'  

|LANGUAGE SQL  

|[NOT] DETERMINISTIC  

|[CONTAINS SQL|NO SQL|READS SQL DATA|MODIFIES SQL DATA}  

|SQL SECURITY {DEFINER|INVOKER}

```

anweisungsblock:
gültige Deklarationen und Anweisungen

Hinweise siehe [Abschnitt 26.2 auf Seite 408](#), zum Löschen `DROP FUNCTION` auf [Seite 429](#) und zum Ändern `ALTER FUNCTION` auf [Seite 419](#).

CREATE INDEX

Anlegen eines Index

```

CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX name
[index_typ]
ON tabellenname (indexspaltenname[, indexspaltenname]*)
[option*]
[algorithmus_typ*]
[lock_typ*]
;

indexspaltenname:
{spaltenname[(länge)|(ausdruck)]} [ASC|DESC]

```

```

index_typ:
  USING {BTREE|HASH}

option:
  KEY_BLOCK_SIZE [=] wert
  |index_typ
  |WITH PARSER parsername
  |COMMENT 'kommentar'
  |{VISIBLE|INVISIBLE}

algorithmus_typ:
  ALGORITHM [=] {DEFAULT|INPLACE|COPY}

lock_typ:
  LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}

```

Legt einen INDEX an. Ein Index kann nicht verändert werden. Er muss gelöscht und neu angelegt werden. Weitere Hinweise siehe [Abschnitt 6.1.2 auf Seite 97](#) und zum Löschen DROP INDEX auf [Seite 429](#).

CREATE LOGFILE GROUP | Anlegen einer Logdateigruppe

```

CREATE LOGFILE GROUP logdateigruppe
  ADD UNDOFILE 'dateiname'
  |INITIAL_SIZE [=] startgröße
  |UNDO_BUFFER_SIZE [=] puffergröße_rückgängig
  |REDO_BUFFER_SIZE [=] puffergröße_wiederholen
  |NODEGROUP [=] knotengruppe_id
  |WAIT]
  |COMMENT [=] kommentar
  ENGINE [=] engine
;

```

Anlegen einer Logdateigruppe, zu denen die Log- oder Datendateien gehören. Diese Gruppe enthält die *Undo*-Datei *dateiname*⁵. Logdateigruppen werden im Zusammenhang mit Clustern verwendet. MySQL unterstützt nur die Engines NDB und NDBCLUSTER. Weitere Hinweise siehe zum Löschen DROP LOGFILE GROUP auf [Seite 430](#) und zum Ändern ALTER LOGFILE GROUP auf [Seite 420](#).

CREATE PROCEDURE | Anlegen einer Prozedur

```

CREATE
  [DEFINER = user]
  PROCEDURE name ([parameter][, parameter]*)
  [option]* anweisungsblock
;

parameter:
  [IN|OUT|INOUT] name typ

typ:
  jeder gültige MySQL Datentyp (siehe Abschnitt 26.1 auf Seite 397)

option:
  COMMENT 'kommentar'
  |LANGUAGE SQL
  |[NOT] DETERMINISTIC
  |{CONTAINS SQL|NO SQL|READS SQL DATA|MODIFIES SQL DATA}
  |SQL SECURITY {DEFINER|INVOKER}

anweisungsblock:

```

⁵ Es gibt also doch ein *Undo!* Finden Sie heraus, wie es funktioniert ;-).

gültige Deklarationen und Anweisungen

Hinweise siehe [Kapitel 20 auf Seite 333](#) und zum Löschen DROP PROCEDURE auf [Seite 430](#).

CREATE SCHEMA

Anlegen einer Datenbank

Siehe CREATE DATABASE auf [Seite 422](#).

CREATE SERVER

Anlegen eines neuen Servers

```
CREATE SERVER server_name
  FOREIGN DATA WRAPPER wrappername
  OPTIONS (option [, option]*)
;

option:
  |HOST 'hostname'
  |DATABASE 'datenbankname'
  |USER 'username'
  |PASSWORD 'passwort'
  |SOCKET 'dateiname'
  |OWNER 'username'
  |PORT nummer
```

Erstellt einen Server für die Engine FEDERATED. Damit können Datenbanken anderer MySQL Server (Engines: MyISAM oder InnoDB) so eingerichtet werden, als ob diese lokal wären.

```
1 CREATE SERVER server_106_test
2   FOREIGN DATA WRAPPER mysql
3   OPTIONS
4   (
5     USER 'root', HOST '192.168.1.106', DATABASE 'test'
6   )
7 ;
```

Auf dieser Serverbindung können nun weitere Befehle abgesetzt werden:

```
1 CREATE TABLE fremdetabelle
2 (
3   s1 INT
4 ) ENGINE=FEDERATED CONNECTION='server_106_test'
5 ;
```

Weitere Hinweise siehe zum Löschen DROP SERVER auf [Seite 430](#) und zum Ändern ALTER SERVER auf [Seite 420](#).

CREATE SPATIAL REFERENCE SYSTEM

Anlegen eines räumlichen Referenzsystems

```
CREATE OR REPLACE SPATIAL REFERENCE SYSTEM
  referenzid attribut[, attribut]*
```

```
CREATE SPATIAL REFERENCE SYSTEM [IF NOT EXISTS]
  referenzid attribut[, attribut]*
```

attribut:

```
{
  NAME name
  |DEFINITION definition
```

```
|ORGANIZATION name IDENTIFIED BY organisationsid
|DESCRIPTION beschreibung
}
```

referenzid, organisationsid: 32-bit unsigned integer

CREATE TABLE

Anlegen einer Tabelle

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tabellenname
  (spalte_eigenschaft[, spalte_eigenschaft]*)  

  [tabellenoptionen]  

  [partitionsoptionen]  

;  

spalte_eigenschaft:  

  spaltenname spaltendefinition  

  [{INDEX|KEY} [indexname] [indextyp]  

   (indexspaltenname[, indexspaltenname]*)[ indexoption]*  

  |{FULLTEXT|SPATIAL} [INDEX|KEY] [indexname]  

   (indexspaltenname[, indexspaltenname]*)[ indexoption]*  

  |[CONSTRAINT [name]] PRIMARY KEY [indextyp]  

   (indexspaltenname[, indexspaltenname]*)[ indexoption]*  

  |[CONSTRAINT [name]] UNIQUE [INDEX|KEY] [indexname] [indextyp]  

   (indexspaltenname[, indexspaltenname]*)[ indexoption]*  

  |[CONSTRAINT [name]] FOREIGN KEY [indexname]  

   (indexspaltenname[, indexspaltenname]*)referenz  

  [check_definition]  

spaltendefinition:  

  datentyp  

  [NOT NULL|NULL] [DEFAULT vorbelegung]  

  [AUTO_INCREMENT] [UNIQUE [KEY]] [[PRIMARY] KEY]  

  [COMMENT kommentar]  

  [COLLATE sortierung]  

  [COLUMN_FORMAT {FIXED|DYNAMIC|DEFAULT}]  

  [STORAGE {DISK|MEMORY}]  

  [referenz]  

  [check_definition]  

datentyp  

  [COLLATE sortierung]  

  [GENERATED ALWAYS] AS (ausdruck)  

  [VIRTUAL|STORED] [NOT NULL|NULL]  

  [UNIQUE [KEY]] [[PRIMARY] KEY]  

  [COMMENT kommentar]  

  [referenz]  

  [check_definition]  

datentyp:  

siehe Abschnitt 26.1 auf Seite 397  

indexspaltenname:  

  {spaltenname( länge)|( ausdruck)} [ASC|DESC]  

indextyp:  

  USING BTREE|HASH  

indexoption:  

  KEY_BLOCK_SIZE [=] wert  

indextyp  

  |WITH PARSER parsername  

  |COMMENT kommentar  

  |{VISIBLE|INVISIBLE}
```

referenz:

```
REFERENCES tabellenname (spaltenname[, spaltenname]*)
  [MATCH FULL|MATCH PARTIAL|MATCH SIMPLE]
  [ON DELETE referenzoption]
  [ON UPDATE referenzoption]
```

referenzoption:

```
RESTRICT|CASCADE|SET NULL|NO ACTION|SET DEFAULT
```

tabellenoptionen:

```
tabellenoption [,] tabellenoption>*
```

tabellenoption:

```
ENGINE [=] engine
|AUTO_INCREMENT [=] wert
|AVG_ROW_LENGTH [=] wert
|[DEFAULT] CHARACTER SET [=] zeichensatz
|CHECKSUM [=] {0|1}
|[DEFAULT] COLLATE [=] sortierung
|COMMENT [=] kommentar
|COMPRESSION [=] {'ZLIB'|'LZ4'|'NONE'}
|CONNECTION [=] verbindungsparameter
|[DATA|INDEX} DIRECTORY [=] absoluter pfad
|DELAY_KEY_WRITE [=] {0|1}
|ENCRYPTION [=] {'Y'|'N'}
|INSERT_METHOD [=] {NO|FIRST|LAST}
|KEY_BLOCK_SIZE [=] wert
|MAX_ROWS [=] wert
|MIN_ROWS [=] wert
|PACK_KEYS [=] {0|1|DEFAULT}
|PASSWORD [=] passwort
|ROW_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}
|STATS_AUTO_RECALC [=] {DEFAULT|0|1}
|STATS_PERSISTENT [=] {DEFAULT|0|1}
|STATS_SAMPLE_PAGES [=] wert
|TABLESPACE tablespacename [STORAGE {DISK|MEMORY}]
|UNION [=] (tabellenname[, tabellenname]*)
```

partitionsoptionen:

```
PARTITION BY
  {[LINEAR] HASH(ausdruck)}
  |[LINEAR] KEY [ALGORITHM={1|2}](spaltenliste)
  |RANGE{(ausdruck)}[COLUMNS (spaltenliste)}
  |LIST{(ausdruck)}[COLUMNS (spaltenliste)}}
[PARTITIONS anzahl]
[SUBPARTITION BY
  {[LINEAR] HASH(ausdruck)}
  |[LINEAR] KEY [ALGORITHM={1|2}](spaltenliste)}
  |[SUBPARTITIONS anzahl]
]
[(partitionsdefinition [, partitionsdefinition]*)]
```

partitionsdefinition:

```
PARTITION partitionsname
  [VALUES
    {LESS THAN {(ausdruck|werteliste)|MAXVALUE}
    |
    IN (werteliste)}]
  [[STORAGE] ENGINE [=] engine]
  |COMMENT [=] kommentar]
  |DATA DIRECTORY [=] datenverzeichnis]
  |INDEX DIRECTORY [=] indexverzeichnis]
  |MAX_ROWS [=] anzahl]
```

```
[MIN_ROWS [=] anzahl]
[TABLESPACE [=] tablespacename]
[(subpartitionsdefinition [, subpartitionsdefinition*)]
```

subpartitionsdefinition:

```
SUBPARTITION name
  [[STORAGE] ENGINE [=] engine]
  [COMMENT [=] kommentar]
  [DATA DIRECTORY [=] datenverzeichnis]
  [INDEX DIRECTORY [=] indexverzeichnis]
  [MAX_ROWS [=] anzahl]
  [MIN_ROWS [=] anzahl]
  [TABLESPACE [=] tablespacename]
```

Weitere Hinweise siehe [Abschnitt 5.3 auf Seite 72](#), zum Ändern ALTER TABLE auf [Seite 420](#) und zum Löschen DROP TABLE auf [Seite 430](#).

CREATE TABLE ...SELECT SELECT-Ergebnis als Tabelle speichern

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tabellename
  (spalte_eigenschaft[, spalte_eigenschaft]*)
  [tabellenoptionen]
  [partitionoptionen]
  selectangabe
;
```

spalte_eigenschaft:

Siehe CREATE TABLE auf [Seite 426](#)

tabellenoptionen:

Siehe CREATE TABLE auf [Seite 426](#)

partitionoptionen:

Siehe CREATE TABLE auf [Seite 426](#)

selectangabe:

[IGNORE|REPLACE] [AS] SELECT *auswahl*

Legt eine neue Tabelle anhand des Ergebnisses einer Auswahl mit SELECT an.

CREATE TABLE ...LIKE Tabellenstruktur kopieren

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tabellename_ziel
  {LIKE tabellename_quelle|(LIKE tabellename_quelle)}
;
```

Legt eine neue leere Tabelle *tabellename_ziel* mit den gleichen Spezifikationen wie *tabellename_quelle* an. Weitere Hinweise siehe [Abschnitt 5.3.7 auf Seite 89](#).

CREATE TABLESPACE Anlegen eines neuen Tablespaces

```
CREATE [UNDO] TABLESPACE tablespacename
  bei InnoDB und NDB:
    ADD DATAFILE 'dateiname'
  nur bei InnoDB:
    [FILE_BLOCK_SIZE = wert]
    [ENCRYPTION [=] {'Y'|'N'}]
  nur bei InnoDB:
    USE LOGFILE GROUP logfilegruppe
    [EXTENT_SIZE [=] erweiterungsgröße]
    [INITIAL_SIZE [=] startgröße]
    [AUTOEXTEND_SIZE [=] automatische_erweiterungsgröße]
    [MAX_SIZE [=] maximalgröße]
```

```
[NODEGROUP [=] knotengruppe_id]
[WAIT]
[COMMENT [=] kommentar]
bei InnoDB und NDB:
ENGINE [=] engine
;
```

Anlegen eines festen Speicherplatzes für Tabellen. Normalerweise wird der Speicherplatz vom Dateisystem automatisch verwaltet. Legt man den Tablespace manuell an, erweitert der sich nicht mehr automatisch, ist aber erheblich schneller im Zugriff. Weitere Hinweise siehe zum Ändern ALTER TABLESPACE auf [Seite 422](#) und zum Löschen DROP TABLESPACE auf [Seite 430](#).

CREATE TRIGGER**Anlegen eines Triggers**

```
CREATE
[DEFINER = user]
TRIGGER name {BEFORE|AFTER} {INSERT|UPDATE|DELETE}
ON tabellenname FOR EACH ROW
[ {FOLLOWS|PRECEDES} ein_anderer_triggername ]
anweisungsblock
```

Hinweise siehe [Kapitel 22 auf Seite 363](#) und zum Löschen DROP TRIGGER auf [Seite 430](#).

CREATE VIEW**Anlegen einer Ansicht**

```
CREATE
[OR REPLACE]
[ALGORITHM = {UNDEFINED|MERGE|TEMPTABLE}]
[DEFINER = user]
[SQL SECURITY {DEFINER|INVOKER}]
VIEW name [(spaltenname[, spaltenname]*)]
AS SELECT auswahl
[WITH [CASCADED|LOCAL] CHECK OPTION]
;
```

Hinweise siehe [Kapitel 16 auf Seite 273](#), zum Ändern ALTER VIEW auf [Seite 422](#) und zum Löschen DROP VIEW auf [Seite 430](#).

DROP DATABASE**Löscht die Datenbank und alle ihre Elemente**

```
DROP DATABASE [IF EXISTS] name;
```

Alle Datenobjekte wie Tabellen, Ansichten etc. werden ohne Rückfrage endgültig gelöscht. Weitere Hinweise siehe CREATE DATABASE auf [Seite 422](#).

DROP EVENT**Löscht ein zeitgesteuertes Ereignis**

```
DROP EVENT [IF EXISTS] name;
```

Hinweise siehe CREATE EVENT auf [Seite 422](#).

DROP FUNCTION**Löscht eine selbsterstellte Funktion**

```
DROP FUNCTION [IF EXISTS] name;
```

Hinweise siehe CREATE FUNCTION auf [Seite 423](#).

DROP INDEX**Löscht einen Index**

```
DROP INDEX name ON tabellenname
[ALGORITHM [=] {DEFAULT|INPLACE|COPY}]
```

```
|LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}];
```

Hinweise siehe CREATE INDEX auf [Seite 423](#).

DROP LOGFILE GROUP

Löscht eine Logdateigruppe

```
DROP LOGFILE GROUP logdateigruppe ENGINE [=] engine;
```

Hinweise siehe CREATE LOGFILE GROUP auf [Seite 424](#).

DROP PROCEDURE

Löscht eine Prozedur

```
DROP PROCEDURE [IF EXISTS] name;
```

Hinweise siehe CREATE PROCEDURE auf [Seite 424](#).

DROP SCHEMA

Löscht die Datenbank und alle ihre Elemente

```
DROP SCHEMA [IF EXISTS] name;
```

Alle Datenobjekte wie Tabellen, Ansichten etc. werden ohne Rückfrage endgültig gelöscht. Weitere Hinweise siehe CREATE SCHEMA auf [Seite 425](#).

DROP SERVER

Löscht einen FEDERATED Server

```
DROP SERVER [IF EXISTS] name;
```

Hinweise siehe CREATE SERVER auf [Seite 425](#).

**DROP SPATIAL REFERENCE
SYSTEM**

Löscht ein räumliches Referenzsystem

```
DROP SPATIAL REFERENCE SYSTEM [IF EXISTS] referenzid;
```

referenzid: 32-bit unsigned integer

Hinweise siehe CREATE SPATIAL REFERENCE SYSTEM auf [Seite 425](#).

DROP TABLE

Löscht eine Tabelle

```
DROP [TEMPORARY] TABLE [IF EXISTS]
tabellename[, tabellename]*  
[RESTRICT|CASCADE]
```

Alle Daten der Tabelle werden ohne Rückfrage endgültig gelöscht. Weitere Hinweise siehe CREATE TABLE auf [Seite 426](#).

DROP TABLESPACE

Löscht einen Tablespace

```
DROP [UNDO] TABLESPACE name ENGINE [=] engine;
```

Hinweise siehe CREATE TABLESPACE auf [Seite 428](#).

DROP TRIGGER

Löscht einen Trigger

```
DROP TRIGGER [IF EXISTS] [datenbankname.]name;
```

Hinweise siehe CREATE TRIGGER auf [Seite 429](#).

DROP VIEW

Löscht eine Ansicht

```
DROP VIEW [IF EXISTS] name[, name]* [RESTRICT|CASCADE];
```

Hinweise siehe CREATE VIEW auf [Seite 429](#).

RENAME TABLE

Ändert einen Tabellennamen

RENAME

TABLE *name_alt* TO *name_neu* [, TABLE *name_alt* TO *name_neu*]*;

Hinweise siehe CREATE TABLE auf [Seite 426](#).

TRUNCATE

Resetet eine Tabelle

TRUNCATE [TABLE] *tabellenname*;

Versetzt die Tabelle in den Startzustand zurück. Die Daten werden gelöscht und der ggf. vorhandene AUTO_INCREMENT-Zähler wird auf 1 gesetzt. Weitere Hinweise siehe [Abschnitt 9.3.6 auf Seite 154](#).

26.4.2 Data Manipulation Language

Mithilfe der Befehle der Data Manipulation Language (DML) werden die Tabelleninhalte ausgewertet oder verändert.

CALL

Aufruf einer Prozedur

CALL *prozedurname*([*parameter*[, *parameter*]*]);

Die Prozedur mit dem Namen *prozedurname* wird mit den angegebenen Parametern aufgerufen. Weitere Hinweise siehe [Kapitel 20 auf Seite 333](#).

DELETE

Inhalte einer Tabelle löschen

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tabellenname [[AS] alias]
      [PARTITION (partitionsname[, partitionsname]*])
      [WHERE bedingung]
      [ORDER BY sortierangabe]
      [LIMIT anzahl];
;
```

Löscht *anzahl* viele Zeilen in der Reihenfolge der *sortierangabe* der Tabelle *tabellenname*, für welche die *bedingung* den Wert TRUE ergibt. Weitere Hinweise siehe [Abschnitt 9.3 auf Seite 150](#).

DELETE

Inhalt einer Tabellenreferenz löschen

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
      tabellenname[.*] [, tabellenname[.*]]*
      FROM tabellenreferenz
      [WHERE bedingung];
;
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
      FROM tabellenname[.*] [, tabellenname[.*]]*
      USING tabellenreferenz
      [WHERE bedingung];
;
```

Hinweise siehe DELETE auf [Seite 431](#).

DO	Ausdrücke auswerten
----	---------------------

`DO ausdruck[, ausdruck]*;`

Mit DO wird der Ausdruck ausgeführt, ohne dass ggf. ermittelte Ergebnisse zurückgeliefert werden. Es handelt sich dabei um eine meist etwas schnellere Ausführung als bei einem SELECT. Fehler werden als Warnung ausgegeben.

INSERT INTO	Zeilen mit VALUES einfügen
-------------	----------------------------

```
INSERT [LOW_PRIORITY|HIGH_PRIORITY] [IGNORE]
[INTO] tabellenname
[PARTITION (partitionsname[, partitionsname]*)]
[spaltenname[, spaltenname]*]
{VALUES|VALUE}
  ((ausdruck|DEFAULT){, {ausdruck|DEFAULT}*})
  [, ((ausdruck|DEFAULT){, {ausdruck|DEFAULT}*})]*
[ON DUPLICATE KEY UPDATE
  spaltenname=ausdruck
  [, spaltenname=ausdruck]*]
;
```

Fügt die Werte hinter VALUES als neue Zeilen in die Tabelle *tabellenname* ein. Wird dabei ein schon vorhandener Schlüssel verwendet, kann man nach ON DUPLICATE KEY UPDATE angeben, wie damit verfahren werden soll.

Diese Version eignet sich gut für das Einfügen vieler Datensätze mit einem Befehl. Weitere Hinweise siehe [Abschnitt 7.2.1 auf Seite 113](#).

INSERT INTO	Zeilen mit SET einfügen
-------------	-------------------------

```
INSERT [LOW_PRIORITY|HIGH_PRIORITY] [IGNORE]
[INTO] tabellenname
[PARTITION (partitionsname[, partitionsname]*)]
SET spaltenname={ausdruck|DEFAULT}{, spaltenname={ausdruck|DEFAULT}}*
[ON DUPLICATE KEY UPDATE
  spaltenname=ausdruck
  [, spaltenname=ausdruck]*]
;
```

Fügt eine neue Zeile in die Tabelle *tabellenname* ein. Dabei werden den Spaltennamen hinter dem SET die Werte zugewiesen. Wird dabei ein schon vorhandener Schlüssel verwendet, kann man nach ON DUPLICATE KEY UPDATE angeben, wie damit verfahren werden soll.

Diese Version eignet sich gut für das Einfügen vieler Datensätze mit einem Befehl. Weitere Hinweise siehe [Abschnitt 7.2.2 auf Seite 114](#).

INSERT INTO	Zeilen mit SELECT einfügen
-------------	----------------------------

```
INSERT [LOW_PRIORITY|HIGH_PRIORITY] [IGNORE]
[INTO] tabellenname
[PARTITION (partitionsname[, partitionsname]*)]
SELECT auswahl
[ON DUPLICATE KEY UPDATE
  spaltenname=ausdruck
  [, spaltenname=ausdruck]*]
;
```

Fügt das Ergebnis der *auswahl* als neue Zeilen der Tabelle *tabellenname* hinzu. Wird dabei ein schon vorhandener Schlüssel verwendet, kann man nach ON DUPLICATE KEY UPDATE angeben, wie damit verfahren werden soll.

Diese Version eignet sich gut für das Einfügen vieler Datensätze mit einem Befehl. Weitere Hinweise siehe [Abschnitt 7.3 auf Seite 120](#).

LOAD DATA | Daten aus einer CSV-Datei einlesen

```
LOAD DATA [LOW_PRIORITY|CONCURRENT] [LOCAL] INFILE 'dateiname'
[REPLACE|IGNORE]
INTO TABLE tabellenname
[PARTITION (partitionsname[, partitionsname]*)]
[CHARACTER SET zeichensatz]
[FIELDS|COLUMNS]
[TERMINATED BY 'zeichenkette']
[[OPTIONALLY] ENCLOSED BY 'zeichen']
[ESCAPED BY 'zeichen']
]
[LINES
[STARTING BY 'zeichenkette']
[TERMINATED BY 'zeichenkette']
]
[IGNORE anzahl {LINES|ROWS}]
[({spaltennamen|ausdruck} [{spaltennamen|ausdruck}]]*)]
[SET spaltenname=ausdruck[, spaltenname=ausdruck]*]
;
```

Lädt die Daten aus einer CSV-Datei in eine Tabelle. Weitere Hinweise siehe [Abschnitt 7.1 auf Seite 105](#).

LOAD XML | Daten aus einer XML-Datei einlesen

```
LOAD XML [LOW_PRIORITY|CONCURRENT] [LOCAL] INFILE 'dateiname'
[REPLACE|IGNORE]
INTO TABLE [datenbankname.]tabellenname
[PARTITION (partitionsname[, partitionsname]*)]
[CHARACTER SET zeichensatz]
[ROWS IDENTIFIED BY '<tagname>']
[IGNORE anzahl {LINES|ROWS}]
[({spaltennamen|ausdruck} [{spaltennamen|ausdruck}]]*)]
[SET spaltenname=ausdruck[, spaltenname=ausdruck]*]
;
```

Lädt Daten aus einer XML-Datei in eine Tabelle. Weitere Hinweise siehe [Abschnitt 20.5 auf Seite 353](#).

SELECT | Daten auswählen

```
SELECT
[ALL|DISTINCT|DISTINCTROW]
[HIGH_PRIORITY]
[STRAIGHT_JOIN]
[SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
[SQL_CACHE] [SQL_CALC_FOUND_ROWS]
{*[spaltenliste|ausdruck]}
[FROM {tabellenreferenzen|tabellenname}
[PARTITION (partitionsname[, partitionsname]*)]
[WHERE bedingung]
[GROUP BY
{spaltenname|ausdruck|position} [ASC|DESC]
[, {spaltenname|ausdruck|position} [ASC|DESC]]*
[WITH ROLLUP]]
[HAVING bedingung]
[WINDOW fenstername AS (fensterdefinition) [, fenstername AS (fensterdefinition)]*]
```

```
[ORDER BY
  {spaltenname|ausdruck|position} [ASC|DESC]
  [, {spaltenname|ausdruck|position} [ASC|DESC]]*
  [WITH ROLLUP]]
[LIMIT {[offset,] anzahl|anzahl OFFSET offset}]
[INTO OUTFILE 'dateiname'
  [CHARACTER SET zeichensatz]
  exportoptionen
  INTO {DUMPFILE 'dateiname'|variable[, variable]*}]
[FOR {UPDATE|SHARE} {OF tabellenname [, tabellenname]*}
  {NOWAIT|SKIP LOCKED}|LOCK IN SHARE MODE]]
;
```

Wählt Zeilen aus einer Tabelle *tabellenname* oder *tabellenreferenz* aus. Eine Tabellenreferenz kann beispielsweise das Ergebnis eines JOIN sein. Die Spalten können dabei mit Konstanten und Ausdrücken aufbereitet werden. Es werden nur die Zeilen ausgewählt, für welche die *bedingung* der WHERE-Klausel den Wert TRUE ergibt. Weitere Hinweise siehe die Kapitel im Teil [IV auf Seite 155](#).

tabellenreferenzen

Referenzen auf Tabellen erstellen

tabellenreferenzen:
*erweiterte_tabellenreferenz[, erweiterte_tabellenreferenz]**

erweiterte_tabellenreferenz:
tabellenreferenz{0J tabellenreferenz}

tabellenreferenz:
tabellenfaktor|verknüpfung

tabellenfaktor:
tabellenname [PARTITION(partitionsnamen)] [[AS] alias] [indexhinweise]
[unterabfrage [AS] alias [(spaltenliste)]
|(tabellenreferenzen)

verknüpfung:
tabellenreferenz [INNER|CROSS] JOIN tabellenfaktor [verknüpfungsbedingung]
|tabellenreferenz STRAIGHT_JOIN tabellenfaktor [verknüpfungsbedingung]
|tabellenreferenz {LEFT|RIGHT} [OUTER] JOIN tabellenreferenz verknüpfungsbedingung
|tabellenreferenz NATURAL [INNER|{LEFT|RIGHT} [OUTER]] JOIN tabellenfaktor

verknüpfungsbedingung:
ON bedingung|USING (spaltenliste)

indexhinweise:
*indexhinweis [, indexhinweis]**

indexhinweis:
USE {INDEX|KEY}
[FOR {JOIN|ORDER BY|GROUP BY}] ((indexname[, indexname]))*
{IGNORE|FORCE} {INDEX|KEY}
[FOR {JOIN|ORDER BY|GROUP BY}] (indexname[, indexname]))*

Tabellenreferenzen sind in der Regel keine Tabellen, die mit CREATE angelegt wurden, sondern solche, die dynamisch bei der Ausführung von Anweisungen entstehen.

UNION**Ergebnisse von Abfragen vereinigen**

```
SELECT auswahl
UNION [ALL|DISTINCT]
SELECT auswahl
[UNION [ALL|DISTINCT]
SELECT auswahl]*
```

;

Hinweise siehe [Abschnitt 14.1 auf Seite 251.](#)

UPDATE**Inhalte einer Tabelle ändern**

```
UPDATE [LOW_PRIORITY] [IGNORE] tabellenreferenz
SET
  spaltenname={ausdruck|DEFAULT}
  [, spaltenname={ausdruck|DEFAULT}]*
  [WHERE bedingung]
  [ORDER BY sortierangabe]
  [LIMIT anzahl]
;
```

Hinweise siehe [Abschnitt 9.2 auf Seite 146.](#)

UPDATE**Inhalte einer Tabellenreferenz ändern**

```
UPDATE [LOW_PRIORITY] [IGNORE] tabellenreferenz
SET
  spaltenname={ausdruck|DEFAULT}
  [, spaltenname={ausdruck|DEFAULT}]*
  [WHERE bedingung]
;
```

Hinweise siehe [Abschnitt 9.2 auf Seite 146.](#)

WITH**Abfragen mit Common Table Expressions**

cte_ausdruck:
WITH [RECURSIVE]
 cte_name [(spaltenliste)] AS (*unterabfrage*)
 [cte_name [(spaltenliste)] AS (*unterabfrage*)]*
anweisung;

Syntaktisch sind CTE etwas schwierig zu beschreiben. Sie werden der entsprechenden SELECT-, INSERT-, UPDATE- oder DELETE-Anweisung direkt – ohne Semikolon – vorangestellt und so weiter verwendet, als wäre das Ergebnis des CTE eine Tabelle.

26.4.3 Benutzerverwaltung

ALTER USER**Benutzer ändern**

```
ALTER USER [IF EXISTS]
  benutzerspezifikation [, benutzerspezifikation]*
  [REQUIRE {NONE|verschlüsselung [[AND] verschlüsselung]*}]
  [WITH resourceoption[ resourceoption]*]
  [passwortoption|sperroption]
```

ALTER USER**Benutzerrolle ändern**

```
ALTER USER [IF EXISTS] user DEFAULT ROLE {NONE|ALL|benutzerrolle[, benutzerrolle]}
```

ALTER USER**Authentifizierung des aktuellen Benutzers ändern**

```
ALTER USER [IF EXISTS] USER() benutzerauthentifizierungoption
```

CREATE ROLE**Benutzerrolle anlegen**

```
CREATE ROLE [IF NOT EXISTS] rollename[, rollename]*
```

Über den frei wählbaren Rollennamen können Kataloge von Benutzerrechten einem Benutzer zugewiesen werden (siehe GRANT auf Seite 436 und REVOKE auf Seite 437).

CREATE USER**Benutzer anlegen**

```
CREATE USER benutzerspezifikation[, benutzerspezifikation]*
```

Hinweise siehe [Abschnitt 24.2 auf Seite 379](#).

DROP ROLE**Benutzerrolle löschen**

```
DROP ROLE [IF EXISTS] rollename[, rollename]*
```

DROP USER**Benutzer löschen**

```
DROP USER [IF EXISTS] benutzername[, benutzername]*
```

Hinweise siehe [Abschnitt 24.2 auf Seite 379](#).

GRANT**Benutzer Rechte zuweisen**

GRANT

benutzerrecht [(*spaltenliste*)][, *benutzerrecht* [(*spaltenliste*)]]*

ON [*objekttyp*] *privilegtiefe*

TO {*benutzername*|*rollename*} [, {*benutzername*|*rollename*}]*

[WITH GRANT OPTION]

[AS *benutzername*]

[WITH ROLE

DEFAULT

|NONE

|ALL

|ALL EXCEPT *rollename*[, *rollename*]*

|*rollename*[, *rollename*]*

]

]

;

Hinweise siehe [24.2 auf Seite 379](#).

GRANT PROXY**Benutzer über Proxy Rechte zuweisen**

```
GRANT PROXY ON {vorlage_benutzername|vorlage_rollename}
```

```
TO {benutzername|rollename} [, {benutzername|rollename}]* [WITH GRANT OPTION]
```

;

Weist einem Benutzernamen oder einer Benutzerrolle die Rechte der Vorlage *vorlage_** zu.

GRANT**Benutzerrolle Rechte zuweisen**

```
GRANT rollename [, rollename]*  
TO {benutzername|rollename} [, {benutzername|rollename}]* [WITH ADMIN OPTION]  
;
```

RENAME USER**Benutzer umbenennen**

```
RENAME USER  
benutzername_alt TO benutzername_neu  
[, benutzername_alt TO benutzername_neu]*  
;
```

REVOKE**Benutzer ein Recht entziehen**

```
REVOKE  
benutzerrecht [(spaltenname[, spaltenname]*)]  
[, benutzerrecht [(spaltenname[, spaltenname]*)]]*  
ON [objekttyp] privilegtiefe  
FROM {benutzername|rollename}[, {benutzername|rollename}]*  
;
```

Hinweise siehe [Abschnitt 24.2 auf Seite 379](#).

REVOKE ALL**Benutzer alle Rechte entziehen**

```
REVOKE ALL [PRIVILEGES] GRANT OPTION  
FROM {benutzername|rollename}[, {benutzername|rollename}]*  
;
```

Hinweise siehe [Abschnitt 24.2 auf Seite 379](#).

REVOKE PROXY**Benutzer Rechte über einen Proxy entziehen**

```
REVOKE PROXY ON {benutzername|rollename}  
FROM {benutzername|rollename}[, {benutzername|rollename}]*  
;
```

REVOKE benutzerrolle**Benutzerrollen Rechte entziehen**

```
REVOKE rollename[, rollename]*  
FROM {benutzername|rollename}[, {benutzername|rollename}]*  
;
```

SET PASSWORD**Ein neues Passwort vergeben**

```
SET PASSWORD [FOR benutzername] = 'passwort'  
[REPLACE 'passwort']  
[RETAIN CURRENT PASSWORD]  
;
```

benutzerspezifikation**Festlegen der Benutzereigenschaften**

benutzerspezifikation:

```
benutzername {  
IDENTIFIED BY 'passwort' [REPLACE 'passwort'] [RETAIN CURRENT PASSWORD]  
| IDENTIFIED WITH authentifizierungsplugin  
| IDENTIFIED WITH authentifizierungsplugin BY 'passwort'  
| [REPLACE 'passwort'] [RETAIN CURRENT PASSWORD]  
| IDENTIFIED WITH authentifizierungsplugin AS 'passwort'  
| DISCARD OLD PASSWORD  
}
```

Hinweise siehe [Abschnitt 24.2 auf Seite 379](#).

benutzerauthentifizierungs-	Optionen der Funktion zur Benutzerauthentifizierung
------------------------------------	--

```
benutzerauthentifizierungsoption:
  IDENTIFIED BY 'passwort'
    [REPLACE 'passwort'] [RETAIN CURRENT PASSWORD]
  |DISCARD OLD PASSWORD
}
```

grantoption	Optionen eines Rechts
--------------------	------------------------------

```
grantoption:
  GRANT OPTION|ressourceoption
```

Hinweise siehe [Abschnitt 24.2 auf Seite 379](#).

sperroption	Angaben zur Sperrung (lock)
--------------------	------------------------------------

```
sperroption:
  ACCOUNT LOCK|ACCOUNT UNLOCK
```

objektyp	Objekt eines Rechts
-----------------	----------------------------

```
objektyp:
  TABLE|FUNCTION|PROCEDURE
```

Hinweise siehe [Abschnitt 24.2 auf Seite 379](#).

passwortoption	Optionen eines Passworts
-----------------------	---------------------------------

```
passwortoption:
  PASSWORD EXPIRE [DEFAULT|NEVER|INTERVAL n DAY]
  |PASSWORD HISTORY [DEFAULT]n
  |PASSWORD REUSE INTERVAL [DEFAULT]n DAY}
  |PASSWORD REQUIRE CURRENT [DEFAULT|OPTIONAL]
```

ressourceoption	Optionen einer Ressource
------------------------	---------------------------------

```
ressourceoption:
  MAX_QUERIES_PER_HOUR anzahl
  |MAX_UPDATES_PER_HOUR anzahl
  |MAX_CONNECTIONS_PER_HOUR anzahl
  |MAX_USER_CONNECTIONS anzahl
```

Hinweise siehe [Abschnitt 24.2 auf Seite 379](#).

privilegtiefe	Ebene, auf der das Recht angewendet wird
----------------------	---

```
privilegtiefe:
  *
  |*.*|
  |datenbankname.*
  |datenbankname.tabellenname
  |tabellenname
  |datenbankname.{funktionsname|procedurename}
```

Weitere Hinweise siehe [Abschnitt 24.2 auf Seite 379](#).

verschlüsselung	Angaben zur Verschlüsselung
------------------------	------------------------------------

```
verschlüsselung:
  SSL|X509|CIPHER 'cipher'|ISSUER 'issuer'|SUBJECT 'subject'
```

27

Ausgewählte Quelltexte



Hinweis: Wenn man im MySQL oder MariaDB Client eine Abfrage ausführt, bekommt man immer eine Ausführungszeit angegeben. Das ist nicht die hier ermittelte. Diese Information liegt nicht auf dem Server, sondern wird vom Client als Zeitraum vom Start bis zum Ende der Ausführung selbst ermittelt.

■ 27.1 DOUBLE versus DECIMAL



Laden Sie dieses Experiment mit SOURCE perfDecimal01.sql.

```
1 -- START Bereitstellen der Testumgebung
2 DROP DATABASE IF EXISTS perfDecimal01;
3 CREATE DATABASE perfDecimal01;
4 USE perfDecimal01;
5
6 -- Tabellen fuer die Testdaten
7 CREATE TABLE testdaten
8 (
9     spalte_double  DOUBLE,
10    spalte_decimal DECIMAL(30,15)
11 );
12
13 CREATE TABLE testdaten_quelle
14 (
15     spalte_double  DOUBLE,
16    spalte_decimal DECIMAL(30,15)
17 );
18
19 -- Tabelle fuer die Messergebnisse
20 CREATE TABLE messung
21 (
22     anzahl_ds BIGINT  UNSIGNED,
23     sekunden_double  DECIMAL(12,8),
24     sekunden_decimal DECIMAL(12,8)
```

```

25 );
26 -- ENDE Bereitstellen der Testumgebung

```

Damit das Experiment garantiert unter den gleichen Datenbedingungen startet, wird die dazugehörige Datenbank auf jeden Fall gelöscht und wieder neu angelegt. Für die Testdaten verwende ich zwei Tabellen: testdaten und testdaten_quelle. Zuerst werden – beispielsweise 10000 – Zufallsdaten in die Tabelle testdaten_quelle eingefügt. Später werden diese Daten in die Tabelle testdaten kopiert. Der Grund für dieses Vorgehen ist, dass das Einfügen eines Datensatzes mit `INSERT INTO ... VALUES` sehr langsam bezogen auf `INSERT INTO ... SELECT` ist.

Die Tabelle messung enthält pro Zeile eine Zeitmessung. Dabei wird die Dauer in Anzahl der Datensätze und die jeweilige Dauer in Sekunden gemessen.

```

28 -- START Einfuegen von Testdaten
29 DELIMITER //
30 CREATE PROCEDURE testdaten_insert
31 (
32     IN iAnzahl INT -- Anzahl der einzufuegenden Testdaten
33 )
34 BEGIN
35     DECLARE vAnzahl    INT DEFAULT 0; -- Schleifenvariable
36     DECLARE vDouble   DOUBLE;
37     DECLARE vDecimal  DECIMAL(30,15);
38
39     WHILE vAnzahl < iAnzahl DO          -- Schleifenbedingung
40         SELECT RAND() * FLOOR(-10000 + (RAND() * 10000)) -- Zufallswert
41             INTO vDecimal;
42         SET vDouble = vDecimal;
43         INSERT INTO testdaten_quelle VALUES(vDouble, vDecimal); -- Einfuegen
44         SET vAnzahl = vAnzahl + 1;      -- Schleifeninkrement
45     END WHILE;
46 END//
47 DELIMITER ;
48 -- ENDE Einfuegen von Testdaten

```

Die Prozedur testdaten_insert baut den Bestand der Tabelle testdaten_quelle auf. Die Anzahl der Zufallswerte wird als Übergabeparameter festgelegt. Passend dazu wird in [Zeile 35](#) eine Schleifenvariable deklariert. Diese startet mit 0 (DEFAULT) und wird bei jedem Schleifendurchlauf um 1 in [Zeile 44](#) erhöht. In [Zeile 39](#) werden beide Variablen in der Schleifenbedingung verwendet.

In [Zeile 40](#) werden Zufallswerte erzeugt. Der dabei verwendete Ausdruck liefert Zufallswerte im Bereich von -10000 bis fast 0. Entscheidend ist, dass Nachkommastellen unterschiedlicher Länge erzeugt werden. Die werden dann in [Zeile 43](#) in die Tabelle eingefügt.

```

51 -- Messung
52 DELIMITER //
53 CREATE PROCEDURE perfDecimal01
54 (
55     IN iAnzahlNeueZeilen  INT,    -- Anzahl der neuen Datensaetze pro Messung
56     IN iAnzahlSchleifen   INT     -- Anzahl der Messschleifen
57 )
58 BEGIN
59     DECLARE vAnzahlSchleifen INT DEFAULT 0; -- Schleifenvariable
60     DECLARE vSekundenDouble DECIMAL(12,8); -- Zeitdauer
61     DECLARE vSekundenDecimal DECIMAL(12,8); -- Zeitdauer
62     DECLARE vAnzahlDS        BIGINT; -- Anzahl der Testdaten

```

```

63  DECLARE vTmpDouble      DOUBLE; -- Hilfsvariable
64  DECLARE vTmpDec        DECIMAL(30,15); -- Hilfsvariable

```

Die Prozedur `perfDecimal01` implementiert die eigentliche Messung. Die beiden Über-gabeparameter bestimmen zum einen die Anzahl der Erhöhung der Testdaten pro Schleifendurchlauf. Das ist gleichbedeutend mit der Anzahl der Zufallswerte in `testdaten_quelle`. Zum anderen wird die Anzahl der Messungen festgelegt. Deshalb gibt es passend zu `iAnzahlSchleifen` die Schleifenvariable `vAnzahlSchleifen`.

Die Variablen `vSekundenDouble` und `vSekundenDecimal` nehmen die Ausführungszeit der jeweiligen Berechnung auf. `vAnzahlDS` wird verwendet, um sich die Anzahl der Datensätze in der Tabelle `testdaten` zu merken, damit diese in die Tabelle `messung` eingefügt werden kann. Die beiden `vTmp`-Variablen sind funktionslos und werden nur gebraucht, um sinnlose Bildschirmausgaben zu verhindern.

```

66  TRUNCATE messung;          -- Alte Daten entfernen
67  TRUNCATE testdaten;        -- Alte Daten entfernen
68  TRUNCATE testdaten_quelle; -- Alte Daten entfernen

```

Die Tabellen werden geleert. Somit ist sichergestellt, dass bei jedem Messstart alles bei 0 beginnt.

```

70  SET PROFILING = 1;          -- Profiling einschalten
71
72  CALL testdaten_insert(iAnzahlNeueZeilen); -- Testdatenquelle aufbauen
73  WHILE vAnzahlSchleifen < iAnzahlSchleifen DO -- Hauptschleife
74    SELECT 'Schleifendurchlauf ', vAnzahlSchleifen, ' von ', iAnzahlSchleifen;
75    INSERT INTO testdaten SELECT * FROM testdaten_quelle; -- Testdaten erweitern
76    -- START Messung DOUBLE
77    SELECT SQL_NO_CACHE AVG(spalte_double)  FROM testdaten INTO vTmpDouble;
78    SELECT SUM(DURATION) FROM INFORMATION_SCHEMA.PROFILING
79    GROUP BY query_id
80    ORDER BY query_id DESC LIMIT 1
81    INTO vSekundenDouble;
82    -- ENDE Messung DOUBLE
83
84    -- Start Messung DECIMAL
85    SELECT SQL_NO_CACHE AVG(spalte_decimal) FROM testdaten INTO vTmpDec;
86    SELECT SUM(DURATION) FROM INFORMATION_SCHEMA.PROFILING
87    GROUP BY query_id
88    ORDER BY query_id DESC LIMIT 1
89    INTO vSekundenDecimal;
90    -- Ende Messung DECIMAL
91
92    -- Sichern der Ergebnisse
93    SELECT COUNT(*) FROM testdaten INTO vAnzahlDS;
94    INSERT INTO messung
95      (anzahl_ds, sekunden_double, sekunden_decimal)
96    VALUES
97      (vAnzahlDS, vSekundenDouble, vSekundenDecimal);
98
99    SET vAnzahlSchleifen = vAnzahlSchleifen + 1;
100   END WHILE; -- Hauptschleifenende
101
102  SET PROFILING = 0;
103  END///
104  DELIMITER ;

```

Um die Ausführungszeiten ermitteln zu können, muss das Profiling für diese Session eingeschaltet werden ([Zeile 70](#)). Anschließend wird die Testdatenquelle aufgebaut, indem die Prozedur `testdaten_insert` passend parametrisiert aufgerufen wird.

Die Hauptschleife ([73-100](#)) geht wie folgt vor: Erweitern des Testdatenbestands, Berechnung mit `DOUBLE`, Zeitmessung der Berechnung mit `DOUBLE`, Berechnung mit `DECIMAL`, Zeitmessung der Berechnung mit `DECIMAL` und Einfügen der Messwerte in die Tabelle `messung`. Schauen wir uns eine Messung mal genauer an.

```

84    -- Start Messung DECIMAL
85    SELECT SQL_NO_CACHE AVG(spalte_decimal) FROM testdaten INTO vTmpDec;
86    SELECT SUM(DURATION) FROM INFORMATION_SCHEMA.PROFILING
87    GROUP BY query_id
88    ORDER BY query_id DESC LIMIT 1
89    INTO vSekundenDecimal;
90    -- Ende Messung DECIMAL

```

Die Berechnung ist eine Mittelwertbildung über die gesamte Testdatentabelle. Der Zusatz `SQL_NO_CACHE` soll verhindern, dass die Messung durch Caching verfälscht wird. Mit `INTO vTempDec` wird verhindert, dass der errechnete Mittelwert auf dem Bildschirm ausgegeben wird¹.

Und jetzt kommt das Herzstück: die Zeitmessung. Ist das Profiling eingeschaltet, wird in der Tabelle `INFORMATION_SCHEMA.PROFILING` für jeden Arbeitsschritt, der zur Ausführung einer Anweisung nötig ist, eine Zeitdauer (`DURATION`) eingefügt.

```

1  SELECT query_id, state, duration
2  FROM INFORMATION_SCHEMA.PROFILING
3  WHERE query_id = 1;
4
5  +-----+-----+-----+
6  | query_id | state           | duration |
7  +-----+-----+-----+
8  |     1 | starting        | 0.000044 |
9  |     1 | checking permissions | 0.000005 |
10 |    1 | Opening tables   | 0.000018 |
11 |    1 | System lock      | 0.000007 |
12 |    1 | init             | 0.000014 |
13 |    1 | optimizing       | 0.000004 |
14 |    1 | statistics        | 0.000007 |
15 |    1 | preparing         | 0.000005 |
16 |    1 | executing        | 0.000004 |
17 |    1 | Sending data     | 0.003052 |
18 |    1 | end              | 0.000008 |
19 |    1 | query end        | 0.000004 |
20 |    1 | closing tables   | 0.000006 |
21 |    1 | freeing items    | 0.000009 |
22 |    1 | logging slow query | 0.000003 |
23 |    1 | cleaning up      | 0.000004 |
24 +-----+-----+-----+

```

Gruppiert man nun die Tabelle nach der `query_id` und summiert die einzelnen Zeiten auf, erhält man die Gesamtdauer. Will man nun die Dauer der letzten Anweisung, so sortiert man die Gruppenergebnisse nach `query_id` absteigend und holt sich nur den ersten Datensatz. Und genau das passiert hier.

¹ Was übrigens eine sehr teure Angelegenheit ist und das Experiment erheblich verlangsamen würde.

Zum Schluss wird noch das Profiling wieder ausgeschaltet und das Experiment aufgerufen. Den Aufruf habe ich allerdings auskommentiert, damit er nicht aus Versehen startet.

```
102 SET PROFILING = 0;
103 END//  
104 DELIMITER ;
105  
106 CALL perfDecimal01(20000, 100);
107 DROP DATABASE IF EXISTS perfDecimal01;
```

■ 27.2 Rundungsfehler



Laden Sie dieses Experiment mit SOURCE rundungsfehler01.sql.

Zuerst wird die Testumgebung gebaut:

```
1 -- START Bereitstellen der Testumgebung
2 DROP DATABASE IF EXISTS rundungsfehler01;
3 CREATE DATABASE rundungsfehler01;
4 USE rundungsfehler01;
5
6 -- Tabellen fuer die Testdaten
7 CREATE TABLE bla
8 (
9     floSpalte FLOAT,
10    decSpalte DECIMAL(30,15)
11 );
12 -- Ende Bereitstellung Testumgebung
```

Anschließend basteln wir uns eine Prozedur, die die Tabelle mit Werten füllt, hier 0.001.

```
14 -- Prozedur zum Aufbau der Testdaten
15 DELIMITER //  
16 CREATE PROCEDURE testdaten(IN iAnzahl INT)
17 BEGIN
18     DECLARE vAnzahl    INT DEFAULT 0;
19     DECLARE floSpalte FLOAT;
20     DECLARE decSpalte DECIMAL(10,4);
21
22     SET floSpalte = 0.001;
23     SET decSpalte = 0.001;
24     WHILE vAnzahl < iAnzahl DO
25         INSERT INTO bla VALUES(floSpalte, decSpalte);
26         SET vAnzahl = 1 + vAnzahl;
27     END WHILE;
28 END//  
29 DELIMITER ;
```

Zum Schluss der Aufruf der Prozedur und die Ausgabe der Summen.

```
32 CALL testdaten(1000); -- Erzeuge 1000 Testdaten
33 SELECT SUM(floSpalte) 'FLOAT', SUM(decSpalte) 'DECIMAL' FROM bla ;
34 DROP DATABASE rundungsfehler01;
```

■ 27.3 NULL versus NOT NULL



Laden Sie dieses Experiment mit SOURCE perfNotNull01.sql.

```

1 -- START Bereitstellen der Testumgebung
2 DROP DATABASE IF EXISTS perfNotNull01;
3 CREATE DATABASE perfNotNull01;
4 USE perfNotNull01;
5
6 -- Tabelle fuer die Testdaten
7 CREATE TABLE testdaten
8 (
9     spalte_notnull VARCHAR(255) NOT NULL DEFAULT '',
10    spalte_null     VARCHAR(255) NULL
11 );
12
13 CREATE TABLE testdaten_quelle
14 (
15     spalte_notnull VARCHAR(255) NOT NULL DEFAULT '',
16     spalte_null     VARCHAR(255) NULL
17 );
18
19 -- Tabelle fuer die Messergebnisse
20 CREATE TABLE messung
21 (
22     anzahl_ds BIGINT  UNSIGNED,
23     sekunden_notnull DECIMAL(12,8),
24     sekunden_null   DECIMAL(12,8)
25 );
26 -- ENDE Bereitstellen der Testumgebung

```

Aufbau und Funktion dieser Zeilen entsprechen denen des Experiments zur Performance-messung von DECIMAL und DOUBLE (siehe [Abschnitt 27.1 auf Seite 439](#)).

```

28 -- Einfuegen von Testdaten
29 -- START Einfuegen von Testdaten
30 DELIMITER //
31 CREATE PROCEDURE testdaten_insert
32 (
33     IN iAnzahl INT -- Anzahl der einzufuegenden Testdaten
34 )
35 BEGIN
36     DECLARE vAnzahl INT DEFAULT 0; -- Schleifenvariable
37
38     WHILE vAnzahl < iAnzahl DO      -- Schleifenbedingung
39         INSERT INTO testdaten_quelle VALUES('', NULL);
40         SET vAnzahl = vAnzahl + 1;    -- Schleifeninkrement
41     END WHILE;
42 END//
43 DELIMITER ;
44 -- ENDE Einfuegen von Testdaten

```

Der Aufbau der Testdaten ist allerdings viel einfacher als oben. Es werden eine bestimmte Anzahl (`iAnzahl`) Datensätze erzeugt. In der ersten Spalte wird dabei ein Leer-String und in der zweiten NULL verwendet.

```
47 -- Messung
48 DELIMITER //
49 CREATE PROCEDURE perfNotNull01
50 (
51     IN iAnzahlNeueZeilen INT, -- Anzahl der Testdatensaetze
52     IN iAnzahlSchleifen INT -- Anzahl der Messungen
53 )
54 BEGIN
55     DECLARE vAnzahlSchleifen INT DEFAULT 0;
56     DECLARE vSekundenNotNull DECIMAL(12,8); -- Zeitdauer
57     DECLARE vSekundenNull DECIMAL(12,8); -- Zeitdauer
58     DECLARE vAnzahlLDS BIGINT;
59     DECLARE vTmp VARCHAR(255);
60
61     TRUNCATE messung;
62     TRUNCATE testdaten;
63     TRUNCATE testdaten_quelle;
64
65     SET PROFILING = ON;          -- Profiling einschalten
66
67     CALL testdaten_insert(iAnzahlNeueZeilen); -- Testdatenquelle aufbauen
68
69     WHILE vAnzahlSchleifen < iAnzahlSchleifen DO -- Hauptschleife
70         SELECT 'Schleifendurchlauf ', vAnzahlSchleifen, ' von ', iAnzahlSchleifen;
71         INSERT INTO testdaten SELECT * FROM testdaten_quelle; -- Testdaten erweitern
72
73     -- START Messung NULL
74     SELECT SQL_NO_CACHE COUNT(*)      -- Testquery
75         FROM testdaten
76         WHERE spalte_null IS NULL INTO vTmp;
77     SELECT SUM(DURATION) FROM INFORMATION_SCHEMA.PROFILING
78         GROUP BY query_id
79         ORDER BY query_id DESC LIMIT 1
80         INTO vSekundenNull;
81     -- ENDE Messung NULL
82
83     -- START Messung NOT NULL
84     SELECT SQL_NO_CACHE COUNT(*)      -- Testquery
85         FROM testdaten
86         WHERE spalte_notnull = '' INTO vTmp;
87     SELECT SUM(DURATION) FROM INFORMATION_SCHEMA.PROFILING
88         GROUP BY query_id
89         ORDER BY query_id DESC LIMIT 1
90         INTO vSekundenNotNull;
91     -- ENDE Messung NOT NULL
92
93     -- Messung
94     -- Sichern der Ergebnisse
95     SELECT COUNT(*) FROM testdaten INTO vAnzahlLDS;
96     INSERT INTO messung (anzahl_ds, sekunden_notnull, sekunden_null)
97         VALUES (vAnzahlLDS, vSekundenNotNull, vSekundenNull);
98
99     SET vAnzahlSchleifen = vAnzahlSchleifen + 1;
100    END WHILE; -- Hauptschleifenende
101
102   SET PROFILING = OFF;          -- Profiling ausschalten
103   END///
104   DELIMITER ;
```

Fast alle Teilschritte der Messung entsprechen dem des Experiments oben (siehe [Abschnitt 27.1 auf Seite 439](#)). Lediglich die zu messenden Anweisungen sind – natürlich – andere:

```
74  SELECT SQL_NO_CACHE COUNT(*)      -- Testquery
75    FROM testdaten
76   WHERE spalte_null IS NULL INTO vTmp;
```

und

```
84  SELECT SQL_NO_CACHE COUNT(*)      -- Testquery
85    FROM testdaten
86   WHERE spalte_notnull = '' INTO vTmp;
```

Ich habe diese beiden Abfragen gewählt, weil sie genau das tun, wo Zeitunterschiede festgestellt werden müssen: Ein Vergleich mit einem Leer-String (= '') und NULL (IS NULL).

Die Messdaten wurden mit folgendem Aufruf ermittelt:

```
106 CALL perfNotNull01(10000, 1000);
```

27.4 Suchen mit und ohne Index



Laden Sie dieses Experiment mit SOURCE perfIndexMitOhne01.sql.

Um die Performanceunterschiede bei der Verwendung und Nichtverwendung von Indizes darzustellen, habe ich Echtdaten benutzt. Es handelt sich um die Liste der Bankleitzahlen, die man bei [\[Bun16\]](#) herunterladen kann. Vorher wird die Datenbank angelegt und ausgewählt.

```
1  -- START Bereitstellen der Testumgebung
2  DROP DATABASE IF EXISTS perfIndexMitOhne01;
3  CREATE DATABASE perfIndexMitOhne01 CHARACTER SET utf8;
4
5  USE perfIndexMitOhne01;
```

Anschließend werden die Testdatentabellen angelegt. Dabei wird auf einer Spalte ein Index erzeugt (`bezeichnung_mit`) und auf der anderen (`bezeichnung_ohne`) nicht.

```
7  -- Tabelle fuer die Testdaten
8  CREATE TABLE bank
9  (
10    blz        CHAR(8),
11    merkmal    CHAR(1),
12    bezeichnung VARCHAR(255),
13    plz        CHAR(5),
14    ort        VARCHAR(255),
15    kurz       VARCHAR(255),
16    pan        CHAR(5),
17    bic        VARCHAR(255),
18    sm         CHAR(2),
19    ds         CHAR(6),
20    kz         CHAR(1),
```

```

21    blzl      CHAR(1),
22    blzn      CHAR(8),
23    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY
24  )
25  ;
26
27 CREATE TABLE testdaten
28 (
29   bezeichnung_mit  VARCHAR(255),
30   bezeichnung_ohne VARCHAR(255)
31 );
32
33 CREATE INDEX idx_testdaten ON testdaten(bezeichnung_mit);

```

Die Messergebnisse werden in einer eigenen Tabelle abgelegt.

```

35 -- Tabelle fuer die Messergebnisse
36 CREATE TABLE messung
37 (
38   anzahl_ds BIGINT  UNSIGNED,
39   sekunden_mitindex DECIMAL(12,8),
40   sekunden_ohneindex DECIMAL(12,8)
41 );

```

Jetzt werden die Bankleitzahlen in die Tabelle bank importiert. Dabei wird eine `id` per `AUTO_INCREMENT` mit erzeugt. Über diese `id` erfolgt später die zufällige Selektion.

```

43 -- Einlesen der Bankdaten
44 SELECT 'LOAD DATA';
45 LOAD DATA INFILE '/home/ralf/daten/privat/Buch/src/blz_20120305.csv'
46 INTO TABLE bank
47 FIELDS TERMINATED BY ','
48 LINES TERMINATED BY '\n'
49 ;
50 -- ENDE Bereitstellen der Testumgebung

```

Die Prozedur `testdaten_insert` kopiert `iAnzahl` viele Datensätze aus der Tabelle `bank` in die Tabelle `testdaten`. Damit es nicht immer die gleichen sind, wird per Zufall eine Zahl ermittelt und diese als `id` verwendet. Damit es diesen Zufallswert auch wirklich in der Banktabelle gibt, wird vorher die Anzahl der Bankdatensätze ermittelt und als Obergrenze für die Zufallszahl benutzt.

```

52 DELIMITER //
53 CREATE PROCEDURE testdaten_insert
54 (
55   iAnzahl INT
56 )
57 BEGIN
58   DECLARE vAnzahl INT;
59   DECLARE vCount INT;
60   DECLARE vId INT UNSIGNED;
61
62   SELECT COUNT(*) FROM bank INTO vCount;
63   SET vAnzahl = 0;
64   WHILE vAnzahl < iAnzahl DO
65     SET vId = FLOOR(RAND() * vCount);
66     INSERT INTO testdaten (bezeichnung_mit, bezeichnung_ohne)
67       SELECT bezeichnung, bezeichnung FROM bank WHERE id = vId;
68     SET vAnzahl = vAnzahl + 1;
69   END WHILE;

```

```

70  END//  

71  DELIMITER ;

```

Die Prozedur `perfIndexMitOhne01` hat als ersten Parameter die Anzahl der Messungen und als zweiten die Größe der Erweiterung der Testdaten pro Messung. Es werden notwendige Variablen deklariert und das Profiling eingeschaltet. Über das Profiling und wie man die Ausführungszeiten ermittelt, erfahren Sie im ersten Experiment oben (siehe [Abschnitt 27.1 auf Seite 439](#)) mehr.

```

73  -- Messung  

74  DELIMITER //  

75  CREATE PROCEDURE perfIndexMitOhne01  

76  (  

77      IN iAnzahlNeueZeilen INT, -- Anzahl der Testdatensaetze  

78      IN iAnzahlSchleifen INT -- Anzahl der Messungen  

79  )  

80 BEGIN  

81     DECLARE vAnzahlSchleifen INT DEFAULT 0;  

82     DECLARE vSekundenMitIndex DECIMAL(12,8); -- Zeitdauer  

83     DECLARE vSekundenOhneIndex DECIMAL(12,8); -- Zeitdauer  

84     DECLARE vAnzahlDS BIGINT;  

85     DECLARE vTmp VARCHAR(255);  

86  

87     TRUNCATE messung;  

88     TRUNCATE testdaten;  

89  

90     SET PROFILING = ON;          -- Profiling einschalten

```

In der Hauptschleife wird nach einer Bildschirmausgabe zuerst der Testdatenbestand erweitert. Dann wird eine Suche auf der Spalte mit und anschließend eine auf der Spalte ohne Index durchgeführt. Die Ausführungszeiten werden ermittelt und in die Variablen `vSekundenMit` und `vSekundenOhne` abgelegt.

Die Messwerte werden dann in die Tabelle `messwert` eingefügt und die Schleifenvariable inkrementiert.

```

92 WHILE vAnzahlSchleifen < iAnzahlSchleifen DO -- Hauptschleife  

93     SELECT 'Schleifendurchlauf ', vAnzahlSchleifen, ' von ', iAnzahlSchleifen;  

94     CALL testdaten_insert(iAnzahlNeueZeilen); -- Testdatenquelle aufbauen  

95  

96     -- START Messung mit Index  

97     SELECT SQL_NO_CACHE COUNT(*)      -- Testquery  

98     FROM testdaten  

99     WHERE bezeichnung_mit = 'ABC'  

100    INTO vTmp;  

101    SELECT SUM(DURATION) FROM INFORMATION_SCHEMA.PROFILING  

102        GROUP BY query_id  

103        ORDER BY query_id DESC LIMIT 1  

104        INTO vSekundenMitIndex;  

105    -- ENDE Messung mit Index  

106  

107    -- START Messung ohne Index  

108    SELECT SQL_NO_CACHE COUNT(*)      -- Testquery  

109    FROM testdaten  

110    WHERE bezeichnung_ohne = 'ABC'  

111    INTO vTmp;  

112    SELECT SUM(DURATION) FROM INFORMATION_SCHEMA.PROFILING  

113        GROUP BY query_id  

114        ORDER BY query_id DESC LIMIT 1

```

```

115      INTO vSekundenOhneIndex;
116      -- ENDE Messung ohne Index
117
118      -- Messung
119      -- Sichern der Ergebnisse
120      SELECT COUNT(*) FROM testdaten INTO vAnzahlDS;
121      INSERT INTO messung
122          (anzahl_ds, sekunden_mitindex, sekunden_ohneindex)
123          VALUES
124          (vAnzahlDS, vSekundenMitIndex, vSekundenOhneIndex);
125
126      SET vAnzahlSchleifen = vAnzahlSchleifen + 1;
127  END WHILE; -- Hauptschleifenende

```

Abschließend wird das Profiling wieder ausgeschaltet und der DELIMITER wieder zurückgesetzt.

```

192 SET PROFILING = OFF;           -- Profiling ausschalten
193 END//                         ;
194 DELIMITER ;

```

Um die Ergebnisse der ersten Messung zu erhalten, rufen Sie die Prozedur wie folgt auf:

```
1 CALL perfIndexMitOhne01(100, 500);
```

Die zweite Messung wurde mit anderen Werten durchgeführt:

```
1 CALL perfIndexMitOhne01(2, 300);
```

■ 27.5 Messen der Performance der Einfügeoperation



Laden Sie dieses Experiment mit SOURCE perfIndexInsert01.sql.

Zuerst wird wieder eine Testumgebung gebastelt. Die Tabelle `testdaten` dient als Kopierquelle des `INSERT`. Es werden dazu die Bankleitzahlendaten verwendet.

```

1 -- START Bereitstellen der Testumgebung
2 DROP DATABASE IF EXISTS perfIndexInsert01;
3 CREATE DATABASE perfIndexInsert01 CHARACTER SET utf8;
4
5 USE perfIndexInsert01;
6
7 -- Tabellen fuer die Testdaten
8 CREATE TABLE testdaten
9 (
10    blz CHAR(8),  merkmal char(1),  bezeichnung varchar(255),
11    plz char(5),  ort varchar(255),  kurz varchar(255),
12    pan char(5),  bic varchar(255),  sm char(2),
13    ds char(6),   kz char(1),     blzl char(1),  blzn char(8)
14 ) Engine=MyISAM;
15

```

```

16  -- Einlesen der Bankdaten
17  LOAD DATA INFILE '/home/ralf/daten/privat/Buch/src/blz_20120305.csv'
18  INTO TABLE testdaten
19  FIELDS TERMINATED BY ';'
20  LINES TERMINATED BY '\n'
21  ;

```

Dann die beiden Tabellen `bank_ohne` und `bank_mit`. Die eine hat keine Indizes und die andere sehr wohl. Ein Index ist ein einfacher und der andere ein aus drei Spalten zusammengesetzter.

```

23  CREATE TABLE bank_ohne
24  (
25    blz CHAR(8), merkmal char(1), bezeichnung varchar(255),
26    plz char(5), ort varchar(255), kurz varchar(255),
27    pan char(5), bic varchar(255), sm char(2),
28    ds char(6), kz char(1), blzl char(1), blzn char(8)
29  ) Engine=MyISAM;
30
31  CREATE TABLE bank_mit
32  (
33    blz CHAR(8), merkmal char(1), bezeichnung varchar(255),
34    plz char(5), ort varchar(255), kurz varchar(255),
35    pan char(5), bic varchar(255), sm char(2),
36    ds char(6), kz char(1), blzl char(1), blzn char(8)
37  ) Engine=MyISAM;
38
39  CREATE INDEX idx_bankmit_bzeichnung ON bank_mit(bezeichnung);
40  CREATE INDEX idx_bankmit_plzortblz ON bank_mit(plz, ort, blz);

```

Jetzt noch die Tabelle für die Messergebnisse:

```

42  -- Tabelle fuer die Messergebnisse
43  CREATE TABLE messung
44  (
45    anzahl_ds BIGINT UNSIGNED,
46    sekunden_mitindex DECIMAL(12,8),
47    sekunden_ohneindex DECIMAL(12,8)
48  );
49  -- ENDE Bereitstellen der Testumgebung

```

Wir haben zwei Prozeduren für den Test: Die erste kopiert anhängig vom Übergabeparameter `iName` den Inhalt der Tabelle `testdaten` nach `bank_ohne` oder `bank_mit`. Anschließend wird die Messung vorgenommen. Das Verfahren dazu wird oben (siehe [Abschnitt 27.1 auf Seite 439](#)) beschrieben. Der ermittelte Wert wird in den Übergabeparameter `oSekunden` geschrieben. Da dieser mit `OUT` deklariert wurde, steht der Wert auch außerhalb der Prozedur zur Verfügung.

```

52  -- Prozedur kopieren der Testdaten
53  DELIMITER //
54  CREATE PROCEDURE kopiere
55  (
56    IN iName VARCHAR(255),
57    OUT oSekunden DECIMAL(12,8) -- Ein OUT Parameter!
58  )
59  BEGIN
60  CASE iName
61  WHEN 'bank_ohne' THEN
62    INSERT INTO bank_ohne SELECT SQL_NO_CACHE * FROM testdaten;

```

```

63 WHEN 'bank_mit' THEN
64   INSERT INTO bank_mit  SELECT SQL_NO_CACHE * FROM testdaten;
65 END CASE;
66 SELECT SUM(DURATION) FROM INFORMATION_SCHEMA.PROFILING
67 GROUP BY query_id
68 ORDER BY query_id DESC LIMIT 1
69   INTO oSekunden;          -- Ergebnis in den OUT-Parameter
70 END///
71 DELIMITER ;

```

Die eigentliche Testprozedur bekommt als Übergabeparameter die Anzahl der Messungen. Vor dem Start werden die entscheidenden Tabellen geleert. Innerhalb der Schleife wird die Prozedur `kopiere` aufgerufen; einmal mit `bank_ohne` und einmal mit `bank_mit`. Anschließend werden die gemessenen Werte in die Messtabelle eingefügt. Die Testergebnisse wurden mit `CALL perfIndexInsert01(50)` erzielt.

```

74 -- Testprozedur
75 DELIMITER //
76 CREATE PROCEDURE perfIndexInsert01
77 (
78   IN iAnzahlSchleifen INT UNSIGNED
79 )
80 BEGIN
81   DECLARE vAnzahlSchleifen    INT DEFAULT 0;
82   DECLARE vSekundenMitIndex  DECIMAL(12,8); -- Zeitdauer
83   DECLARE vSekundenOhneIndex DECIMAL(12,8); -- Zeitdauer
84   DECLARE vAnzahlDS           BIGINT;
85
86   TRUNCATE messung;
87   TRUNCATE bank_mit;
88   TRUNCATE bank_ohne;
89
90   SET PROFILING = ON;          -- Profiling einschalten
91
92   SET vAnzahlSchleifen = 0;
93   -- Hauptschleife
94   WHILE vAnzahlSchleifen < iAnzahlSchleifen DO
95     SELECT 'Durchlauf ', vAnzahlSchleifen, ' von ', iAnzahlSchleifen;
96
97     -- START Messung
98     CALL kopiere('bank_ohne', vSekundenOhneIndex);
99     CALL kopiere('bank_mit', vSekundenMitIndex);
100    -- Ende Messung
101
102    -- Sichern der Messergebnisse
103    SELECT COUNT(*) FROM bank_ohne INTO vAnzahlDS;
104    INSERT INTO messung (anzahl_ds, sekunden_mitindex, sekunden_ohneindex)
105      VALUES (vAnzahlDS, vSekundenMitIndex, vSekundenOhneIndex);
106
107    SET vAnzahlSchleifen = 1 + vAnzahlSchleifen;
108  END WHILE;
109
110  SET PROFILING = OFF;         -- Profiling ausschalten
111
112  END///
113 DELIMITER ;
114
115 CALL perfIndexInsert01(50);

```

Bei diesem Versuch wurde die Engine MyISAM verwendet. Ich hatte eigentlich keinen besonderen Grund, diese zu wählen, fragte mich aber, ob die InnoDB die gleichen Ergebnisse liefern würden. Also: Umbau des Versuchs; die Engine wird von MyISAM nach InnoDB umgestellt und der Versuch neu gestartet. Das Ergebnis ist in Bild 27.1 zu sehen. Ohne Index einzufügen ist bei großen Datenmengen signifikant billiger als mit Index.

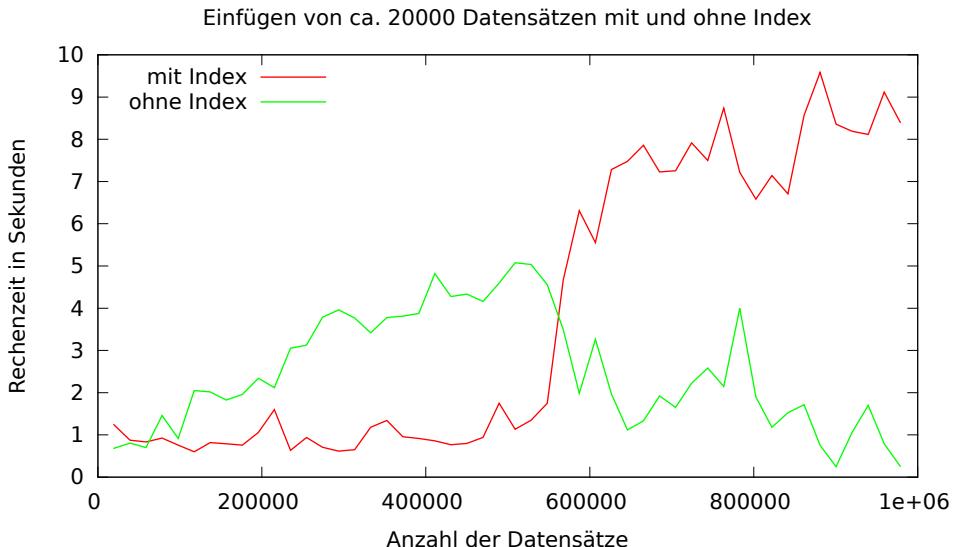


Bild 27.1 Zeitdauer Einfügeoperationen mit und ohne Index bei der InnoDB

Erstaunlicherweise ist das Einfügen mit Index bei der InnoDB bis ca. 550.000 Datensätze billiger als ohne. Erst danach wird das Einfügen ohne Index billiger. Überraschend ist für mich auch gewesen, dass es sogar billiger ist, 700.000 Datensätze einzufügen statt 400.000! Weitere Tests haben mir gezeigt, dass sich der Trend danach genauso fortsetzt.

Leider kann ich keine Erklärung dafür finden. Ich vermute allerdings, dass ab einer gewissen Datenmenge die interne Verarbeitungsstrategie der Tabellen verändert wird. Vielleicht können bis zur genannten Grenze die Daten im Arbeitsspeicher gehalten werden, und danach muss ausgelagert werden und wird seitenweise auf die Tabellen zugegriffen. Vielleicht ... *Wer eine gute Begründung hat, möge sich doch bitte melden.*

■ 27.6 Messen der Indexselektivität



Laden Sie dieses Experiment mit SOURCE indexSelektiv01.sql.

Eine Testumgebung muss her. Diese beinhaltet die Tabelle `testdaten`, in welche ich die Bankleitzahlen importiere.

```

2  DROP DATABASE IF EXISTS indexSelektiv01;
3  CREATE DATABASE indexSelektiv01 CHARACTER SET utf8;
4
5  USE indexSelektiv01;
6
7  CREATE TABLE testdaten
8  (
9    blz CHAR(8),  merkmal char(1),  bezeichnung varchar(255),
10   plz char(5),  ort varchar(255),  kurz varchar(255),
11   pan char(5),  bic varchar(255),  sm char(2),
12   ds char(6),   kz char(1),     blzl char(1),  blzn char(8)
13 );
14
15 LOAD DATA INFILE 'blz_20120305.csv'
16   INTO TABLE testdaten
17   FIELDS TERMINATED BY ','
18   LINES TERMINATED BY '\n'
19 ;

```

In der Tabelle **messung** wird die Länge des Index und seine Selektivität abgespeichert.

```

22  CREATE TABLE messung (
23    laenge      INT UNSIGNED,
24    selektiv    DOUBLE
25 );

```

In der eigentlichen Messprozedur wird erst mal ein Haufen Variablen bereitgestellt. Die Variable **vLaengeMax** wird die längste String-Länge enthalten, die im Index möglich ist. In **vLaengeAkt** steht die aktuelle Indexlänge, die in der Schleife bei 1 beginnend bis **vLaengeMax** inkrementiert wird.

vSelektiv nimmt die berechnete Indexselektivität auf, damit diese später in der Tabelle **messung** abgespeichert werden kann. Verbleibt die Info, wie viele Zeilen überhaupt als Testdaten zur Verfügung stehen. Dazu ist **vAnzahl** da.

```

29  DELIMITER //
30  CREATE PROCEDURE indexSelektiv01()
31  BEGIN
32
33  DECLARE vLaengeAkt      INT UNSIGNED; -- Schleifenvariable
34  DECLARE vLaengeMax      INT UNSIGNED; -- Maximale Stringlaenge
35  DECLARE vSelektiv       DOUBLE;        -- Gemessene Seletivitaet
36  DECLARE vAnzahl         INT UNSIGNED; -- Anzahl der Testdaten

```

Jetzt können die Gesamtanzahl und die maximale String-Länge bestimmt werden.

```

41  -- Testdatenanzahl ermitteln
42  SELECT COUNT(*) FROM testdaten INTO vAnzahl;
43  -- Maximale Stringlaenge ermitteln
44  SELECT MAX(LENGTH(bezeichnung)) FROM testdaten INTO vLaengeMax;

```

In der Hauptschleife wird die Anzahl der unterscheidbaren Inhalte der Länge **vLaengeAkt** in der Spalte **bezeichnung** gezählt. Diese muss durch die Anzahl aller Zeilen dividiert werden (siehe Formel zur Indexselektivität in [Definition 32 auf Seite 102](#)). Die Schleife selbst erhöht die Länge bei jeder Iteration um 1.

```

43  -- Hauptschleife
44  SET vLaengeAkt = 1;
45  WHILE vLaengeAkt <= vLaengeMax DO
46    -- Messung der Seletivitaet

```

```

47  SELECT COUNT(DISTINCT LEFT(bezeichnung, vLaengeAkt)) / vAnzahl
48  FROM testdaten INTO vSelektiv;
49
50  -- Speichern der Messung
51  INSERT INTO messung(laenge, selektiv) VALUES (vLaengeAkt, vSelektiv);
52  SELECT 'Laenge ', vLaengeAkt, ' von ', vLaengeMax, ' mit ', vSelektiv;
53
54  SET vLaengeAkt = 1 + vLaengeAkt;
55  END WHILE;

```

Verbleibt der Rest.

```

56  END///
57  DELIMITER ;

```

■ 27.7 Sortieren ohne und mit Index



Laden Sie dieses Experiment mit SOURCE perfIndexOrderBy01.sql.

Zuerst wird – wie üblich – eine Testumgebung bereitgestellt. Dazu gehören eine Datenbank, eine Tabelle ohne Index, eine Tabelle mit Index und eine Tabelle für die Messergebnisse.

```

2  DROP DATABASE IF EXISTS perfIndexOrderBy01;
3  CREATE DATABASE perfIndexOrderBy01 CHARACTER SET utf8;
4
5  USE perfIndexOrderBy01;
6
7  -- Tabelle ohne Index
8  CREATE TABLE name_ohne
9  (
10    id      INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
11    nachname VARCHAR(255),
12    vorname  VARCHAR(255)
13 ) Engine=InnoDB;
14
15 -- Tabelle mit Index
16 CREATE TABLE name_mit
17 (
18    id      INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
19    nachname VARCHAR(255),
20    vorname  VARCHAR(255)
21 ) Engine=InnoDB;
22
23 -- Index erstellen
24 CREATE INDEX idx_namemit_nnvn ON name_mit(nachname, vorname);
25
26 -- Tabelle fuer die Messergebnisse
27 CREATE TABLE messung
28 (
29    anzahl_ds BIGINT  UNSIGNED,
30    sekunden_ohne DECIMAL(12,8),

```

```

31     sekunden_mit  DECIMAL(12,8)
32 );

```

In der Prozedur datenerzeugen werden, gesteuert durch den Übergabeparameter iAnzahl, zufällige Testdaten in die beiden Tabellen eingefügt. Die Inhalte der Datensätze sind dabei gleich, nur der Index unterscheidet die beiden Tabellen.

```

36 DELIMITER //
37 CREATE PROCEDURE datenerzeugen
38 (
39     IN iAnzahl INT UNSIGNED
40 )
41 BEGIN
42     DECLARE vAnzahl INT UNSIGNED DEFAULT 0;
43     DECLARE vNachname VARCHAR(255);
44     DECLARE vVorname VARCHAR(255);
45
46     WHILE vAnzahl < iAnzahl DO
47         SELECT MD5(RAND()) INTO vNachname;
48         SELECT MD5(RAND()) INTO vVorname;
49         INSERT INTO name_ohne (nachname, vorname) VALUES (vNachname, vVorname);
50         INSERT INTO name_mit (nachname, vorname) VALUES (vNachname, vVorname);
51         SET vAnzahl = 1 + vAnzahl;
52     END WHILE;
53 END//
54 DELIMITER ;

```

Die Prozedur dauer ermittelt den Zeitverbrauch der zuletzt durchgeführten Anweisung (Näheres in [Abschnitt 27.1 auf Seite 439](#)). Der Zeitverbrauch wird in die OUT-Variable oSekunden geschrieben und steht daher der aufrufenden Prozedur zur Verfügung.

```

56 -- Ermitteln des Verbrauchs der letzten Aktion
57 DELIMITER //
58 CREATE PROCEDURE dauer
59 (
60     OUT oSekunden DECIMAL(12,8)
61 )
62 BEGIN
63     SELECT SUM(DURATION) FROM INFORMATION_SCHEMA.PROFILING
64     GROUP BY query_id
65     ORDER BY query_id DESC LIMIT 1
66     INTO oSekunden;
67 END//
68 DELIMITER ;

```

Das eigentliche Experiment geht nach bekanntem Muster vor. Die Variable vAnzahl ist die Schleifenvariable für die Messschleife. Sie wird bei jedem Schleifendurchlauf um 1 inkrementiert und mit dem Übergabeparameter iAnzahl in der Schleifenbedingung verglichen. In vAnzahlDS wird die aktuelle Anzahl der Testdaten abgelegt. vDauerMit und vDauerOhne enthalten die Ausführungszeiten der jeweiligen Sortieroperationen. Der vDummy wird nur dazu gebraucht, das Ergebnis des SELECT irgendwo hinzuspeichern, damit es die Bildschirmausgabe nicht stört. Anschließend werden alle Tabellen zurückgesetzt und das Profiling eingeschaltet.

```

71 -- Das eigentliche Experiment
72 DELIMITER //
73 CREATE PROCEDURE perfIndexOrderBy01
74 (

```

```

75      IN iAnzahl          INT UNSIGNED
76    )
77 BEGIN
78  DECLARE vAnzahl          INT UNSIGNED DEFAULT 0;
79  DECLARE vAnzahlDS        BIGINT UNSIGNED DEFAULT 0;
80  DECLARE vDauerOhne      DECIMAL(12,8);
81  DECLARE vDauerMit       DECIMAL(12,8);
82  DECLARE vDummy          VARCHAR(255);
83
84  TRUNCATE messung;      -- Alte Daten entfernen
85  TRUNCATE name_ohne;   -- Alte Daten entfernen
86  TRUNCATE name_mit;    -- Alte Daten entfernen
87
88  SET PROFILING = 1;     -- Profiling einschalten

```

In der Hauptschleife wird jeweils ein SELECT mit einem ORDER BY abgesetzt: einmal auf die Tabelle ohne und einmal auf die Tabelle mit Index. Unmittelbar nach dem SELECT wird die Zeit der Ausführung gemessen. Zuletzt werden die Messergebnisse in die Tabelle `messung` eingefügt und die Schleifenvariable inkrementiert.

```

90  -- Hauptschleife
91  SET vAnzahl = 1;
92  WHILE vAnzahl <= iAnzahl DO
93    SELECT 'Durchlauf ', vAnzahl, ' von ', iAnzahl;
94    -- Testdaten erweitern
95    CALL datenerzeugen(10);
96
97    -- Messung ohne
98    SELECT SQL_NO_CACHE id
99      FROM name_ohne
100     ORDER BY nachname, vorname
101    LIMIT 1 INTO vDummy;
102    CALL dauer(vDauerOhne);
103    -- Messung mit
104    SELECT SQL_NO_CACHE id
105      FROM name_mit
106     ORDER BY nachname, vorname
107    LIMIT 1 INTO vDummy;
108    CALL dauer(vDauerMit);
109
110   -- Sichern der Messergebnisse
111   SELECT COUNT(*) FROM name_ohne INTO vAnzahlDS;
112   INSERT INTO messung(anzahl_ds, sekunden_ohne, sekunden_mit)
113     VALUES (vAnzahlDS, vDauerOhne, vDauerMit);
114
115   SET vAnzahl = 1 + vAnzahl;
116 END WHILE;

```

Nach der Schleife wird das Profiling wieder ausgeschaltet und der DELIMITER wieder auf das Semikolon gesetzt. Wenn Sie die Testergebnisse nachvollziehen wollen, müssen Sie `CALL perfIndexOrderBy01(100);` ausführen.

```

118  SET PROFILING = 0;      -- Profiling ausschalten
119  END///
120  DELIMITER ;
121
122  CALL perfIndexOrderBy01(100);

```

■ 28.1 Für Deutsch relevante Zeichensätze

Zeichensatz	Beschreibung	Standardsortierung	Max. Zeichenlänge
dec8	DEC West European	dec8_swedish_ci	1
cp850	DOS West European	cp850_general_ci	1
cp852	DOS Central European	cp852_general_ci	1
cp1250	Windows Central European	cp1250_general_ci	1
hp8	HP West European	hp8_english_ci	1
latin1	cp1252 West European	latin1_swedish_ci	1
latin2	ISO 8859-2 Central European	latin2_general_ci	1
macce	Mac Central European	macce_general_ci	1
macroman	Mac West European	macroman_general_ci	1
ucs2	UCS-2 Unicode	ucs2_general_ci	2
utf8	UTF-8 Unicode	utf8_general_ci	3
utf8mb4	UTF-8 Unicode	utf8mb4_general_ci	4
utf16	UTF-16 Unicode	utf16_general_ci	4
utf16le	UTF-16 Unicode	utf16_general_ci	4
utf32	UTF-32 Unicode	utf32_general_ci	4

Eine vollständige Liste erhalten Sie mit `SHOW CHARACTER SET;`.

Die Abkürzung `ci` steht für *case insensitive* (Groß- und Kleinschreibung wird ignoriert) und `cs` für *case sensitive* (Groß- und Kleinschreibung wird beachtet). Die Abkürzung `le` steht für *little endian* und gibt an, dass entgegen der sonst üblichen Reihenfolge der Unicode Bytes *big endian* diese in der anderen Reihenfolge angeordnet sind.

Die Zeichensätze `utf8` und `utf8mb4` unterscheiden sich wie folgt: `utf8` verwendet maximal 3 Bytes zur Kodierung und unterstützt nur Unicode-Zeichen der *Basic Multilingual Plane (BMP)*. Dies sind die üblichen Sprachzeichensysteme (lateinisch, griechisch, kyrillisch, japanisch etc.) und die üblichen Sonderzeichen. Der Zeichensatz `utf8mb4` verwendet maximal 4 Bytes pro Zeichen und enthält neben allen Zeichen von `utf8` auch die Unicode-Zeichen der *Supplementary Multilingual Plane (SMP)*. Mehr finden Sie unter [Wik16].

■ 28.2 Für Deutsch relevante Sortierungen

Sortierung	Zeichensatz	Id	Vorbelegung	Sortierlänge
dec8_bin	dec8	69		1
cp1250_bin	cp1250	66		1
cp1250_general_ci	cp1250	26	Yes	1
cp850_bin	cp850	80		1
cp850_general_ci	cp850	4	Yes	1
cp852_bin	cp852	81		1
cp852_general_ci	cp852	40	Yes	1
hp8_bin	hp8	72		1
latin1_bin	latin1	47		1
latin1_general_ci	latin1	48		1
latin1_general_cs	latin1	49		1
latin1_german1_ci	latin1	5		1
latin1_german2_ci	latin1	31		2
latin2_bin	latin2	77		1
latin2_general_ci	latin2	9	Yes	1
macce_bin	macce	43		1
macce_general_ci	macce	38	Yes	1
macroman_bin	macroman	53		1
macroman_general_ci	macroman	39	Yes	1
ucs2_bin	ucs2	90		1
ucs2_general_ci	ucs2	35	Yes	1
ucs2_german2_ci	ucs2	148	Yes	1
ucs2_unicode_ci	ucs2	128		8
utf8_bin	utf8	83		1
utf8_general_ci	utf8	33	Yes	1
utf8_german2_ci	utf8	212	Yes	1
utf8_unicode_ci	utf8	192		8
utf8mb4_bin	utf8mb4	46		1
utf8mb4_general_ci	utf8mb4	45	Yes	1
utf8mb4_german2_ci	utf8mb4	244	Yes	1
utf8mb4_unicode_ci	utf8mb4	224		8
utf16_bin	utf16	55		1
utf16_general_ci	utf16	54	Yes	1
utf16_german2_ci	utf16	121	Yes	1
utf16_unicode_ci	utf16	101		8

Fortsetzung auf der nächsten Seite ...

Fortsetzung der vorherigen Seite ...

Sortierung	Zeichensatz	Id	Vorbelegung	Sortierlänge
utf16le_bin	utf16	56		1
utf16le_general_ci	utf16	62	Yes	1
utf32_bin	utf32	61		1
utf32_general_ci	utf32	60	Yes	1
utf32_german2_ci	utf32	180	Yes	1
utf32_unicode_ci	utf32	160		8

Eine vollständige Liste der verfügbaren Sortierungen erhalten Sie mit `SHOW COLLATION;`.

Sortierungen mit der Endung `bin` unterscheiden zwischen Groß- und Kleinschreibungen.
Eine sehr schöne Übersicht der Zeichensätze und ihrer Collations finden Sie unter [\[Bar10\]](#).

Damit Sie direkt die Quelltexte und Beispiele studieren können, ohne dass dazu ein Rechner parallel zum E-Book laufen muss, sind diese hier weitestgehend abgedruckt. Natürlich sind diese nicht selbsterklärend und nur zum Nachschlagen geeignet.

■ 29.1 MySQL/MariaDB

29.1.1 Quelltexte zu Teil II

Listing 29.1 mysql/listing01.sql

```
1 -- Warum formatiere ich den Quelltext so ganz anders, als beispielsweise diese
   -- Beautyfier?
2 -- * http://www.driver.com/pp/sqlformat.htm
3 -- * https://codebeautify.org/sqlformatter
4 -- Ich finde die Arbeit dieser Programme sehr gut und auch die Ergebnisse sind
   -- ansprechend,
5 -- aber sie widersprechen meinem Anspruch an die leichte Verständlichkeit der
   -- Befehle.
6 -- Zwei Beispiele:
7 -- * Es werden keine Einrückungen eingefügt, die den inhaltlichen
   -- Zusammenhang
8 -- zwischen dem Befehl und seinen Parametern abbilden:
9 -- SELECT blz,
10 --       bankname
11 -- FROM  bank
12 -- ORDER BY blz DESC
13 -- LIMIT 1;
14 --
15 -- anstelle von
16 --
17 -- SELECT
18 --       blz, bankname
19 -- FROM  bank
20 -- ORDER BY blz DESC
21 -- LIMIT 1
22 -- ;
23
```

```

24 -- * Hinter dem FROM wird ein Tabellenname gestellt, obwohl die Auswertung aus
25 -- dem ganzen JOIN erfolgt:
26 --   SELECT k.kunde_id,
27 --         k.nachname,
28 --         k.vorname,
29 --         b.datum
30 --   FROM bestellung b
31 --     INNER JOIN kunde k USING (kunde_id)
32 --   ORDER BY k.nachname,
33 --           k.vorname;
34
35 -- anstelle von
36 --   SELECT
37 --     k.kunde_id, k.nachname, k.vorname, b.datum
38 --   FROM
39 --     bestellung b INNER JOIN kunde k USING (kunde_id)
40 --   ORDER BY
41 --     k.nachname, k.vorname
42 --   ;
43
44 -- Ich werde daher meine Darstellung beibehalten.
45
46
47 -- Anlegen der Datenbank auf der grünen Wiese
48 SELECT 'Start listing01.sql' //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////';
49 SET NAMES utf8;
50 SET AUTOCOMMIT=ON;
51 SELECT 'Löschen einer ggf. vorhandenen Datenbank oshop (Seite 68)';
52 DROP DATABASE IF EXISTS oshop;
53
54 SELECT 'Anzeigen der verfügbaren Zeichenkodierungen (Seite 69)';
55 SHOW CHARACTER SET;
56 SHOW CHARACTER SET LIKE 'latin';
57 SHOW CHARACTER SET LIKE 'utf%';
58
59 SELECT 'Anzeigen der verfügbaren Sortierungen (Seite 71)';
60 SHOW COLLATION;
61 SHOW COLLATION LIKE 'latin1%';
62 SHOW COLLATION LIKE 'utf8%';
63
64 SELECT 'Anlegen der Datenbank mit Zeichensatz und Sortierung (Seite 66)';
65 CREATE DATABASE oshop
66   DEFAULT CHARACTER SET utf8
67   COLLATE utf8_unicode_ci
68 ;
69
70 SELECT 'Anzeigen aller Datenbanken (Seite 72)';
71 SHOW DATABASES;
72 SELECT 'Ende listing01.sql' //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////';

```

Listing 29.2 mysql/listing02.sql

```

1 -- Ausführen der vorherigen Befehle
2 SOURCE listing01.sql
3
4 SELECT 'Start listing02.sql' //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////';
5

```

```
6 -- Erspare mir ein wenig Tipparbeit
7 USE oshop; -- Seite 75
8
9 -- adresse
10 SELECT 'Tabelle adresse anlegen (Seite 82)';
11 CREATE TABLE adresse (
12     adresse_id          INT UNSIGNED AUTO_INCREMENT,
13     strasse              VARCHAR(255) NOT NULL DEFAULT '',
14     hnr                  VARCHAR(255) NOT NULL DEFAULT '',
15     lkz                  CHAR(2) NOT NULL DEFAULT '',
16     plz                  CHAR(9) NOT NULL DEFAULT '',
17     ort                  VARCHAR(255) NOT NULL DEFAULT '',
18     deleted              TINYINT UNSIGNED NOT NULL DEFAULT 0,
19     PRIMARY KEY(adresse_id)
20 ) ENGINE=InnoDB;    -- Nicht wundern, die Engine ist XtraDB.
21
22 -- kunde
23 SELECT 'Tabelle kunde mit Fremdschlüssel anlegen (Seite 87)';
24 CREATE TABLE kunde (
25     kunde_id            INT UNSIGNED AUTO_INCREMENT,
26     nachname             VARCHAR(255) NOT NULL DEFAULT '',
27     vorname              VARCHAR(255) NOT NULL DEFAULT '',
28     rechnung_adresse_id INT UNSIGNED,
29     liefer_adresse_id   INT UNSIGNED,
30     bezahlart            INT UNSIGNED NOT NULL DEFAULT 0,
31     art                  ENUM('unb', 'prv', 'gsch') NOT NULL DEFAULT 'unb',
32     deleted              TINYINT UNSIGNED NOT NULL DEFAULT 0,
33     PRIMARY KEY(kunde_id),
34     FOREIGN KEY (rechnung_adresse_id)
35     REFERENCES adresse(adresse_id)
36     ON UPDATE CASCADE
37     ON DELETE RESTRICT,
38     FOREIGN KEY (liefer_adresse_id)
39     REFERENCES adresse(adresse_id)
40     ON UPDATE CASCADE
41     ON DELETE SET NULL
42 ) ENGINE=InnoDB;
43
44 -- bank
45 SELECT 'Tabelle bank anlegen (Seite 76)';
46 CREATE TABLE bank (
47     bank_id              CHAR(12),
48     bankname             VARCHAR(255) NOT NULL DEFAULT '',
49     lkz                  CHAR(2) NOT NULL DEFAULT 'DE',
50     deleted              TINYINT UNSIGNED NOT NULL DEFAULT 0,
51     PRIMARY KEY(bank_id)
52 ) ENGINE=InnoDB;
53
54 -- bankverbindung
55 SELECT 'Tabelle bankverbindung mit Fremdschlüssel anlegen (Seite 76)';
56 CREATE TABLE bankverbindung (
57     kunde_id            INT UNSIGNED,
58     bankverbindung_nr  INT UNSIGNED,
59     bank_id              CHAR(12) NOT NULL,
60     kontonummer          CHAR(25) NOT NULL DEFAULT '',
61     iban                 CHAR(34) NOT NULL DEFAULT '',
62     deleted              TINYINT UNSIGNED NOT NULL DEFAULT 0,
63     PRIMARY KEY(kunde_id,bankverbindung_nr),
64     FOREIGN KEY (kunde_id)
```

```
65      REFERENCES kunde(kunde_id)
66      ON UPDATE RESTRICT
67      ON DELETE RESTRICT,
68  FOREIGN KEY (bank_id)
69      REFERENCES bank(bank_id)
70      ON UPDATE RESTRICT
71      ON DELETE RESTRICT
72 ) ENGINE=InnoDB;
73
74
75 -- artikel
76 SELECT 'Tabelle artikel anlegen';
77 CREATE TABLE artikel (
78     artikel_id          INT UNSIGNED AUTO_INCREMENT,
79     bezeichnung         VARCHAR(255) NOT NULL DEFAULT '',
80     einzelpreis        DECIMAL(14,6) NOT NULL DEFAULT 0.0,
81     waehrung           CHAR(3) DEFAULT 'EUR',
82     deleted            TINYINT UNSIGNED NOT NULL DEFAULT 0,
83     PRIMARY KEY(artikel_id)
84 ) ENGINE=InnoDB;
85
86
87 -- warengruppe
88 SELECT 'Tabelle warengruppe anlegen';
89 CREATE TABLE warengruppe (
90     warengruppe_id     INT UNSIGNED AUTO_INCREMENT,
91     bezeichnung        VARCHAR(255) NOT NULL DEFAULT '',
92     deleted            TINYINT UNSIGNED NOT NULL DEFAULT 0,
93     PRIMARY KEY(warengruppe_id)
94 ) ENGINE=InnoDB;
95
96
97 -- lieferant
98 SELECT 'Tabelle lieferant mit Fremdschlüssel anlegen';
99 CREATE TABLE lieferant (
100    lieferant_id        INT UNSIGNED AUTO_INCREMENT,
101    firmenname          VARCHAR(255) NOT NULL DEFAULT '',
102    adresse_id          INT UNSIGNED NOT NULL DEFAULT 0,
103    deleted             TINYINT UNSIGNED NOT NULL DEFAULT 0,
104    PRIMARY KEY(lieferant_id),
105    FOREIGN KEY (adresse_id)
106    REFERENCES adresse(adresse_id)
107    ON UPDATE RESTRICT
108    ON DELETE RESTRICT
109 ) ENGINE=InnoDB;
110
111
112 -- Hilfstabelle artikel zu warengruppe
113 SELECT 'Hilfstabelle artikel_nm_warengruppe anlegen';
114 CREATE TABLE artikel_nm_warengruppe (
115     warengruppe_id     INT UNSIGNED,
116     artikel_id          INT UNSIGNED,
117     PRIMARY KEY(warengruppe_id, artikel_id),
118     FOREIGN KEY (warengruppe_id)
119     REFERENCES warengruppe(warengruppe_id)
120     ON UPDATE RESTRICT
121     ON DELETE RESTRICT,
122     FOREIGN KEY (artikel_id)
123     REFERENCES artikel(artikel_id)
```

```
124      ON UPDATE RESTRICT
125      ON DELETE RESTRICT
126  ) ENGINE=InnoDB;
127
128
129 -- Hilfstabelle artikel zu lieferant
130 SELECT 'Hilfstabelle artikel_nm_lieferant anlegen';
131 CREATE TABLE artikel_nm_lieferant (
132     lieferant_id      INT UNSIGNED,
133     artikel_id        INT UNSIGNED,
134     PRIMARY KEY(lieferant_id, artikel_id),
135     FOREIGN KEY (lieferant_id)
136         REFERENCES lieferant(lieferant_id)
137         ON UPDATE RESTRICT
138         ON DELETE RESTRICT,
139     FOREIGN KEY (artikel_id)
140         REFERENCES artikel(artikel_id)
141         ON UPDATE RESTRICT
142         ON DELETE RESTRICT
143  ) ENGINE=InnoDB;
144
145
146 -- bestellung
147 SELECT 'Tabelle bestellung mit Fremdschlüssel anlegen';
148 CREATE TABLE bestellung (
149     bestellung_id      INT UNSIGNED AUTO_INCREMENT,
150     kunde_id          INT UNSIGNED NOT NULL DEFAULT 0,
151     adresse_id         INT UNSIGNED NOT NULL DEFAULT 0,
152     datum              DATETIME NOT NULL DEFAULT '1000-01-01 00:00:00',
153     status              ENUM('offen', 'versendet', 'angekommen', 'retour', 'bezahlt') NOT NULL DEFAULT 'offen',
154     deleted             TINYINT UNSIGNED NOT NULL DEFAULT 0,
155     PRIMARY KEY(bestellung_id),
156     FOREIGN KEY (kunde_id)
157         REFERENCES kunde(kunde_id)
158         ON UPDATE RESTRICT
159         ON DELETE RESTRICT,
160     FOREIGN KEY (adresse_id)
161         REFERENCES adresse(adresse_id)
162         ON UPDATE RESTRICT
163         ON DELETE RESTRICT
164  ) ENGINE=InnoDB;
165
166 -- position der bestellung
167 SELECT 'Tabelle bestellung_position mit Fremdschlüssel anlegen';
168 CREATE TABLE bestellung_position (
169     bestellung_id      INT UNSIGNED,
170     position_nr        INT UNSIGNED,
171     artikel_id         INT UNSIGNED NOT NULL DEFAULT 0,
172     menge              DECIMAL(14,6) NOT NULL DEFAULT 0.0,
173     deleted             TINYINT UNSIGNED NOT NULL DEFAULT 0,
174     PRIMARY KEY(bestellung_id, position_nr),
175     FOREIGN KEY (bestellung_id)
176         REFERENCES bestellung(bestellung_id)
177         ON UPDATE RESTRICT
178         ON DELETE RESTRICT,
179     FOREIGN KEY (artikel_id)
180         REFERENCES artikel(artikel_id)
181         ON UPDATE RESTRICT
```

```

182      ON DELETE RESTRICT
183  ) ENGINE=InnoDB;
184
185
186  -- rechnung
187  SELECT 'Tabelle rechnung mit Fremdschlüssel anlegen';
188  CREATE TABLE rechnung (
189    rechnung_id          INT UNSIGNED AUTO_INCREMENT,
190    kunde_id             INT UNSIGNED NOT NULL DEFAULT 0,
191    bestellung_id        INT UNSIGNED NOT NULL DEFAULT 0,
192    adresse_id           INT UNSIGNED NOT NULL DEFAULT 0,
193    bezahlart            ENUM('lastschrift', 'rechnung', 'kredit', 'bar') NOT NULL
194              DEFAULT 'lastschrift',
195    datum                DATETIME NOT NULL DEFAULT '1000-01-01 00:00:00',
196    status               ENUM('offen', 'gestellt', 'mahnung1', 'inkasso', 'storno
197              ', 'beglichen') NOT NULL DEFAULT 'offen',
198    rabatt               DECIMAL(6,2) NOT NULL DEFAULT 0.0,
199    skonto               DECIMAL(6,2) NOT NULL DEFAULT 0.0,
200    deleted              TINYINT UNSIGNED NOT NULL DEFAULT 0,
201    PRIMARY KEY(rechnung_id),
202    FOREIGN KEY (kunde_id)
203      REFERENCES kunde(kunde_id)
204      ON UPDATE RESTRICT
205      ON DELETE RESTRICT,
206    FOREIGN KEY (adresse_id)
207      REFERENCES adresse(adresse_id)
208      ON UPDATE RESTRICT
209      ON DELETE RESTRICT
210  ) ENGINE=InnoDB;

211
212  -- position der rechnung
213  SELECT 'Tabelle rechnung_position mit Fremdschlüssel anlegen';
214  CREATE TABLE rechnung_position (
215    rechnung_id          INT UNSIGNED,
216    position_nr          INT UNSIGNED,
217    artikel_id           INT UNSIGNED NOT NULL DEFAULT 0,
218    steuer               DECIMAL(14,6) NOT NULL DEFAULT 19.0,
219    menge                DECIMAL(14,6) NOT NULL DEFAULT 0.0,
220    deleted              TINYINT UNSIGNED NOT NULL DEFAULT 0,
221    PRIMARY KEY(rechnung_id, position_nr),
222    FOREIGN KEY (rechnung_id)
223      REFERENCES rechnung(rechnung_id)
224      ON UPDATE RESTRICT
225      ON DELETE RESTRICT,
226    FOREIGN KEY (artikel_id)
227      REFERENCES artikel(artikel_id)
228      ON UPDATE RESTRICT
229      ON DELETE RESTRICT
230  ) ENGINE=InnoDB;

231  SELECT 'Neue Tabelle mit Struktur einer alten anlegen (Seite 89)';
232  CREATE TABLE wurstbrot LIKE adresse;
233
234  SELECT 'Neue Tabelle sofort wieder löschen, da nicht mehr gebraucht';
235  DROP TABLE wurstbrot;
236
237  SELECT 'Temporäre Tabelle anlegen (Seite 89)';
238  CREATE TEMPORARY TABLE tmp_kunde LIKE kunde;

```

```
239  
240 SHOW TABLES;  
241 DROP TABLE tmp_kunde;  
242 SELECT 'Ende listing02.sql //////////////////////////////';
```

Listing 29.3 mysql/listing03.sql

```
1 -- Ausführen der vorherigen Befehle  
2 SOURCE listing02.sql  
3  
4 SELECT 'Start listing03.sql //////////////////////////////';  
5  
6 -- Seite 95  
7 SHOW INDEX FROM kunde\G  
8  
9 -- Anlegen der Indizes nach Liste auf Seite 97  
10 SELECT 'CREATE INDEX idx_kunde_nachname_vorname';  
11 CREATE INDEX idx_kunde_nachname_vorname  
12   ON kunde (nachname, vorname)  
13 ;  
14  
15 SELECT 'CREATE INDEX idx_bank_bankname';  
16 CREATE INDEX idx_bank_bankname  
17   ON bank (bankname)  
18 ;  
19  
20 SELECT 'CREATE INDEX idx_bankverbindung_bankid_kontonummer';  
21 CREATE INDEX idx_bankverbindung_bankid_kontonummer  
22   ON bankverbindung (bank_id, kontonummer)  
23 ;  
24  
25 SELECT 'CREATE UNIQUE INDEX idx_bankverbindung_iban';  
26 CREATE UNIQUE INDEX idx_bankverbindung_iban  
27   ON bankverbindung (iban)  
28 ;  
29  
30 SELECT 'CREATE INDEX idx_artikel_bezeichnung';  
31 CREATE INDEX idx_artikel_bezeichnung  
32   ON artikel (bezeichnung)  
33 ;  
34  
35 SELECT 'CREATE INDEX idx_warengruppe_bezeichnung';  
36 CREATE INDEX idx_warengruppe_bezeichnung  
37   ON warengruppe (bezeichnung)  
38 ;  
39  
40 SELECT 'CREATE INDEX idx_lieferant_firmenname';  
41 CREATE INDEX idx_lieferant_firmenname  
42   ON lieferant (firmenname)  
43 ;  
44  
45 SELECT 'CREATE INDEX idx_bestellung_kundeid_datum';  
46 CREATE INDEX idx_bestellung_kundeid_datum  
47   ON bestellung (kunde_id, datum)  
48 ;  
49  
50  
51 SELECT 'CREATE UNIQUE INDEX idx_adresse_dublette (Seite 100)';
```

```

52 CREATE UNIQUE INDEX idx_adresse_dublette
53   ON adresse (strasse(100), hnr(100), plz);
54
55 -- Start Selektivität
56 SELECT 'Indexselektivität (Seite 103)';
57 SELECT 'CREATE TEMPORARY TABLE testdaten_bank';
58 DROP TABLE IF EXISTS testdaten_bank;
59 CREATE TEMPORARY TABLE testdaten_bank
60 (
61   blz      CHAR(8),
62   merkmal  CHAR(1),
63   bezeichnung VARCHAR(255),
64   plz      CHAR(5),
65   ort      VARCHAR(255),
66   kurz     VARCHAR(255),
67   pan      CHAR(5),
68   bic      VARCHAR(255),
69   sm       CHAR(2),
70   ds       CHAR(6),
71   kz       CHAR(1),
72   blzl     CHAR(1),
73   blzn     CHAR(8)
74 );
75
76 SELECT 'LOAD DATA';
77 SET GLOBAL local_infile = 1;
78 -- Hier Dateiname und Pfad einfügen
79 LOAD DATA LOCAL INFILE 'blz_20120305.csv'
80   INTO TABLE testdaten_bank
81   FIELDS TERMINATED BY ','
82   LINES TERMINATED BY '\n'
83 ;
84
85 CREATE INDEX idx_testdaten_bank_bezeichnung
86   ON testdaten_bank (bezeichnung(30))
87 ;
88 SHOW INDEX FROM testdaten_bank\G
89
90 -- Ende Selektivität
91
92 -- ENABLE / DISABLE
93 -- SELECT 'DISABLE/ENABLE KEYS: Erwarte zwei Warnungen!';
94 -- ALTER TABLE bankverbindung DISABLE KEYS; SHOW WARNINGS\G
95 -- ALTER TABLE bankverbindung ENABLE KEYS; SHOW WARNINGS\G
96
97 SELECT 'Ende listing03.sql /////////////////////////////////';
```

Listing 29.4 mysql/listing04.sql

```

1 -- Ausführen der vorherigen Befehle
2 SOURCE listing03.sql
3
4 SELECT 'Start listing04.sql /////////////////////////////////';
5 -- Seite 108
6 -- Alternativ: SET @@local_infile = 1;
7 SET GLOBAL local_infile = 1;
8
9 SELECT 'Importieren von artikel01.csv (Seite 108)';
```

```
10 LOAD DATA LOCAL INFILE 'artikel01.csv'  
11   INTO TABLE oshop.artikel  
12   FIELDS  
13     TERMINATED BY ';'  
14   LINES  
15     TERMINATED BY '\n'  
16 ;  
17  
18 SELECT 'Löschen der schlecht importierten Daten (Seite 109)';  
19 DELETE FROM oshop.artikel;  
20  
21 SELECT 'Importieren von artikel01.csv (Seite 109)';  
22 LOAD DATA LOCAL INFILE 'artikel01.csv'  
23   INTO TABLE oshop.artikel  
24   FIELDS  
25     TERMINATED BY ';'  
26   LINES  
27     TERMINATED BY '\n'  
28   IGNORE 1 LINES  
29   (artikel_id, bezeichnung, einzelpreis, waehrung)  
30   SET deleted=0  
31 ;  
32  
33 -- Start REPLACE oder IGNORE  
34 SELECT 'Beispiel artikel02.csv mit IGNORE (Seite 111)';  
35 LOAD DATA LOCAL INFILE 'artikel02.csv' IGNORE  
36   INTO TABLE oshop.artikel  
37   FIELDS  
38     TERMINATED BY ';'  
39   LINES  
40     TERMINATED BY '\n'  
41   IGNORE 1 LINES  
42   (artikel_id, bezeichnung, einzelpreis, waehrung)  
43   SET deleted=0  
44 ;  
45 SELECT * FROM artikel;  
46  
47 SELECT 'Beispiel artikel02.csv mit IGNORE (Seite 111)';  
48 DELETE FROM artikel;  
49  
50 LOAD DATA LOCAL INFILE 'artikel01.csv'  
51   INTO TABLE oshop.artikel  
52   FIELDS  
53     TERMINATED BY ';'  
54   LINES  
55     TERMINATED BY '\n'  
56   IGNORE 1 LINES  
57   (artikel_id, bezeichnung, einzelpreis, waehrung)  
58   SET deleted=0  
59 ;  
60  
61 LOAD DATA LOCAL INFILE 'artikel02.csv' REPLACE  
62   INTO TABLE oshop.artikel  
63   FIELDS  
64     TERMINATED BY ';'  
65   LINES  
66     TERMINATED BY '\n'  
67   IGNORE 1 LINES  
68   (artikel_id, bezeichnung, einzelpreis, waehrung)
```

```
69     SET deleted=0
70 ;
71
72 SELECT * FROM artikel;
73
74 -- Ende REPLACE oder IGNORE
75
76 -- Start INSERT
77
78 SELECT 'Anlegen mit INSERT INTO ... VALUES (' Seite 113 ')';
79 SELECT 'Warengruppen anlegen';
80 INSERT INTO warengruppe (warengruppe_id, bezeichnung)
81   VALUES
82     (1, 'Bürobedarf')
83     ,(2, 'Pflanzen')
84     ,(3, 'Gartenbedarf')
85     ,(4, 'Werkzeug')
86 ;
87 SELECT * FROM warengruppe;
88
89 SELECT 'Anlegen mit INSERT INTO ... SET (' Seite 115 ')';
90 SELECT 'Warengruppen verbinden';
91 INSERT INTO artikel_nm_warengruppe
92   SET
93     artikel_id=3001
94     ,warengruppe_id=1
95 ;
96 INSERT INTO artikel_nm_warengruppe
97   SET
98     artikel_id=3005
99     ,warengruppe_id=1
100 ;
101 INSERT INTO artikel_nm_warengruppe
102   SET
103     artikel_id=3006
104     ,warengruppe_id=1
105 ;
106 INSERT INTO artikel_nm_warengruppe
107   SET
108     artikel_id=3007
109     ,warengruppe_id=1
110 ;
111 INSERT INTO artikel_nm_warengruppe
112   SET
113     artikel_id=3010
114     ,warengruppe_id=1
115 ;
116 INSERT INTO artikel_nm_warengruppe
117   SET
118     artikel_id=7856
119     ,warengruppe_id=2
120 ;
121 INSERT INTO artikel_nm_warengruppe
122   SET
123     artikel_id=7856
124     ,warengruppe_id=3
125 ;
126 INSERT INTO artikel_nm_warengruppe
127   SET
```

```
128     artikel_id=7863
129     ,warengruppe_id=2
130 ;
131 INSERT INTO artikel_nm_warengruppe
132     SET
133     artikel_id=7863
134     ,warengruppe_id=3
135 ;
136 INSERT INTO artikel_nm_warengruppe
137     SET
138     artikel_id=9010
139     ,warengruppe_id=3
140 ;
141 INSERT INTO artikel_nm_warengruppe
142     SET
143     artikel_id=9010
144     ,warengruppe_id=4
145 ;
146 INSERT INTO artikel_nm_warengruppe
147     SET
148     artikel_id=9015
149     ,warengruppe_id=3
150 ;
151 INSERT INTO artikel_nm_warengruppe
152     SET
153     artikel_id=9015
154     ,warengruppe_id=4
155 ;
156 -- Ende INSERT
157
158 -- Start Kundendaten
159 SELECT 'Testdaten INSERT adresse';
160
161 -- adresse
162 INSERT INTO adresse (adresse_id, strasse, hnr, lkz, plz, ort)
163 VALUES
164     (1, 'Beutelhaldenweg', '5', 'AL', '67676', 'Hobbingen')
165     ,(2, 'Beutelhaldenweg', '1', 'AL', '67676', 'Hobbingen')
166     ,(3, 'Auf der Feste', '1', 'GO', '54786', 'Minas Tirith')
167     ,(4, 'Letztes Haus', '4', 'ER', '87567', 'Bruchtal')
168     ,(5, 'Baradur', '1', 'MO', '62519', 'Lugburz')
169     ,(10, 'Hochstrasse', '4a', 'DE', '44879', 'Bochum')
170     ,(11, 'Industriegebiet', '8', 'DE', '44878', 'Bochum')
171 ;
172
173 SELECT 'Testdaten INSERT kunde';
174 -- kunde
175 INSERT INTO kunde (kunde_id, rechnung_adresse_id, nachname, vorname, art)
176 VALUES
177     (1, 1, 'Gamdschie', 'Samweis', 'prv')
178     ,(2, 2, 'Beutlin', 'Frodo', 'prv')
179     ,(3, 2, 'Beutlin', 'Bilbo', 'prv')
180     ,(4, 3, 'Telcontar', 'Elessar', 'prv')
181     ,(5, 4, 'Earendilionn', 'Elrond', 'gsch')
182 ;
183
184 SELECT 'Lieferadresse bei zwei Adressen ändern';
185 UPDATE kunde SET liefer_adresse_id = 2 WHERE kunde_id = 1;
186 UPDATE kunde SET liefer_adresse_id = 4 WHERE kunde_id = 3;
```

```
187 -- Ende Kundendaten
188
189 -- Start Aufbau von 2 Bestellungen
190 SELECT 'Aufbau von 2 Bestellungen: Bestelldaten';
191 INSERT INTO bestellung (kunde_id, adresse_id, datum)
192     VALUES
193         (1, 1, '2012-03-24 17:41:00')
194         ,(2, 2, '2012-03-23 16:11:00')
195 ;
196
197 SELECT 'Aufbau von 2 Bestellungen: Bestellpositionen';
198 INSERT INTO bestellung_position (bestellung_id, position_nr, artikel_id, menge
199 )
200     VALUES
201         (1, 1, 7856, 30)
202         ,(1, 2, 7863, 50)
203         ,(1, 3, 9015, 1)
204         ,(2, 1, 7856, 10)
205         ,(2, 2, 9010, 5)
206 ;
207 -- Ende Aufbau Bestellung
208
209 -- SELECT 'Drei Fehlermeldung wegen Constraints (Seite 116)!';
210 SELECT bestellung_id, kunde_id, adresse_id, datum FROM bestellung;
211 -- INSERT INTO bestellung (datum) VALUES (NOW());
212 -- INSERT INTO bestellung (adresse_id, datum) VALUES (10, NOW());
213 -- INSERT INTO adresse (adresse_id, strasse) VALUES (10, 'Oberer Weg');
214
215 -- Bildtabelle
216 SELECT 'Bildtabelle anlegen (Seite 117)';
217 CREATE TABLE bild
218 (
219     bild_id      INT          UNSIGNED AUTO_INCREMENT PRIMARY KEY,
220     bild        BLOB,
221     dateiname    VARCHAR(255),
222     dateityp     VARCHAR(255),
223     dateigroesse BIGINT       UNSIGNED,
224     artikel_id   INT          UNSIGNED REFERENCES artikel(artikel_id)
225 );
226
227 SELECT 'Bild importieren (Seite 119)';
228 INSERT INTO bild
229     SET
230         bild = LOAD_FILE('/var/lib/mysql/interchange/spaten01.png'),
231         dateiname = 'spaten01.png',
232         dateityp = 'png',
233         dateigroesse = 7168,
234         artikel_id = 9015
235 ;
236
237 INSERT INTO bild
238     SET
239         bild = LOAD_FILE('/var/lib/mysql/interchange/spaten02.png'),
240         dateiname = 'spaten02.png',
241         dateityp = 'png',
242         dateigroesse = 6910,
243         artikel_id = 9015
244 ;
```

```
245 INSERT INTO bild
246   SET
247     bild = LOAD_FILE('/var/lib/mysql/interchange/spaten03.png'),
248     dateiname = 'spaten03.png',
249     dateityp = 'png',
250     dateigroesse = 7095,
251     artikel_id = 9015
252 ;
253
254 SELECT 'Daten kopieren (Seite 121)';
255 DROP TABLE IF EXISTS kunde_beutlin;
256 CREATE TEMPORARY TABLE kunde_beutlin LIKE kunde;
257 INSERT INTO kunde_beutlin
258   SELECT * FROM kunde WHERE nachname='Beutlin';
259 SELECT * FROM kunde_beutlin;
260
261 SELECT 'Daten kopieren (Seite 121)';
262 DROP TABLE IF EXISTS kunde_namen;
263 CREATE TEMPORARY TABLE kunde_namen LIKE kunde;
264 INSERT INTO kunde_namen (vorname, nachname)
265   SELECT vorname, nachname FROM kunde;
266 SELECT * FROM kunde_namen;
267
268
269 -- Start DROP (auskommentiert)
270 -- SELECT * FROM warengruppe INTO OUTFILE 'bla.txt';
271 -- DROP DATABASE oshop;
272 -- SELECT 'Löschen Sie nun MySQL-Datenverzeichnis oshop/bla.txt!';
273 -- Ende DROP
274
275 SELECT 'Ende listing04.sql /////////////////////////////////';
```

29.1.2 Quelltexte zu Teil III

Listing 29.5 mysql/listing05.sql

```
1 -- Ausführen der vorherigen Befehle
2 SOURCE listing04.sql
3
4 SELECT 'Start listing05.sql /////////////////////////////////';
5
6 SELECT 'Aktuelle Datenbankeinstellung (Seite 126)';
7 SHOW CREATE DATABASE oshop\G
8
9 SELECT 'DATABASE löschen (Seite 127)';
10 CREATE DATABASE wurstbrot;
11 SHOW DATABASES;
12 DROP DATABASE wurstbrot;
13 SHOW DATABASES;
14
15
16 -- Start Tabellen umbenennen
17 SELECT 'Tabellen umbenennen (Seite 130)';
18 DROP TABLE IF EXISTS wurstbrot;
19 SHOW TABLES;
20 ALTER TABLE adresse RENAME TO wurstbrot;
```

```
21 SHOW TABLES;
22 SHOW CREATE TABLE kunde\G
23 ALTER TABLE wurstbrot RENAME TO adresse;
24 -- Ende Tabellen umbenennen
25
26
27 -- Start Spalten verändern
28 SELECT 'Spalte hinzufügen (Seite 131)';
29 DESCRIBE kunde;
30 ALTER TABLE kunde
31     ADD firmenname VARCHAR(255) NOT NULL DEFAULT '' AFTER vorname
32 ;
33 DESCRIBE kunde;
34
35 SELECT 'Spalte Datentyp ändern (Seite 133)';
36 ALTER TABLE kunde
37     MODIFY
38         bezahlart ENUM('rechnung', 'bankeinzug', 'nachname') DEFAULT 'rechnung'
39 ;
40 DESCRIBE kunde;
41
42 SELECT 'Spaltenlänge ändern (Seite 133)';
43 DROP TABLE IF EXISTS bla ;
44 CREATE TEMPORARY TABLE bla LIKE artikel;
45 INSERT INTO bla SELECT * FROM artikel;
46 -- ALTER TABLE bla MODIFY bezeichnung CHAR(10);
47 SELECT MAX(CHAR_LENGTH(bezeichnung)) FROM artikel;
48
49 SELECT 'Zeichen nach Zahl ändern (Seite 134)';
50 DROP TABLE IF EXISTS wurstbrot ;
51 CREATE TEMPORARY TABLE wurstbrot LIKE adresse;
52 INSERT INTO wurstbrot SELECT * FROM adresse;
53 SELECT * FROM wurstbrot;
54 ALTER TABLE wurstbrot MODIFY plz INT;
55 SELECT * FROM wurstbrot;
56
57 -- ALTER TABLE wurstbrot MODIFY plz TINYINT;
58 -- ALTER TABLE wurstbrot MODIFY hnr INT;
59 -- ELECT * FROM wurstbrot;
60
61 SELECT 'Datum- und Zeitspalten ändern (Seite 136)';
62 DROP TABLE IF EXISTS wurstbrot ;
63 CREATE TEMPORARY TABLE wurstbrot
64 (
65     a DATETIME,
66     b TIME,
67     c YEAR
68 );
69
70 INSERT INTO wurstbrot (a, b, c)
71     VALUES ('2012-03-24 14:57:00', '14:57:00', '2012')
72 ;
73
74 ALTER TABLE wurstbrot
75     MODIFY a BIGINT,
76     MODIFY b BIGINT,
77     MODIFY c BIGINT
78 ;
79 SELECT * FROM wurstbrot;
```

```
80
81 ALTER TABLE wurstbrot
82   MODIFY a DATETIME,
83   MODIFY b TIME,
84   MODIFY c YEAR
85 ;
86 SELECT * FROM wurstbrot;
87
88 ALTER TABLE wurstbrot
89   MODIFY a VARCHAR(255),
90   MODIFY b VARCHAR(255),
91   MODIFY c VARCHAR(255)
92 ;
93 SELECT * FROM wurstbrot;
94
95 ALTER TABLE wurstbrot
96   MODIFY a DATETIME,
97   MODIFY b TIME,
98   MODIFY c YEAR
99 ;
100 SELECT * FROM wurstbrot;
101
102 SELECT 'Spalten löschen (Seite 137)';
103 DROP TABLE IF EXISTS wurstbrot;
104 CREATE TEMPORARY TABLE wurstbrot AS SELECT * FROM adresse;
105 DESCRIBE wurstbrot;
106 ALTER TABLE wurstbrot
107   DROP deleted,
108   DROP strasse,
109   DROP ort
110 ;
111 DESCRIBE wurstbrot;
112
113 -- Ende Spalten verändern
114
115 -- Start Tabellen löschen
116 SELECT 'Tabelle löschen (Seite 139)';
117 SHOW TABLES;
118 -- DROP TABLE
119 --   rechnung_position
120 --   ,rechnung
121 --   ,bestellung_position
122 --   ,bestellung
123 -- ;
124 SHOW TABLES;
125 -- Ende Tabellen löschen
126 SELECT 'Ende listing05.sql //////////////////////////////';
```

Listing 29.6 mysql/listing06.sql

```
1 -- Ausführen der vorherigen Befehle
2 SOURCE listing05.sql
3
4 SELECT 'Start listing06.sql //////////////////////////////';
5
6 SELECT 'WHERE mit TRUE / FALSE (Seite 142)';
7 SELECT * FROM artikel WHERE TRUE;
8 SELECT * FROM artikel WHERE FALSE;
```

```
9
10
11 -- Start Groß-/Kleinschreibung
12 SELECT 'Groß-/Kleinschreibung (Seite 143)';
13 DROP TABLE IF EXISTS wurstbrot;
14 CREATE TEMPORARY TABLE wurstbrot (
15     name VARCHAR(255)
16 );
17
18 INSERT INTO wurstbrot
19     VALUES
20         ('Jaqueline')
21     ,('Kevin')
22     ,('kevin')
23     ,('jaqueline')
24 ;
25
26 SELECT * FROM wurstbrot WHERE name = 'kevin';
27
28 DROP TABLE wurstbrot;
29
30 CREATE TEMPORARY TABLE wurstbrot (
31     name VARCHAR(255) BINARY
32 );
33
34 INSERT INTO wurstbrot
35     VALUES
36         ('Jaqueline')
37     ,('Kevin')
38     ,('kevin')
39     ,('jaqueline')
40 ;
41
42 SELECT * FROM wurstbrot WHERE name = 'kevin';
43 -- Ende Groß- Kleinschreibung
44
45 -- Start Einfache Wertzuweisung
46 SELECT 'Einfache Wertzuweisung (Seite 148)';
47 SELECT * FROM bestellung_position;
48 UPDATE bestellung_position
49 SET menge = 2
50 WHERE bestellung_id = 1 AND position_nr = 3
51 ;
52 SELECT * FROM bestellung_position;
53 -- Ende Einfache Wertzuweisung
54
55 -- Start Wertberechnung
56 SELECT 'Wertberechnung (Seite 148)';
57 SELECT * FROM artikel;
58 UPDATE artikel SET einzelpreis = einzelpreis + einzelpreis / 100.0;
59 SELECT * FROM artikel;
60 UPDATE artikel SET einzelpreis = ROUND(einzelpreis, 2);
61 SELECT * FROM artikel;
62 -- Ende Wertberechnung
63
64 -- Start Gebastelte Zeichenketten
65 SELECT 'Gebastelte Zeichenketten: Anrede (Seite 149)';
66 ALTER TABLE kunde ADD anrede VARCHAR(255) AFTER nachname;
67 UPDATE kunde
```

```

68 SET anrede = CONCAT('Sehr geehrte/r Frau/Herr ', nachname)
69 WHERE nachname <> ''
70 ;
71 SELECT * FROM kunde;
72 -- Ende Gebastelte Zeichenketten
73
74
75 -- Start DELETE
76 SELECT 'Löschen mit DELETE (Seite 150)';
77 SELECT * FROM bestellung_position;
78 DELETE FROM bestellung_position WHERE bestellung_id = 1;
79 SELECT * FROM bestellung_position;
80
81 -- Start Wiederherstellen der Daten
82 INSERT INTO bestellung_position (
83 bestellung_id, position_nr, artikel_id, menge)
84 VALUES
85     (1, 1, 7856, 30)
86     ,(1, 2, 7863, 50)
87     ,(1, 3, 9015, 1)
88 ;
89 -- Ende Wiederherstellen der Daten
90
91 -- SELECT 'Löschversuch mit Constraint.';
92 -- SELECT 'Erwarte eine Fehlermeldung! (Seite 152)';
93 -- DELETE FROM kunde WHERE kunde_id = 1;
94
95
96 SELECT 'Test AUTO_INCREMENT (Seite 152)';
97 SELECT kunde_id, nachname FROM kunde;
98 DELETE FROM kunde WHERE kunde_id IN (3,5);
99 INSERT INTO kunde (nachname, vorname) VALUES ('Eichenschild', 'Thorin');
100 SELECT kunde_id, nachname FROM kunde;
101 -- Start Wiederherstellen der Daten
102 INSERT INTO kunde (kunde_id, rechnung_adresse_id, nachname, vorname, art)
103 VALUES
104     (3, 2, 'Beutlin',      'Bilbo',    'prv')
105     ,(5, 4, 'Earendillionn','Elrond',   'gsch')
106 ;
107 -- Ende Wiederherstellen der Daten
108
109 SELECT 'Tabelle leeren (Seite 154)';
110 -- DELETE FROM bestellung_position;
111 -- TRUNCATE bestellung_position;
112 -- Ende DELETE
113 SELECT 'Ende listing06.sql /////////////////////////////////';
```

29.1.3 Quelltexte zu Teil IV

Listing 29.7 mysql/listing07.sql

```

1 -- Ausführen der vorherigen Befehle
2 SOURCE listing06.sql
3
4 SELECT 'Start listing07.sql /////////////////////////////////';
5
```

```
6  SELECT 'Ausdrücke (Seite 158)';
7
8  SELECT 5, 'Beginn der Auswertung';
9  SELECT -5, 9 + 4, 9 - 4, 9 * 4, 9 / 4, 9 DIV 4, 9 % 4, 9 MOD 4;
10 SELECT 3 * 4 DIV 3, (3 * 4) DIV 3, 3 * (4 DIV 3);
11 SELECT 9 / 0, 9 * NULL, SQRT(-5);
12
13 SELECT 'Zufallszahlen (Seite 160)';
14 SELECT RAND(1);
15 SELECT RAND(1);
16 SELECT RAND(1) FROM artikel;
17 -- FLOOR(a + RAND() * (b - a))
18
19 SELECT 'Variablen (Seite 161)';
20
21 SELECT POWER(2,8) INTO @x;
22 SELECT @x - 1;
23
24 -- SELECT 'Ungültiger Versuch: erwarte eine Fehlermeldung (Seite 161)';
25 -- SELECT einzelpreis FROM artikel INTO @y;
26
27 SET @artnr=3010;
28 SELECT * FROM artikel WHERE artikel_id=@artnr;
29
30
31 SELECT 'Spalten- und Zeilenwahl (Seite 162)';
32 SELECT * FROM artikel;
33 SELECT
34   artikel_id, bezeichnung, einzelpreis, waehrung, deleted
35   FROM artikel
36 ;
37
38 SELECT
39   artikel_id + 10000 AS Unsinn, deleted AS Löschkennzeichen
40   FROM artikel
41 ;
42
43 SELECT
44   artikel_id, bezeichnung
45   FROM artikel
46   WHERE einzelpreis BETWEEN 10 AND 100
47 ;
48
49 SELECT 'Sortierung (Seite 163)';
50 SELECT
51   artikel_id, bezeichnung, einzelpreis
52   FROM artikel
53   ORDER BY einzelpreis ASC
54 ;
55
56 SELECT
57   nachname, vorname
58   FROM kunde
59   ORDER BY nachname, vorname
60 ;
61
62 SELECT
63   nachname, vorname
64   FROM kunde
```

```
65 ORDER BY nachname DESC, vorname ASC
66 ;
67
68 DROP TABLE IF EXISTS wurstbrot;
69 SET NAMES cp850;
70 CREATE TEMPORARY TABLE wurstbrot ( name VARCHAR(255) CHARACTER SET 'cp850' );
71 INSERT INTO wurstbrot
72 VALUES
73     ('winfried')
74     ,('achim')
75     ,('olga')
76     ,('zechine')
77     ,('ägidius')
78 ;
79 SELECT * FROM wurstbrot ORDER BY name;
80
81 DROP TABLE IF EXISTS wurstbrot;
82 SET NAMES utf8;
83 CREATE TEMPORARY TABLE wurstbrot ( name VARCHAR(255) CHARACTER SET 'utf8' );
84 INSERT INTO wurstbrot
85 VALUES
86     ('winfried')
87     ,('achim')
88     ,('olga')
89     ,('zechine')
90     ,('ägidius')
91 ;
92 SELECT * FROM wurstbrot ORDER BY name;
93
94 DROP TABLE IF EXISTS wurstbrot;
95 CREATE TEMPORARY TABLE wurstbrot (name VARCHAR(255));
96 INSERT INTO wurstbrot
97 VALUES
98     ('winfried')
99     ,('achim')
100    ,('olga')
101    ,('Olga')
102    ,('zechine')
103 ;
104 SELECT * FROM wurstbrot ORDER BY name;
105 SELECT * FROM wurstbrot ORDER BY name DESC;
106
107 DROP TABLE IF EXISTS wurstbrot;
108 CREATE TEMPORARY TABLE wurstbrot (name VARCHAR(255) BINARY);
109 INSERT INTO wurstbrot
110 VALUES
111     ('winfried')
112     ,('achim')
113     ,('olga')
114     ,('Olga')
115     ,('zechine')
116 ;
117 SELECT * FROM wurstbrot ORDER BY name;
118 SELECT * FROM wurstbrot ORDER BY name DESC;
119
120
121 SELECT
122     bestellung_id, datum
123     FROM bestellung
```

```

124 ORDER BY datum;
125
126 SELECT
127 bestellung_id, TIME(datum) Uhrzeit
128 FROM bestellung
129 ORDER BY uhrzeit;
130
131 EXPLAIN SELECT * FROM kunde ORDER BY nachname, vorname\G
132
133 SELECT 'Mehrfachausgaben unterbinden (Seite 173)';
134
135 SELECT ort FROM adresse ORDER BY ort;
136 SELECT DISTINCT ort FROM adresse ORDER BY ort;
137 SELECT DISTINCT nachname FROM kunde ORDER BY nachname;
138
139 -- Start Bankimport
140 SELECT 'Bankdaten importieren (Seite 174)';
141 DROP TABLE IF EXISTS bank_import;
142 CREATE TEMPORARY TABLE bank_import (
143 bankleitzahl CHAR(8),
144 bezeichnung VARCHAR(255)
145 );
146
147 LOAD DATA LOCAL INFILE 'blz_20120305.csv' -- Hier ggf. anpassen!
148 INTO TABLE bank_import
149 FIELDS
150 TERMINATED BY ';'
151 LINES
152 TERMINATED BY '\n'
153 IGNORE 1 LINES
154 (bankleitzahl, @dummy, bezeichnung)
155 ;
156 -- SELECT DISTINCT * FROM bank_import;
157 -- SELECT DISTINCT bankleitzahl FROM bank_import;
158 -- SELECT DISTINCT bankleitzahl, bezeichnung FROM bank_import;
159
160 ALTER TABLE bankverbindung
161 DROP FOREIGN KEY bankverbindung_ibfk_1,
162 DROP FOREIGN KEY bankverbindung_ibfk_2,
163 DROP INDEX idx_bankverbindung_bankid_kontonummer,
164 DROP PRIMARY KEY
165 ;
166 ALTER TABLE bank
167 DROP PRIMARY KEY
168 ;
169
170 ALTER TABLE bank
171 MODIFY bank_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
172 ADD blz CHAR(12) NOT NULL DEFAULT '' AFTER bank_id
173 ;
174
175 ALTER TABLE bankverbindung
176 MODIFY bank_id INT UNSIGNED,
177 ADD PRIMARY KEY(kunde_id, bankverbindung_nr),
178 ADD INDEX idx_bankverbindung_bankid_kontonummer (bank_id, kontonummer),
179 ADD FOREIGN KEY (kunde_id)
180 REFERENCES kunde(kunde_id)
181 ON UPDATE RESTRICT
182 ON DELETE RESTRICT,

```

```
183 ADD FOREIGN KEY (bank_id)
184 REFERENCES bank(bank_id)
185 ON UPDATE RESTRICT
186 ON DELETE RESTRICT
187 ;
188
189 INSERT INTO bank (blz, bankname)
190 SELECT DISTINCT bankleitzahl, bezeichnung FROM bank_import
191 ;
192
193 CREATE INDEX idx_bank_blbzbankname
194 ON bank (blz, bankname)
195 ;
196
197 -- Beginn Für die vorhandenen Kunden eine Bankverbindung bauen
198
199 INSERT INTO bankverbindung (kunde_id, bankverbindung_nr, bank_id, kontonummer,
200 iban)
201 VALUES
202 (1, 1, 2, '1111111111', '100100101111111111'),
203 (1, 2, 2, '1111111112', '100100101111111112'),
204 (2, 1, 35, '2222222221', '100601982222222221'),
205 (3, 1, 35, '3333333331', '100601983333333331'),
206 (4, 1, 90, '4444444441', '120700004444444441'),
207 (5, 1, 90, '5555555551', '120700005555555551')
208 ;
209 -- Ende Für die vorhandenen Kunden eine Bankverbindung bauen
210
211 -- Ende Bankimport
212
213 SELECT 'Ausschneiden mit LIMIT (Seite 177)';
214 -- SELECT * FROM bank;
215 SELECT blz, bankname FROM bank ORDER BY blz LIMIT 3;
216 SELECT
217 blz, bankname
218 FROM bank
219 ORDER BY blz DESC
220 LIMIT 1
221 ;
222 SELECT
223 blz
224 FROM bank
225 ORDER BY blz DESC
226 LIMIT 1
227 INTO @blzmin
228 ;
229 SELECT
230 bankname
231 FROM bank
232 WHERE blz = @blzmin
233 ;
234 SELECT blz, bankname FROM bank ORDER BY blz LIMIT 2 OFFSET 1;
235 SELECT blz, bankname FROM bank ORDER BY blz LIMIT 1;
236 SELECT blz, bankname FROM bank ORDER BY blz LIMIT 1 OFFSET 0;
237
238 SELECT 'CSV-Datei exportieren (Seite 180)';
239
240 -- SELECT
```

```

241 -- blz Bankleitzahl, bankname Bankname
242 -- INTO OUTFILE '/var/lib/mysql/interchange/bank.csv'
243 -- FIELDS
244 -- TERMINATED BY ';'
245 -- ENCLOSED BY ""
246 -- LINES
247 -- TERMINATED BY '\r\n'
248 -- FROM
249 -- bank
250 -- ORDER BY
251 -- Bankleitzahl, Bankname;
252 SELECT 'Ende listing07.sql /////////////////////////////////';

```

Listing 29.8 mysql/listing08.sql

```

1 -- Ausführen der vorherigen Befehle
2 SOURCE listing07.sql
3
4 SELECT 'Start listing08.sql /////////////////////////////////';
5
6 -- Start INNER JOIN
7 SELECT 'CROSS JOIN (Seite 184)';
8 SELECT
9   kunde_id, nachname, vorname, rechnung_adresse_id, adresse_id
10  FROM
11    kunde, adresse
12 ;
13
14 SELECT 'CROSS JOIN mit WHERE (Seite ??)';
15 SELECT
16   kunde_id, nachname, vorname, rechnung_adresse_id, adresse_id
17  FROM
18    kunde, adresse
19 WHERE
20   rechnung_adresse_id = adresse_id
21 ;
22
23 SELECT 'INNER JOIN (Seite 187)';
24 SELECT
25   kunde_id, nachname, vorname, rechnung_adresse_id, adresse_id
26  FROM
27    kunde INNER JOIN adresse
28    ON rechnung_adresse_id = adresse_id
29 ;
30
31 SELECT
32   kunde_id, kontonummer, blz, bankname
33  FROM
34    bankverbindung INNER JOIN bank
35    ON bankverbindung.bank_id = bank.bank_id
36 ORDER BY
37   kunde_id, kontonummer
38 ;
39 SELECT
40   nachname, vorname, kontonummer
41  FROM
42    bankverbindung INNER JOIN kunde ON bankverbindung.kunde_id = kunde.kunde_id
43 ORDER BY
44   nachname, vorname, kontonummer

```

```
45 ;
46
47 SELECT 'Abkürzung mit USING (Seite 192)';
48 SELECT
49   kunde_id, kontonummer, blz, bankname
50   FROM
51     bankverbindung INNER JOIN bank USING (bank_id)
52 ;
53
54 SELECT 'Abkürzung mit NATURAL JOIN (Seite 192)';
55 SELECT
56   kunde_id, kontonummer, blz, bankname
57   FROM
58     bankverbindung NATURAL JOIN bank
59 ;
60 -- Ende INNER JOIN
61
62 -- Start Datenquelle
63 SELECT 'JOIN als Datenquelle (Seite 193)';
64 SELECT
65   k.kunde_id, k.nachname, k.vorname, a.strasse, a.hnr, a.plz, a.ort
66   FROM
67   kunde k INNER JOIN adresse a
68     ON k.rechnung_adresse_id = a.adresse_id
69 ;
70 DROP TABLE IF EXISTS tmp_kadresse;
71 CREATE TEMPORARY TABLE tmp_kadresse
72 SELECT
73   k.kunde_id, k.nachname, k.vorname, a.strasse, a.hnr, a.plz, a.ort
74   FROM
75   kunde k INNER JOIN adresse a
76     ON k.rechnung_adresse_id = a.adresse_id
77 ;
78 SELECT
79   t.kunde_id, t.nachname, t.ort, b.bestellung_id, DATE(b.datum)
80   FROM
81   bestellung b INNER JOIN tmp_kadresse t USING (kunde_id)
82 ;
83 DROP TABLE IF EXISTS tmp_kbank;
84 CREATE TEMPORARY TABLE tmp_kbank
85 SELECT
86   bv.kunde_id, bv.bankverbindung_nr, bv.kontonummer,
87   bv.iban, ba.blz, ba.bankname
88   FROM
89   bankverbindung bv INNER JOIN bank ba USING (bank_id)
90 ;
91 DROP TABLE IF EXISTS tmp_kbankeinzug;
92 CREATE TEMPORARY TABLE tmp_kbankeinzug
93 SELECT
94   ka.* , kb.bankverbindung_nr, kb.kontonummer,
95   kb.iban, kb.blz, kb.bankname
96   FROM
97   tmp_kadresse ka INNER JOIN tmp_kbank kb USING(kunde_id);
98
99 SELECT kunde_id, nachname, ort, bankname FROM tmp_kbankeinzug ;
100 -- Ende Datenquelle
101
102 -- Start Keine Schlüssel
103 SELECT 'Ohne Schlüssel (Seite 196)';
```

```

104 DESCRIBE tmp_kadresse;
105 DESCRIBE tmp_kbank;
106 SELECT
107 k.kunde_id, k.vorname, a.strasse, a.hnr, a.plz, a.ort
108 FROM
109     kunde k INNER JOIN adresse a
110         ON k.rechnung_adresse_id <= a.adresse_id
111 ;
112 -- Ende Kein Schlüssel
113
114 -- Start mehr als zwei Tabellen
115 SELECT 'Mit mehr als 2 Tabellen (Seite 198)';
116 SELECT
117     a.bezeichnung, nm.warengruppe_id
118 FROM
119     artikel_nm_warengruppe nm NATURAL JOIN artikel a
120 ;
121 SELECT
122     w.bezeichnung, nm.artikel_id
123 FROM
124     artikel_nm_warengruppe nm NATURAL JOIN warengruppe w
125 ;
126 SELECT
127     w.bezeichnung, nm.artikel_id
128 FROM
129     (artikel_nm_warengruppe nm INNER JOIN warengruppe w USING(warengruppe_id))
130 ;
131 SELECT
132     w.bezeichnung, a.bezeichnung
133 FROM
134     (artikel_nm_warengruppe nm INNER JOIN warengruppe w USING (warengruppe_id)
135             INNER JOIN artikel a USING(artikel_id))
136 ;
137 SELECT
138     w.bezeichnung Warengruppe, a.bezeichnung Artikel
139 FROM
140     artikel_nm_warengruppe nm INNER JOIN warengruppe w USING (warengruppe_id)
141             INNER JOIN artikel a USING(artikel_id)
142 ;
143 -- Ende mehr als zwei Tabellen
144
145 -- Start Aufbau Lieferanten
146 SELECT 'Aufbau der Lieferanten';
147 INSERT INTO lieferant (firmenname, adresse_id)
148 VALUES
149     ('Gartenbedarf AllesGrün', 10)
150     ,('Office International', 11)
151     ,('Bürohengst GmbH', 11)
152 ;
153
154 INSERT INTO artikel_nm_lieferant (lieferant_id, artikel_id)
155 VALUES
156     (1, 7856)
157     ,(1, 7863)
158     ,(1, 9010)
159     ,(1, 9015)
160     ,(3, 3001)
161     ,(3, 3005)
162     ,(3, 3006)

```

```
163      ,(3, 3007)
164      ,(3, 3010)
165 ;
166 -- Ende Aufbau Lieferanten
167
168 -- Start OUTER JOIN
169 SELECT 'OUTER JOIN (Seite 200)';
170 SELECT
171   k.kunde_id, k.nachname, k.vorname, b.datum
172   FROM
173     bestellung b INNER JOIN kunde k USING (kunde_id)
174 ORDER BY
175   k.nachname, k.vorname
176 ;
177 SELECT
178   k.kunde_id, k.nachname, k.vorname, b.datum
179   FROM
180     bestellung b RIGHT OUTER JOIN kunde k USING (kunde_id)
181 ORDER BY
182   k.nachname, k.vorname
183 ;
184 SELECT
185   k.kunde_id, k.nachname, k.vorname, b.datum
186   FROM
187     kunde k LEFT OUTER JOIN bestellung b USING (kunde_id)
188 ORDER BY
189   k.nachname, k.vorname
190 ;
191 SELECT
192   l.lieferant_id, l.firmenname, a.artikel_id, a.bezeichnung
193   FROM
194     artikel_nm_lieferant nm INNER JOIN artikel a USING(artikel_id)
195           INNER JOIN lieferant l USING(lieferant_id)
196 ORDER BY firmenname
197 ;
198 SELECT
199   l.lieferant_id, l.firmenname, a.artikel_id, a.bezeichnung
200   FROM
201     artikel_nm_lieferant nm INNER JOIN artikel a USING(artikel_id)
202           RIGHT JOIN lieferant l USING(lieferant_id)
203 ORDER BY firmenname
204 ;
205 SELECT
206   l.firmenname
207   FROM
208     artikel_nm_lieferant nm INNER JOIN artikel a USING(artikel_id)
209           RIGHT JOIN lieferant l USING(lieferant_id)
210 WHERE artikel_id IS NULL
211 ORDER BY firmenname
212 ;
213 SELECT k.kunde_id, k.nachname, k.vorname
214   FROM
215     bestellung b RIGHT JOIN kunde k USING(kunde_id)
216 WHERE b.bestellung_id IS NULL
217 ;
218 -- Ende OUTER JOIN
219
220 -- Start Aufbau Forum
221 SELECT 'SELF JOIN (Seite 205)' Hinweis;
```

```

222 SELECT 'CREATE account';
223 CREATE TABLE account (
224     kunde_id INT UNSIGNED NOT NULL DEFAULT 0,
225     name VARCHAR(255) NOT NULL DEFAULT '',
226     passwort CHAR(34) NOT NULL DEFAULT '',
227     status ENUM('aktiv','gesperrt') NOT NULL DEFAULT 'aktiv',
228     kommentar TEXT ,
229     admin BOOL          NOT NULL DEFAULT FALSE,
230     deleted      TINYINT UNSIGNED NOT NULL DEFAULT 0,
231 PRIMARY KEY(kunde_id),
232 FOREIGN KEY (kunde_id)
233     REFERENCES kunde(kunde_id)
234     ON UPDATE RESTRICT
235     ON DELETE RESTRICT
236 ) ENGINE=InnoDB;
237
238 SELECT 'CREATE beitrag';
239 CREATE TABLE beitrag (
240     beitrag_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
241     account_id INT UNSIGNED NOT NULL,
242     bezug_beitrag_id INT UNSIGNED NOT NULL DEFAULT 1,
243     zeitstempel DATETIME NOT NULL DEFAULT '1000-01-01 00:00:00',
244     nachricht TEXT ,
245     deleted      TINYINT UNSIGNED NOT NULL DEFAULT 0,
246 PRIMARY KEY(beitrag_id),
247 FOREIGN KEY (account_id)
248     REFERENCES account(kunde_id)
249     ON UPDATE RESTRICT
250     ON DELETE RESTRICT,
251 FOREIGN KEY (bezug_beitrag_id)
252     REFERENCES beitrag(beitrag_id)
253     ON UPDATE RESTRICT
254     ON DELETE RESTRICT
255 ) ENGINE=InnoDB;
256
257 SELECT 'account daten';
258 INSERT INTO account (kunde_id, name, passwort, admin)
259 VALUES
260     (1, 'admin', md5('rosi'), TRUE)
261     ,(2, 'frodo', md5('elbereth'), FALSE)
262     ,(3, 'bilbo', md5('blitzerschlagen'), FALSE)
263     ,(5, 'elle', md5('feanor'), FALSE)
264 ;
265
266
267 SELECT 'beitrag daten';
268 INSERT INTO beitrag (beitrag_id, account_id, bezug_beitrag_id, zeitstempel,
269     nachricht)
270 VALUES
271     (1, 1, 1, '1000-01-01 00:00:00', '')
272     ,(2, 2, 1, '2011-05-01 14:13:00', 'Der Lieferservice ist super.')
273     ,(3, 3, 2, '2011-05-02 11:45:00', 'Das finde ich auch.')
274     ,(4, 5, 2, '2011-05-01 17:01:00', 'Aber ein wenig langsam.')
275     ,(5, 2, 4, '2011-05-01 17:15:00', 'Finde ich nicht.')
276     ,(6, 5, 1, '2011-06-12 09:07:00', 'Angebot könnte besser sein.')
277 ;
278 -- Ende Aufbau Forum
279 -- Start SELF JOIN

```

```
280 SELECT 'SELF JOIN (Seite 206)';
281 SELECT
282   kunde_id, name, status, admin
283   FROM
284     account
285 ;
286 SELECT
287   beitrag_id, account_id, bezug_beitrag_id, LEFT(nachricht, 20)
288   FROM
289     beitrag
290 ;
291 -- SELECT 'Erwarte eine Fehlermeldung (Seite 207)';
292 -- SELECT nachricht
293 --   FROM
294 --   beitrag INNER JOIN beitrag ON bezug_beitrag_id = beitrag_id
295 -- WHERE beitrag_id = 2
296 -- ;
297 SELECT ant.nachricht 'Antwort'
298   FROM
299     beitrag ant INNER JOIN beitrag orig ON ant.bezug_beitrag_id = orig.
300       beitrag_id
301 WHERE orig.beitrag_id = 2
302 ;
303 SELECT 'CTE rekursiv (Seite 208)';
304 WITH RECURSIVE ant_auf AS
305 (
306   -- nicht rekuriver Teil
307   SELECT * FROM beitrag WHERE bezug_beitrag_id = 2
308 UNION ALL
309   -- rekuriver Teil
310   SELECT ant.*
311     FROM
312       beitrag ant INNER JOIN ant_auf orig
313         ON ant.bezug_beitrag_id = orig.beitrag_id
314 )
315 SELECT beitrag_id, bezug_beitrag_id, nachricht FROM ant_auf;
316 -- Ende SELF JOIN
317
318 -- Start Redundant
319 SELECT 'Beschleunigen mit Redundanzen (Seite 209)';
320 ALTER TABLE
321   kunde
322   ADD r_strasse VARCHAR(255),
323   ADD r_ort VARCHAR(255),
324   ADD r_aktuell BOOL NOT NULL DEFAULT TRUE,
325   ADD l_strasse VARCHAR(255),
326   ADD l_ort VARCHAR(255),
327   ADD l_aktuell BOOL NOT NULL DEFAULT TRUE
328 ;
329 UPDATE kunde INNER JOIN adresse ON rechnung_adresse_id = adresse_id
330   SET
331     r_strasse = CONCAT(strasse, ' ', hnr),
332     r_ort = CONCAT(lkz, '-', plz, ' ', ort),
333     r_aktuell = TRUE
334 ;
335 SELECT
336   kunde_id, r_strasse, r_ort, r_aktuell
337   FROM kunde
```

```

338 ;
339 -- Ende Redundant
340 SELECT 'Ende listing08.sql /////////////////////////////////';

```

Listing 29.9 mysql/listing09.sql

```

1 -- Ausführen der vorherigen Befehle
2 SOURCE listing08.sql
3
4 SELECT 'Start listing09.sql /////////////////////////////////';
5
6 SELECT 'Aggregatfunktionen (Seite 212)';
7 SELECT AVG(einzelpreis) FROM artikel;
8 SELECT COUNT(*) FROM kunde;
9 SELECT COUNT(liefer_adresse_id) FROM kunde;
10 SELECT COUNT(DISTINCT(rechnung_adresse_id)) FROM kunde;
11 SELECT MAX(einzelpreis), MIN(einzelpreis) FROM artikel;
12 SELECT SUM(menge) FROM bestellung_position;
13
14     bp.menge, a.einzelpreis
15     FROM
16         bestellung_position bp INNER JOIN artikel a USING(artikel_id)
17 ;
18
19     SELECT
20         bp.menge * a.einzelpreis 'Positionswert'
21     FROM
22         bestellung_position bp INNER JOIN artikel a USING(artikel_id)
23 ;
24
25     -- Start Aufbau Lagerverwaltung
26     SELECT 'Aufbau Lagerbestand (Seite 214)';
27     SELECT 'CREATE lagerbestand';
28     DROP TABLE IF EXISTS lagerbestand;
29     CREATE TABLE lagerbestand (
30         artikel_id          INT UNSIGNED NOT NULL DEFAULT 0,
31         menge_mindest      DECIMAL(14,6) NOT NULL DEFAULT 0.0,
32         menge_aktuell       DECIMAL(14,6) NOT NULL DEFAULT 0.0,
33         deleted             TINYINT UNSIGNED NOT NULL DEFAULT 0,
34         PRIMARY KEY(artikel_id),
35         FOREIGN KEY (artikel_id)
36             REFERENCES artikel(artikel_id)
37             ON UPDATE RESTRICT
38             ON DELETE RESTRICT
39     ) ENGINE=InnoDB;
40
41     SELECT 'Lagerbestand festlegen';
42     INSERT INTO lagerbestand (artikel_id, menge_aktuell, menge_mindest)
43     VALUES
44         (7856, 500, 100)
45         ,(7863, 400, 100)
46         ,(9010, 30, 10)
47         ,(9015, 25, 10)
48         ,(3001, 5000, 1000)
49         ,(3005, 250, 200)
50         ,(3006, 250, 200)
51         ,(3007, 149, 200)
52         ,(3010, 3, 100)
53 ;
54     -- Ende Aufbau Lagerverwaltung

```

```
54
55
56 SELECT 'GROUP BY (Seite 215)';
57 -- Start Group by
58 SELECT
59   COUNT(*)
60   FROM
61   bestellung_position
62 ;
63 SELECT
64   bestellung_id Bestellnummer, COUNT(*) 'Anzahl der Positionen'
65   FROM
66   bestellung_position
67 GROUP BY
68   bestellung_id
69 ;
70 SELECT
71   a.bezeichnung, bp.menge
72   FROM
73   bestellung_position bp INNER JOIN artikel a USING (artikel_id)
74 ;
75 SELECT
76   a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
77   FROM
78   bestellung_position bp INNER JOIN artikel a USING (artikel_id)
79 GROUP BY
80   artikel_id
81 ;
82 SELECT
83   a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
84   FROM
85   bestellung_position bp RIGHT JOIN artikel a USING (artikel_id)
86 GROUP BY
87   artikel_id
88 ;
89 SELECT
90   a.bezeichnung 'Artikelname',
91   CASE
92     WHEN SUM(bp.menge) IS NULL THEN 0
93     ELSE SUM(bp.menge)
94   END AS 'Anzahl bestellter Artikel'
95   FROM
96   bestellung_position bp RIGHT JOIN artikel a USING (artikel_id)
97 GROUP BY
98   artikel_id
99 ;
100 -- SELECT 'Eine Fehlermeldung! (Seite 218)';
101 -- SELECT
102 -- a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
103 --   FROM
104 --   bestellung_position bp INNER JOIN artikel a USING (artikel_id)
105 --   WHERE
106 --     SUM(bp.menge) > 10
107 --   GROUP BY
108 --     artikel_id
109 -- ;
110
111 SELECT 'Gruppenergebnisse mit WITH ROLLUP summieren (Seite 217)';
112 SELECT
```

```
113 artikel_id,
114 CASE
115 WHEN SUM(bp.menge) IS NULL THEN 0
116 ELSE SUM(bp.menge)
117 END AS 'Anzahl bestellter Artikel'
118 FROM
119 bestellung_position bp RIGHT JOIN artikel a USING (artikel_id)
120 GROUP BY
121 artikel_id WITH ROLLUP
122 ;
123
124
125 SELECT 'Gruppenergebnisse mit HAVING filtern (Seite 219)';
126 SELECT
127 a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
128 FROM
129 bestellung_position bp INNER JOIN artikel a USING (artikel_id)
130 GROUP BY
131 artikel_id
132 HAVING
133 SUM(bp.menge) > 10
134 ;
135 SELECT
136 a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
137 FROM
138 bestellung_position bp INNER JOIN artikel a USING (artikel_id)
139 WHERE
140 a.bezeichnung LIKE 'Silber%'
141 GROUP BY
142 artikel_id
143 HAVING
144 SUM(bp.menge) > 10
145 ;
146
147 SELECT 'Fragen zu Gruppen (Seite 220)';
148 SELECT
149 SUBSTRING(blz, 1, 1) 'Clearinggebiet', COUNT(*) 'Anzahl'
150 FROM
151 bank
152 GROUP BY
153 'Clearinggebiet'
154 ORDER BY 'Anzahl' DESC
155 ;
156 -- Start Bestellungen erweitern
157 SELECT 'Bestellungen erweitern';
158 INSERT INTO bestellung (bestellung_id, kunde_id, adresse_id, datum)
159 VALUES
160 (3, 1, 1, '2011-01-15 16:43:00')
161 ,(4, 1, 1, '2011-01-16 09:15:00')
162 ,(5, 1, 1, '2011-01-16 09:16:00')
163 ,(6, 2, 2, '2012-04-01 13:11:00')
164 ;
165
166 INSERT INTO bestellung_position (
167 bestellung_id, position_nr, artikel_id, menge)
168 VALUES
169 (3, 1, 7856, 10)
170 ,(3, 2, 7863, 10)
171 ,(4, 1, 3006, 1)
```

```

172  ,(4, 2, 3010, 4)
173  ,(5, 1, 3001, 100)
174  ,(5, 2, 3010, 5)
175  ,(5, 3, 3006, 1)
176  ,(5, 4, 3005, 4)
177  ;
178 -- Ende Bestellungen erweitern
179 SELECT
180   CONCAT(YEAR(datum), '/', MONTHNAME(datum)) 'Monat', COUNT(*) 'Anzahl'
181   FROM bestellung
182   GROUP BY
183     YEAR(datum), MONTH(datum)
184   ORDER BY
185     YEAR(datum) DESC, MONTH(datum) DESC
186 ;
187 EXPLAIN
188 SELECT
189   SUBSTRING(blz, 1, 1) 'Clearinggebiet', COUNT(*) 'Anzahl'
190   FROM
191     bank
192   GROUP BY
193     'Clearinggebiet'
194   ORDER BY
195     'Anzahl' DESC\G
196 EXPLAIN
197 SELECT
198   CONCAT(lkz, SUBSTRING(blz, 6, 3)) 'Sinnlos', COUNT(*) 'Anzahl'
199   FROM
200     bank
201   GROUP BY
202     'Sinnlos'
203   ORDER BY
204     'Sinnlos'\G
205
206 -- Ende Group by
207 SELECT 'Ende listing09.sql /////////////////////////////////';

```

Listing 29.10 mysql/listing10.sql

```

1  -- Ausführen der vorherigen Befehle
2  SOURCE listing09.sql
3
4  SELECT 'Start listing10.sql /////////////////////////////////';
5
6  SELECT 'Start Aufbau Rechnungswesen';
7  USE oshop;
8
9
10 INSERT INTO rechnung
11   (kunde_id, bestellung_id, adresse_id, datum)
12   SELECT kunde_id, bestellung_id, adresse_id, DATE_ADD(datum, INTERVAL 1 DAY)
13   FROM bestellung;
14
15
16 INSERT INTO rechnung (rechnung_id, kunde_id, bestellung_id, adresse_id, datum,
17   bezahlart, status)
17 VALUES
18   (7, 3, 0, 2, '2012-04-06 12:11:00', 'kredit', 'gestellt')
19   ,(8, 3, 0, 2, '2012-04-07 13:12:00', 'kredit', 'mahnung1')

```

```

20 , (9, 5, 0, 4, '2012-04-08 14:13:00', 'rechnung', 'storno')
21 , (10, 5, 0, 4, '2012-04-09 15:14:00', 'rechnung', 'beglichen')
22 , (11, 5, 0, 4, '2012-04-10 16:15:00', 'rechnung', 'beglichen')
23 ;
24 UPDATE rechnung SET status = 'inkasso' WHERE rechnung_id = 2;
25 INSERT INTO rechnung_position (rechnung_id, position_nr, artikel_id, menge)
26 SELECT bestellung_id, position_nr, artikel_id, menge FROM bestellung_position
27 ;
28 INSERT INTO rechnung_position
29 (rechnung_id, position_nr, artikel_id, menge)
30 VALUES
31     (7, 1, 3001, 5)
32 , (7, 2, 3005, 1)
33 , (8, 1, 3006, 7)
34 , (8, 2, 3007, 1)
35 , (10, 1, 3010, 15)
36 , (10, 2, 3001, 5)
37 , (11, 1, 3005, 20)
38 ;
39
40 UPDATE rechnung SET rabatt = 7 WHERE kunde_id = 1;
41 UPDATE rechnung SET skonto = 3 WHERE kunde_id = 5;
42
43 SELECT 'Ende Aufbau Rechnungswesen';
44
45 SELECT 'Problem und Lösung (Seite 225)';
46 SELECT
47 bp.bestellung_id, SUM(bp.menge * a.einzelpreis) 'Bestellwert'
48 FROM
49 bestellung_position bp INNER JOIN artikel a USING(artikel_id)
50 GROUP BY
51 bp.bestellung_id
52 ;
53
54 SELECT
55 k.nachname, k.vorname, SUM(bp.menge * a.einzelpreis) 'Bestellwert'
56 FROM
57 bestellung_position bp INNER JOIN artikel a      USING(artikel_id)
58                      INNER JOIN bestellung b USING(bestellung_id)
59                      INNER JOIN kunde k      USING(kunde_id)
60 GROUP BY
61 bp.bestellung_id
62 ;
63
64 DROP TABLE IF EXISTS tmp_bestellwert;
65 CREATE TEMPORARY TABLE tmp_bestellwert
66 SELECT
67 k.nachname, k.vorname, SUM(bp.menge * a.einzelpreis) 'bestellwert'
68 FROM
69 bestellung_position bp INNER JOIN artikel a USING(artikel_id)
70                      INNER JOIN bestellung b USING(bestellung_id)
71                      INNER JOIN kunde k USING(kunde_id)
72 GROUP BY
73 bp.bestellung_id
74 ;
75
76 SELECT * FROM tmp_bestellwert;
77 SELECT

```

```
78  nachname, vorname, SUM(bestellwert) 'Umsatz'
79  FROM tmp_bestellwert
80  GROUP BY nachname, vorname
81 ;
82
83 SELECT
84  bw.nachname, bw.vorname, SUM(bw.bestellwert) 'Umsatz'
85  FROM
86  (
87   SELECT
88    k.nachname, k.vorname, SUM(bp.menge * a.einzelpreis) 'bestellwert'
89    FROM
90      bestellung_position bp INNER JOIN artikel a USING(artikel_id)
91          INNER JOIN bestellung b USING(bestellung_id)
92          INNER JOIN kunde k USING(kunde_id)
93    GROUP BY
94      bp.bestellung_id
95  ) AS 'bw'
96  GROUP BY nachname, vorname
97 ;
98
99 -- Start nicht korrelierende Unterabfrage
100 SELECT 'Nicht korrelierende Unterabfrage (Seite 228)';
101 SELECT MAX(blz) FROM bank;
102 SELECT bankname
103  FROM bank
104 WHERE blz = '37010050'
105 ;
106 SELECT bankname
107  FROM bank
108 WHERE blz =
109 (
110   SELECT MAX(blz)
111   FROM bank
112 )
113 ;
114
115 SELECT
116  AVG(einzelpreis) AS 'durchschnittspreis'
117  FROM artikel
118 ;
119
120 SELECT *
121  FROM artikel
122 WHERE
123   einzelpreis > 30
124 ;
125
126 SELECT *
127  FROM artikel
128 WHERE
129   einzelpreis >
130   (
131   SELECT
132     AVG(einzelpreis) AS 'durchschnittspreis'
133     FROM artikel
134   )
135 ;
136
```

```
137 SELECT
138   bp.bestellung_id, SUM(bp.menge * a.einzelpreis) 'bestellwert'
139   FROM
140     bestellung_position bp INNER JOIN artikel a USING(artikel_id)
141   GROUP BY
142     bp.bestellung_id
143   ;
144
145 SELECT AVG(bw.bestellwert)
146   FROM (
147     SELECT
148       bp.bestellung_id, SUM(bp.menge * a.einzelpreis) 'bestellwert'
149       FROM
150         bestellung_position bp INNER JOIN artikel a USING(artikel_id)
151       GROUP BY
152         bp.bestellung_id
153     ) AS 'bw'
154   ;
155
156 SELECT
157   bp1.bestellung_id, SUM(bp1.menge * a1.einzelpreis) 'bestellwert1'
158   FROM
159     bestellung_position bp1 INNER JOIN artikel a1 USING(artikel_id)
160   GROUP BY
161     bp1.bestellung_id
162   HAVING 'bestellwert1' > 100
163   ;
164
165 SELECT
166   bp1.bestellung_id, SUM(bp1.menge * a1.einzelpreis) 'bwert1'
167   FROM
168     bestellung_position bp1 INNER JOIN artikel a1 USING(artikel_id)
169   GROUP BY
170     bp1.bestellung_id
171   HAVING
172     'bwert1' >
173     (
174       SELECT AVG(bw.bwert2)
175       FROM
176       (
177         SELECT
178           bp2.bestellung_id, SUM(bp2.menge * a2.einzelpreis) 'bwert2'
179           FROM
180             bestellung_position bp2 INNER JOIN artikel a2
181               USING(artikel_id)
182             GROUP BY
183               bp2.bestellung_id
184             ) AS 'bw'
185       )
186   ;
187
188 SELECT bestellung_id FROM bestellung;
189 SELECT rechnung_id
190   FROM rechnung
191 WHERE bestellung_id IN
192   (
193   SELECT bestellung_id FROM bestellung
194   )
195   ;
```

```
196
197 SELECT a.einzelpreis
198   FROM
199     artikel_nm_warengruppe INNER JOIN artikel a      USING(artikel_id)
200                           INNER JOIN warengruppe w USING(warengruppe_id)
201 WHERE
202   w.bezeichnung = 'Gartenbedarf'
203 ;
204 SELECT
205   a.bezeichnung, a.einzelpreis
206   FROM
207     artikel a
208 WHERE
209   a.einzelpreis > ALL(SELECT 10)
210 ;
211
212 SELECT
213   a.bezeichnung, a.einzelpreis
214   FROM
215     artikel a
216 WHERE
217   a.einzelpreis > ALL
218   (
219     SELECT a.einzelpreis
220       FROM
221         artikel_nm_warengruppe INNER JOIN artikel a USING(artikel_id)
222                           INNER JOIN warengruppe w USING(warengruppe_id)
223 WHERE
224   w.bezeichnung = 'Gartenbedarf'
225   )
226 ;
227
228 SELECT
229   rechnung_id, kunde_id, SUM(einzelpreis*menge) AS 'umsatz'
230   FROM
231     rechnung_position INNER JOIN artikel  USING(artikel_id)
232                           INNER JOIN rechnung USING(rechnung_id)
233 GROUP BY
234   rechnung_id
235 ;
236
237 SELECT SUM(umsatz) AS 'umsatzsumme'
238   FROM
239   (
240     SELECT
241       rechnung_id, kunde_id, SUM(einzelpreis*menge) AS 'umsatz'
242     FROM
243       rechnung_position INNER JOIN artikel  USING(artikel_id)
244                           INNER JOIN rechnung USING(rechnung_id)
245     GROUP BY
246       rechnung_id
247   ) AS 'ksum'
248 GROUP BY
249   kunde_id
250 ;
251
252 SELECT SUM(umsatz) AS 'umsatzsumme'
253   FROM
254   (
```

```

255     SELECT
256         rechnung_id, kunde_id, SUM(einzelpreis*menge) AS 'umsatz'
257     FROM
258         rechnung_position INNER JOIN artikel USING(artikel_id)
259             INNER JOIN rechnung USING(rechnung_id)
260                 INNER JOIN kunde USING(kunde_id)
261     WHERE nachname = 'beutlin'
262     GROUP BY
263         rechnung_id
264 ) AS 'ksum'
265 GROUP BY
266     kunde_id
267 ;
268
269 SELECT kunde_id, SUM(umsatz) AS 'umsatzsumme'
270 FROM
271 (
272     SELECT
273         rechnung_id, kunde_id, SUM(einzelpreis*menge) AS 'umsatz'
274     FROM
275         rechnung_position INNER JOIN artikel USING(artikel_id)
276             INNER JOIN rechnung USING(rechnung_id)
277     GROUP BY
278         rechnung_id
279 ) AS 'ksum'
280 GROUP BY
281     kunde_id
282 HAVING umsatzsumme > ALL (SELECT 100)
283 ;
284
285 SELECT kunde_id, kunde.nachname, kunde.vorname, SUM(umsatz) AS 'umsatzsumme'
286 FROM
287 (
288     SELECT
289         rechnung_id, kunde_id, SUM(einzelpreis*menge) AS 'umsatz'
290     FROM
291         rechnung_position INNER JOIN artikel USING(artikel_id)
292             INNER JOIN rechnung USING(rechnung_id)
293     GROUP BY
294         rechnung_id
295 ) AS 'ksum'
296     INNER JOIN kunde USING (kunde_id)
297 GROUP BY
298     kunde_id
299 HAVING umsatzsumme > ALL
300 (
301     SELECT SUM(umsatz) AS 'umsatzsumme'
302     FROM
303 (
304     SELECT
305         rechnung_id, kunde_id, SUM(einzelpreis*menge) AS 'umsatz'
306     FROM
307         rechnung_position INNER JOIN artikel USING(artikel_id)
308             INNER JOIN rechnung USING(rechnung_id)
309                 INNER JOIN kunde USING(kunde_id)
310     WHERE nachname = 'beutlin'
311     GROUP BY
312         rechnung_id
313 ) AS 'ksum'

```

```
314     GROUP BY
315         kunde_id
316     )
317 ;
318
319 SELECT a.einzelpreis
320   FROM
321     artikel_nm_warengruppe INNER JOIN artikel a      USING(artikel_id)
322                               INNER JOIN warengruppe w USING(warengruppe_id)
323 WHERE
324   w.bezeichnung = 'Bürobedarf'
325 ;
326
327 SELECT
328   a.bezeichnung, a.einzelpreis
329   FROM
330     artikel a
331 WHERE
332   a.einzelpreis > ANY(SELECT 10)
333 ;
334 SELECT
335   a.bezeichnung, a.einzelpreis
336   FROM
337     artikel a
338 WHERE
339   a.einzelpreis > ANY
340   (
341     SELECT a.einzelpreis
342       FROM
343         artikel_nm_warengruppe INNER JOIN artikel a USING(artikel_id)
344                               INNER JOIN warengruppe w USING(warengruppe_id)
345 WHERE
346   w.bezeichnung = 'Bürobedarf'
347   )
348 ;
349
350 SELECT
351   a.bezeichnung, a.einzelpreis
352   FROM
353     artikel a
354 WHERE
355   a.artikel_id NOT IN
356   (
357     SELECT
358       artikel_id
359       FROM
360         artikel_nm_warengruppe nm  INNER JOIN warengruppe w USING(warengruppe_id)
361       WHERE w.bezeichnung = 'Bürobedarf'
362   )
363 AND
364   a.einzelpreis > ANY
365   (
366     SELECT a.einzelpreis
367       FROM
368         artikel_nm_warengruppe nm INNER JOIN artikel a USING(artikel_id)
369                               INNER JOIN warengruppe w USING(warengruppe_id)
370 WHERE
371   w.bezeichnung = 'Bürobedarf'
372   )
```

```
373 ;
374
375 SELECT DISTINCT k.nachname, k.vorname
376   FROM
377     rechnung r INNER JOIN kunde k USING(kunde_id)
378   WHERE
379     YEAR(r.datum) = '2011' AND (k.nachname, k.vorname) IN
380   (
381     SELECT
382       k.nachname, k.vorname
383     FROM
384       rechnung r INNER JOIN kunde k USING(kunde_id)
385     WHERE
386       YEAR(r.datum) = '2012'
387   )
388 ;
389 -- Ende nicht korrelierende Unterabfrage
390
391 -- Start korrelierende Unterabfrage
392 SELECT 'Korrelierende Unterabfrage (Seite 240)';
393   SELECT rechnung_id
394     FROM rechnung r
395   WHERE
396   (
397     SELECT
398       COUNT(position_nr)
399     FROM rechnung_position rp
400     WHERE
401       rp.rechnung_id = r.rechnung_id
402     ) >= 3
403 ;
404   SELECT rechnung_id
405     FROM rechnung r
406   WHERE
407     EXISTS
408     (
409       SELECT
410         bestellung_id
411       FROM bestellung b
412       WHERE b.bestellung_id = r.bestellung_id
413     )
414 ;
415 -- Ende korrelierende Unterabfrage
416
417 -- Start Fallstudie
418 SELECT 'Fallstudie Datenimport (Seite 242)';
419 DROP TABLE IF EXISTS tmp_import;
420 CREATE TEMPORARY TABLE tmp_import
421 (
422   nachname VARCHAR(255),
423   vorname  VARCHAR(255),
424   strasse   VARCHAR(255),
425   hnr      VARCHAR(255),
426   lkz      VARCHAR(255),
427   plz      VARCHAR(255),
428   ort      VARCHAR(255),
429   ktNr    VARCHAR(255),
430   blz      VARCHAR(255)
431 );
```

```
432
433 LOAD DATA LOCAL INFILE 'kunden01.csv'
434   INTO TABLE tmp_import
435   FIELDS
436     TERMINATED BY ';'
437   LINES
438     TERMINATED BY '\n'
439   IGNORE 1 LINES
440   (nachname, vorname, strasse, hnr, lkz, plz, ort, ktnr, blz)
441 ;
442
443 INSERT INTO kunde (nachname, vorname)
444   SELECT DISTINCT nachname, vorname
445     FROM tmp_import tmp
446   WHERE
447     (tmp.vorname, tmp.nachname) NOT IN
448     (
449       SELECT vorname, nachname FROM kunde
450     )
451 ;
452
453 INSERT INTO adresse (strasse, hnr, lkz, plz, ort)
454   SELECT DISTINCT strasse, hnr, lkz, plz, ort
455     FROM tmp_import tmp
456   WHERE
457     (tmp.strasse, tmp.hnr, tmp.lkz, tmp.plz, tmp.ort) NOT IN
458     (
459       SELECT strasse, hnr, lkz, plz, ort
460         FROM adresse
461     )
462 ;
463
464 ALTER TABLE tmp_import ADD kunde_id    INT UNSIGNED,
465                           ADD adresse_id INT UNSIGNED,
466                           ADD bank_id    CHAR(12)
467 ;
468
469 UPDATE tmp_import t
470   SET
471     t.kunde_id =
472     (
473       SELECT kunde_id
474         FROM kunde k
475       WHERE t.vorname = k.vorname AND t.nachname = k.nachname
476     ),
477     t.adresse_id =
478     (
479       SELECT adresse_id
480         FROM adresse a
481       WHERE
482         t.strasse = a.strasse
483           AND t.hnr = a.hnr
484           AND t.plz = a.plz
485           AND t.ort = a.ort
486     ),
487     t.bank_id =
488     (
489       SELECT bank_id FROM bank WHERE t.blz = bank.blz LIMIT 1
490     )
```

```
491 ;
492
493 INSERT INTO
494 bankverbindung (kunde_id, bankverbindung_nr, bank_id, kontonummer, iban)
495 SELECT DISTINCT kunde_id, 1, bank_id, ktnr, CONCAT(blz, ktnr)
496 FROM tmp_import tmp
497 WHERE
498 (tmp.kunde_id, 1, tmp.bank_id, tmp.ktnr) NOT IN
499 (
500     SELECT kunde_id, 1, bank_id, kontonummer
501     FROM bankverbindung
502 )
503 ;
504
505 ALTER TABLE tmp_import
506 ADD bankverbindung_nr INT UNSIGNED NOT NULL DEFAULT 1
507 ;
508
509 UPDATE kunde k SET
510     k.rechnung_adresse_id =
511     (
512         SELECT adresse_id FROM tmp_import tmp WHERE tmp.kunde_id = k.kunde_id
513     )
514 WHERE k.rechnung_adresse_id IS NULL
515 ;
516
517 SELECT kunde_id, nachname, vorname, strasse, ort
518 FROM
519     kunde LEFT JOIN adresse ON kunde.rechnung_adresse_id = adresse_id
520 ;
521 -- Ende Fallstudie
522
523 SELECT 'Unterabfragen intern (Seite 245)';
524 SELECT
525     bp1.bestellung_id, SUM(bp1.menge * a1.einzelpreis) 'bwert1'
526     FROM
527         bestellung_position bp1 INNER JOIN artikel a1 USING(artikel_id)
528     GROUP BY
529         bp1.bestellung_id
530     HAVING
531         'bwert1' >
532     (
533         SELECT AVG(bw.bwert2)
534         FROM
535     (
536         SELECT
537             bp2.bestellung_id, SUM(bp2.menge * a2.einzelpreis) 'bwert2'
538             FROM
539                 bestellung_position bp2 INNER JOIN artikel a2
540                     USING(artikel_id)
541                 GROUP BY
542                     bp2.bestellung_id
543             ) AS 'bw'
544     )
545 ;
546
547 EXPLAIN
548 SELECT
549     bp1.bestellung_id, SUM(bp1.menge * a1.einzelpreis) 'bwert1'
```

```
550   FROM
551     bestellung_position bp1 INNER JOIN artikel a1 USING(artikel_id)
552 GROUP BY
553   bp1.bestellung_id
554 HAVING
555   'bwert1' >
556   (
557     SELECT AVG(bw.bwert2)
558     FROM
559   (
560     SELECT
561       bp2.bestellung_id, SUM(bp2.menge * a2.einzelpreis) 'bwert2'
562     FROM
563       bestellung_position bp2 INNER JOIN artikel a2
564         USING(artikel_id)
565     GROUP BY
566       bp2.bestellung_id
567   ) AS 'bw'
568 )
569 \G
570
571 SELECT 'Ende listing10.sql //////////////////////////////';
```

Listing 29.11 mysql/listing11.sql

```
1 -- Ausführen der vorherigen Befehle
2 SOURCE listing10.sql
3
4 SELECT 'Start listing11.sql //////////////////////////////';
5
6 SELECT 'UNION (Seite 251)';
7 SELECT
8   strasse, hnr, lkz, plz, ort
9   FROM
10  kunde INNER JOIN adresse ON rechnung_adresse_id = adresse_id
11 ;
12 SELECT
13   strasse, hnr, lkz, plz, ort
14   FROM
15  lieferant INNER JOIN adresse USING(adresse_id)
16 ;
17 SELECT
18   strasse, hnr, lkz, plz, ort
19   FROM
20  kunde INNER JOIN adresse ON rechnung_adresse_id = adresse_id
21 UNION
22 SELECT
23   strasse, hnr, lkz, plz, ort
24   FROM
25  lieferant INNER JOIN adresse USING(adresse_id)
26 ;
27 SELECT
28   strasse, hnr, lkz, plz, ort
29   FROM
30  kunde INNER JOIN adresse ON rechnung_adresse_id = adresse_id
31 UNION ALL
32 SELECT
33   strasse, hnr, lkz, plz, ort
34   FROM
```

```

35     lieferant INNER JOIN adresse USING(adresse_id)
36 ;
37
38 SELECT 'INTERSECT (Seite 254)';
39 SELECT 'Schnittmenge mit Unterabfrage (Seite 255)';
40 SELECT kunde_id FROM rechnung WHERE YEAR(datum) = '2011';
41 SELECT DISTINCT kunde_id
42   FROM rechnung
43   WHERE YEAR(datum) = '2012' AND kunde_id IN (1, 2, 3)
44 ;
45 SELECT DISTINCT kunde_id
46   FROM rechnung
47 WHERE
48   YEAR(datum) = '2012' AND kunde_id IN
49   (
50     SELECT kunde_id FROM rechnung WHERE YEAR(datum) = '2011'
51   )
52 ;
53
54 SELECT 'EXCEPT (Seite 256)';
55
56 SELECT 'Differenz mit Unterabfrage (Seite 256)';
57 SELECT
58   strasse, hnr, lkz, plz, ort
59   FROM
60   adresse
61 ;
62 SELECT
63   strasse, hnr, lkz, plz, ort
64   FROM
65   kunde INNER JOIN adresse
66   ON liefer_adresse_id = adresse_id
67 ;
68 SELECT kunde_id
69   FROM rechnung
70   WHERE YEAR(datum) <> '2011' AND YEAR(datum) = '2012'
71 ;
72 SELECT kunde_id FROM rechnung WHERE YEAR(datum) = '2011';
73 SELECT DISTINCT kunde_id
74   FROM rechnung
75 WHERE
76   YEAR(datum) = '2012' AND kunde_id NOT IN (1, 2, 3)
77 ;
78 SELECT DISTINCT kunde_id
79   FROM rechnung
80 WHERE
81   YEAR(datum) = '2012' AND kunde_id NOT IN
82   (
83     SELECT kunde_id FROM rechnung WHERE YEAR(datum) = '2011'
84   )
85 ;
86 SELECT 'Ende listing11.sql /////////////////////////////////';

```

Listing 29.12 mysql/listing12.sql

```

1 -- Ausführen der vorherigen Befehle
2 SOURCE listing11.sql
3
4 SELECT 'Start listing12.sql /////////////////////////////////';

```

```
5
6 SELECT kunde_id, nachname, vorname, art FROM kunde;
7 SELECT 'Start Aufbau einer temporären art-Tabelle (Seite 262)';
8
9 DROP TABLE IF EXISTS tmp_art;
10 CREATE TEMPORARY TABLE tmp_art
11 (
12     art VARCHAR(255),
13     k_art VARCHAR(255)
14 )
15 ;
16
17 INSERT INTO tmp_art
18     VALUES
19         ('prv', 'Privatkunde')
20         ,('gsch', 'Geschäftskunde')
21         ,('unb', 'Unbekannt')
22 ;
23
24 SELECT kunde_id, nachname, vorname, k_art
25     FROM
26     kunde INNER JOIN tmp_art USING(art)
27 ;
28 SELECT 'Ende Aufbau einer temporaeren art-Tabelle';
29
30 SELECT 'Lösung mit UNIONs (Seite 262)';
31 SELECT kunde_id, nachname, vorname, k_art
32     FROM
33     kunde INNER JOIN
34     (
35         SELECT 'prv' AS art, 'Privatkunde' AS k_art
36         UNION
37         SELECT 'gsch' AS art, 'Geschäftskunde' AS k_art
38         UNION
39         SELECT 'unb' AS art, 'Unbekannt' AS k_art
40     ) t
41     USING(art)
42 ;
43
44 SELECT 'Nö (Seite 263)';
45 SELECT kunde_id, nachname, vorname,
46 CASE art
47     WHEN 'prv' THEN 'Privatkunde'
48     WHEN 'gsch' THEN 'Geschäftskunde'
49     ELSE 'Unbekannt'
50 END AS k_art
51 FROM kunde
52 ;
53
54 -- Start Einfacher CASE
55 SELECT 'Einfacher CASE (Seite 264)';
56 SELECT artikel_id, bezeichnung,
57 CASE waehrung
58     WHEN 'EUR' THEN LPAD(CONCAT(FORMAT(einzelpreis, 2, 'de_DE'), ' €'), 7, ' ')
59     WHEN 'USD' THEN LPAD(CONCAT(FORMAT(einzelpreis, 2, 'de_DE'), ' $'), 7, ' ')
60     ELSE '?????'
61 END AS preis
62 FROM artikel
63 ;
```

```

64 UPDATE artikel SET waehrung = 'XYZ' WHERE artikel_id = 3001;
65 SELECT artikel_id, bezeichnung,
66 CASE waehrung
67 WHEN 'EUR' THEN LPAD(CONCAT(FORMAT(einzelpreis, 2, 'de_DE'), ' €'), 7, ' ')
68 WHEN 'USD' THEN LPAD(CONCAT(FORMAT(einzelpreis, 2, 'de_DE'), ' $'), 7, ' ')
69 END AS preis
70 FROM artikel
71 ;
72 UPDATE artikel SET waehrung = 'EUR' WHERE artikel_id = 3001;
73 -- Ende Einfacher CASE
74
75 -- Start Searched CASE
76 SELECT 'Searched CASE (Seite 265)';
77 SELECT beitrag_id,
78 CASE
79 WHEN bezug_beitrag_id > 1 THEN 'Antwort'
80 WHEN bezug_beitrag_id <= 1 THEN 'Keine Antwort'
81 ELSE '??????'
82 END AS Typ
83 FROM beitrag
84 ORDER BY beitrag_id
85 ;
86 SELECT beitrag_id,
87 CASE
88 WHEN bezug_beitrag_id > 1
89 THEN CONCAT(nachricht, ' Antwort auf ', bezug_beitrag_id, ': >',
90 (
91     SELECT nachricht
92         FROM beitrag b2
93         WHERE b2.beitrag_id = b1.bezug_beitrag_id
94 )
95 ) -- Ende CONCAT
96 WHEN bezug_beitrag_id <= 1 THEN nachricht
97 ELSE '??????'
98 END AS Inhalt
99 FROM beitrag b1
100 WHERE beitrag_id > 1
101 ;
102 SELECT beitrag_id,
103 CASE
104 WHEN bezug_beitrag_id >= 0 THEN 'Antwort'
105 WHEN bezug_beitrag_id > 1 THEN 'Keine Antwort'
106 ELSE '??????'
107 END AS Typ
108 FROM beitrag
109 ORDER BY beitrag_id
110 ;
111 -- Ende Searched CASE
112
113 -- Start Fallbeispiele
114 SELECT 'Fallbeispiel Lagerbestand (Seite 267)';
115 SELECT * FROM lagerbestand;
116 SELECT a.bezeichnung,
117 CASE
118 WHEN l.menge_aktuell <= l.menge_mindest
119     THEN 'Artikel nachbestellen'
120 ELSE 'Bestand ausreichend'
121 END AS lagerstand
122 FROM lagerbestand l INNER JOIN artikel a USING(artikel_id)

```

```
123 ORDER BY l.menge_aktuell / l.menge_mindest
124 ;
125
126 SELECT 'Fallbeispiel Kundengruppen (Seite 268)';
127 SELECT DISTINCT k.kunde_id, k.nachname, k.vorname,
128 CASE
129 WHEN 3 <
130 (
131     SELECT COUNT(rechnung_id) FROM rechnung r
132     WHERE r.kunde_id = k.kunde_id
133 ) THEN 'Prämiumkunde'
134 WHEN 2 <
135 (
136     SELECT COUNT(rechnung_id) FROM rechnung r
137     WHERE r.kunde_id = k.kunde_id
138 ) THEN 'Guter Kunde'
139 ELSE 'Kleinkunde'
140 END kundenart
141 FROM
142     rechnung r RIGHT OUTER JOIN kunde k USING(kunde_id)
143 ;
144 SELECT k.kunde_id, k.nachname, k.vorname,
145 (
146     SELECT SUM(a.einzelpreis * rp.menge)
147     FROM rechnung r INNER JOIN rechnung_position rp
148             USING (rechnung_id)
149             INNER JOIN artikel a
150                     USING (artikel_id)
151     WHERE r.kunde_id = k.kunde_id
152 ) rechnungswert,
153 CASE
154 WHEN
155     kunde_id NOT IN (SELECT kunde_id FROM rechnung)
156 THEN 'Kleinkunde'
157 WHEN
158 (
159     SELECT SUM(a.einzelpreis * rp.menge) rechnungswert
160     FROM rechnung r INNER JOIN rechnung_position rp
161             USING (rechnung_id)
162             INNER JOIN artikel a
163                     USING (artikel_id)
164     WHERE r.kunde_id = k.kunde_id
165 ) BETWEEN 0 AND 300 THEN 'Kleinkunde'
166 WHEN
167 (
168     SELECT SUM(a.einzelpreis * rp.menge) rechnungswert
169     FROM rechnung r INNER JOIN rechnung_position rp
170             USING (rechnung_id)
171             INNER JOIN artikel a
172                     USING (artikel_id)
173     WHERE r.kunde_id = k.kunde_id
174 ) BETWEEN 300 AND 1000 THEN 'Guter Kunde'
175 ELSE 'Prämiumkunde'
176 END AS kundenart
177 FROM kunde k
178 ORDER BY
179 (
180     SELECT SUM(a.einzelpreis * rp.menge) rechnungswert
181     FROM rechnung r INNER JOIN rechnung_position rp
```

```

182           USING (rechnung_id)
183           INNER JOIN artikel a
184           USING (artikel_id)
185   WHERE r.kunde_id = k.kunde_id
186 ) DESC
187 ;
188 DROP TABLE IF EXISTS tmp_umsatz;
189 CREATE TEMPORARY TABLE tmp_umsatz
190   SELECT kunde_id, SUM(a.einzelpreis * rp.menge) rechnungswert
191     FROM rechnung r INNER JOIN rechnung_position rp USING (rechnung_id)
192           INNER JOIN artikel a USING (artikel_id)
193   GROUP BY kunde_id
194 ;
195   SELECT k.kunde_id, k.nachname, k.vorname, tmp.rechnungswert,
196   CASE
197     WHEN
198       tmp.kunde_id IS NULL THEN 'Kleinkunde'
199     WHEN
200       rechnungswert BETWEEN 0 AND 300 THEN 'Kleinkunde'
201     WHEN
202       rechnungswert BETWEEN 300 AND 1000 THEN 'Guter Kunde'
203     ELSE 'Premiumkunde'
204   END AS kundenart
205   FROM kunde k LEFT JOIN tmp_umsatz tmp USING(kunde_id)
206   ORDER BY rechnungswert DESC
207 ;
208
209 WITH tmp_umsatz (kunde_id, rechnungswert)
210 AS (
211   SELECT kunde_id, SUM(a.einzelpreis * rp.menge) rechnungswert
212     FROM rechnung r INNER JOIN rechnung_position rp USING (rechnung_id)
213           INNER JOIN artikel a USING (artikel_id)
214   GROUP BY kunde_id
215 )
216   SELECT k.kunde_id, k.nachname, k.vorname, tmp.rechnungswert,
217   CASE
218     WHEN
219       tmp.kunde_id IS NULL THEN 'Kleinkunde'
220     WHEN
221       rechnungswert BETWEEN 0 AND 300 THEN 'Kleinkunde'
222     WHEN
223       rechnungswert BETWEEN 300 AND 1000 THEN 'Guter Kunde'
224     ELSE 'Premiumkunde'
225   END AS kundenart
226   FROM kunde k LEFT JOIN tmp_umsatz tmp USING(kunde_id)
227   ORDER BY rechnungswert DESC
228 ;
229
230 SELECT 'Fallbeispiel Lieferanten (Seite 271)';
231 SELECT l.firmenname,
232   CASE
233     WHEN
234       l.lieferant_id IN
235       (
236         SELECT lieferant_id FROM artikel_nm_lieferant
237       ) THEN 'aktiv'
238     ELSE 'inaktiv'
239   END AS status
240   FROM lieferant l

```

```

241 ORDER BY firmenname
242 ;
243 SELECT a.bezeichnung,
244 CASE
245     WHEN l.menge_aktuell <= l.menge_mindest THEN 'Artikel sofort nachbestellen
246         ,
247     WHEN l.menge_aktuell <= l.menge_mindest + 0.3 * l.menge_mindest THEN '
248         Artikel bald nachbestellen'
249     ELSE 'Bestand ausreichend' END AS lagerstand FROM lagerbestand l INNER
250         JOIN artikel a USING(artikel_id)
251 ORDER BY l.menge_aktuell / l.menge_mindest
252 ;
253 -- Ende Fallbeispiele
254 SELECT 'Ende listing12.sql /////////////////////////////////';

```

Listing 29.13 mysql/listing13.sql

```

1 -- Ausführen der vorherigen Befehle
2 SOURCE listing12.sql
3
4 SELECT 'Start listing13.sql /////////////////////////////////';
5
6 SELECT 'Meine erste Ansicht (Seite 274)';
7 CREATE VIEW
8 view_kundenrechnungsadresse (id, nachname, vorname, strasse, ort)
9 AS
10    SELECT
11        kunde_id, nachname, vorname,
12        CONCAT_WS(' ', strasse, hnr),
13        CONCAT_WS(' ', lkz, plz, ort)
14    FROM
15        Kunde INNER JOIN Adresse
16            ON rechnung_adresse_id = adresse_id
17    WHERE kunde.deleted = 0
18 ;
19 SELECT * FROM view_kundenrechnungsadresse;
20 SHOW TABLES;
21
22 SELECT 'VIEW-Verarbeitung (Seite 277)';
23 CREATE VIEW view_artikel_aktiv
24 AS
25    SELECT * FROM artikel
26        WHERE deleted = 0
27 ;
28 SELECT FLOOR(artikel_id / 1000) gruppe, AVG(einzelpreis)
29     FROM view_artikel_aktiv
30     GROUP BY gruppe
31 ;
32 EXPLAIN SELECT FLOOR(artikel_id / 1000) gruppe, AVG(einzelpreis)
33     FROM view_artikel_aktiv
34     GROUP BY gruppe
35 \G
36
37 SELECT artikel_id, bezeichnung FROM
38 view_artikel_aktiv
39 WHERE
40     artikel_id BETWEEN 7000 AND 9000
41 ;
42 EXPLAIN SELECT artikel_id, bezeichnung FROM

```

```
43 view_artikel_aktiv
44 WHERE
45 artikel_id BETWEEN 7000 AND 9000
46 \G
47
48 CREATE ALGORITHM=TEMPTABLE VIEW view_artikel_aktiv_tmp
49 AS
50   SELECT * FROM artikel
51   WHERE deleted = 0
52 ;
53 SELECT artikel_id, bezeichnung
54   FROM view_artikel_aktiv_tmp
55   WHERE artikel_id BETWEEN 7000 AND 8000
56 ;
57 SELECT artikel_id, bezeichnung
58   FROM view_artikel_aktiv_tmp
59   WHERE artikel_id BETWEEN 3000 AND 4000
60 ;
61 EXPLAIN SELECT artikel_id, bezeichnung
62   FROM view_artikel_aktiv_tmp
63   WHERE artikel_id BETWEEN 3000 AND 4000
64 \G
65
66 -- Start DROP VIEW
67 SELECT 'Ansicht löschen (Seite 279)';
68 SHOW TABLES;
69 DROP VIEW view_artikel_aktiv_tmp;
70 SHOW TABLES;
71
72 DROP DATABASE IF EXISTS tmp1;
73 CREATE DATABASE tmp1;
74 USE tmp1;
75 CREATE TABLE a (i INT);
76 CREATE TABLE b (i INT);
77 CREATE VIEW ab
78 AS
79   SELECT * FROM a INNER JOIN b USING(i)
80 ;
81 SHOW TABLES;
82 DROP TABLE a;
83 SHOW TABLES;
84 -- SELECT 'Erwarte eine Fehlermeldung';
85 -- SELECT * FROM ab;
86 CHECK TABLE ab\G
87 DROP DATABASE tmp1;
88
89 -- Ende DROP VIEW
90 USE oshop;
91 SELECT 'Anwendungsgebiet: Vereinfachung (Seite 283)';
92 CREATE OR REPLACE
93   ALGORITHM = TEMPTABLE
94   VIEW view_artikel_aktiv
95   AS
96     SELECT * FROM artikel
97     WHERE deleted = 0
98 ;
99 CREATE OR REPLACE VIEW view_artikel_verfuegbar
100 AS
101   SELECT artikel_id, bezeichnung
```

```
102     FROM artikel INNER JOIN lagerbestand USING(artikel_id)
103 WHERE menge_aktuell >= menge_mindest
104 ORDER BY artikel_id
105 ;
106 SELECT * FROM view_artikel_verfuegbar;
107 CREATE OR REPLACE
108   ALGORITHM = TEMPTABLE
109   VIEW view_lagerbestand_aktiv
110     AS
111       SELECT * FROM lagerbestand
112         WHERE deleted = 0
113 ;
114 CREATE OR REPLACE VIEW view_lagerbestand_artikelbezeichnung
115 AS
116   SELECT l.*, a.bezeichnung
117     FROM
118       view_lagerbestand_aktiv l INNER JOIN view_artikel_aktiv a
119         USING(artikel_id)
120 ;
121 SELECT * FROM view_lagerbestand_artikelbezeichnung;
122
123 -- SELECT 'Grenzen der VIEW. Erwarte 3 Fehlermeldungen (Seite 286)';
124 -- CREATE OR REPLACE VIEW v
125 --   AS SELECT bezeichnung FROM (SELECT * FROM artikel) a;
126 -- SET @b = 3000;
127 -- CREATE OR REPLACE VIEW v
128 --   AS SELECT * FROM artikel WHERE artikel_id > @b;
129 DROP TABLE IF EXISTS tmp;
130 CREATE TEMPORARY TABLE tmp AS SELECT * FROM artikel;
131 -- CREATE OR REPLACE VIEW view_tmp AS SELECT artikel_id FROM tmp;
132 DROP TABLE IF EXISTS view_tmp;
133 CREATE OR REPLACE VIEW view_tmp
134   AS SELECT nachname FROM kunde ORDER BY nachname DESC;
135 SELECT * FROM view_tmp;
136 SELECT * FROM view_tmp ORDER BY nachname;
137 CREATE OR REPLACE VIEW view_tmp_eins
138   AS
139     SELECT nachname FROM kunde
140 ;
141 CREATE OR REPLACE VIEW view_tmp_zwei
142   AS
143     SELECT rechnung_id, COUNT(*) anzahl
144       FROM rechnung_position
145      GROUP BY rechnung_id
146 ;
147 UPDATE view_tmp_eins
148   SET nachname = 'Streicher'
149   WHERE nachname = 'Telcontar'
150 ;
151
152 -- SELECT 'Grenzen der VIEW. Erwarte Fehlermeldungen';
153 -- UPDATE view_tmp_zwei
154 --   SET anzahl = 2
155 --   WHERE anzahl > 2
156 -- ;
157 -- Rückbau
158 UPDATE view_tmp_eins
159   SET nachname = 'Telcontar'
160   WHERE nachname = 'Streicher'
```

```

161 ;
162 -- UPDATE view_tmp_zwei SET rechnung_id = 1 WHERE rechnung_id > 1;
163 INSERT INTO view_tmp_eins VALUES ('Wurst'), ('Brot');
164 SELECT * FROM view_tmp_eins;
165 DELETE FROM view_tmp_eins WHERE nachname IN ('Wurst', 'Brot');
166
167 SELECT 'Ende listing13.sql' //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////;

```

Listing 29.14 mysql/listing13NoSQL.js

```

1 // Globale Variablen (Seite 294)
2 var connect = { host: 'localhost'
3                 , port: 33060
4                 , user: 'root'
5                 , password: 'weinschlauchX10' };
6
7 var mysqlx = require('mysqlx');
8
9 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
10 // Skript 1: Verbinden, anlegen und schließen (Seite 294)
11 function warenkorbCollectionAnlegen(mySession) {
12     var db0shop = mySession.getSchema('oshop');
13     db0shop.dropCollection('warenkörbe');
14     return db0shop.createCollection('warenkörbe');
15 }
16
17 var session = mysqlx.getSession(connect);
18 warenkorbCollectionAnlegen(session);
19 session.close();
20
21
22 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
23 // Skript 2: Leere Warenkörbe verteilen (Seite 295)
24 function warenkorbLeeren(mySession) {
25     var db0shop = mySession.getSchema('oshop');
26     var collWarenkörbe = db0shop.getCollection('warenkörbe');
27
28     collWarenkörbe.remove({_id>0}).execute();
29
30     // Für alle Kunden einen leeren Warenkorb anlegen.
31     var tabKunde = db0shop.getTable("kunde");
32     var alleKunden = tabKunde.select("kunde_id").execute();
33
34     var zeilen = alleKunden.fetchAll();
35     for (index in zeilen) {
36         var zeile = zeilen[index];
37         collWarenkörbe.add({
38             _id: zeile.kunde_id,
39             warenkorb: {
40                 kunde_id: zeile.kunde_id,
41                 positionen: new Array(),
42                 warenwerte: new Array()
43             }
44         }).execute();
45     }
46 }
47
48 var session = mysqlx.getSession(connect);
49 warenkorbLeeren(session);

```

```
50 mySession.close();
51
52
53 ///////////////////////////////////////////////////////////////////
54 // Skript 3: Kunde 2 legt drei neue Warenkorbposition an (Seite 296)
55 function warenkorbPositionEinfügen(mySession, myKundeId, myArtikelId, myMenge)
56 {
57     var dbOshop = mySession.getSchema('oshop');
58     var collWarenkörbe = dbOshop.getCollection('warenkörbe');
59     var sql = "SELECT";
60     sql += " artikel_id, bezeichnung, einzelpreis, waehrung";
61     sql += " FROM oshop.artikel";
62     sql += " WHERE artikel_id=?";
63     var artikel = mySession.sql(sql).bind(myArtikelId).execute().fetchOne();
64     var objArtikel = {
65         artikel_id: artikel.artikel_id,
66         artikelname: artikel.bezeichnung,
67         preis: artikel.einzelpreis
68     };
69     var objPosition = {
70         artikel: objArtikel,
71         menge: myMenge,
72         gesamtpreis: myMenge * artikel.einzelpreis,
73         währung: artikel.waehrung
74     };
75     collWarenkörbe.modify({_id=:param}).arrayInsert('warenkorb.positionen[0]', {
76         position: objPosition
77     }).bind('param', myKundeId).execute();
78 }
79
80 var session = mysqlx.getSession(connect);
81 warenkorbPositionEinfügen(session, 2, 3007, 10);
82 warenkorbPositionEinfügen(session, 2, 9010, 5);
83 warenkorbPositionEinfügen(session, 2, 3010, 2);
84 session.close();
85
86 ///////////////////////////////////////////////////////////////////
87 // Skript 4: Aktualisierung von Gesamtpreis und Warenwert pro Währung (Seite 298)
88 function aktualisiereGesamtpreisWarenwert(myWarenkorb) {
89     var ergebnis = Object();
90     var posindizes = Object.keys(myWarenkorb.warenkorb.positionen);
91     posindizes.forEach(function(posindex) { // Positionen durchwandern
92         var myPos = myWarenkorb.warenkorb.positionen[posindex].position;
93         // gesamtpreis der Position berechnen
94         myPos.gesamtpreis = myPos.artikel.preis * myPos.menge;
95         if (ergebnis.hasOwnProperty(myPos.währung)) {
96             ergebnis[myPos.währung] += myPos.gesamtpreis;
97         } else {
98             ergebnis[myPos.währung] = myPos.gesamtpreis;
99         }
100    });
101    var warenwert = new Array();
102    for (index in Object.keys(ergebnis)) {
103        var währung = Object.keys(ergebnis)[index];
104        var wert = ergebnis[währung];
105        warenwert.push({ [währung]: wert });
106    }
}
```

```

107     myWarenkorb.warenkorb.warenwerte = warenwert;
108     return myWarenkorb;
109 }
110 }
111
112
113 ///////////////////////////////////////////////////////////////////
114 // Skript 3a: insert mit Aktualisierung (Seite 298)
115 function warenkorbPositionEinfügen(mySession, myKundeId, myArtikelId, myMenge)
116 {
117     var dbOshop = mySession.getSchema('oshop');
118     var collWarenkörbe = dbOshop.getCollection('warenkörbe');
119     var sql = "SELECT";
120     sql += " artikel_id, bezeichnung, einzelpreis, waehrung";
121     sql += " FROM oshop.artikel";
122     sql += " WHERE artikel_id=?";
123     var artikel = mySession.sql(sql).bind(myArtikelId).execute().fetchOne();
124     var objArtikel = {
125         artikel_id: artikel.artikel_id,
126         artikelname: artikel.bezeichnung,
127         preis: artikel.einzelpreis
128     };
129     var objPosition = {
130         artikel: objArtikel,
131         menge: myMenge,
132         gesamtpreis: myMenge * artikel.einzelpreis,
133         währung: artikel.waehrung
134     };
135     collWarenkörbe.modify('_id=:param').arrayInsert('warenkorb.positionen[0]', {
136         position: objPosition
137     }).bind('param', myKundeId).execute();
138     var wk = collWarenkörbe.find("_id=:param").bind('param', myKundeId).execute
139         ().fetchOne();
140     wk = aktualisiereGesamtpreisWarenwert(wk);
141     collWarenkörbe.modify('_id=:param').set('$', wk).bind('param', myKundeId).
142         execute();
143
144     var session = mysqlx.getSession(connect);
145     warenkorbPositionEinfügen(session, 2, 3007, 10);
146     warenkorbPositionEinfügen(session, 2, 9010, 5);
147     warenkorbPositionEinfügen(session, 2, 3010, 2);
148     session.close();
149
150
151 ///////////////////////////////////////////////////////////////////
152 // Skript 5: Auswertung Warenkorb mit Warengruppe (Seite 300)
153 function printWarenkorbMitWarengruppen(mySession, myKundeId) {
154     var dbOshop = mySession.getSchema('oshop');
155     var collWarenkörbe = dbOshop.getCollection('warenkörbe');
156
157     var select = "SELECT";
158     select += " artikel.artikel_id,";
159     select += " GROUP_CONCAT(warengruppe.bezeichnung) 'warengruppen'";
160     select += " FROM";
161     select += " oshop.artikel INNER JOIN oshop.artikel_nm_warengruppe";
162     select += " USING(artikel_id)";

```

```
163     select += "    INNER JOIN";
164     select += "    oshop.warengruppe";
165     select += "    USING(warengruppe_id)";
166     select += " GROUP BY artikel.artikel_id";
167     var alleArtikel = mySession.sql(select).execute().fetchAll();
168     var arrArtikel = new Array();
169
170     for (index in alleArtikel) {
171         var zeile = alleArtikel[index];
172         arrArtikel.push({
173             artikel_id: zeile["artikel_id"],
174             warengruppen: zeile["warengruppen"]
175         });
176     }
177
178     var wk = collWarenkörbe.find({_id=:param}).bind('param', myKundeId).execute
179         ().fetchOne();
180     var n = 1;
181     var items = Object.keys(wk.warenkorb.positionen);
182     items.forEach(function(item) {
183         var myPos = wk.warenkorb.positionen[item].position;
184         var wg = arrArtikel.find(function(element) {
185             return element.artikel_id == myPos.artikel.artikel_id
186         });
187         print('Warenkorbposition ' + n++ + ':');
188         print(' ' + myPos.artikel.artikel_id);
189         print(' ' + myPos.artikel.artikelname);
190         print(' ' + wg.warengruppen + "'");
191         print(' ' + myPos.artikel.preis);
192         print(' ' + myPos.menge);
193         print(' ' + myPos.gesamtpreis);
194         print(' ' + myPos.währung + "\n");
195     });
196     var session = mysqlx.getSession(connect);
197     printWarenkorbMitWarengruppen(session, 2);
198     session.close();
199
200
201 //////////////////////////////////////////////////////////////////
202 // Skript 6: Auswertung Warenkorb 2 Anzahl der Bestellungen pro Warengruppe (Seite 301)
203 function erweiternUmWarengruppen(mySession, myKundeId) {
204     var db0shop = mySession.getSchema('oshop');
205     var collWarenkörbe = db0shop.getCollection('warenkörbe');
206     var wk = collWarenkörbe.find({_id=:param}).bind('param', myKundeId).execute
207         ().fetchOne();
208
209     var items = Object.keys(wk.warenkorb.positionen);
210     items.forEach(function(item) {
211         var myPos = wk.warenkorb.positionen[item].position;
212
213         var select = "SELECT";
214         select += " warengruppe_id, bezeichnung";
215         select += " FROM";
216         select += " oshop.artikel_nm_warengruppe";
217         select += " INNER JOIN";
218         select += " oshop.warengruppe";
219         select += " USING(warengruppe_id)";
```

```

219     select += " WHERE artikel_id = ?";
220
221     var wgs = mySession.sql(select).bind(myPos.artikel.artikel_id).execute().
222         fetchAll();
223     arr = new Array();
224     for (index in wgs) {
225         var zeile = wgs[index];
226         arr.push({
227             warengruppe_id: zeile.warengruppe_id,
228             bezeichnung: zeile.bezeichnung
229         });
230     }
231     myPos.artikel.warenguppen = arr;
232 };
233 collWarenkörbe.modify('_id=:param').set('$', wk).bind('param', myKundeId).
234     execute();
235
236
237 var session = mysqlx.getSession(connect);
238 erweiternUmWarenguppen(session, 2);
239 session.close();
240
241
242 ///////////////////////////////////////////////////////////////////
243 // Skript 7: Auswertung Anzahl der Bestellungen pro Warengruppe (Seite 302)
244 function getAnzahlWarengruppe(mySession, myKundeId) {
245     var wk = erweiternUmWarenguppen(mySession, myKundeId);
246
247     var ergebnis = Object();
248     var posindizes = Object.keys(wk.warenkorb.positionen);
249     posindizes.forEach(function(posindex) {
250         var myPos = wk.warenkorb.positionen[posindex].position;
251         var wgs = myPos.artikel.warenguppen;
252         var wgindizes = Object.keys(wgs);
253         wgindizes.forEach(function(wgindex) {
254             if (ergebnis.hasOwnProperty(wgs[wgindex].bezeichnung)) {
255                 ergebnis[wgs[wgindex].bezeichnung]++;
256             } else {
257                 ergebnis[wgs[wgindex].bezeichnung] = 1;
258             }
259         });
260     });
261
262     return ergebnis;
263 }
264
265 var session = mysqlx.getSession(connect);
266 print(getAnzahlBestellungenProWarengruppe(session, 2));
267 session.close();
268
269
270 ///////////////////////////////////////////////////////////////////
271 // Skript 8: Ändern von Mengenangaben (Seite 303)
272 function mengenÄndern(mySession, myKundeId, myArtikelId, myMenge) {
273     var dbOshop = mySession.getSchema('oshop');
274     var collWarenkörbe = dbOshop.getCollection('warenkörbe');

```

```

275     var wk = collWarenkörbe.find("_id=:param").bind('param', myKundeId).execute
276         ().fetchOne();
277
278     for (index in Object.keys(wk.warenkorb.positionen)) {
279         if (wk.warenkorb.positionen[index].position.artikel.artikel_id ==
280             myArtikelId) {
281             wk.warenkorb.positionen[index].position.menge = myMenge;
282         }
283     wk = aktualisiereGesamtpreisWarenwert(wk);
284     collWarenkörbe.modify('_id=:param').set('$', wk).bind('param', myKundeId).
285         execute();
286
287     return wk;
288 }
289
290 var session = mysqlx.getSession(connect);
291 mengenÄndern(session, 2, 3010, 0);
292 mengenÄndern(session, 2, 3007, 1);
293 mengenÄndern(session, 2, 9010, 2);
294 session.close();
295
296 // Skript 9: Löschen von Positionen (Seite 303)
297 function warenkorbPositionLöschen(mySession, myKundeId, myArtikelId) {
298     var db0shop = mySession.getSchema('oshop');
299     var collWarenkörbe = db0shop.getCollection('warenkörbe');
300     var wk = collWarenkörbe.find("_id=:param").bind('param', myKundeId).execute
301         ().fetchOne();
302
303     for (index in Object.keys(wk.warenkorb.positionen)) {
304         if (wk.warenkorb.positionen[index].position.artikel.artikel_id ==
305             myArtikelId) {
306             var str = 'warenkorb.positionen[' + index + ']';
307             collWarenkörbe.modify('_id=:param').arrayDelete(str).bind('param',
308                 myKundeId).execute();
309             wk = collWarenkörbe.find("_id=:param").bind('param', myKundeId).execute
310                 ().fetchOne();
311             wk = aktualisiereGesamtpreisWarenwert(wk);
312         }
313     }
314
315     var session = mysqlx.getSession(connect);
316     warenkorbPositionLöschen(session, 2, 3007);
317     session.close();

```

Listing 29.15 mysql/listing13NoSQL.sql

```

1 -- Ausführen der vorherigen Befehle
2 -- SOURCE listing12.sql
3
4 SELECT 'Start listingNoSQL.sql //////////////////////////////';
5 SELECT 'Hinzufügen der JSON-Spalte in kunde (Seite 304)';
6
7 ALTER TABLE kunde

```

```

8   ADD warenkorb JSON AFTER art
9 ;
10
11 UPDATE kunde
12   SET warenkorb = JSON_OBJECT(
13     'warenkorb',
14       JSON_OBJECT(
15         'kunde_id', kunde_id,
16         'positionen', JSON_ARRAY(),
17           'warenwerte', JSON_ARRAY()
18         )
19       )
20 ;
21
22 SELECT 'Hinzufügen von Warenkorbpositionen (Seite 304)';
23 SELECT 'Kunde 2 10mal 3007';
24 SELECT 2 INTO @kunde_id;
25 SELECT 3007 INTO @artikel_id;
26 SELECT 10 INTO @menge;
27
28 SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id INTO @wk;
29 SELECT JSON_EXTRACT(@wk, "$.warenkorb.positionen");
30
31 SELECT
32   JSON_OBJECT(
33     'position', JSON_OBJECT(
34       'artikel', JSON_OBJECT(
35         'artikel_id', CAST(artikel_id AS CHAR),
36           'bezeichnung', bezeichnung,
37             'einzelpreis', einzelpreis
38           ),
39         'menge', @menge,
40           'gesamtpreis', @menge * einzelpreis,
41             'währung', waehrung
42           )
43     )
44   FROM artikel
45   WHERE artikel_id = @artikel_id
46   INTO @position
47 ;
48
49 UPDATE kunde
50   SET warenkorb = JSON_ARRAY_APPEND(
51     @wk,
52       "$.warenkorb.positionen",
53         CAST(@position AS JSON)
54       )
55   WHERE kunde_id = @kunde_id
56 ;
57
58 SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id\G
59
60 SELECT 'Kunde 2 2mal 3010';
61 SELECT 2 INTO @kunde_id;
62 SELECT 3010 INTO @artikel_id;
63 SELECT 2 INTO @menge;
64
65 SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id INTO @wk;
66 SELECT JSON_EXTRACT(@wk, "$.warenkorb.positionen");

```

```
67
68 SELECT
69   JSON_OBJECT(
70     'position', JSON_OBJECT(
71       'artikel', JSON_OBJECT(
72         'artikel_id', CAST(artikel_id AS CHAR),
73         'bezeichnung', bezeichnung,
74         'einzelpreis', einzelpreis
75       ),
76       'menge', @menge,
77       'gesamtpreis', @menge * einzelpreis,
78       'währung', waehrung
79     )
80   )
81 FROM artikel
82 WHERE artikel_id = @artikel_id
83 INTO @position
84 ;
85
86 UPDATE kunde
87 SET warenkorb = JSON_ARRAY_APPEND(
88   @wk,
89   "$.warenkorb.positionen",
90   CAST(@position AS JSON)
91 )
92 WHERE kunde_id = @kunde_id
93 ;
94
95 SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id\G
96
97 SELECT 'Kunde 2 5mal 9010';
98 SELECT 2 INTO @kunde_id;
99 SELECT 9010 INTO @artikel_id;
100 SELECT 5 INTO @menge;
101
102 SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id INTO @wk;
103 SELECT JSON_EXTRACT(@wk, "$.warenkorb.positionen");
104
105 SELECT
106   JSON_OBJECT(
107     'position', JSON_OBJECT(
108       'artikel', JSON_OBJECT(
109         'artikel_id', CAST(artikel_id AS CHAR),
110         'bezeichnung', bezeichnung,
111         'einzelpreis', einzelpreis
112       ),
113       'menge', @menge,
114       'gesamtpreis', @menge * einzelpreis,
115       'währung', waehrung
116     )
117   )
118 FROM artikel
119 WHERE artikel_id = @artikel_id
120 INTO @position
121 ;
122
123 UPDATE kunde
124 SET warenkorb = JSON_ARRAY_APPEND(
125   @wk,
```

```

126          "$.warenkorb.positionen",
127                      CAST(@position AS JSON)
128      )
129 WHERE kunde_id = @kunde_id
130 ;
131
132 SELECT JSON_PRETTY(warenkorb) FROM kunde WHERE kunde_id = @kunde_id\G
133
134
135 SELECT 'Auswerten von Warenkorbpositionen';
136
137 SELECT 2 INTO @kunde_id;
138 SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id INTO @wk;
139
140 SELECT @kunde_id AS 'kunde_id', tt.*
141   FROM
142 JSON_TABLE(
143     @wk,
144     '$.warenkorb.positionen[*]'
145     COLUMNS (
146       posnr      FOR ORDINALITY,
147       artikel_id INT           PATH '$.position.artikel.artikel_id',
148       einzelpreis DECIMAL(12,2) PATH '$.position.artikel.einzelpreis',
149       menge      INT           PATH '$.position.menge',
150       währung    CHAR(3)        PATH '$.position.währung'
151     )
152   ) AS tt
153 ;
154
155 SELECT bezeichnung AS 'warengruppe', COUNT(*) AS 'anzahl'
156   FROM
157 JSON_TABLE(
158   @wk,
159   '$.warenkorb.positionen[*]'
160   COLUMNS (
161     posnr      FOR ORDINALITY,
162     artikel_id INT           PATH '$.position.artikel.artikel_id',
163     einzelpreis DECIMAL(12,2) PATH '$.position.artikel.einzelpreis',
164     menge      INT           PATH '$.position.menge',
165     währung    CHAR(3)        PATH '$.position.währung'
166   )
167   ) AS wk
168 INNER JOIN artikel_nm_warengruppe USING(artikel_id)
169 INNER JOIN warengruppe USING(warengruppe_id)
170 GROUP BY bezeichnung
171 ;
172
173 WITH wk AS (
174   SELECT * FROM JSON_TABLE(
175     @wk,
176     '$.warenkorb.positionen[*]'
177     COLUMNS (
178       posnr FOR ORDINALITY,
179       artikel_id INT PATH '$.position.artikel.artikel_id',
180       einzelpreis DECIMAL(12,2) PATH '$.position.artikel.einzelpreis',
181       menge INT PATH '$.position.menge',
182       währung CHAR(3) PATH '$.position.währung'
183     )
184   ) AS t

```

```
185 )
186 SELECT bezeichnung AS 'warengruppe', COUNT(*) AS 'anzahl'
187   FROM
188     wk INNER JOIN artikel_nm_warengruppe USING(artikel_id)
189       INNER JOIN warengruppe USING(warengruppe_id)
190 GROUP BY bezeichnung
191 ;
192
193
194 SELECT 'Ändern von Warenkorbpositionen (Seite 307)';
195
196 SELECT 'Kunde 2 1mal 3007';
197 SELECT 2 INTO @kunde_id;
198 SELECT 3007 INTO @artikel_id;
199 SELECT 1 INTO @menge;
200
201 SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id INTO @wk;
202 SELECT JSON_SEARCH(@wk, 'one', @artikel_id) INTO @path;
203
204 SELECT CONCAT(
205   SUBSTRING(
206     @path,
207     2,
208     22+LOCATE('.', LEFT(@path, 13))
209   ),
210   '.position.menge'
211 )
212 INTO @path_to_pos;
213
214 UPDATE kunde
215 SET warenkorb = JSON_REPLACE(@wk, @path_to_pos, @menge)
216 WHERE kunde_id = @kunde_id
217 ;
218 SELECT JSON_PRETTY(warenkorb) FROM kunde WHERE kunde_id = @kunde_id\G
219
220 SELECT 'Kunde 2 0mal 3010';
221 SELECT 2 INTO @kunde_id;
222 SELECT 3010 INTO @artikel_id;
223 SELECT 0 INTO @menge;
224
225 SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id INTO @wk;
226 SELECT JSON_SEARCH(@wk, 'one', @artikel_id) INTO @path;
227 SELECT CONCAT(
228   SUBSTRING(
229     @path,
230     2,
231     22+LOCATE('.', LEFT(@path, 13))
232   ),
233   '.position.menge'
234 )
235 INTO @path_to_pos;
236
237 SELECT JSON_REPLACE(@wk, @path_to_pos, @menge);
238 UPDATE kunde
239 SET warenkorb = JSON_REPLACE(@wk, @path_to_pos, @menge)
240 WHERE kunde_id = @kunde_id
241 ;
242 SELECT JSON_PRETTY(warenkorb) FROM kunde WHERE kunde_id = @kunde_id\G
243
```

```

244 SELECT 'Kunde 2 10mal 9010';
245 SELECT 2 INTO @kunde_id;
246 SELECT 9010 INTO @artikel_id;
247 SELECT 10 INTO @menge;
248
249 SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id INTO @wk;
250 SELECT JSON_SEARCH(@wk, 'one', @artikel_id) INTO @path;
251 SELECT CONCAT(
252     SUBSTRING(
253         @path,
254         2,
255         22+LOCATE('.', LEFT(@path, 13))
256     ),
257     '.position.menge'
258 )
259 INTO @path_to_pos;
260
261 SELECT JSON_REPLACE(@wk, @path_to_pos, @menge);
262 UPDATE kunde
263 SET warenkorb = JSON_REPLACE(@wk, @path_to_pos, @menge)
264 WHERE kunde_id = @kunde_id
265 ;
266 SELECT JSON_PRETTY(warenkorb) FROM kunde WHERE kunde_id = @kunde_id\G
267
268
269 SELECT 'Kunde 2 1mal 3007 mit Aktualisierungen (Seite 309)';
270 SELECT 2 INTO @kunde_id;
271 SELECT 3007 INTO @artikel_id;
272 SELECT 1 INTO @menge;
273
274 SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id INTO @wk;
275 SELECT JSON_SEARCH(@wk, 'one', @artikel_id) INTO @path;
276 SELECT CONCAT(SUBSTRING(@path, 2, 22+LOCATE('.', LEFT(@path, 13))),'.position.
    menge')
277 INTO @path_to_pos;
278 SELECT CONCAT(SUBSTRING(@path, 2, 22+LOCATE('.', LEFT(@path, 13))),'.position.
    gesamtpreis')
279 INTO @path_to_gesamtpreis;
280 SELECT '$.warenkorb.warenwerte' INTO @path_to_warenwerte;
281
282 CREATE TEMPORARY TABLE wk
283 SELECT * FROM JSON_TABLE(
284     @wk,
285     '$.warenkorb.positionen[*]'
286     COLUMNS (
287         posnr FOR ORDINALITY,
288         artikel_id INT PATH '$.position.artikel.artikel_id',
289         einzelpreis DECIMAL(12,2) PATH '$.position.artikel.einzelpreis',
290         menge INT PATH '$.position.menge',
291         währung CHAR(3) PATH '$.position.währung'
292     )
293 ) AS t
294 ;
295
296 SELECT einzelpreis*menge
297 FROM wk
298 WHERE artikel_id = @artikel_id
299 INTO @gesamtpreis
300 ;

```

```

301
302 SELECT JSON_ARRAYAGG(warenwert)
303   FROM
304     (SELECT JSON_OBJECT(währung, SUM(einzelpreis*menge)) warenwert
305      FROM wk
306     GROUP BY währung) AS a
307 INTO @warenwerte
308 ;
309
310 UPDATE kunde
311   SET warenkorb = JSON_REPLACE(@wk,
312                               @path_to_pos, @menge,
313                               @path_to_gesamtpreis, @gesamtpreis,
314                               @path_to_warenwerte, CAST(@warenwerte AS JSON))
315 WHERE kunde_id = @kunde_id
316 ;
317 SELECT JSON_PRETTY(warenkorb) FROM kunde WHERE kunde_id = @kunde_id\G
318
319 SELECT 'Löschen von Warenkorbpositionen (Seite 312)';
320
321 SELECT 'Kunde 2 Position mit Artikel 9010';
322 SELECT 2 INTO @kunde_id;
323 SELECT 9010 INTO @artikel_id;
324
325 SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id INTO @wk;
326 SELECT JSON_SEARCH(@wk, 'one', @artikel_id) INTO @path;
327
328 SELECT SUBSTRING(@path, 2, 22+LOCATE('.', LEFT(@path, 13)))
329   INTO @path_to_pos;
330 UPDATE kunde
331   SET warenkorb = JSON_REMOVE(@wk, @path_to_pos)
332 WHERE kunde_id = @kunde_id
333 ;
334 SELECT JSON_PRETTY(warenkorb) FROM kunde WHERE kunde_id = @kunde_id\G
335
336 SELECT 'Kunde 2 Warenkorb leeren';
337 SELECT 2 INTO @kunde_id;
338
339 SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id INTO @wk;
340
341 UPDATE kunde
342   SET warenkorb = JSON_REPLACE(@wk,
343                               '$.warenkorb.positionen', JSON_ARRAY(),
344                               '$.warenkorb.warenwerte', JSON_ARRAY())
345 WHERE kunde_id = @kunde_id
346 ;
347 SELECT JSON_PRETTY(warenkorb) FROM kunde WHERE kunde_id = @kunde_id\G
348 SELECT 'Ende listingNoSQL.sql /////////////////////////////////';

```

29.1.4 Quelltexte zu Teil V

Listing 29.16 mysql/listing14.sql

```

1 -- Ausführen der vorherigen Befehle
2 SOURCE listing13.sql
3

```

```

4  SELECT 'Start listing14.sql //////////////////////////////';
5  SELECT 'Lagerbestand (Seite 315)';
6  UPDATE lagerbestand
7  SET menge_aktuell = menge_aktuell - 5
8  WHERE artikel_id = 9015;
9
10 UPDATE lagerbestand
11 SET menge_aktuell = menge_aktuell - 3
12 WHERE artikel_id = 9015;
13
14 SELECT * FROM lagerbestand WHERE artikel_id = 9015;
15
16 -- Start Lock
17 SELECT 'LOCK (Seite 318)';
18 LOCK TABLES artikel WRITE;
19 SELECT COUNT(*) FROM artikel;
20 -- SELECT 'Erwarte eine Fehlermeldung (Seite 290)';
21 -- SELECT COUNT(*) FROM warengruppe;
22 UNLOCK TABLES;
23 SELECT COUNT(*) FROM warengruppe;
24 -- Ende Lock
25 SELECT 'Ende listing14.sql //////////////////////////////';

```

Listing 29.17 mysql/listing15.sql

```

1  -- Ausführen der vorherigen Befehle
2  SOURCE listing14.sql
3
4  SELECT 'Start listing15.sql //////////////////////////////';
5  SELECT 'Das Problem (Seite 320)';
6  INSERT INTO artikel
7  (bezeichnung, einzelpreis, waehrung)
8  VALUES
9  ('Säge', 17.85, 'EUR')
10 ;
11 SET @id = LAST_INSERT_ID();
12 INSERT INTO artikel_nm_warengruppe
13 VALUES
14 (3, @id),
15 (4, @id)
16 ;
17 UPDATE lagerbestand
18 SET menge_aktuell = menge_aktuell - 1
19 WHERE artikel_id = 9015
20 ;
21 UPDATE bestellung SET status = 'versendet' WHERE bestellung_id = 1;
22
23 SELECT 'Isolationsebenen (Seite 324)';
24 SHOW VARIABLES LIKE 'AUTOCOMMIT';
25 SET AUTOCOMMIT = OFF;
26 SHOW VARIABLES LIKE 'AUTOCOMMIT';
27
28
29 SELECT 'READ UNCOMMITTED (Seite 325)';
30 -- Session 1
31 SET AUTOCOMMIT = 0;
32
33 SELECT artikel_id, menge_aktuell
34   FROM lagerbestand

```

```
35 WHERE artikel_id = 9015
36 ;
37 UPDATE lagerbestand
38 SET menge_aktuell = menge_aktuell - 1
39 WHERE artikel_id = 9015
40 ;
41
42 SELECT artikel_id, menge_aktuell
43 FROM lagerbestand
44 WHERE artikel_id = 9015
45 ;
46 ROLLBACK;
47 SELECT artikel_id, menge_aktuell
48 FROM lagerbestand
49 WHERE artikel_id = 9015
50 ;
51 -- Session 2
52 SET SESSION TRANSACTION ISOLATION LEVEL
53 READ UNCOMMITTED;
54
55 SELECT artikel_id, menge_aktuell
56 FROM lagerbestand
57 WHERE artikel_id = 9015
58 ;
59 SELECT artikel_id, menge_aktuell
60 FROM lagerbestand
61 WHERE artikel_id = 9015
62 ;
63 SELECT artikel_id, menge_aktuell
64 FROM lagerbestand
65 WHERE artikel_id = 9015
66 ;
67 -- Ende read uncommitted
68
69 SELECT 'READ COMMITTED (Seite 326)';
70 -- Session 1
71 SET AUTOCOMMIT = 0;
72 SELECT artikel_id, menge_aktuell
73 FROM lagerbestand
74 WHERE artikel_id = 9015
75 ;
76 UPDATE lagerbestand
77 SET menge_aktuell = menge_aktuell - 1
78 WHERE artikel_id = 9015
79 ;
80 SELECT artikel_id, menge_aktuell
81 FROM lagerbestand
82 WHERE artikel_id = 9015
83 ;
84 COMMIT;
85 SELECT artikel_id, menge_aktuell
86 FROM lagerbestand
87 WHERE artikel_id = 9015;
88 -- Session 2
89 SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
90 SELECT artikel_id, menge_aktuell
91 FROM lagerbestand
92 WHERE artikel_id = 9015
93 ;
```

```
94  SELECT artikel_id, menge_aktuell
95    FROM lagerbestand
96   WHERE artikel_id = 9015
97 ;
98  SELECT artikel_id, menge_aktuell
99    FROM lagerbestand
100  WHERE artikel_id = 9015
101 ;
102 -- Ende read committed
103
104 SELECT 'REPEATABLE READ (Seite 327)';
105 -- Session 1
106 SET AUTOCOMMIT = 0;
107 SELECT artikel_id, menge_aktuell
108   FROM lagerbestand
109  WHERE artikel_id = 9015
110 ;
111 UPDATE lagerbestand
112   SET menge_aktuell = menge_aktuell - 1
113  WHERE artikel_id = 9015
114 ;
115 COMMIT;
116 UPDATE lagerbestand
117   SET menge_aktuell = menge_aktuell - 1
118  WHERE artikel_id = 9015
119 ;
120 COMMIT;
121 SELECT artikel_id, menge_aktuell
122   FROM lagerbestand
123  WHERE artikel_id = 9015
124 ;
125 -- Session 2
126 SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
127 START TRANSACTION;
128 SELECT artikel_id, menge_aktuell
129   FROM lagerbestand
130  WHERE artikel_id = 9015
131 ;
132 SELECT artikel_id, menge_aktuell
133   FROM lagerbestand
134  WHERE artikel_id = 9015
135 ;
136 COMMIT;
137 SELECT artikel_id, menge_aktuell
138   FROM lagerbestand
139  WHERE artikel_id = 9015
140 ;
141 -- Ende repeatable read
142
143 SELECT 'SERIALIZABLE (Seite 328)';
144 -- Session 1
145 SET AUTOCOMMIT = 0;
146 SELECT COUNT(*) FROM artikel;
147 INSERT INTO artikel VALUES (1, 'X', 0.0, 'EUR', 0);
148 COMMIT;
149
150 -- Session 2
151 SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
152 START TRANSACTION;
```

```
153 DELETE FROM artikel WHERE artikel_id = 1;
154 DELETE FROM artikel WHERE artikel_id = 1;
155 SELECT COUNT(*) FROM artikel;
156 -- Ende Serializable
157 SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
158 SELECT 'Ende listing15.sql /////////////////////////////////';
```

Listing 29.18 mysql/listing16.sql

```
1 -- Ausführen der vorherigen Befehle
2 SOURCE listing15.sql
3
4 SELECT 'Start listing16.sql /////////////////////////////////';
5
6 -- Start Prozedur insert_artikel Version 1
7 SELECT 'Prozedur insert_artikel Version 1 (Seite 337)';
8 DELIMITER //
9 CREATE PROCEDURE insert_artikel
10 (
11     IN iArtikelname VARCHAR(255),
12     IN iWarengruppe VARCHAR(255)
13 )
14
15 BEGIN
16
17     DECLARE v_artikel_id      INT DEFAULT 0;
18     DECLARE v_warengruppe_id INT DEFAULT 0;
19
20
21     INSERT INTO artikel
22         (bezeichnung)
23     VALUES (iArtikelname);
24
25     SET v_artikel_id = LAST_INSERT_ID();          -- Variable setzen
26
27     SELECT warengruppe_id
28         FROM warengruppe
29         WHERE bezeichnung = iWarengruppe
30         INTO v_warengruppe_id;                    -- Oder so
31
32     INSERT INTO artikel_nm_warengruppe        -- Eintrag Hilfstabelle
33         (warengruppe_id, artikel_id)
34     VALUES
35         (v_warengruppe_id, v_artikel_id);
36
37 END//                                     -- Anderer DELIMITER
38 DELIMITER ;
39 -- Start Prozedur insert_artikel Version 1 Seite 338
40 CALL insert_artikel ('Schlauch', 'Gartenbedarf');
41 SELECT artikel_id, bezeichnung FROM artikel ORDER BY artikel_id;
42 SELECT warengruppe_id, bezeichnung FROM warengruppe;
43 SELECT * FROM artikel_nm_warengruppe;
44 SHOW PROCEDURE STATUS\G
45
46 -- SELECT 'Verzweigung. Erwarte eine Fehlermeldung (Seite 339) ';
47 -- CALL insert_artikel ('Fleischwurst', 'Lebensmittel');
48
49 -- Start Prozedur insert_artikel Version 2 (Seite 340)
50 DROP PROCEDURE IF EXISTS insert_artikel;
```

```

51  DELIMITER //
52  CREATE PROCEDURE insert_artikel
53  (
54      IN iArtikelname VARCHAR(255),
55      IN iWarengruppe VARCHAR(255)
56  )
57  )
58
59 BEGIN
60
61  DECLARE v_artikel_id      INT DEFAULT -1;
62  DECLARE v_warengruppe_id  INT DEFAULT -1;
63  DECLARE v_count_zuordnung INT DEFAULT 0;
64
65  -- Teste, ob Warengruppe vorhanden ist
66  SELECT warengruppe_id
67      FROM warengruppe
68      WHERE bezeichnung = iWarengruppe
69      INTO v_warengruppe_id;
70
71  -- Wenn nein, füge eine ein
72  IF v_warengruppe_id < 0 THEN          -- Einfache Verzweigung
73      INSERT INTO warengruppe (bezeichnung) VALUES (iWarengruppe);
74      SET v_warengruppe_id = LAST_INSERT_ID();
75  END IF;
76
77  -- Teste, ob artikel vorhanden ist
78  SELECT artikel_id
79      FROM artikel
80      WHERE bezeichnung = iArtikelname
81      INTO v_artikel_id;
82
83  -- Wenn nein, teste auf füge ein
84  IF v_artikel_id < 0 THEN
85      INSERT INTO artikel (bezeichnung) VALUES (iArtikelname);
86      SET v_artikel_id = LAST_INSERT_ID();
87  END IF;
88
89  -- Test, ob Zuordnung schon vorhanden
90  SELECT COUNT(*)
91      FROM artikel_nm_warengruppe
92      WHERE artikel_id = v_artikel_id AND warengruppe_id = v_warengruppe_id
93      INTO v_count_zuordnung;
94
95  -- Wenn nein, füge eine hinzu
96  IF v_count_zuordnung <= 0 THEN
97      INSERT INTO artikel_nm_warengruppe
98          (warengruppe_id, artikel_id)
99          VALUES
100             (v_warengruppe_id, v_artikel_id);
101 ELSE
102     SELECT 'Zuordnung schon vorhanden';      -- Hinweis
103 END IF;
104 END///
105 DELIMITER ;
106 -- Ende Prozedur insert_artikel Version 2
107 CALL insert_artikel ('Fleischwurst', 'Lebensmittel');
108 CALL insert_artikel ('Fleischwurst', 'Lebensmittel');
109

```

```
110 SELECT 'Mehrfachverzweigung (Seite 342)';
111 -- Start Prozedur Umsatzreport
112 DELIMITER //
113 CREATE PROCEDURE umsatzreport
114 (
115     IN iZeitraum INT
116 )
117 BEGIN
118     DROP TABLE IF EXISTS umsatzzahlen;
119
120     CASE iZeitraum
121         WHEN 1 THEN      -- Total
122             CREATE TABLE umsatzzahlen
123                 SELECT 'Total', SUM(einzelpreis * menge) 'Umsatz'
124                 FROM
125                     rechnung_position INNER JOIN artikel USING (artikel_id)
126                         INNER JOIN rechnung USING (rechnung_id)
127 ;
128         WHEN 2 THEN      -- Pro Jahr
129             CREATE TABLE umsatzzahlen
130                 SELECT YEAR(datum) 'Jahr', SUM(einzelpreis * menge) 'Umsatz'
131                 FROM
132                     rechnung_position INNER JOIN artikel USING (artikel_id)
133                         INNER JOIN rechnung USING (rechnung_id)
134                 GROUP BY 'Jahr'
135                 ORDER BY 'Jahr' DESC
136 ;
137         WHEN 3 THEN      -- Pro Monat
138             CREATE TABLE umsatzzahlen
139                 SELECT YEAR(datum) 'Jahr', MONTH(datum) 'Monat', SUM(einzelpreis * menge)
140                     'Umsatz'
141                 FROM
142                     rechnung_position INNER JOIN artikel USING (artikel_id)
143                         INNER JOIN rechnung USING (rechnung_id)
144                 GROUP BY 'Jahr', 'Monat'
145                 ORDER BY 'Jahr' DESC, 'Monat' DESC
146 ;
147         ELSE            -- Sowas sollte es immer geben
148             CREATE TABLE umsatzzahlen
149                 SELECT 'Unbekannte Berichtsart', iZeitraum;
150     END CASE;
151 END//
152 DELIMITER ;
153 -- Ende Prozedur Umsatzreport
154 CALL umsatzreport(1); SELECT * FROM umsatzzahlen;
155 CALL umsatzreport(2); SELECT * FROM umsatzzahlen;
156 CALL umsatzreport(3); SELECT * FROM umsatzzahlen;
157 CALL umsatzreport(0); SELECT * FROM umsatzzahlen;
158
159 -- Start Prozedur b (Seite 344)
160 DELIMITER //
161 CREATE PROCEDURE b()
162 BEGIN
163     DECLARE vZeitraum INT ;
164     SET vZeitraum = 2;
165     CASE
166         WHEN vZeitraum = 2 THEN SELECT 'A';
167         WHEN vZeitraum > 0 THEN SELECT 'B';
168         WHEN vZeitraum < 2 THEN SELECT 'C';
169 
```

```
168     END CASE;
169 END//  
170 DELIMITER ;  
171  
172 CALL b();  
173 -- Ende Prozedur b  
174  
175 SELECT 'Schleifen (Seite 346)';  
176 -- Start Prozedur warte_auf_ueberlauf  
177 DELIMITER //  
178 CREATE PROCEDURE warte_auf_ueberlauf()  
179 BEGIN  
180     DECLARE vInt TINYINT;  
181  
182     SET vInt = 0;  
183     LOOP                      -- Schleifenstart  
184         SELECT vInt;  
185         SET vInt = vInt + 1;  
186     END LOOP;                  -- Schleifenende  
187 END;  
188 //  
189 DELIMITER ;  
190  
191 -- CALL warte_auf_ueberlauf();  
192 -- Ende Prozedur warte_auf_ueberlauf  
193  
194 -- Start Prozedur min_max  
195 DELIMITER //  
196 CREATE PROCEDURE min_max()  
197 BEGIN  
198     DECLARE vInt TINYINT;  
199     DECLARE vMin INT;  
200     DECLARE vMax INT;  
201  
202     SET vInt = 0;  
203     maxsuche: LOOP           -- Jetzt mit Label  
204         SET vMax = vInt;  
205         SET vInt = vInt + 1;  
206         IF vMax = vInt THEN  
207             LEAVE maxsuche;    -- Verlasse Label  
208         END IF;  
209     END LOOP maxsuche;  
210  
211     SET vInt = 0;  
212     minsuche: LOOP  
213         SET vMin = vInt;  
214         SET vInt = vInt - 1;  
215         IF vMin = vInt THEN  
216             LEAVE minsuche;  
217         END IF;  
218     END LOOP minsuche;  
219  
220     SELECT 'MIN = ', vMin, 'MAX = ', vMax;  
221 END;  
222 //  
223 DELIMITER ;  
224 CALL min_max();  
225 -- Ende Prozedur min_max  
226
```

```
227 -- Start Prozedur Sieb (Seite 348)
228 DELIMITER //
229 CREATE PROCEDURE sieb
230 (
231     iGrenze BIGINT UNSIGNED
232 )
233 BEGIN
234     DECLARE vZahl BIGINT UNSIGNED DEFAULT 2;          -- Start mit 2
235     DECLARE vPosition BIGINT UNSIGNED DEFAULT 2;
236
237     -- Aufbau der Tabelle
238     DROP TABLE IF EXISTS zahlenstrahl;
239     CREATE TABLE zahlenstrahl
240     (
241         position BIGINT UNSIGNED,
242         zahl BIGINT UNSIGNED,
243         PRIMARY KEY(position)
244     );
245
246     WHILE vZahl <= iGrenze DO
247         INSERT INTO zahlenstrahl VALUES (vZahl, vZahl);
248         SET vZahl = vZahl + 1;
249     END WHILE;
250
251     -- Sieb
252     WHILE vPosition < CEILING(SQRT(iGrenze)) DO
253         SELECT zahl FROM zahlenstrahl WHERE position = vPosition INTO vZahl;
254         IF vZahl > 0 THEN
255             UPDATE zahlenstrahl SET zahl = 0 WHERE (position > vPosition) AND (zahl %
256                                         vZahl = 0);
257         END IF;
258         SET vPosition = vPosition + 1;
259     END WHILE;
260
261     -- Bereinigen der Tabelle
262     DELETE FROM zahlenstrahl WHERE zahl = 0;
263 END//
264 DELIMITER ;
265 CALL sieb(20);
266 SELECT zahl FROM zahlenstrahl ORDER BY zahl;
267 -- Ende Prozedur Sieb
268
269 -- Start Prozedur kopiere_blz (Seite 349)
270 DROP DATABASE IF EXISTS tmpSchleife;
271 CREATE DATABASE tmpSchleife CHARACTER SET utf8;
272
273 USE tmpSchleife;
274
275 CREATE TABLE bank_quelle
276 (
277     blz           CHAR(8),
278     merkmal      CHAR(1),
279     bezeichnung  VARCHAR(255),
280     plz          CHAR(5),
281     ort          VARCHAR(255),
282     kurz         VARCHAR(255),
283     pan          CHAR(5),
284     bic          VARCHAR(255),
```

```

285    sm          CHAR(2),
286    ds          CHAR(6),
287    kz          CHAR(1),
288    blzl        CHAR(1),
289    blzn        CHAR(8)
290  )
291 ;
292
293 LOAD DATA LOCAL INFILE 'blz_20120305.csv'
294   INTO TABLE bank_quelle
295   FIELDS TERMINATED BY ','
296   LINES TERMINATED BY '\n'
297 ;
298 CREATE TABLE bank_ziel
299 (
300   blz          CHAR(8),
301   bezeichnung  VARCHAR(255),
302   adresse      TEXT
303 )
304 ;
305
306 DELIMITER //
307 CREATE PROCEDURE kopiere_blz
308 (
309   iAnzahl INT UNSIGNED           -- Anzahl der Zeilen pro Kopiervorgang
310 )
311 BEGIN
312   DECLARE vOffset INT UNSIGNED;    -- Aktueller Startpunkt
313   DECLARE vCount  INT UNSIGNED;    -- Anzahl der zu kopierenden Zeilen
314
315   SELECT COUNT(*) FROM bank_quelle INTO vCount;
316
317   SET vOffset = 0;
318   WHILE vOffset < vCount DO        -- Start der Schleife
319     SELECT 'vOffset ', vOffset;    -- Nur zum gucken
320
321     INSERT INTO bank_ziel (blz, bezeichnung, adresse)
322       SELECT blz, bezeichnung, CONCAT(plz, ' ', ort)
323         FROM bank_quelle
324         LIMIT vOffset, iAnzahl;
325     SET vOffset = vOffset + iAnzahl; -- Neuer Startwert
326   END WHILE;                      -- Ende der Schleife
327 END///
328 DELIMITER ;
329
330 CALL kopiere_blz(1000);
331 USE oshop;
332 -- Ende Prozedur kopiere_blz
333
334 -- Start Prozedur ggt (Seite 351)
335 DELIMITER //
336 CREATE PROCEDURE ggt
337 (
338   iZahl1 INT UNSIGNED,
339   iZahl2 INT UNSIGNED
340 )
341 BEGIN
342   SET @m = iZahl1;      -- Deklaration durch
343   SET @n = iZahl2;      -- eine Zuweisung

```

```
344
345 REPEAT          -- Schleifenstart
346   SET @t = @m % @n;
347   SET @m = @n;
348   SET @n = @t;
349 UNTIL @n = 0 END REPEAT; -- Schleifenende
350
351 SELECT CONCAT('GGT(',iZahl1,', ',iZahl2,') = ') 'GgT()', @m;
352 END// 
353 DELIMITER ;
354
355 CALL ggt(20, 15);
356 -- Ende Prozedur ggt
357
358 SELECT 'Transaktion innerhalb einer Prozedur (Seite 352)';
359 -- Start Prozedur insert_artikel Version 3
360 DROP PROCEDURE insert_artikel;
361
362 DELIMITER //
363 CREATE PROCEDURE insert_artikel
364 (
365   IN iArtikelname VARCHAR(255),
366   IN iWarengruppe VARCHAR(255)
367 )
368
369 BEGIN
370   DECLARE v_artikel_id      INT DEFAULT -1;
371   DECLARE v_warengruppe_id  INT DEFAULT -1;
372   DECLARE v_count_zuordnung INT DEFAULT 0;
373
374   DECLARE EXIT HANDLER FOR SQLEXCEPTION  -- Handle fuer Fehler
375   BEGIN
376     SELECT 'Wegen Fehler ROLLBACK';
377     ROLLBACK;
378   END;
379
380   START TRANSACTION;
381
382   -- Teste, ob Warengruppe vorhanden ist
383   SELECT warengruppe_id
384   FROM warengruppe
385   WHERE bezeichnung = iWarengruppe
386   INTO v_warengruppe_id;
387
388   -- Wenn nein, fuege eine ein
389   IF v_warengruppe_id < 0 THEN          -- Einfache Verzweigung
390     INSERT INTO warengruppe (bezeichnung) VALUES (iWarengruppe);
391     SET v_warengruppe_id = LAST_INSERT_ID();
392   END IF;
393
394   -- Teste, ob artikel vorhanden ist
395   SELECT artikel_id
396   FROM artikel
397   WHERE bezeichnung = iArtikelname
398   INTO v_artikel_id;
399
400   -- Wenn nein, teste auf fuege ein
401   IF v_artikel_id < 0 THEN
402     INSERT INTO artikel (bezeichnung) VALUES (iArtikelname);
```

```

403     SET v_artikel_id = LAST_INSERT_ID();
404 END IF;
405
406 -- Test, ob Zuordnung schon vorhanden
407 SELECT COUNT(*)
408   FROM artikel_nm_warengruppe
409 WHERE artikel_id = v_artikel_id AND warengruppe_id = v_warengruppe_id
410   INTO v_count_zuordnung;
411
412 -- Wenn nein, füge eine hinzu
413 IF v_count_zuordnung <= 0 THEN
414   INSERT INTO artikel_nm_warengruppe
415     (warengruppe_id, artikel_id)
416   VALUES
417     (v_warengruppe_id, v_artikel_id);
418 ELSE
419   SELECT 'Zuordnung schon vorhanden';          -- Hinweis
420 END IF;
421
422 COMMIT;
423
424 END///
425 DELIMITER ;
426 -- Ende Prozedur insert_artikel Version 3
427
428 SELECT 'Cursor (Seite 356)';
429 -- Start import mit CURSOR
430 SOURCE importMitCursor01.sql;
431 SELECT bestellung_id, kunde_id, adresse_id, datum, status FROM tmp_bestellung;
432 SELECT bestellung_id, position_nr, artikel_id, menge, position_id FROM
433   tmp_position;
434 CALL importBestellung();
435 SELECT * FROM bestellung;
436 SELECT * from bestellung_position;
437 -- Ende import mit CURSOR
438
439 SELECT 'Ende listing16.sql /////////////////////////////////';

```

Listing 29.19 mysql/listing17.sql

```

1  -- Ausführen der vorherigen Befehle
2 SOURCE listing16.sql
3
4 SELECT 'Start listing17.sql /////////////////////////////////';
5
6 SELECT 'Funktionen (Seite 362)';
7 DROP FUNCTION IF EXISTS preis_anpassen;
8 DELIMITER //
9 CREATE FUNCTION preis_anpassen
10 (
11   iPreisOriginal DECIMAL(14,6),      -- Originalpreis
12   iProzentsatz  DECIMAL(7,3)        -- Anpassung in Prozent (0-100)
13 )
14 RETURNS DECIMAL(14,6) DETERMINISTIC
15 BEGIN
16   DECLARE vPreisNeu DECIMAL(14,6);    -- Neuer Preis
17
18   IF iPreisOriginal IS NULL THEN      -- Test auf NULL

```

```

19      RETURN NULL;
20  END IF;
21  IF iProzentsatz IS NULL THEN          -- Test auf NULL
22      RETURN NULL;
23  END IF;
24  IF iProzentsatz NOT BETWEEN 0 AND 100 THEN -- Wertebereich Prozent
25      RETURN NULL;
26  END IF;
27  SET vPreisNeu = iPreisOriginal + ((iPreisOriginal * iProzentsatz) / 100.0);
28  SET vPreisNeu = ROUND(vPreisNeu, 2);
29  RETURN vPreisNeu;
30 END //
31 DELIMITER ;
32
33 SELECT 'Ende listing17.sql /////////////////////////////////';

```

Listing 29.20 mysql/listing18.sql

```

1  -- Ausfuehren der vorherigen Befehle
2 SOURCE listing17.sql
3
4 SELECT 'Start listing18.sql /////////////////////////////////';
5
6 SELECT 'INSERT Trigger (Seite 365)';
7 DROP TRIGGER IF EXISTS tri_bestellung_after_insert;
8 DELIMITER //
9 CREATE TRIGGER tri_bestellung_after_insert -- ein schöner Name
10 AFTER INSERT                         -- wann?
11 ON bestellung                        -- bei wem?
12 FOR EACH ROW
13 BEGIN
14   INSERT INTO rechnung                -- was?
15   SET
16     kunde_id      = NEW.kunde_id,
17     bestellung_id = NEW.bestellung_id,
18     adresse_id    = NEW.adresse_id,
19     datum         = NEW.datum;
20 END//
21 DELIMITER ;
22 INSERT INTO bestellung (kunde_id, adresse_id, datum) VALUES (1, 1, NOW());
23 SELECT bestellung_id, kunde_id, adresse_id, datum FROM bestellung;
24 SELECT rechnung_id, kunde_id, bestellung_id, adresse_id, datum FROM rechnung\G
25
26 SELECT 'UPDATE Trigger (Seite 366)';
27 DROP TRIGGER IF EXISTS tri_bestellung_after_update;
28 DELIMITER //
29 CREATE TRIGGER tri_bestellung_after_update
30 AFTER UPDATE
31 ON bestellung
32 FOR EACH ROW
33 BEGIN
34   DECLARE vBestellungId INT;
35
36   SET vBestellungId = OLD.bestellung_id;    -- Damit wir es wiederfinden
37
38   IF OLD.kunde_id != NEW.kunde_id THEN      -- Nur, wenn nötig
39     UPDATE rechnung
40     SET kunde_id = NEW.kunde_id
41     WHERE bestellung_id = vBestellungId;

```

```
42    END IF;
43
44    IF OLD.adresse_id != NEW.adresse_id THEN
45        UPDATE rechnung
46            SET adresse_id = NEW.adresse_id
47            WHERE bestellung_id = vBestellungId;
48    END IF;
49
50    IF OLD.datum != NEW.datum THEN
51        UPDATE rechnung
52            SET datum = NEW.datum
53            WHERE bestellung_id = vBestellungId;
54    END IF;
55
56    END//  

57 DELIMITER ;
58 UPDATE bestellung SET adresse_id = 2 WHERE kunde_id = 1;
59 SELECT bestellung_id, kunde_id, adresse_id, datum
60     FROM bestellung
61 WHERE kunde_id=1;
62 SELECT rechnung_id, kunde_id, bestellung_id, adresse_id
63     FROM rechnung
64 WHERE kunde_id = 1;
65
66 SELECT 'DELETE Trigger (Seite 368)';
67 DROP TRIGGER IF EXISTS tri_bestellung_before_delete;
68 DELIMITER //  

69 CREATE TRIGGER tri_bestellung_before_delete
70     BEFORE DELETE
71     ON bestellung
72     FOR EACH ROW
73 BEGIN
74     DECLARE vBestellungId INT;
75     DECLARE vRechnungId INT;
76
77     SET vBestellungId = OLD.bestellung_id;    -- Damit wir es wiederfinden
78     SELECT rechnung_id
79         FROM rechnung
80     WHERE bestellung.id = vBestellungId
81     INTO vRechnungId;
82
83     DELETE FROM rechnung_position    WHERE rechnung_id    = vRechnungId;
84     DELETE FROM rechnung           WHERE rechnung_id    = vRechnungId;
85     DELETE FROM bestellung_position WHERE bestellung_id = vBestellungId;
86 END//  

87 DELIMITER ;
88 DELETE FROM bestellung WHERE bestellung_id = 5;
89 SELECT
90 (
91     SELECT COUNT(*) FROM bestellung WHERE bestellung_id = 5
92 ) AS AnzBestellung,
93 (
94     SELECT COUNT(*) FROM bestellung_position WHERE bestellung_id = 5
95 ) AS AnzPosBestellung,
96 (
97     SELECT COUNT(*) FROM rechnung WHERE bestellung_id = 5
98 ) AS AnzRechnung,
99 (
100    SELECT COUNT(*)
```

```
101    FROM
102      rechnung_position INNER JOIN rechnung USING (rechnung_id)
103      WHERE bestellung_id = 5
104    ) AS AnzPosRechnung
105  ;
106  SELECT 'Ende listing18.sql /////////////////////////////////';
```

Listing 29.21 mysql/listing19.sql

```
1  -- Ausfuehren der vorherigen Befehle
2  SOURCE listing18.sql
3
4  SELECT 'Start listing19.sql /////////////////////////////////';
5
6  SELECT 'EVENT anlegen (Seite 372)';
7
8  DROP EVENT IF EXISTS event_umsatz_woche;
9  CREATE EVENT event_umsatz_woche
10  ON SCHEDULE EVERY 1 WEEK
11  DO
12    CALL umsatzreport(1);
13
14 SHOW VARIABLES LIKE 'event%';
15 SET GLOBAL event_scheduler = ON;
16 ALTER EVENT event_umsatz_woche DISABLE;
17 DROP EVENT IF EXISTS event_umsatz_woche; -- Wegen Vorsicht!
18
19 DROP TABLE IF EXISTS a;
20 CREATE TABLE a
21 (
22   zeit DATETIME
23 );
24 DROP EVENT IF EXISTS event_zeit_sekunde;
25 CREATE EVENT event_zeit_sekunde
26   ON SCHEDULE EVERY 2 SECOND
27   DO
28     INSERT INTO a VALUES (NOW());
29
30 -- Warte 20 Sekunden
31 SELECT * FROM a;
32
33
34 SELECT 'EVENT loeschen ([label={lst19_006}]);'
35 DROP EVENT IF EXISTS event_zeit_sekunde;
36 SET GLOBAL event_scheduler = OFF;
37 DROP TABLE IF EXISTS a;
38 SELECT 'Ende listing19.sql /////////////////////////////////';
```

■ 29.2 PostgreSQL

29.2.1 Quelltexte zu Teil II

Listing 29.22 postgresql/listing01.sql

```
1 -- Warum formatiere ich den Quelltext so ganz anders, als beispielsweise diese
   -- Beautyfier?
2 -- * http://www.dpriver.com/pp/sqlformat.htm
3 -- * https://codebeautify.org/sqlformatter
4 -- Ich finde die Arbeit dieser Programme sehr gut und auch die Ergebnisse sind
   -- ansprechend,
5 -- aber sie widersprechen meinem Anspruch an die leichte Verständlichkeit der
   -- Befehle.
6 -- Zwei Beispiele:
7 -- * Es werden keine Einrückungen eingefügt, die den inhaltlichen
   -- Zusammenhang
8 -- zwischen dem Befehl und seinen Parametern abbilden:
9 -- SELECT blz,
10 --       bankname
11 -- FROM  bank
12 -- ORDER BY blz DESC
13 -- LIMIT 1;
14 --
15 -- anstelle von
16 --
17 -- SELECT
18 --   blz, bankname
19 --   FROM bank
20 --   ORDER BY blz DESC
21 --   LIMIT 1
22 -- ;
23
24 -- * Hinter dem FROM wird ein Tabellenname gestellt, obwohl die Auswertung aus
   -- dem ganzen JOIN erfolgt:
25 -- SELECT k.kunde_id,
26 --       k.nachname,
27 --       k.vorname,
28 --       b.datum
29 -- FROM bestellung b
30 --   INNER JOIN kunde k USING (kunde_id)
31 -- ORDER BY k.nachname,
32 --           k.vorname;
33
34 -- anstelle von
35
36 -- SELECT
37 --   k.kunde_id, k.nachname, k.vorname, b.datum
38 --   FROM
39 --     bestellung b INNER JOIN kunde k USING (kunde_id)
40 --   ORDER BY
41 --     k.nachname, k.vorname
42 -- ;
43
44 -- Ich werde daher meine Darstellung beibehalten.
45
46
```

```

47 -- Anlegen der Datenbank auf der grünen Wiese
48 \echo 'Start listing01.sql //////////////////////////////'
49 SELECT 'Loeschen einer ggf. vorhandenen Datenbank oshop (Seite 67)' Hinweis;
50 \connect postgres
51
52 DROP DATABASE IF EXISTS oshop;
53
54 -- In PostgreSQL gibt es keine Möglichkeit, die Zeichensätze und Sortierungen
      dynamisch zu ermitteln
55
56 SELECT 'Anlegen der Datenbank mit Zeichensatz und Sortierung (Seite 66)'
      Hinweis;
57 CREATE DATABASE oshop
58   ENCODING 'UTF8'
59   LC_COLLATE='en_US.utf8'
60   LC_CTYPE='en_US.utf8'
61 ;
62
63 SELECT 'Anzeigen aller Datenbanken (Seite 72)';
64 \l
65
66 \echo 'Ende listing01.sql //////////////////////////////'

```

Listing 29.23 postgresql/listing02.sql

```

1 -- Ausführen der vorherigen Befehle
2 SET client_encoding = 'UTF8';
3 SET standard_conforming_strings = on;
4 \i listing01.sql
5
6 \echo 'Start listing02.sql //////////////////////////////'
7
8 -- Erspare mir ein wenig Tipparbeit (Seite 75)
9 \connect oshop
10
11 -- adresse
12 SELECT 'Tabelle adresse anlegen (Seite 82)' Hinweis;
13 CREATE TABLE adresse (
14   adresse_id          SERIAL,
15   strasse              VARCHAR(255) NOT NULL DEFAULT '',
16   hnr                  VARCHAR(255) NOT NULL DEFAULT '',
17   lkz                  CHAR(2) NOT NULL DEFAULT '',
18   plz                  CHAR(9) NOT NULL DEFAULT '',
19   ort                  VARCHAR(255) NOT NULL DEFAULT '',
20   deleted              SMALLINT NOT NULL DEFAULT 0,
21   PRIMARY KEY(adresse_id)
22 );
23
24 -- kunde
25 SELECT 'Tabelle kunde mit Fremdschlüssel anlegen (Seite 87)' Hinweis;
26 CREATE TYPE kundenart AS ENUM('unb', 'prv', 'gsch');
27 CREATE TABLE kunde (
28   kunde_id              SERIAL,
29   nachname              VARCHAR(255) NOT NULL DEFAULT '',
30   vorname               VARCHAR(255) NOT NULL DEFAULT '',
31   rechnung_adresse_id  INT,
32   liefer_adresse_id    INT,
33   bezahlart              INT NOT NULL DEFAULT 0,
34   art                   kundenart NOT NULL DEFAULT 'unb',

```

```

35      deleted          SMALLINT NOT NULL DEFAULT 0,
36      PRIMARY KEY(kunde_id),
37      FOREIGN KEY (rechnung_adresse_id)
38          REFERENCES adresse(adresse_id)
39          ON UPDATE CASCADE
40          ON DELETE RESTRICT,
41      FOREIGN KEY (liefer_adresse_id)
42          REFERENCES adresse(adresse_id)
43          ON UPDATE CASCADE
44          ON DELETE SET NULL
45  );
46
47 -- bank
48 SELECT 'Tabelle bank anlegen (Seite 76)' Hinweis;
49 CREATE TABLE bank (
50     bank_id           CHAR(12),
51     bankname          VARCHAR(255) NOT NULL DEFAULT '',
52     lkz               CHAR(2) NOT NULL DEFAULT 'DE',
53     deleted           SMALLINT NOT NULL DEFAULT 0,
54     PRIMARY KEY(bank_id)
55  );
56
57 -- bankverbindung
58 SELECT 'Tabelle bankverbindung mit Fremdschlüssel anlegen (Seite 76)' Hinweis;
59 CREATE TABLE bankverbindung (
60     kunde_id          INT,
61     bankverbindung_nr INT,
62     bank_id            CHAR(12) NOT NULL,
63     kontonummer        CHAR(25) NOT NULL DEFAULT '',
64     iban              CHAR(34) NOT NULL DEFAULT '',
65     deleted           SMALLINT NOT NULL DEFAULT 0,
66     PRIMARY KEY(kunde_id,bankverbindung_nr),
67     FOREIGN KEY (kunde_id)
68         REFERENCES kunde(kunde_id)
69         ON UPDATE RESTRICT
70         ON DELETE RESTRICT,
71     FOREIGN KEY (bank_id)
72         REFERENCES bank(bank_id)
73         ON UPDATE RESTRICT
74         ON DELETE RESTRICT
75  );
76
77
78 -- artikel
79 SELECT 'Tabelle artikel anlegen' Hinweis;
80 CREATE TABLE artikel (
81     artikel_id         SERIAL,
82     bezeichnung        VARCHAR(255) NOT NULL DEFAULT '',
83     einzelpreis        DECIMAL(14,6) NOT NULL DEFAULT 0.0,
84     waehrung          CHAR(3) DEFAULT 'EUR',
85     deleted           SMALLINT NOT NULL DEFAULT 0,
86     PRIMARY KEY(artikel_id)
87  );
88
89
90 -- warengruppe
91 SELECT 'Tabelle warengruppe anlegen' Hinweis;
92 CREATE TABLE warengruppe (
93     warengruppe_id    SERIAL,

```

```
94  bezeichnung          VARCHAR(255) NOT NULL DEFAULT '',
95  deleted              SMALLINT NOT NULL DEFAULT 0,
96  PRIMARY KEY(warengruppe_id)
97 );
98
99
100 -- lieferant
101 SELECT 'Tabelle lieferant mit Fremdschlüssel anlegen' Hinweis;
102 CREATE TABLE lieferant (
103  lieferant_id        SERIAL,
104  firmenname          VARCHAR(255) NOT NULL DEFAULT '',
105  adresse_id          INT NOT NULL DEFAULT 0,
106  deleted              SMALLINT NOT NULL DEFAULT 0,
107  PRIMARY KEY(lieferant_id),
108  FOREIGN KEY (adresse_id)
109    REFERENCES adresse(adresse_id)
110    ON UPDATE RESTRICT
111    ON DELETE RESTRICT
112 );
113
114
115 -- Hilfstabelle artikel zu warengruppe
116 SELECT 'Hilfstabelle artikel_nm_warengruppe anlegen' Hinweis;
117 CREATE TABLE artikel_nm_warengruppe (
118  warengruppe_id      INT,
119  artikel_id          INT,
120  PRIMARY KEY(warengruppe_id, artikel_id),
121  FOREIGN KEY (warengruppe_id)
122    REFERENCES warengruppe(warengruppe_id)
123    ON UPDATE RESTRICT
124    ON DELETE RESTRICT,
125  FOREIGN KEY (artikel_id)
126    REFERENCES artikel(artikel_id)
127    ON UPDATE RESTRICT
128    ON DELETE RESTRICT
129 );
130
131
132 -- Hilfstabelle artikel zu lieferant
133 SELECT 'Hilfstabelle artikel_nm_lieferant anlegen' Hinweis;
134 CREATE TABLE artikel_nm_lieferant (
135  lieferant_id         INT,
136  artikel_id           INT,
137  PRIMARY KEY(lieferant_id, artikel_id),
138  FOREIGN KEY (lieferant_id)
139    REFERENCES lieferant(lieferant_id)
140    ON UPDATE RESTRICT
141    ON DELETE RESTRICT,
142  FOREIGN KEY (artikel_id)
143    REFERENCES artikel(artikel_id)
144    ON UPDATE RESTRICT
145    ON DELETE RESTRICT
146 );
147
148
149 -- bestellung
150 SELECT 'Tabelle bestellung mit Fremdschlüssel anlegen ' Hinweis;
151 CREATE TYPE bestellstatus AS ENUM('offen', 'versendet', 'angekommen', 'retour',
152   , 'bezahlt');
```

```

152 CREATE TABLE bestellung (
153     bestellung_id      SERIAL,
154     kunde_id          INT NOT NULL DEFAULT 0,
155     adresse_id         INT NOT NULL DEFAULT 0,
156     datum              TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
157     status              bestellstatus NOT NULL DEFAULT 'offen',
158     deleted             SMALLINT NOT NULL DEFAULT 0,
159     PRIMARY KEY(bestellung_id),
160     FOREIGN KEY (kunde_id)
161       REFERENCES kunde(kunde_id)
162       ON UPDATE RESTRICT
163       ON DELETE RESTRICT,
164     FOREIGN KEY (adresse_id)
165       REFERENCES adresse(adresse_id)
166       ON UPDATE RESTRICT
167       ON DELETE RESTRICT
168 );
169
170 -- position der bestellung
171 SELECT 'Tabelle bestellung_position mit Fremdschlüssel anlegen' Hinweis;
172 CREATE TABLE bestellung_position (
173     bestellung_id      INT,
174     position_nr        INT,
175     artikel_id         INT NOT NULL DEFAULT 0,
176     menge              DECIMAL(14,6) NOT NULL DEFAULT 0.0,
177     deleted             SMALLINT NOT NULL DEFAULT 0,
178     PRIMARY KEY(bestellung_id, position_nr),
179     FOREIGN KEY (bestellung_id)
180       REFERENCES bestellung(bestellung_id)
181       ON UPDATE RESTRICT
182       ON DELETE RESTRICT,
183     FOREIGN KEY (artikel_id)
184       REFERENCES artikel(artikel_id)
185       ON UPDATE RESTRICT
186       ON DELETE RESTRICT
187 );
188
189
190 -- rechnung
191 SELECT 'Tabelle rechnung mit Fremdschlüssel anlegen' Hinweis;
192 CREATE TYPE bezahlarten AS ENUM('lastschrift', 'rechnung', 'kredit', 'bar');
193 CREATE TYPE rechnungstatus AS ENUM('offen', 'gestellt', 'mahnung1', 'inkasso',
194                                         'storno', 'beglichen');
195 CREATE TABLE rechnung (
196     rechnung_id        SERIAL,
197     kunde_id          INT NOT NULL DEFAULT 0,
198     bestellung_id     INT NOT NULL DEFAULT 0,
199     adresse_id         INT NOT NULL DEFAULT 0,
200     bezahlart          bezahlarten NOT NULL DEFAULT 'lastschrift',
201     datum              TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
202     status              rechnungstatus NOT NULL DEFAULT 'offen',
203     rabatt             DECIMAL(6,2) NOT NULL DEFAULT 0.0,
204     skonto             DECIMAL(6,2) NOT NULL DEFAULT 0.0,
205     deleted             SMALLINT NOT NULL DEFAULT 0,
206     PRIMARY KEY(rechnung_id),
207     FOREIGN KEY (kunde_id)
208       REFERENCES kunde(kunde_id)
209       ON UPDATE RESTRICT
210       ON DELETE RESTRICT,

```

```

210 FOREIGN KEY (adresse_id)
211 REFERENCES adresse(adresse_id)
212 ON UPDATE RESTRICT
213 ON DELETE RESTRICT
214 );
215
216
217 -- position der rechnung
218 SELECT 'Tabelle rechnung_position mit Fremdschlüssel anlegen' Hinweis;
219 CREATE TABLE rechnung_position (
220 rechnung_id      INT,
221 position_nr     INT,
222 artikel_id      INT NOT NULL DEFAULT 0,
223 steuer          DECIMAL(14,6) NOT NULL DEFAULT 19.0,
224 menge           DECIMAL(14,6) NOT NULL DEFAULT 0.0,
225 deleted         SMALLINT NOT NULL DEFAULT 0,
226 PRIMARY KEY(rechnung_id, position_nr),
227 FOREIGN KEY (rechnung_id)
228 REFERENCES rechnung(rechnung_id)
229 ON UPDATE RESTRICT
230 ON DELETE RESTRICT,
231 FOREIGN KEY (artikel_id)
232 REFERENCES artikel(artikel_id)
233 ON UPDATE RESTRICT
234 ON DELETE RESTRICT
235 );
236
237 SELECT 'Neue Tabelle mit Struktur einer alten anlegen (Seite 89)' Hinweis;
238 CREATE TABLE wurstbrot (LIKE adresse);
239
240 SELECT 'Neue Tabelle sofort wieder löschen, da nicht mehr gebraucht' Hinweis;
241 DROP TABLE wurstbrot;
242
243 -- Zeige mir das Ergebnis
244 \dt
245 \echo 'Ende listing02.sql /////////////////////////////////'
```

Listing 29.24 postgresql/listing03.sql

```

1 -- Ausführen der vorherigen Befehle
2 \i listing02.sql
3
4 \echo 'Start listing03.sql /////////////////////////////////'
5
6 -- Seite 95
7 \d kunde
8
9 -- Anlegen der Indizes nach Liste auf Seite 97
10 SELECT 'CREATE INDEX idx_kunde_nachname_vorname' Hinweis;
11 CREATE INDEX idx_kunde_nachname_vorname
12   ON kunde (nachname, vorname)
13 ;
14
15 SELECT 'CREATE INDEX idx_bank_bankname' Hinweis;
16 CREATE INDEX idx_bank_bankname
17   ON bank (bankname)
18 ;
19
20 SELECT 'CREATE INDEX idx_bankverbindung_bankid_kontonummer' Hinweis;
```

```
21 CREATE INDEX idx_bankverbindung_bankid_kontonummer  
22   ON bankverbindung (bank_id, kontonummer)  
23 ;  
24  
25 SELECT 'CREATE UNIQUE INDEX idx_bankverbindung_iban' Hinweis;  
26 CREATE UNIQUE INDEX idx_bankverbindung_iban  
27   ON bankverbindung (iban)  
28 ;  
29  
30 SELECT 'CREATE INDEX idx_artikel_bezeichnung' Hinweis;  
31 CREATE INDEX idx_artikel_bezeichnung  
32   ON artikel (bezeichnung)  
33 ;  
34  
35 SELECT 'CREATE INDEX idx_warengruppe_bezeichnung' Hinweis;  
36 CREATE INDEX idx_warengruppe_bezeichnung  
37   ON warengruppe (bezeichnung)  
38 ;  
39  
40 SELECT 'CREATE INDEX idx_lieferant_firmenname' Hinweis;  
41 CREATE INDEX idx_lieferant_firmenname  
42   ON lieferant (firmenname)  
43 ;  
44  
45 SELECT 'CREATE INDEX idx_bestellung_kundeid_datum' Hinweis;  
46 CREATE INDEX idx_bestellung_kundeid_datum  
47   ON bestellung (kunde_id, datum)  
48 ;  
49  
50  
51 SELECT 'CREATE UNIQUE INDEX idx_adresse_dublette (Seite 100)' Hinweis;  
52 CREATE UNIQUE INDEX idx_adresse_dublette  
53   ON adresse (substring(strasse from 1 for 100), substring(hnr from 1 for 100)  
      , plz);  
54  
55 -- Start Selektivität  
56 SELECT 'Indexselektivität (Seite 103)' Hinweis;  
57 SELECT 'CREATE TEMPORARY TABLE testdaten_bank' Hinweis;  
58 DROP TABLE IF EXISTS testdaten_bank;  
59 CREATE TABLE testdaten_bank  
60 (  
61   blz      CHAR(8),  
62   merkmal  CHAR(1),  
63   bezeichnung VARCHAR(255),  
64   plz      CHAR(5),  
65   ort      VARCHAR(255),  
66   kurz     VARCHAR(255),  
67   pan      CHAR(5),  
68   bic      VARCHAR(255),  
69   sm       CHAR(2),  
70   ds       CHAR(6),  
71   kz       CHAR(1),  
72   blzl    CHAR(1),  
73   blzn    CHAR(8)  
74 );  
75  
76 SELECT 'COPY' Hinweis;  
77   -- Hier Dateiname und Pfad einfuegen  
78
```

```

79 \copy testdaten_bank FROM './blz_20120305.csv' WITH DELIMITER ',' CSV HEADER
80 SELECT 'CREATE INDEX' Hinweis;
81 CREATE INDEX idx_testdaten_bank_bezeichnung
82   ON testdaten_bank (substring(bezeichnung from 1 for 30))
83 ;
84 -- Ende Selektivitaet
85
86 \echo 'Ende listing03.sql /////////////////////////////////';

```

Listing 29.25 postgresql/listing04.sql

```

1  -- Ausfren der vorherigen Befehle
2 \i listing03.sql
3
4 \echo 'Start listing04.sql /////////////////////////////////'
5
6 SELECT 'Importieren von artikel01.csv (Seite 111)' Hinweis;
7 \copy artikel(artikel_id, bezeichnung, einzelpreis, waehrung) FROM './
     artikel02.csv' WITH DELIMITER ',' CSV HEADER
8
9 SELECT * FROM artikel;
10
11 -- Start INSERT
12 SELECT 'Anlegen mit INSERT INTO ... VALUES (Seite 113)' Hinweis;
13 SELECT 'Warengruppen anlegen' Hinweis;
14 INSERT INTO warengruppe (warengruppe_id, bezeichnung)
15   VALUES
16     (1, 'Brobedarf')
17     ,(2, 'Pflanzen')
18     ,(3, 'Gartenbedarf')
19     ,(4, 'Werkzeug')
20 ;
21 SELECT * FROM warengruppe;
22
23 SELECT 'Anlegen mit INSERT INTO ... SET (Seite 115) geht in PostgreSQL nicht'
     Hinweis;
24 SELECT 'Warengruppen verbinden' Hinweis;
25 INSERT INTO artikel_mm_warengruppe (artikel_id, warengruppe_id)
26   VALUES
27     (3001,1),(3005,1),(3006,1),(3007,1),(3010,1), (7856,2),(7856,3),(7863,2)
     ,(7863,3),(9010,3),(9010,4),(9015,3),(9015,4)
28 ;
29 -- Ende INSERT
30
31 -- Start Kundendaten
32 SELECT 'Testdaten INSERT adresse' Hinweis;
33
34 -- adresse
35 INSERT INTO adresse (adresse_id, strasse, hnr, lkz, plz, ort)
36   VALUES
37     (1, 'Beutelhaldenweg', '5', 'AL', '67676', 'Hobbingen')
38     ,(2, 'Beutelhaldenweg', '1', 'AL', '67676', 'Hobbingen')
39     ,(3, 'Auf der Feste', '1', 'GO', '54786', 'Minas Tirith')
40     ,(4, 'Letztes Haus', '4', 'ER', '87567', 'Bruchtal')
41     ,(5, 'Baradur', '1', 'MO', '62519', 'Lugburz')
42     ,(10, 'Hochstrasse', '4a', 'DE', '44879', 'Bochum')
43     ,(11, 'Industriegebiet','8', 'DE', '44878', 'Bochum')
44 ;
45

```

```

46 SELECT 'Testdaten INSERT kunde' Hinweis;
47 -- kunde
48 INSERT INTO kunde (kunde_id, rechnung_adresse_id, nachname, vorname, art)
49 VALUES
50     (1, 1, 'Gamdschie',    'Samweis', 'prv')
51     ,(2, 2, 'Beutlin',      'Frodo',   'prv')
52     ,(3, 2, 'Beutlin',      'Bilbo',   'prv')
53     ,(4, 3, 'Telcontar',   'Elessar',  'prv')
54     ,(5, 4, 'Earendilionn','Elrond',  'gsch')
55 ;
56
57 SELECT 'Lieferadresse bei zwei Adressen ändern' Hinweis;
58 UPDATE kunde SET liefer_adresse_id = 2 WHERE kunde_id = 1;
59 UPDATE kunde SET liefer_adresse_id = 4 WHERE kunde_id = 3;
60 -- Ende Kundendaten
61
62 -- Start Aufbau von 2 Bestellungen
63 SELECT 'Aufbau von 2 Bestellungen: Bestelldaten' Hinweis;
64 INSERT INTO bestellung (kunde_id, adresse_id, datum)
65 VALUES
66     (1, 1, '2012-03-24 17:41:00')
67     ,(2, 2, '2012-03-23 16:11:00')
68 ;
69
70 SELECT 'Aufbau von 2 Bestellungen: Bestellpositionen' Hinweis;
71 INSERT INTO bestellung_position (bestellung_id, position_nr, artikel_id, menge
72 )
72 VALUES
73     (1, 1, 7856, 30)
74     ,(1, 2, 7863, 50)
75     ,(1, 3, 9015, 1)
76     ,(2, 1, 7856, 10)
77     ,(2, 2, 9010, 5)
78 ;
79 -- Ende Aufbau Bestellung
80
81 -- SELECT 'Drei Fehlermeldung wegen Constraints (Seite 116)!';
82 -- SELECT bestellung_id, kunde_id, adresse_id, datum FROM bestellung;
83 -- INSERT INTO bestellung (datum) VALUES (NOW());
84 -- INSERT INTO bestellung (adresse_id, datum) VALUES (10, NOW());
85 -- INSERT INTO adresse (adresse_id, strasse) VALUES (10, 'Oberer Weg');
86
87 -- Bildtabelle
88 SELECT 'Bildtabelle anlegen (Seite 117)' Hinweis;
89 CREATE TABLE bild
90 (
91     bild_id          SERIAL PRIMARY KEY,
92     bild             BYTEA,
93     dateiname        VARCHAR(255),
94     dateityp         VARCHAR(255),
95     dateigroesse    BIGINT,
96     artikel_id       INT REFERENCES artikel(artikel_id)
97 );
98
99 SELECT 'Daten kopieren (Seite 121)' Hinweis;
100 DROP TABLE IF EXISTS kunde_beutlin;
101 CREATE TEMPORARY TABLE kunde_beutlin (LIKE kunde);
102 INSERT INTO kunde_beutlin
103     SELECT * FROM kunde WHERE nachname='Beutlin';

```

```

104 SELECT * FROM kunde_beutlin;
105
106
107 -- SELECT DROP (auskommentiert)
108 -- SELECT * FROM warengruppe INTO OUTFILE 'bla.txt';
109 -- DROP DATABASE oshop;
110 -- SELECT 'Löschen Sie nun MySQL-Datenverzeichnis oshop/bla.txt!'
111 -- Ende DROP
112
113 \echo 'Ende listing04.sql /////////////////////////////////'

```

29.2.2 Quelltexte zu Teil III

Listing 29.26 postgresql/listing05.sql

```

1 -- Ausführen der vorherigen Befehle
2 \i listing04.sql
3
4 \echo 'Start listing05.sql /////////////////////////////////'
5
6 SELECT 'Aktuelle Schemaeinstellung (Seite 126)' Hinweis;
7 \l+
8
9 SELECT 'DATABASE loeschen (Seite 127)' Hinweis;
10 CREATE DATABASE wurstbrot;
11 \list+
12 DROP DATABASE wurstbrot;
13 \list+
14
15
16 -- Start Tabellen umbenennen
17 SELECT 'Tabellen umbenennen (Seite 130)' Hinweis;
18 DROP TABLE IF EXISTS wurstbrot;
19 \dt
20 ALTER TABLE adresse RENAME TO wurstbrot;
21 \dt
22 ALTER TABLE wurstbrot RENAME TO adresse;
23 -- Ende Tabellen umbenennen
24
25
26
27 -- Start Spalten verändern
28 SELECT 'Spalte hinzufügen (Seite 131)' Hinweis;
29 \d+ kunde
30 ALTER TABLE kunde
31     ADD firmename VARCHAR(255) NOT NULL DEFAULT ''
32 ;
33 \d+ kunde
34
35 SELECT 'Spalte Datentyp aendern (Seite 133)' Hinweis;
36 -- Konvertierung geht in PostgreSQL nur unter Zuhilfenahme einer Funktion.
37 -- Aus diesem Grunde verzichte ich hier auf die Darstellung der entsprechenden
38 -- Konvertierungseffekte, da in den selbsterstellten Funktionen alle
39 -- Fehlersituationen
40 -- abgefangen werden können.
41 -- Hier ein nicht erklärttes, aber hoffentlich selbsterklärendes Beispiel.

```

```

41 CREATE OR REPLACE FUNCTION IntToBezahlarten(wert INT)
42     RETURNS bezahlarten AS
43 $BODY$
44     SELECT CASE wert
45         WHEN 1 THEN 'lastschrift'::bezahlarten
46         WHEN 2 THEN 'rechnung'::bezahlarten
47         WHEN 3 THEN 'kredit'::bezahlarten
48         WHEN 4 THEN 'bar'::bezahlarten
49         ELSE 'rechnung'::bezahlarten
50     END;
51 $BODY$
52 IMMUTABLE STRICT LANGUAGE SQL
53 ;
54 ALTER TABLE kunde
55     ALTER COLUMN bezahlart
56     DROP DEFAULT
57 ;
58 ALTER TABLE kunde
59     ALTER COLUMN bezahlart
60     SET DATA TYPE bezahlarten USING IntToBezahlarten(bezahlart)
61 ;
62 ALTER TABLE kunde
63     ALTER COLUMN bezahlart
64     SET NOT NULL
65 ;
66 ALTER TABLE kunde
67     ALTER COLUMN bezahlart
68     SET DEFAULT 'rechnung'::bezahlarten
69 ;
70 ;
71 DROP FUNCTION IF EXISTS IntToBezahlarten(int);
72 \d+ kunde
73
74 SELECT 'Spaltenlänge ändern (Seite 133)' Hinweis;
75 DROP TABLE IF EXISTS bla ;
76 CREATE TEMPORARY TABLE bla (LIKE artikel);
77 INSERT INTO bla SELECT * FROM artikel;
78 -- Hinweis: siehe oben.
79 CREATE OR REPLACE FUNCTION Cut10(wert VARCHAR(255))
80     RETURNS CHAR(10) AS
81 $BODY$
82     SELECT LEFT(wert, 10);
83 $BODY$
84 IMMUTABLE STRICT LANGUAGE SQL
85 ;
86 ALTER TABLE bla ALTER COLUMN bezeichnung SET DATA TYPE CHAR(10) USING Cut10(
87     bezeichnung);
88 SELECT * FROM bla;
89 -- SHOW WARNINGS;
90 SELECT MAX(CHAR_LENGTH(bezeichnung)) FROM artikel;
91
92 SELECT 'Zeichen nach Zahl ändern (Seite 134)' Hinweis;
93 DROP TABLE IF EXISTS wurstbrot ;
94 CREATE TEMPORARY TABLE wurstbrot (LIKE adresse);
95 INSERT INTO wurstbrot SELECT * FROM adresse;
96 SELECT * FROM wurstbrot;
97 ALTER TABLE wurstbrot ALTER COLUMN plz SET DATA TYPE INT USING plz::int;
98 \d+ wurstbrot
99 SELECT * FROM wurstbrot;

```

```

99
100 ALTER TABLE wurstbrot DROP COLUMN plz;
101
102 -- Ende Spalten verändern
103
104 -- Start Tabellen löschen
105 -- DROP TABLE
106 --   rechnung_position
107 --   ,rechnung
108 --   ,bestellung_position
109 --   ,bestellung
110 -- ;
111 \dt
112 \echo 'Ende listing05.sql /////////////////////////////////'
```

Listing 29.27 postgresql/listing06.sql

```

1 -- Ausführen der vorherigen Befehle
2 \i listing05.sql
3
4 \echo 'Start listing06.sql /////////////////////////////////'
5
6 SELECT 'WHERE mit TRUE / FALSE (Seite 142)' Hinweis;
7 SELECT * FROM artikel WHERE TRUE;
8 SELECT * FROM artikel WHERE FALSE;
9
10
11 -- Start Groß-/Kleinschreibung
12 SELECT 'Groß/Kleinschreibung (Seite 143)' Hinweis;
13 DROP TABLE IF EXISTS wurstbrot;
14 CREATE TEMPORARY TABLE wurstbrot (
15   name VARCHAR(255)
16 );
17
18 INSERT INTO wurstbrot
19   VALUES
20     ('Jaqueline'),
21     ('Kevin'),
22     ('kevin'),
23     ('jaqueline')
24 ;
25 SELECT * FROM wurstbrot WHERE LOWER(name) = 'kevin';
26
27 SELECT * FROM wurstbrot WHERE name = 'kevin';
28 -- Ende Groß- Kleinschreibung
29
30 -- Start Einfache Wertzuweisung
31 SELECT 'Einfache Wertzuweisung (Seite 148)' Hinweis;
32 SELECT * FROM bestellung_position;
33 UPDATE bestellung_position
34   SET menge = 2
35   WHERE bestellung_id = 1 AND position_nr = 3
36 ;
37 SELECT * FROM bestellung_position;
38 -- Ende Einfache Wertzuweisung
39
40 -- Start Wertberechnung
41 SELECT 'Wertberechnung (Seite 148)' Hinweis;
42 SELECT * FROM artikel;
```

```

43 UPDATE artikel SET einzelpreis = einzelpreis + einzelpreis / 100.0;
44 SELECT * FROM artikel;
45 UPDATE artikel SET einzelpreis = ROUND(einzelpreis, 2);
46 SELECT * FROM artikel;
47 -- Ende Wertberechnung
48
49 -- Start Gebastelte Zeichenketten
50 SELECT 'Gebastelte Zeichenketten: Anrede (Seite 149)' Hinweis;
51 ALTER TABLE kunde ADD anrede VARCHAR(255);
52 UPDATE kunde
53 SET anrede = 'Sehr geehrte/r Frau/Herr ' || nachname
54 WHERE nachname <> ''
55 ;
56 SELECT * FROM kunde;
57 -- Ende Gebastelte Zeichenketten
58
59
60 -- Start DELETE
61 SELECT 'Löschen mit DELETE (Seite 150)' Hinweis;
62 SELECT * FROM bestellung_position;
63 DELETE FROM bestellung_position WHERE bestellung_id = 1;
64 SELECT * FROM bestellung_position;
65
66 -- Start Wiederherstellen der Daten
67 INSERT INTO bestellung_position (bestellung_id, position_nr, artikel_id, menge
68 )
69 VALUES
70     (1, 1, 7856, 30)
71 , (1, 2, 7863, 50)
72 , (1, 3, 9015, 1)
73 ;
74 -- Ende Wiederherstellen der Daten
75
76 -- SELECT 'Löschversuch mit Constraint. Erwarte eine Fehlermeldung! (Seite 152)';
77 -- DELETE FROM kunde WHERE kunde_id = 1;
78
79 SELECT 'Test AUTO_INCREMENT (Seite 152)' Hinweis;
80 SELECT kunde_id, nachname FROM kunde;
81 DELETE FROM kunde WHERE kunde_id IN (3,5);
82 -- Liefert in PostgreSQL eine Fehlermeldung, PK-Wert ermittelt werden muss
83 INSERT INTO kunde (kunde_id, nachname, vorname) VALUES (6, 'Eichenschild', 'Thorin');
84 SELECT kunde_id, nachname FROM kunde;
85 -- Start Wiederherstellen der Daten
86 INSERT INTO kunde (kunde_id, rechnung_adresse_id, nachname, vorname, art)
87 VALUES
88     (3, 2, 'Beutlin', 'Bilbo', 'prv')
89 , (5, 4, 'Earendillionn', 'Elrond', 'gsch')
90 ;
91 -- Ende Wiederherstellen der Daten
92
93 SELECT 'Tabelle leeren (Seite 154)' Hinweis;
94 -- DELETE FROM bestellung_position;
95 -- TRUNCATE bestellung_position;
96 -- Ende DELETE
97 \echo 'Ende listing06.sql /////////////////////////////////'

```

29.2.3 Quelltexte zu Teil IV

Listing 29.28 postgresql/listing07.sql

```
1 -- Ausführen der vorherigen Befehle
2 \i listing06.sql
3
4 \echo 'Start listing07.sql /////////////////////////////////'
5
6 SELECT 'Ausdrücke (Seite 158)' Hinweis;
7
8 SELECT 5, 'Beginn der Auswertung';
9 SELECT -5, 9 + 4, 9 - 4, 9 * 4, 9.0 / 4.0, DIV(9,4), 9 % 4, MOD(9,4);
10 SELECT 3 * 4 + 3, (3 * 4) + 3, 3 * (4 + 3); -- Beispiel auf PostgreSQL
    angepasst, da DIV dort ein Funktionsaufruf und kein Operator ist.
11 -- SELECT 9 / 0, 9 * NULL, SQRT(-5) ; -- Erzeugt in PostgreSQL eine
    Fehlermeldung.
12
13 SELECT 'Zufallszahlen (Seite 160)' Hinweis;
14 SELECT SETSEED(0.5);
15 SELECT RANDOM();
16 SELECT SETSEED(0.5);
17 SELECT RANDOM();
18 SELECT SETSEED(0.5);
19 SELECT RANDOM() FROM artikel;
20
21 SELECT 'Variablen (Seite 161)';
22 DO $$ -- Zum Deklarieren einer Variablen muss man in einem Anweisungsblock
    sein. Daher hier der anonyme Anweisungsblock.
23 DECLARE x double precision := 0;
24 BEGIN
25     SELECT POWER(2,8) INTO x;
26     PERFORM x - 1; -- Ergebnis nicht in der Konsole sichtbar.
27 END $$;
28
29 -- SELECT 'Ungültiger Versuch: erwarte eine Fehlermeldung (Seite 161)'
30 -- DO $$
31 -- DECLARE artnr INT := 3010;
32 -- BEGIN
33 --     PERFORM * FROM artikel WHERE artikel_id = artnr; -- Ergebnis nicht in der
    Konsole sichtbar.
34 -- END $$;
35
36
37 SELECT 'Spalten- und Zeilenwahl (Seite 162)' Hinweis;
38 SELECT * FROM artikel;
39 SELECT
40     artikel_id, bezeichnung, einzelpreis, waehrung, deleted
41     FROM artikel
42 ;
43
44 SELECT
45     artikel_id + 10000 AS Unsinn, deleted AS Löschkennzeichen
46     FROM artikel
47 ;
48
49 SELECT
50     artikel_id, bezeichnung
```

```
51  FROM artikel
52  WHERE einzelpreis BETWEEN 10 AND 100
53 ;
54
55 SELECT 'Sortierung (Seite 163)' Hinweis;
56 SELECT
57   artikel_id, bezeichnung, einzelpreis
58  FROM artikel
59 ORDER BY einzelpreis ASC
60 ;
61
62 SELECT
63   nachname, vorname
64  FROM kunde
65 ORDER BY nachname, vorname
66 ;
67
68 SELECT
69   nachname, vorname
70  FROM kunde
71 ORDER BY nachname DESC, vorname ASC
72 ;
73
74 -- CP850 wird von PostgreSQL nicht unterstützt. Beispiel ist daher nicht
    verwendbar.
75 -- DROP TABLE IF EXISTS wurstbrot;
76 -- SET NAMES 'cp850';
77 -- CREATE TEMPORARY TABLE wurstbrot ( name VARCHAR(255) CHARACTER SET 'cp850' )
    ;
78 -- INSERT INTO wurstbrot
79 -- VALUES
80 --   ('winfried')
81 --   ,('achim')
82 --   ,('olga')
83 --   ,('zechine')
84 --   ,('ägidius')
85 -- ;
86 -- SELECT * FROM wurstbrot ORDER BY name;
87 --
88 -- DROP TABLE IF EXISTS wurstbrot;
89 -- SET NAMES 'utf8';
90 -- CREATE TEMPORARY TABLE wurstbrot ( name VARCHAR(255) CHARACTER SET 'utf8' );
91 -- INSERT INTO wurstbrot
92 -- VALUES
93 --   ('winfried')
94 --   ,('achim')
95 --   ,('olga')
96 --   ,('zechine')
97 --   ,('ägidius')
98 -- ;
99 -- SELECT * FROM wurstbrot ORDER BY name;
100 --
101 DROP TABLE IF EXISTS wurstbrot;
102 CREATE TEMPORARY TABLE wurstbrot (name VARCHAR(255));
103 INSERT INTO wurstbrot
104 VALUES
105   ('winfried')
106   ,('achim')
107   ,('olga')
```

```
108      ,('Olga')
109      ,('zechine')
110 ;
111 SELECT * FROM wurstbrot ORDER BY name;
112 SELECT * FROM wurstbrot ORDER BY name DESC;
113
114 -- In PostgreSQL wird immer zwischen Groß- und Kleinschreibung unterschieden.
115 -- DROP TABLE IF EXISTS wurstbrot;
116 -- CREATE TEMPORARY TABLE wurstbrot (name VARCHAR(255) BINARY);
117 -- INSERT INTO wurstbrot
118 --   VALUES
119 --     ('winfried')
120 --     ,('achim')
121 --     ,('olga')
122 --     ,('Olga')
123 --     ,('zechine')
124 -- ;
125 -- SELECT * FROM wurstbrot ORDER BY name;
126 -- SELECT * FROM wurstbrot ORDER BY name DESC;
127
128 EXPLAIN SELECT * FROM kunde ORDER BY nachname, vorname;
129
130 SELECT
131   bestellung_id, datum
132   FROM bestellung
133   ORDER BY datum;
134
135 EXPLAIN SELECT * FROM kunde ORDER BY nachname, vorname;
136
137 -- In PostgreSQL gibt es keinen Datentyp TIME. Geht nur über Hilfsfunktionen
138 -- in Text umzuwandeln.
139 -- SELECT
140 --   bestellung_id, TIME(datum) Uhrzeit
141 --   FROM bestellung
142 --   ORDER BY uhrzeit;
143
144 SELECT 'Mehrfachausgaben unterbinden (Seite 173)' Hinweis;
145
146 SELECT ort FROM adresse ORDER BY ort;
147 SELECT DISTINCT ort FROM adresse ORDER BY ort;
148 SELECT DISTINCT nachname FROM kunde ORDER BY nachname;
149
150 -- Das Fallbeispiel Bankimport müsste in PostgreSQL ganz anders als
151 -- in MySQL oder MariaDB gelöst werden.
152 -- Daher hier nur der INSERT, damit die Bankdaten vorliegen.
153 \d+ bank
154 ALTER TABLE bank ADD blz CHAR(12) NOT NULL DEFAULT '';
155 \i blz_20120305.sql
156
157 CREATE INDEX idx_bank_blzbankname
158   ON bank (blz, bankname)
159 ;
160
161 -- Beginn Für die vorhandenen Kunden eine Bankverbindung bauen
162 INSERT INTO bankverbindung (kunde_id, bankverbindung_nr, bank_id, kontonummer,
163                             iban)
164   VALUES
165   (1, 1, 2, '1111111111', '100100101111111111'),
```

```

165 (1, 2, 2, '1111111112', '100100101111111112'),
166 (2, 1, 35, '2222222221', '100601982222222221'),
167 (3, 1, 35, '3333333331', '100601983333333331'),
168 (4, 1, 90, '4444444441', '120700004444444441'),
169 (5, 1, 90, '5555555551', '120700005555555551')
170 ;
171 -- Ende Für die vorhandenen Kunden eine Bankverbindung bauen
172
173 -- Ende Bankimport
174
175 SELECT 'Ausschneiden mit LIMIT (Seite 177)' Hinweis;
176
177 SELECT blz, bankname FROM bank ORDER BY blz LIMIT 3;
178
179 SELECT
180   blz, bankname
181   FROM bank
182   ORDER BY blz DESC
183   LIMIT 1
184 ;
185 -- Hier per Unterabfrage, da es mit Variablen in PostgreSQL nicht so einfach
186 -- geht.
187
188 SELECT
189   bankname
190   FROM bank
191   WHERE blz = (
192     SELECT
193       blz
194       FROM bank
195       ORDER BY blz DESC
196       LIMIT 1
197     );
198
199
199 SELECT blz, bankname FROM bank ORDER BY blz LIMIT 2 OFFSET 1;
200 SELECT blz, bankname FROM bank ORDER BY blz LIMIT 1;
201 SELECT blz, bankname FROM bank ORDER BY blz LIMIT 1 OFFSET 0;
202 \echo 'Ende listing07.sql //////////////////////////////////////////////////////////////////'

```

Listing 29.29 postgresql/listing08.sql

```

1 -- Ausführen der vorherigen Befehle
2 \i listing07.sql
3
4 \echo 'Start listing08.sql //////////////////////////////////////////////////////////////////'
5
6 -- Start INNER JOIN
7 SELECT 'CROSS JOIN (Seite 184)' Hinweis;
8
9 SELECT
10   kunde_id, nachname, vorname, rechnung_adresse_id, adresse_id
11   FROM
12     kunde, adresse
13   ;
14
15 SELECT 'CROSS JOIN mit WHERE (Seite ??)' Hinweis;
16
17 SELECT
18   kunde_id, nachname, vorname, rechnung_adresse_id, adresse_id
19   FROM
20     kunde, adresse
21   WHERE
22     rechnung_adresse_id = adresse_id

```

```
21 ;
22
23 SELECT 'INNER JOIN (Seite 187)' Hinweis;
24 SELECT
25   kunde_id, nachname, vorname, rechnung_adresse_id, adresse_id
26   FROM
27     kunde INNER JOIN adresse
28       ON rechnung_adresse_id = adresse_id
29 ;
30
31 SELECT
32   kunde_id, kontonummer, blz, bankname
33   FROM
34     bankverbindung INNER JOIN bank
35       ON bankverbindung.bank_id = bank.bank_id
36 ;
37 SELECT
38   nachname, vorname, kontonummer
39   FROM
40     bankverbindung INNER JOIN kunde ON bankverbindung.kunde_id = kunde.kunde_id
41 ;
42
43 SELECT 'Abkürzung mit USING (Seite 192)' Hinweis;
44 SELECT
45   kunde_id, kontonummer, blz, bankname
46   FROM
47     bankverbindung INNER JOIN bank USING (bank_id)
48 ;
49
50 SELECT 'Abkürzung mit NATURAL JOIN (Seite 192)' Hinweis;
51 SELECT
52   kunde_id, kontonummer, blz, bankname
53   FROM
54     bankverbindung NATURAL JOIN bank
55 ;
56 -- Ende INNER JOIN
57
58 -- Start Datenquelle
59 SELECT 'JOIN als Datenquelle (Seite 193)' Hinweis;
60 SELECT
61   k.kunde_id, k.nachname, k.vorname, a.strasse, a.hnr, a.plz, a.ort
62   FROM
63     kunde k INNER JOIN adresse a
64       ON k.rechnung_adresse_id = a.adresse_id
65 ;
66 DROP TABLE IF EXISTS tmp_kadresse;
67 CREATE TEMPORARY TABLE tmp_kadresse AS
68 SELECT
69   k.kunde_id, k.nachname, k.vorname, a.strasse, a.hnr, a.plz, a.ort
70   FROM
71     kunde k INNER JOIN adresse a
72       ON k.rechnung_adresse_id = a.adresse_id
73 ;
74 SELECT
75   t.kunde_id, t.nachname, t.ort, b.bestellung_id, DATE(b.datum)
76   FROM
77     bestellung b INNER JOIN tmp_kadresse t USING (kunde_id)
78 ;
79 DROP TABLE IF EXISTS tmp_kbank;
```

```

80  CREATE TEMPORARY TABLE tmp_kbank AS
81    SELECT
82      bv.kunde_id, bv.bankverbindung_nr, bv.kontonummer,
83      bv.iban, ba.blz, ba.bankname
84    FROM
85      bankverbindung bv INNER JOIN bank ba USING (bank_id)
86    ;
87  DROP TABLE IF EXISTS tmp_kbankeinzug;
88  CREATE TEMPORARY TABLE tmp_kbankeinzug AS
89    SELECT
90      ka.*, kb.bankverbindung_nr, kb.kontonummer,
91      kb.iban, kb.blz, kb.bankname
92    FROM
93      tmp_kadresse ka INNER JOIN tmp_kbank kb USING(kunde_id);
94
95  SELECT kunde_id, nachname, ort, bankname FROM tmp_kbankeinzug ;
96  -- Ende Datenquelle
97
98  -- Start Keine Schlüssel
99  SELECT 'Ohne Schlüssel (Seite 196)' Hinweis;
100 \d+ tmp_kadresse;
101 \d+ tmp_kbank;
102 SELECT
103   k.kunde_id, k.vorname, a.strasse, a.hnr, a.plz, a.ort
104   FROM
105     kunde k INNER JOIN adresse a
106       ON k.rechnung_adresse_id <= a.adresse_id
107   ;
108  -- Ende Kein Schlüssel
109
110 -- Start mehr als zwei Tabellen
111 SELECT 'Mit mehr als 2 Tabellen (Seite 198)' Hinweis;
112 SELECT
113   a.bezeichnung, nm.warengruppe_id
114   FROM
115     artikel_nm_warengruppe nm NATURAL JOIN artikel a
116   ;
117 SELECT
118   w.bezeichnung, nm.artikel_id
119   FROM
120     artikel_nm_warengruppe nm NATURAL JOIN warengruppe w
121   ;
122 SELECT
123   w.bezeichnung, nm.artikel_id
124   FROM
125     (artikel_nm_warengruppe nm INNER JOIN warengruppe w USING(warengruppe_id))
126   ;
127 SELECT
128   w.bezeichnung, a.bezeichnung
129   FROM
130     (artikel_nm_warengruppe nm INNER JOIN warengruppe w USING (warengruppe_id))
131                           INNER JOIN artikel a USING(artikel_id)
132   ;
133 SELECT
134   w.bezeichnung Warengruppe, a.bezeichnung Artikel
135   FROM
136     artikel_nm_warengruppe nm INNER JOIN warengruppe w USING (warengruppe_id)
137                           INNER JOIN artikel a USING(artikel_id)
138   ;

```

```
139 -- Ende mehr als zwei Tabellen
140
141 -- Start Aufbau Lieferanten
142 SELECT 'Aufbau der Lieferanten' Hinweis;
143 INSERT INTO lieferant (firmenname, adresse_id)
144     VALUES
145     ('Gartenbedarf AllesGrün', 10),
146     ('Office International', 11),
147     ('Bürohengst GmbH', 11)
148 ;
149
150 INSERT INTO artikel_nm_lieferant (lieferant_id, artikel_id)
151     VALUES
152     (1, 7856)
153     ,(1, 7863)
154     ,(1, 9010)
155     ,(1, 9015)
156     ,(3, 3001)
157     ,(3, 3005)
158     ,(3, 3006)
159     ,(3, 3007)
160     ,(3, 3010)
161 ;
162 -- Ende Aufbau Lieferanten
163
164 -- Start OUTER JOIN
165 SELECT 'OUTER JOIN (Seite 200)' Hinweis;
166 SELECT
167     k.kunde_id, k.nachname, k.vorname, b.datum
168     FROM
169     bestellung b INNER JOIN kunde k USING (kunde_id)
170     ORDER BY
171         k.nachname, k.vorname
172 ;
173 SELECT
174     k.kunde_id, k.nachname, k.vorname, b.datum
175     FROM
176     bestellung b RIGHT OUTER JOIN kunde k USING (kunde_id)
177     ORDER BY
178         k.nachname, k.vorname
179 ;
180 SELECT
181     k.kunde_id, k.nachname, k.vorname, b.datum
182     FROM
183     kunde k LEFT OUTER JOIN bestellung b USING (kunde_id)
184     ORDER BY
185         k.nachname, k.vorname
186 ;
187 SELECT
188     l.lieferant_id, l.firmenname, a.artikel_id, a.bezeichnung
189     FROM
190     artikel_nm_lieferant nm INNER JOIN artikel a USING(artikel_id)
191             INNER JOIN lieferant l USING(lieferant_id)
192     ORDER BY firmenname
193 ;
194 SELECT
195     l.lieferant_id, l.firmenname, a.artikel_id, a.bezeichnung
196     FROM
197     artikel_nm_lieferant nm INNER JOIN artikel a USING(artikel_id)
```

```
198          RIGHT JOIN lieferant l USING(lieferant_id)
199 ORDER BY firmenname
200 ;
201 SELECT
202   l.firmenname
203   FROM
204     artikel_nm_lieferant nm INNER JOIN artikel a USING(artikel_id)
205           RIGHT JOIN lieferant l USING(lieferant_id)
206 WHERE artikel_id IS NULL
207 ORDER BY firmenname
208 ;
209 SELECT k.kunde_id, k.nachname, k.vorname
210   FROM
211     bestellung b RIGHT JOIN kunde k USING(kunde_id)
212 WHERE b.bestellung_id IS NULL
213 ;
214 SELECT k.kunde_id, k.nachname, k.vorname, b.bestellung_id, b.datum
215   FROM
216     bestellung b FULL OUTER JOIN kunde k USING(kunde_id)
217 ;
218 -- Ende OUTER JOIN
219
220 -- Start Aufbau Forum
221 SELECT 'SELF JOIN (Seite 205)' Hinweis;
222 SELECT 'CREATE account' Hinweis;
223 CREATE TYPE accountart AS ENUM('aktiv','gesperrt');
224 CREATE TABLE account (
225   kunde_id INT,
226   name      VARCHAR(255) NOT NULL DEFAULT '',
227   passwort  CHAR(34) NOT NULL DEFAULT '',
228   status    accountart NOT NULL DEFAULT 'aktiv',
229   kommentar TEXT ,
230   admin     BOOL NOT NULL DEFAULT FALSE,
231   deleted   SMALLINT NOT NULL DEFAULT 0,
232   PRIMARY KEY(kunde_id),
233   FOREIGN KEY (kunde_id)
234     REFERENCES kunde(kunde_id)
235     ON UPDATE RESTRICT
236     ON DELETE RESTRICT
237 ) ;
238
239 SELECT 'CREATE beitrag' Hinweis;
240 CREATE TABLE beitrag (
241   beitrag_id SERIAL,
242   account_id INT NOT NULL,
243   bezug_beitrag_id INT NOT NULL DEFAULT 1,
244   zeitstempel TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
245   nachricht TEXT ,
246   deleted   SMALLINT NOT NULL DEFAULT 0,
247   PRIMARY KEY(beitrag_id),
248   FOREIGN KEY (account_id)
249     REFERENCES account(kunde_id)
250     ON UPDATE RESTRICT
251     ON DELETE RESTRICT,
252   FOREIGN KEY (bezug_beitrag_id)
253     REFERENCES beitrag(beitrag_id)
254     ON UPDATE RESTRICT
255     ON DELETE RESTRICT
256 ) ;
```

```
257
258 SELECT 'account daten' Hinweis;
259 INSERT INTO account (kunde_id, name, passwort, admin)
260   VALUES
261     (1, 'admin', md5('rosi'), TRUE)
262 ,  (2, 'frodo', md5('elbereth'), FALSE)
263 ,  (3, 'bilbo', md5('blitzerschlagen'), FALSE)
264 ,  (5, 'elle', md5('feanor'), FALSE)
265 ;
266
267
268 SELECT 'beitrag daten' Hinweis;
269 INSERT INTO beitrag (beitrag_id, account_id, bezug_beitrag_id, nachricht)
270   VALUES
271     (1, 1, 1, '')
272 ;
273 INSERT INTO beitrag (beitrag_id, account_id, bezug_beitrag_id, zeitstempel,
274   nachricht)
275   VALUES
276     (2, 2, 1, '2011-05-01 14:13:00', 'Der Lieferservice ist super.')
277 ,  (3, 3, 2, '2011-05-02 11:45:00', 'Das finde ich auch.')
278 ,  (4, 5, 2, '2011-05-01 17:01:00', 'Aber ein wenig langsam.')
279 ,  (5, 2, 4, '2011-05-01 17:15:00', 'Finde ich nicht.')
280 ,  (6, 5, 1, '2011-06-12 09:07:00', 'Angebot könnte besser sein.')
281 ;
282 -- Ende Aufbau Forum
283
284 -- Start SELF JOIN
285 SELECT 'SELF JOIN (Seite 206)' Hinweis;
286 SELECT
287   kunde_id, name, status, admin
288   FROM
289   account
290 ;
291 SELECT
292   beitrag_id, account_id, bezug_beitrag_id, LEFT(nachricht, 20)
293   FROM
294   beitrag
295 ;
296 -- Erwarte eine Fehlermeldung (Seite 207);
297 -- SELECT nachricht
298 --  FROM
299 --   beitrag INNER JOIN beitrag ON bezug_beitrag_id = beitrag_id
300 -- WHERE beitrag_id = 2
301 -- ;
302 SELECT ant.nachricht Antwort
303   FROM
304   beitrag ant INNER JOIN beitrag orig ON ant.bezug_beitrag_id = orig.
305     beitrag_id
306 WHERE orig.beitrag_id = 2
307 ;
308
309 SELECT 'CTE rekursiv (Seite 208)';
310 WITH RECURSIVE ant_auf AS
311 (
312   -- nicht rekuriver Teil
313   SELECT * FROM beitrag WHERE bezug_beitrag_id = 2
314   UNION ALL
315   -- rekuriver Teil
```

```

314  SELECT ant.*  
315    FROM  
316      beitrag ant INNER JOIN ant_auf orig  
317        ON ant.bezug_beitrag_id = orig.beitrag_id  
318    )  
319  SELECT beitrag_id, bezug_beitrag_id, nachricht FROM ant_auf;  
320  
321  -- Ende SELF JOIN  
322  
323  -- Start Redundant  
324  SELECT 'Beschleunigen mit Redundanzen (Seite 209)' Hinweis;  
325  ALTER TABLE  
326    kunde  
327    ADD r_strasse VARCHAR(255),  
328    ADD r_ort VARCHAR(255),  
329    ADD r_aktuell BOOL NOT NULL DEFAULT TRUE,  
330    ADD l_strasse VARCHAR(255),  
331    ADD l_ort VARCHAR(255),  
332    ADD l_aktuell BOOL NOT NULL DEFAULT TRUE  
333  ;  
334  
335  -- In PostgreSQL kann kein Update auf einer abgeleiteten Tabelle durchgeführt  
     werden.  
336  -- UPDATE kunde INNER JOIN adresse ON rechnung_adresse_id = adresse_id  
337  -- SET  
338  --   r_strasse = CONCAT(strasse, ' ', hnr),  
339  --   r_ort = CONCAT(lkz, '-', plz, ' ', ort),  
340  --   r_aktuell = TRUE  
341  -- ;  
342  SELECT  
343    kunde_id, r_strasse, r_ort, r_aktuell  
344  FROM kunde  
345  ;  
346  -- Ende Redundant  
347 \echo 'Ende listing08.sql //////////////////////////////////////////////////////////////////'

```

Listing 29.30 postgresql/listing09.sql

```

1  -- Ausführen der vorherigen Befehle  
2  \i listing08.sql  
3  
4  \echo 'Start listing09.sql //////////////////////////////////////////////////////////////////'  
5  
6  SELECT 'Aggregatfunktionen (Seite 212)' Hinweis;  
7  SELECT AVG(einzelpreis) FROM artikel;  
8  SELECT COUNT(*) FROM kunde;  
9  SELECT COUNT(liefer_adresse_id) FROM kunde;  
10 SELECT COUNT(DISTINCT(rechnung_adresse_id)) FROM kunde;  
11 SELECT MAX(einzelpreis), MIN(einzelpreis) FROM artikel;  
12 SELECT SUM(menge) FROM bestellung_position;  
13  
14 SELECT  
15   bp.menge, a.einzelpreis  
16  FROM  
17    bestellung_position bp INNER JOIN artikel a USING(artikel_id)  
18  ;  
19 SELECT  
20   bp.menge * a.einzelpreis Positionswert  
21  FROM

```

```
22     bestellung_position AS bp INNER JOIN artikel AS a USING(artikel_id)
23 ;
24
25 -- Start Aufbau Lagerverwaltung
26 SELECT 'Aufbau Lagerbestand (Seite 214)' Hinweis;
27 SELECT 'CREATE lagerbestand' Hinweis;
28 DROP TABLE IF EXISTS lagerbestand;
29 CREATE TABLE lagerbestand (
30     artikel_id          INT,
31     menge_mindest      DECIMAL(14,6) NOT NULL DEFAULT 0.0,
32     menge_aktuell       DECIMAL(14,6) NOT NULL DEFAULT 0.0,
33     deleted             SMALLINT NOT NULL DEFAULT 0,
34     PRIMARY KEY(artikel_id),
35     FOREIGN KEY (artikel_id)
36         REFERENCES artikel(artikel_id)
37         ON UPDATE RESTRICT
38         ON DELETE RESTRICT
39 );
40
41 SELECT 'Lagerbestand festlegen' Hinweis;
42 INSERT INTO lagerbestand (artikel_id, menge_aktuell, menge_mindest)
43 VALUES
44     (7856, 500, 100)
45     ,(7863, 400, 100)
46     ,(9010, 30, 10)
47     ,(9015, 25, 10)
48     ,(3001, 5000, 1000)
49     ,(3005, 250, 200)
50     ,(3006, 250, 200)
51     ,(3007, 149, 200)
52     ,(3010, 3, 100)
53 ;
54 -- Ende Aufbau Lagerverwaltung
55
56
57 SELECT 'GROUP BY (Seite 215)' Hinweis;
58 -- Start Group by
59 SELECT
60     COUNT(*)
61     FROM
62     bestellung_position
63 ;
64 SELECT
65     bestellung_id Bestellnummer, COUNT(*) Anzahl_der_Positionen
66     FROM
67     bestellung_position
68     GROUP BY
69     bestellung_id
70 ;
71 SELECT
72     a.bezeichnung, bp.menge
73     FROM
74     bestellung_position bp INNER JOIN artikel a USING (artikel_id)
75 ;
76 SELECT
77     a.bezeichnung Artikelname, SUM(bp.menge) Anzahl_bestellter_Artikel
78     FROM
79     bestellung_position bp INNER JOIN artikel a USING (artikel_id)
80     GROUP BY
```

```
81     a.bezeichnung
82 ;
83
84 SELECT
85     a.bezeichnung Artikelname, SUM(bp.menge) Anzahl_bestellter_Artikel
86     FROM
87         bestellung_position bp RIGHT JOIN artikel a USING (artikel_id)
88     GROUP BY
89         a.bezeichnung
90 ;
91 SELECT
92     a.bezeichnung Artikelname,
93     CASE
94         WHEN SUM(bp.menge) IS NULL THEN 0
95         ELSE SUM(bp.menge)
96     END AS Anzahl_bestellter_Artikel
97     FROM
98         bestellung_position bp RIGHT JOIN artikel a USING (artikel_id)
99     GROUP BY
100        a.bezeichnung
101 ;
102 -- SELECT 'Eine Fehlermeldung! (Seite 218)';
103 -- SELECT
104 --     a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
105 --     FROM
106 --         bestellung_position bp INNER JOIN artikel a USING (artikel_id)
107 --     WHERE
108 --         SUM(bp.menge) > 10
109 --     GROUP BY
110 --         a.bezeichnung
111 -- ;
112
113 SELECT 'Gruppenergebnisse mit WITH ROLLUP summieren (Seite 217)';
114 SELECT
115     artikel_id, SUM(bp.menge) AS
116     FROM
117         bestellung_position bp RIGHT JOIN artikel a USING (artikel_id)
118     GROUP BY ROLLUP (artikel_id)
119 ;
120
121 SELECT 'Gruppenergebnisse mit HAVING filtern (Seite 219)' Hinweis;
122 SELECT
123     a.bezeichnung Artikelname, SUM(bp.menge) Anzahl_bestellter_Artikel
124     FROM
125         bestellung_position bp INNER JOIN artikel a USING (artikel_id)
126     GROUP BY
127         a.bezeichnung
128     HAVING
129         SUM(bp.menge) > 10
130 ;
131 SELECT
132     a.bezeichnung Artikelname, SUM(bp.menge) Anzahl_bestellter_Artikel
133     FROM
134         bestellung_position bp INNER JOIN artikel a USING (artikel_id)
135     WHERE
136         a.bezeichnung LIKE 'Silber%'
137     GROUP BY
138         a.bezeichnung
139     HAVING
```

```
140     SUM(bp.menge) > 10
141 ;
142
143 SELECT 'Fragen zu Gruppen (Seite 220)' Hinweis;
144 SELECT
145     SUBSTRING(blz, 1, 1) Clearinggebiet, COUNT(*) Anzahl
146     FROM
147         bank
148     GROUP BY
149         Clearinggebiet
150     ORDER BY Anzahl DESC
151 ;
152 -- Start Bestellungen erweitern
153 SELECT 'Bestellungen erweitern' Hinweis;
154 INSERT INTO bestellung (bestellung_id, kunde_id, adresse_id, datum)
155     VALUES
156         (3, 1, 1, '2011-01-15 16:43:00')
157         ,(4, 1, 1, '2011-01-16 09:15:00')
158         ,(5, 1, 1, '2011-01-16 09:16:00')
159         ,(6, 2, 2, '2012-04-01 13:11:00')
160 ;
161
162 INSERT INTO bestellung_position (bestellung_id, position_nr, artikel_id, menge
163     )
164     VALUES
165         (3, 1, 7856, 10)
166         ,(3, 2, 7863, 10)
167         ,(4, 1, 3006, 1)
168         ,(4, 2, 3010, 4)
169         ,(5, 1, 3001, 100)
170         ,(5, 2, 3010, 5)
171         ,(5, 3, 3006, 1)
172         ,(5, 4, 3005, 4)
173 ;
174 -- Ende Bestellungen erweitern
175 SELECT
176     EXTRACT(ISOYEAR FROM datum) || '/' || EXTRACT(MONTH FROM datum) Monat, COUNT
177         (*) Anzahl
178     FROM bestellung
179     GROUP BY
180         EXTRACT(ISOYEAR FROM datum), EXTRACT(MONTH FROM datum)
181     ORDER BY
182         EXTRACT(ISOYEAR FROM datum) DESC, EXTRACT(MONTH FROM datum) DESC
183 ;
184
185 EXPLAIN
186 SELECT
187     SUBSTRING(blz, 1, 1) Clearinggebiet, COUNT(*) Anzahl
188     FROM
189         bank
190     GROUP BY
191         Clearinggebiet
192     ORDER BY
193         Anzahl DESC;
194
195 EXPLAIN
196 SELECT
197     lkz || SUBSTRING(blz, 6, 3) Sinnlos, COUNT(*) Anzahl
198     FROM
```

```

197     bank
198     GROUP BY
199     Sinnlos
200     ORDER BY
201     Sinnlos;
202 -- Ende Group by
203 \echo 'Ende listing09.sql /////////////////////////////////'
```

Listing 29.31 postgresql/listing10.sql

```

1  -- Ausführen der vorherigen Befehle
2  \i listing09.sql
3
4  \echo 'Start listing10.sql /////////////////////////////////'
5
6  SELECT 'Start Aufbau Rechnungswesen' Hinweis;
7
8  INSERT INTO rechnung
9    (kunde_id, bestellung_id, adresse_id, datum)
10   SELECT kunde_id, bestellung_id, adresse_id, datum + INTERVAL '1 days' FROM
11      bestellung;
12
13  INSERT INTO rechnung (rechnung_id, kunde_id, bestellung_id, adresse_id, datum,
14    bezahlart, status)
15    VALUES
16      (7, 3, 0, 2, '2012-04-06 12:11:00', 'kredit', 'gestellt')
17      ,(8, 3, 0, 2, '2012-04-07 13:12:00', 'kredit', 'mahnung1')
18      ,(9, 5, 0, 4, '2012-04-08 14:13:00', 'rechnung', 'storno')
19      ,(10, 5, 0, 4, '2012-04-09 15:14:00', 'rechnung', 'beglichen')
20      ,(11, 5, 0, 4, '2012-04-10 16:15:00', 'rechnung', 'beglichen')
21 ;
22 UPDATE rechnung SET status = 'inkasso' WHERE rechnung_id = 2;
23 INSERT INTO rechnung_position (rechnung_id, position_nr, artikel_id, menge)
24   SELECT bestellung_id, position_nr, artikel_id, menge FROM bestellung_position
25 ;
26
27  INSERT INTO rechnung_position
28    (rechnung_id, position_nr, artikel_id, menge)
29    VALUES
30      (7, 1, 3001, 5)
31      ,(7, 2, 3005, 1)
32      ,(8, 1, 3006, 7)
33      ,(8, 2, 3007, 1)
34      ,(10, 1, 3010, 15)
35      ,(10, 2, 3001, 5)
36      ,(11, 1, 3005, 20)
37 ;
38
39 UPDATE rechnung SET rabatt = 7 WHERE kunde_id = 1;
40 UPDATE rechnung SET skonto = 3 WHERE kunde_id = 5;
41
42 SELECT 'Ende Aufbau Rechnungswesen' Hinweis;
43
44 SELECT 'Problem und Loesung (Seite 225)' Hinweis;
45 SELECT
46   bp.bestellung_id, SUM(bp.menge * a.einzelpreis) Bestellwert
47   FROM
48     bestellung_position bp INNER JOIN artikel a USING(artikel_id)
49   GROUP BY
```

```
47     bp.bestellung_id
48 ;
49 SELECT
50     k.nachname, k.vorname, SUM(bp.menge * a.einzelpreis) Bestellwert
51 FROM
52     bestellung_position bp INNER JOIN artikel a      USING(artikel_id)
53                         INNER JOIN bestellung b USING(bestellung_id)
54                         INNER JOIN kunde k      USING(kunde_id)
55 GROUP BY
56     bp.bestellung_id, k.nachname, k.vorname
57 ;
58 DROP TABLE IF EXISTS tmp_bestellwert;
59 CREATE TEMPORARY TABLE tmp_bestellwert
60 AS
61     SELECT
62         bp.bestellung_id, k.nachname, k.vorname, SUM(bp.menge * a.einzelpreis)
63             bestellwert
64     FROM
65         bestellung_position bp INNER JOIN artikel a USING(artikel_id)
66                         INNER JOIN bestellung b USING(bestellung_id)
67                         INNER JOIN kunde k USING(kunde_id)
68 GROUP BY
69     bp.bestellung_id, k.nachname, k.vorname
70 ;
71 SELECT * FROM tmp_bestellwert;
72 SELECT
73     nachname, vorname, SUM(bestellwert) Umsatz
74     FROM tmp_bestellwert
75     GROUP BY nachname, vorname
76 ;
77 SELECT
78     bw.nachname, bw.vorname, SUM(bw.bestellwert) Umsatz
79     FROM
80     (
81         SELECT
82             bp.bestellung_id, k.nachname, k.vorname, SUM(bp.menge * a.einzelpreis)
83                 bestellwert
84             FROM
85                 bestellung_position bp INNER JOIN artikel a USING(artikel_id)
86                         INNER JOIN bestellung b USING(bestellung_id)
87                         INNER JOIN kunde k USING(kunde_id)
88             GROUP BY
89                 bp.bestellung_id, k.nachname, k.vorname
90     ) AS bw
91     GROUP BY nachname, vorname
92 ;
93 -- Start nicht korrelierende Unterabfrage
94 SELECT 'Nicht korrelierende Unterabfrage (Seite 228) Hinweis';
95 SELECT MAX(blz) FROM bank;
96 SELECT bankname
97     FROM bank
98     WHERE blz = '37010050'
99 ;
100 SELECT bankname
101    FROM bank
102   WHERE blz =
103     (
104         SELECT MAX(blz)
```

```
104      FROM bank
105  )
106 ;
107 SELECT
108   AVG(einzelpreis) AS durchschnittspreis
109  FROM artikel
110 ;
111 SELECT *
112  FROM artikel
113 WHERE
114   einzelpreis > 30
115 ;
116 SELECT *
117  FROM artikel
118 WHERE
119   einzelpreis >
120   (
121   SELECT
122     AVG(einzelpreis) AS durchschnittspreis
123    FROM artikel
124   )
125 ;
126 SELECT
127   bp.bestellung_id, SUM(bp.menge * a.einzelpreis) bestellwert
128  FROM
129    bestellung_position bp INNER JOIN artikel a USING(artikel_id)
130  GROUP BY
131    bp.bestellung_id
132 ;
133 SELECT AVG(bw.bestellwert)
134  FROM (
135   SELECT
136     bp.bestellung_id, SUM(bp.menge * a.einzelpreis) bestellwert
137    FROM
138      bestellung_position bp INNER JOIN artikel a USING(artikel_id)
139    GROUP BY
140      bp.bestellung_id
141    ) AS bw
142 ;
143 SELECT
144   bp1.bestellung_id, SUM(bp1.menge * a1.einzelpreis) bestellwert1
145  FROM
146    bestellung_position bp1 INNER JOIN artikel a1 USING(artikel_id)
147  GROUP BY
148    bp1.bestellung_id
149  HAVING SUM(bp1.menge * a1.einzelpreis) > 100
150 ;
151 SELECT
152   bp1.bestellung_id, SUM(bp1.menge * a1.einzelpreis) bwert1
153  FROM
154    bestellung_position bp1 INNER JOIN artikel a1 USING(artikel_id)
155  GROUP BY
156    bp1.bestellung_id
157  HAVING
158    SUM(bp1.menge * a1.einzelpreis) >
159    (
160      SELECT AVG(bw.bwert2)
161      FROM
162      (
```

```
163     SELECT
164         bp2.bestellung_id, SUM(bp2.menge * a2.einzelpreis) bwert2
165     FROM
166         bestellung_position bp2 INNER JOIN artikel a2
167             USING(artikel_id)
168     GROUP BY
169         bp2.bestellung_id
170     ) AS bw
171 )
172 ;
173 SELECT bestellung_id FROM bestellung;
174 SELECT rechnung_id
175     FROM rechnung
176 WHERE bestellung_id IN
177 (
178     SELECT bestellung_id FROM bestellung
179 )
180 ;
181 SELECT a.einzelpreis
182     FROM
183         artikel_nm_warengruppe INNER JOIN artikel a      USING(artikel_id)
184                         INNER JOIN warengruppe w USING(warengruppe_id)
185     WHERE
186         w.bezeichnung = 'Gartenbedarf'
187 ;
188 SELECT
189     a.bezeichnung, a.einzelpreis
190     FROM
191         artikel a
192     WHERE
193         a.einzelpreis > ALL(SELECT 10)
194 ;
195 SELECT
196     a.bezeichnung, a.einzelpreis
197     FROM
198         artikel a
199     WHERE
200         a.einzelpreis > ALL
201     (
202         SELECT a.einzelpreis
203             FROM
204                 artikel_nm_warengruppe INNER JOIN artikel a USING(artikel_id)
205                         INNER JOIN warengruppe w USING(warengruppe_id)
206             WHERE
207                 w.bezeichnung = 'Gartenbedarf'
208     )
209 ;
210 SELECT
211     rechnung_id, kunde_id, SUM(einzelpreis*menge) AS umsatz
212     FROM
213         rechnung_position INNER JOIN artikel  USING(artikel_id)
214                         INNER JOIN rechnung USING(rechnung_id)
215     GROUP BY
216         rechnung_id, kunde_id
217 ;
218 SELECT SUM(umsatz) AS umsatzsumme
219     FROM
220     (
221         SELECT
```

```
222     rechnung_id, kunde_id, SUM(einzelpreis*menge) AS umsatz
223     FROM
224         rechnung_position INNER JOIN artikel USING(artikel_id)
225             INNER JOIN rechnung USING(rechnung_id)
226             GROUP BY
227                 rechnung_id, kunde_id
228     ) AS ksum
229     GROUP BY
230         kunde_id
231 ;
232 SELECT SUM(umsatz) AS umsatzsumme
233     FROM
234 (
235     SELECT
236         rechnung_id, kunde_id, SUM(einzelpreis*menge) AS umsatz
237         FROM
238             rechnung_position INNER JOIN artikel USING(artikel_id)
239                 INNER JOIN rechnung USING(rechnung_id)
240                     INNER JOIN kunde USING(kunde_id)
241             WHERE nachname = 'beutlin'
242             GROUP BY
243                 rechnung_id, kunde_id
244     ) AS ksum
245     GROUP BY
246         kunde_id
247 ;
248 SELECT kunde_id, SUM(umsatz) AS umsatzsumme
249     FROM
250 (
251     SELECT
252         rechnung_id, kunde_id, SUM(einzelpreis*menge) AS umsatz
253         FROM
254             rechnung_position INNER JOIN artikel USING(artikel_id)
255                 INNER JOIN rechnung USING(rechnung_id)
256             GROUP BY
257                 rechnung_id, kunde_id
258     ) AS ksum
259     GROUP BY
260         kunde_id
261     HAVING SUM(umsatz) > ALL (SELECT 100)
262 ;
263 SELECT kunde_id, kunde.nachname, kunde.vorname, SUM(umsatz) AS umsatzsumme
264     FROM
265 (
266     SELECT
267         rechnung_id, kunde_id, SUM(einzelpreis*menge) AS umsatz
268         FROM
269             rechnung_position INNER JOIN artikel USING(artikel_id)
270                 INNER JOIN rechnung USING(rechnung_id)
271             GROUP BY
272                 rechnung_id, kunde_id
273     ) AS ksum
274     INNER JOIN kunde USING (kunde_id)
275     GROUP BY
276         kunde_id, kunde.nachname, kunde.vorname
277     HAVING SUM(umsatz) > ALL
278     (
279         SELECT SUM(umsatz) AS umsatzsumme
280             FROM
```

```
281      (
282      SELECT
283          rechnung_id, kunde_id, SUM(einzelpreis*menge) AS umsatz
284          FROM
285              rechnung_position INNER JOIN artikel USING(artikel_id)
286                  INNER JOIN rechnung USING(rechnung_id)
287                      INNER JOIN kunde USING(kunde_id)
288              WHERE nachname = 'beutlin'
289              GROUP BY
290                  rechnung_id, kunde_id
291      ) AS ksum
292      GROUP BY
293          kunde_id, kunde.nachname, kunde.vorname
294  )
295 ;
296 SELECT a.einzelpreis
297   FROM
298       artikel_nm_warengruppe INNER JOIN artikel a      USING(artikel_id)
299                           INNER JOIN warengruppe w USING(warengruppe_id)
300 WHERE
301     w.bezeichnung = 'Bürobedarf'
302 ;
303 SELECT
304     a.bezeichnung, a.einzelpreis
305   FROM
306     artikel a
307 WHERE
308     a.einzelpreis > ANY(SELECT 10)
309 ;
310 SELECT
311     a.bezeichnung, a.einzelpreis
312   FROM
313     artikel a
314 WHERE
315     a.einzelpreis > ANY
316     (
317         SELECT a.einzelpreis
318           FROM
319               artikel_nm_warengruppe INNER JOIN artikel a USING(artikel_id)
320                               INNER JOIN warengruppe w USING(warengruppe_id)
321             WHERE
322                 w.bezeichnung = 'Bürobedarf'
323     )
324 ;
325 SELECT
326     a.bezeichnung, a.einzelpreis
327   FROM
328     artikel a
329 WHERE
330     a.artikel_id NOT IN
331     (
332         SELECT
333             artikel_id
334           FROM
335               artikel_nm_warengruppe nm  INNER JOIN warengruppe w USING(warengruppe_id)
336             WHERE w.bezeichnung = 'Bürobedarf'
337     )
338 AND
339     a.einzelpreis > ANY
```

```

340      (
341          SELECT a.einzelpreis
342              FROM
343                  artikel_nm_warengruppe nm INNER JOIN artikel a USING(artikel_id)
344                                  INNER JOIN warengruppe w USING(warengruppe_id)
345              WHERE
346                  w.bezeichnung = 'Bürobedarf'
347      )
348 ;
349 SELECT DISTINCT k.nachname, k.vorname
350     FROM
351         rechnung r INNER JOIN kunde k USING(kunde_id)
352     WHERE
353         EXTRACT(ISOYEAR FROM r.datum) = '2011' AND (k.nachname, k.vorname) IN
354     (
355         SELECT
356             k.nachname, k.vorname
357             FROM
358                 rechnung r INNER JOIN kunde k USING(kunde_id)
359             WHERE
360                 EXTRACT(ISOYEAR FROM r.datum) = '2012'
361     )
362 ;
363 -- Ende nicht korrelierende Unterabfrage
364
365 -- Start korrelierende Unterabfrage
366 SELECT 'Korrelierende Unterabfrage (Seite 240)' Hinweis;
367 SELECT rechnung_id
368     FROM rechnung r
369     WHERE
370     (
371         SELECT
372             COUNT(position_nr)
373             FROM rechnung_position rp
374             WHERE
375                 rp.rechnung_id = r.rechnung_id
376         ) >= 3
377 ;
378 SELECT rechnung_id
379     FROM rechnung r
380     WHERE
381         EXISTS
382     (
383         SELECT
384             bestellung_id
385             FROM bestellung b
386             WHERE b.bestellung_id = r.bestellung_id
387     )
388 ;
389 -- Ende korrelierende Unterabfrage
390
391 -- Start Fallstudie
392 SELECT 'Fallstudie Datenimport (Seite 242)' Hinweis;
393 DROP TABLE IF EXISTS tmp_import;
394 CREATE TEMPORARY TABLE tmp_import
395 (
396     nachname VARCHAR(255),
397     vorname  VARCHAR(255),
398     strasse  VARCHAR(255),

```

```
399     hnr      VARCHAR(255) ,
400     lkz      VARCHAR(255) ,
401     plz      VARCHAR(255) ,
402     ort      VARCHAR(255) ,
403     ktnr     VARCHAR(255) ,
404     blz      VARCHAR(255)
405   );
406
407 \copy tmp_import FROM './kunden01.csv' WITH DELIMITER ';' CSV HEADER
408 SELECT SETVAL('kunde_kunde_id_seq', 6); -- PK Wert manuel anpassen
409 INSERT INTO kunde (nachname, vorname)
410   SELECT DISTINCT nachname, vorname
411   FROM tmp_import tmp
412   WHERE
413     (tmp.vorname, tmp.nachname) NOT IN
414     (
415       SELECT vorname, nachname FROM kunde
416     )
417 ;
418 SELECT * FROM tmp_import;
419 SELECT SETVAL('adresse_adresse_id_seq', 100); -- PK Wert manuel anpassen
420 INSERT INTO adresse (strasse, hnr, lkz, plz, ort)
421   SELECT DISTINCT strasse, hnr, lkz, plz, ort
422   FROM tmp_import tmp
423   WHERE
424     (tmp.strasse, tmp.hnr, tmp.lkz, tmp.plz, tmp.ort) NOT IN
425     (
426       SELECT strasse, hnr, lkz, plz, ort
427       FROM adresse
428     )
429 ;
430 ALTER TABLE tmp_import ADD kunde_id    INT ,
431                      ADD adresse_id INT ,
432                      ADD bank_id    CHAR(12)
433 ;
434 UPDATE tmp_import SET
435   kunde_id =
436   (
437     SELECT kunde_id
438     FROM kunde k
439     WHERE tmp_import.vorname = k.vorname AND tmp_import.nachname = k.nachname
440   ),
441   adresse_id =
442   (
443     SELECT adresse_id
444     FROM adresse a
445     WHERE
446       tmp_import.strasse = a.strasse
447       AND tmp_import.hnr = a.hnr
448       AND tmp_import.plz = a.plz
449       AND tmp_import.ort = a.ort
450   ),
451   bank_id =
452   (
453     SELECT bank_id FROM bank WHERE tmp_import.blz = bank.blz LIMIT 1
454   )
455 ;
456
457 INSERT INTO
```

```

458 bankverbindung (kunde_id, bankverbindung_nr, bank_id, kontonummer, iban)
459 SELECT DISTINCT kunde_id, 1, bank_id, ktnr, blz || ktnr
460   FROM tmp_import tmp
461 WHERE
462   (tmp.kunde_id, 1, tmp.bank_id, tmp.ktnr) NOT IN
463   (
464     SELECT kunde_id, 1, bank_id, kontonummer
465       FROM bankverbindung
466   )
467 ;
468 ALTER TABLE tmp_import
469   ADD bankverbindung_nr INT NOT NULL DEFAULT 1
470 ;
471 UPDATE kunde k SET
472   rechnung_adresse_id =
473   (
474     SELECT adresse_id FROM tmp_import tmp WHERE tmp.kunde_id = k.kunde_id
475   )
476 WHERE k.rechnung_adresse_id IS NULL
477 ;
478 SELECT kunde_id, nachname, vorname, strasse, ort
479   FROM
480     kunde LEFT JOIN adresse ON kunde.rechnung_adresse_id = adresse_id
481 ;
482 -- Ende Fallstudie
483
484 SELECT 'Unterabfragen intern (Seite 245) Hinweis';
485 SELECT
486   bp1.bestellung_id, SUM(bp1.menge * a1.einzelpreis) bwert1
487   FROM
488     bestellung_position bp1 INNER JOIN artikel a1 USING(artikel_id)
489   GROUP BY
490     bp1.bestellung_id
491   HAVING
492     SUM(bp1.menge * a1.einzelpreis) >
493   (
494     SELECT AVG(bw.bwert2)
495     FROM
496     (
497       SELECT
498         bp2.bestellung_id, SUM(bp2.menge * a2.einzelpreis) bwert2
499         FROM
500           bestellung_position bp2 INNER JOIN artikel a2
501             USING(artikel_id)
502           GROUP BY
503             bp2.bestellung_id
504       ) AS bw
505   );
506
507 EXPLAIN
508 SELECT
509   bp1.bestellung_id, SUM(bp1.menge * a1.einzelpreis) bwert1
510   FROM
511     bestellung_position bp1 INNER JOIN artikel a1 USING(artikel_id)
512   GROUP BY
513     bp1.bestellung_id
514   HAVING
515     SUM(bp1.menge * a1.einzelpreis) >
516   (

```

```

517     SELECT AVG(bw.bwert2)
518     FROM
519     (
520         SELECT
521             bp2.bestellung_id, SUM(bp2.menge * a2.einzelpreis) bwert2
522             FROM
523                 bestellung_position bp2 INNER JOIN artikel a2
524                     USING(artikel_id)
525             GROUP BY
526                 bp2.bestellung_id
527         ) AS bw
528     );
529
530 \echo 'Ende listing10.sql /////////////////////////////////'

```

Listing 29.32 postgresql/listing11.sql

```

1 -- Ausführen der vorherigen Befehle
2 \include listing10.sql
3
4 \echo 'Start listing11.sql /////////////////////////////////'
5
6 SELECT 'UNION (Seite 251)' Hinweis;
7 SELECT
8     strasse, hnr, lkz, plz, ort
9     FROM
10    kunde INNER JOIN adresse ON rechnung_adresse_id = adresse_id
11 ;
12 SELECT
13     strasse, hnr, lkz, plz, ort
14     FROM
15    lieferant INNER JOIN adresse USING(adresse_id)
16 ;
17 SELECT
18     strasse, hnr, lkz, plz, ort
19     FROM
20    kunde INNER JOIN adresse ON rechnung_adresse_id = adresse_id
21 UNION
22 SELECT
23     strasse, hnr, lkz, plz, ort
24     FROM
25    lieferant INNER JOIN adresse USING(adresse_id)
26 ;
27 SELECT
28     strasse, hnr, lkz, plz, ort
29     FROM
30    kunde INNER JOIN adresse ON rechnung_adresse_id = adresse_id
31 UNION ALL
32 SELECT
33     strasse, hnr, lkz, plz, ort
34     FROM
35    lieferant INNER JOIN adresse USING(adresse_id)
36 ;
37
38 SELECT 'INTERSECT (Seite 254)' Hinweis;
39
40 SELECT kunde_id FROM rechnung WHERE EXTRACT(ISOYEAR FROM datum) = '2011'
41 INTERSECT
42 SELECT kunde_id FROM rechnung WHERE EXTRACT(ISOYEAR FROM datum) = '2012';

```

```

43
44  SELECT 'Schnittmenge mit Unterabfrage (Seite 251)' Hinweis;
45  SELECT kunde_id FROM rechnung WHERE EXTRACT(ISOYEAR FROM datum) = '2011';
46  SELECT DISTINCT kunde_id
47    FROM rechnung
48    WHERE EXTRACT(ISOYEAR FROM datum) = '2012' AND kunde_id IN (1, 2, 3)
49  ;
50  SELECT DISTINCT kunde_id
51    FROM rechnung
52    WHERE
53      EXTRACT(ISOYEAR FROM datum) = '2012' AND kunde_id IN
54      (
55        SELECT kunde_id FROM rechnung WHERE EXTRACT(ISOYEAR FROM datum) = '2011'
56      )
57  ;
58
59  SELECT 'EXCEPT (Seite 256)' Hinweis;
60  SELECT strasse, hnr, lkz, plz, ort FROM adresse
61  EXCEPT
62  SELECT strasse, hnr, lkz, plz, ort
63    FROM kunde INNER JOIN adresse ON liefer_adresse_id = adresse_id
64  ;
65
66  SELECT 'Differenz mit Unterabfrage (Seite 256)' Hinweis;
67  SELECT
68    strasse, hnr, lkz, plz, ort
69    FROM
70      adresse
71  ;
72  SELECT
73    strasse, hnr, lkz, plz, ort
74    FROM
75      kunde INNER JOIN adresse
76        ON liefer_adresse_id = adresse_id
77  ;
78  SELECT kunde_id
79    FROM rechnung
80    WHERE EXTRACT(ISOYEAR FROM datum) <> '2011' AND EXTRACT(ISOYEAR FROM datum) =
81      '2012'
82  ;
83  SELECT kunde_id FROM rechnung WHERE EXTRACT(ISOYEAR FROM datum) = '2011';
84  SELECT DISTINCT kunde_id
85    FROM rechnung
86    WHERE
87      EXTRACT(ISOYEAR FROM datum) = '2012' AND kunde_id NOT IN (1, 2, 3)
88  ;
89  SELECT DISTINCT kunde_id
90    FROM rechnung
91    WHERE
92      EXTRACT(ISOYEAR FROM datum) = '2012' AND kunde_id NOT IN
93      (
94        SELECT kunde_id FROM rechnung WHERE EXTRACT(ISOYEAR FROM datum) = '2011'
95      )
96  ;
97  \echo 'Ende listing11.sql /////////////////////////////////'

```

Listing 29.33 postgresql/listing12.sql

```
1 -- Ausführen der vorherigen Befehle
```

```
2 \include listing11.sql
3
4 \echo 'Start listing12.sql //////////////////////////////'
5
6 SELECT kunde_id, nachname, vorname, art FROM kunde;
7 SELECT 'Start Aufbau einer temporaeren art-Tabelle (Seite 262)' Hinweis;
8
9 DROP TABLE IF EXISTS tmp_art;
10 CREATE TEMPORARY TABLE tmp_art
11 (
12     art kundenart,
13     k_art VARCHAR(255)
14 )
15 ;
16
17 INSERT INTO tmp_art
18     VALUES
19         ('prv', 'Privatkunde')
20         ,('gsch', 'Geschäftskunde')
21         ,('unb', 'Unbekannt')
22 ;
23
24 SELECT kunde_id, nachname, vorname, k_art
25     FROM
26     kunde INNER JOIN tmp_art USING(art)
27 ;
28 SELECT 'Ende Aufbau einer temporaeren art-Tabelle' Hinweis;
29
30 SELECT 'Lösung mit UNIONs (Seite 262)' Hinweis;
31 SELECT kunde_id, nachname, vorname, k_art
32     FROM
33     kunde INNER JOIN
34     (
35         SELECT 'prv' AS art, 'Privatkunde' AS k_art
36         UNION
37         SELECT 'gsch' AS art, 'Geschäftskunde' AS k_art
38         UNION
39         SELECT 'unb' AS art, 'Unbekannt' AS k_art
40     ) t
41     ON kunde.art::text = t.art
42 ;
43
44 SELECT 'Nö (Seite 263)' Hinweis;
45 SELECT kunde_id, nachname, vorname,
46     CASE art
47         WHEN 'prv' THEN 'Privatkunde'
48         WHEN 'gsch' THEN 'Geschäftskunde'
49         ELSE 'Unbekannt'
50     END AS k_art
51     FROM kunde
52 ;
53
54 -- Start Einfacher CASE
55 SELECT 'Einfacher CASE (Seite 264)' Hinweis;
56 SELECT artikel_id, bezeichnung,
57     CASE waehrung
58         WHEN 'EUR' THEN ROUND(einzelpreis, 2) || ' €'
59         WHEN 'USD' THEN ROUND(einzelpreis, 2) || ' $'
60         ELSE '?????'
```

```

61  END AS preis
62  FROM artikel
63 ;
64 UPDATE artikel SET waehrung = 'XYZ' WHERE artikel_id = 3001;
65 SELECT artikel_id, bezeichnung,
66 CASE waehrung
67   WHEN 'EUR' THEN ROUND(einzelpreis, 2) || ' €'
68   WHEN 'USD' THEN ROUND(einzelpreis, 2) || ' $'
69 END AS preis
70 FROM artikel
71 ;
72 UPDATE artikel SET waehrung = 'EUR' WHERE artikel_id = 3001;
73 -- Ende Einfacher CASE
74
75 -- Start Searched CASE
76 SELECT 'Searched CASE (Seite 265)' Hinweis;
77 SELECT beitrag_id,
78 CASE
79   WHEN bezug_beitrag_id > 1 THEN 'Antwort'
80   WHEN bezug_beitrag_id <= 1 THEN 'Keine Antwort'
81   ELSE '??????'
82 END AS Typ
83 FROM beitrag
84 ORDER BY beitrag_id
85 ;
86 SELECT beitrag_id,
87 CASE
88   WHEN bezug_beitrag_id > 1
89     THEN nachricht || ' Antwort auf ' || bezug_beitrag_id || ': >' ||
90   (
91     SELECT nachricht
92       FROM beitrag b2
93      WHERE b2.beitrag_id = b1.bezug_beitrag_id
94   )
95   WHEN bezug_beitrag_id <= 1 THEN nachricht
96   ELSE '??????'
97 END AS Inhalt
98 FROM beitrag b1
99 WHERE beitrag_id > 1
100 ;
101 SELECT beitrag_id,
102 CASE
103   WHEN bezug_beitrag_id >= 0 THEN 'Antwort'
104   WHEN bezug_beitrag_id > 1 THEN 'Keine Antwort'
105   ELSE '??????'
106 END AS Typ
107 FROM beitrag
108 ORDER BY beitrag_id
109 ;
110 -- Ende Searched CASE
111
112 -- Start Fallbeispiele
113 SELECT 'Fallbeispiel Lagerbestand (Seite 267)' Hinweis;
114 SELECT * FROM lagerbestand;
115 SELECT a.bezeichnung,
116 CASE
117   WHEN l.menge_aktuell <= l.menge_mindest
118     THEN 'Artikel nachbestellen'
119   ELSE 'Bestand ausreichend'

```

```

120  END AS lagerstand
121  FROM lagerbestand l INNER JOIN artikel a USING(artikel_id)
122  ORDER BY l.menge_aktuell / l.menge_mindest
123 ;
124
125 SELECT 'Fallbeispiel Kundengruppen (Seite 268)' Hinweis;
126 SELECT DISTINCT k.kunde_id, k.nachname, k.vorname,
127 CASE
128 WHEN 3 <
129 (
130   SELECT COUNT(rechnung_id) FROM rechnung r
131   WHERE r.kunde_id = k.kunde_id
132 ) THEN 'Prämiumkunde'
133 WHEN 2 <
134 (
135   SELECT COUNT(rechnung_id) FROM rechnung r
136   WHERE r.kunde_id = k.kunde_id
137 ) THEN 'Guter Kunde'
138 ELSE 'Kleinkunde'
139 END kundenart
140 FROM
141   rechnung r RIGHT OUTER JOIN kunde k USING(kunde_id)
142 ;
143 SELECT k.kunde_id, k.nachname, k.vorname,
144 (
145   SELECT SUM(a.einzelpreis * rp.menge)
146   FROM rechnung r INNER JOIN rechnung_position rp
147     USING (rechnung_id)
148     INNER JOIN artikel a
149       USING (artikel_id)
150   WHERE r.kunde_id = k.kunde_id
151 ) rechnungswert,
152 CASE
153 WHEN
154   kunde_id NOT IN (SELECT kunde_id FROM rechnung)
155   THEN 'Kleinkunde'
156 WHEN
157 (
158   SELECT SUM(a.einzelpreis * rp.menge) rechnungswert
159   FROM rechnung r INNER JOIN rechnung_position rp
160     USING (rechnung_id)
161     INNER JOIN artikel a
162       USING (artikel_id)
163   WHERE r.kunde_id = k.kunde_id
164 ) BETWEEN 0 AND 300 THEN 'Kleinkunde'
165 WHEN
166 (
167   SELECT SUM(a.einzelpreis * rp.menge) rechnungswert
168   FROM rechnung r INNER JOIN rechnung_position rp
169     USING (rechnung_id)
170     INNER JOIN artikel a
171       USING (artikel_id)
172   WHERE r.kunde_id = k.kunde_id
173 ) BETWEEN 300 AND 1000 THEN 'Guter Kunde'
174 ELSE 'Prämiumkunde'
175 END AS kundenart
176 FROM kunde k
177 ORDER BY
178 (

```

```

179    SELECT SUM(a.einzelpreis * rp.menge) rechnungswert
180      FROM rechnung r INNER JOIN rechnung_position rp
181          USING (rechnung_id)
182          INNER JOIN artikel a
183              USING (artikel_id)
184      WHERE r.kunde_id = k.kunde_id
185  ) DESC
186 ;
187 DROP TABLE IF EXISTS tmp_umsatz;
188 CREATE TEMPORARY TABLE tmp_umsatz AS
189     SELECT kunde_id, SUM(a.einzelpreis * rp.menge) rechnungswert
190      FROM rechnung r INNER JOIN rechnung_position rp USING (rechnung_id)
191          INNER JOIN artikel a USING (artikel_id)
192  GROUP BY kunde_id
193 ;
194 SELECT k.kunde_id, k.nachname, k.vorname, tmp.rechnungswert,
195 CASE
196 WHEN
197     tmp.kunde_id IS NULL THEN 'Kleinkunde'
198 WHEN
199     rechnungswert BETWEEN 0 AND 300 THEN 'Kleinkunde'
200 WHEN
201     rechnungswert BETWEEN 300 AND 1000 THEN 'Guter Kunde'
202 ELSE 'Premiumkunde'
203 END AS kundenart
204 FROM kunde k LEFT JOIN tmp_umsatz tmp USING(kunde_id)
205 ORDER BY rechnungswert DESC
206 ;
207
208 WITH tmp_umsatz (kunde_id, rechnungswert)
209 AS (
210     SELECT kunde_id, SUM(a.einzelpreis * rp.menge) rechnungswert
211       FROM rechnung r INNER JOIN rechnung_position rp USING (rechnung_id)
212           INNER JOIN artikel a USING (artikel_id)
213  GROUP BY kunde_id
214 )
215 SELECT k.kunde_id, k.nachname, k.vorname, tmp.rechnungswert,
216 CASE
217 WHEN
218     tmp.kunde_id IS NULL THEN 'Kleinkunde'
219 WHEN
220     rechnungswert BETWEEN 0 AND 300 THEN 'Kleinkunde'
221 WHEN
222     rechnungswert BETWEEN 300 AND 1000 THEN 'Guter Kunde'
223 ELSE 'Premiumkunde'
224 END AS kundenart
225 FROM kunde k LEFT JOIN tmp_umsatz tmp USING(kunde_id)
226 ORDER BY rechnungswert DESC
227 ;
228
229 SELECT 'Fallbeispiel Lieferanten (Seite 271)' Hinweis;
230 SELECT l.firmenname,
231 CASE
232 WHEN
233     l.lieferant_id IN
234     (
235         SELECT lieferant_id FROM artikel_nm_lieferant
236     ) THEN 'aktiv'
237 ELSE 'inaktiv'

```

```

238 END AS status
239 FROM lieferant l
240 ORDER BY firmenname
241 ;
242
243 SELECT a.bezeichnung,
244 CASE
245 WHEN l.menge_aktuell <= l.menge_mindest THEN 'Artikel sofort nachbestellen
246 ,
247 WHEN l.menge_aktuell <= l.menge_mindest + 0.3 * l.menge_mindest THEN '
248     Artikel bald nachbestellen'
249 ELSE 'Bestand ausreichend' END AS lagerstand FROM lagerbestand l INNER
250 JOIN artikel a USING(artikel_id)
251 ORDER BY l.menge_aktuell / l.menge_mindest
252 ;
253 -- Ende Fallbeispiele
254 \echo 'Ende listing12.sql //////////////////////////////////////////////////////////////////'

```

Listing 29.34 postgresql/listing13.sql

```

1 -- Ausführen der vorherigen Befehle
2 \include listing12.sql
3
4 \echo 'Start listing13.sql //////////////////////////////////////////////////////////////////'
5
6 SELECT 'Meine erste Ansicht (Seite 274)' Hinweis;
7 CREATE VIEW
8 view_kundenrechnungsadresse (id, nachname, vorname, strasse, ort)
9 AS
10 SELECT
11     kunde_id, nachname, vorname,
12     CONCAT_WS(' ', strasse, hnr),
13     CONCAT_WS(' ', lkz, plz, ort)
14     FROM
15         kunde INNER JOIN adresse
16             ON rechnung_adresse_id = adresse_id
17     WHERE kunde.deleted = 0
18 ;
19 SELECT * FROM view_kundenrechnungsadresse;
20 \dv
21
22 SELECT 'VIEW-Verarbeitung (Seite 277)' Hinweis;
23 CREATE VIEW view_artikel_aktiv
24 AS
25     SELECT * FROM artikel
26     WHERE deleted = 0;
27 ;
28 SELECT FLOOR(artikel_id / 1000) gruppe, AVG(einzelpreis)
29     FROM view_artikel_aktiv
30     GROUP BY gruppe
31 ;
32 EXPLAIN SELECT FLOOR(artikel_id / 1000) gruppe, AVG(einzelpreis)
33     FROM view_artikel_aktiv
34     GROUP BY gruppe
35 ;
36 SELECT artikel_id, bezeichnung FROM
37 view_artikel_aktiv
38 WHERE
39     artikel_id BETWEEN 7000 AND 9000

```

```
40 ;
41 EXPLAIN SELECT artikel_id, bezeichnung FROM
42 view_artikel_aktiv
43 WHERE
44 artikel_id BETWEEN 7000 AND 9000
45 ;
46 CREATE VIEW view_artikel_aktiv_tmp
47 AS
48 SELECT * FROM artikel
49 WHERE deleted = 0
50 ;
51 SELECT artikel_id, bezeichnung
52 FROM view_artikel_aktiv_tmp
53 WHERE artikel_id BETWEEN 7000 AND 8000
54 ;
55 SELECT artikel_id, bezeichnung
56 FROM view_artikel_aktiv_tmp
57 WHERE artikel_id BETWEEN 3000 AND 4000
58 ;
59 EXPLAIN SELECT artikel_id, bezeichnung
60 FROM view_artikel_aktiv_tmp
61 WHERE artikel_id BETWEEN 3000 AND 4000
62 ;
63
64 -- Start DROP VIEW
65 SELECT 'Ansicht löschen (Seite 279)' Hinweis;
66 \dv
67 DROP VIEW view_artikel_aktiv_tmp;
68 \dv
69
70 DROP DATABASE IF EXISTS tmp1;
71 CREATE DATABASE tmp1;
72 \connect tmp1
73 CREATE TABLE a (i INT);
74 CREATE TABLE b (i INT);
75 CREATE VIEW ab
76 AS
77   SELECT * FROM a INNER JOIN b USING(i)
78 ;
79 \dt
80 -- \dv
81 -- DROP TABLE a;
82 \dv
83 DROP TABLE a CASCADE;
84 \dv
85 \connect oshop
86 DROP DATABASE tmp1;
87
88 -- Ende DROP VIEW
89
90 SELECT 'Anwendungsgebiet: Vereinfachung (Seite 283)' Hinweis;
91 CREATE OR REPLACE
92 VIEW view_artikel_aktiv
93 AS
94   SELECT * FROM artikel
95   WHERE deleted = 0
96 ;
97 CREATE OR REPLACE VIEW view_artikel_verfuegbar
98 AS
```

```
99  SELECT artikel_id, bezeichnung
100   FROM artikel INNER JOIN lagerbestand USING(artikel_id)
101  WHERE menge_aktuell >= menge_mindest
102  ORDER BY artikel_id
103 ;
104 SELECT * FROM view_artikel_verfuegbar;
105 CREATE OR REPLACE
106 VIEW view_lagerbestand_aktiv
107   AS
108   SELECT * FROM lagerbestand
109  WHERE deleted = 0
110 ;
111 CREATE OR REPLACE VIEW view_lagerbestand_artikelbezeichnung
112 AS
113   SELECT l.*, a.bezeichnung
114     FROM
115       view_lagerbestand_aktiv l INNER JOIN view_artikel_aktiv a
116      USING(artikel_id)
117 ;
118 SELECT * FROM view_lagerbestand_artikelbezeichnung;
119
120 -- SELECT 'Grenzen der VIEW. Erwarte Fehlermeldungen (Seite 286)';
121 DROP TABLE IF EXISTS tmp;
122 CREATE TEMPORARY TABLE tmp AS SELECT * FROM artikel;
123 CREATE OR REPLACE VIEW view_tmp AS SELECT artikel_id FROM tmp;
124 DROP VIEW IF EXISTS view_tmp;
125 CREATE OR REPLACE VIEW view_tmp AS SELECT nachname FROM kunde ORDER BY
126   nachname DESC;
127 SELECT * FROM view_tmp;
128 SELECT * FROM view_tmp ORDER BY nachname;
129 CREATE OR REPLACE VIEW view_tmp_eins
130   AS
131   SELECT nachname FROM kunde
132 ;
133 CREATE OR REPLACE VIEW view_tmp_zwei
134   AS
135   SELECT rechnung_id, COUNT(*) anzahl
136     FROM rechnung_position
137    GROUP BY rechnung_id
138 ;
139 UPDATE view_tmp_eins
140   SET nachname = 'Streicher'
141  WHERE nachname = 'Telcontar'
142 ;
143 -- SELECT 'Grenzen der VIEW. Erwarte Fehlermeldungen';
144 -- UPDATE view_tmp_zwei
145 -- SET anzahl = 2
146 -- WHERE anzahl > 2
147 -- ;
148 UPDATE view_tmp_eins
149   SET nachname = 'Telcontar'
150  WHERE nachname = 'Streicher'
151 ;
152 -- UPDATE view_tmp_zwei SET rechnung_id = 1 WHERE rechnung_id > 1;
153 INSERT INTO view_tmp_eins VALUES ('Wurst'), ('Brot');
154 SELECT * FROM view_tmp_eins;
155 DELETE FROM view_tmp_eins WHERE nachname IN ('Wurst', 'Brot');
156
```

```
157 \echo 'Ende listing13.sql ///////////////////////////////'
```

29.2.4 Quelltexte zu Teil V

Listing 29.35 postgresql/listing14.sql

```
1 -- Ausführen der vorherigen Befehle
2 \include listing13.sql
3 \echo 'Start listing14.sql ///////////////////////////////'
4 SELECT 'Lagerbestand (Seite 315)' Hinweis;
5 UPDATE lagerbestand
6   SET menge_aktuell = menge_aktuell - 5
7 WHERE artikel_id = 9015;
8
9 UPDATE lagerbestand
10  SET menge_aktuell = menge_aktuell - 3
11 WHERE artikel_id = 9015;
12
13 SELECT * FROM lagerbestand WHERE artikel_id = 9015;
14
15 -- Start Lock
16 BEGIN WORK;      -- Lock innerhalb einer Transaktion
17 SELECT 'LOCK (Seite 318)' Hinweis;
18 LOCK TABLE artikel IN ACCESS EXCLUSIVE MODE;
19 SELECT COUNT(*) FROM artikel;
20 -- Ein SELECT COUNT(*) FROM warengruppe; von einer anderen Transaktion aus würde eine FM provozieren.
21 COMMIT WORK;
22 -- Ende Lock
23 \echo 'Ende listing14.sql ///////////////////////////////'
```

Listing 29.36 postgresql/listing15.sql

```
1 -- Ausführen der vorherigen Befehle
2 \i listing14.sql
3 \c oshop
4 \echo 'Start listing15.sql ///////////////////////////////'
5
6 SELECT 'Das Problem (Seite 320)' Hinweis;
7 INSERT INTO artikel
8   (bezeichnung, einzelpreis, waehrung)
9 VALUES
10  ('Säge', 17.85, 'EUR')
11 ;
12 INSERT INTO artikel_nm_warengruppe
13 VALUES
14  (3, CURRVAL('artikel_artikel_id_seq')), 
15  (4, CURRVAL('artikel_artikel_id_seq'))
16 ;
17
18 UPDATE lagerbestand
19   SET menge_aktuell = menge_aktuell - 1
20 WHERE artikel_id = 9015
21 ;
22 UPDATE bestellung SET status = 'versendet' WHERE bestellung_id = 1;
23 SELECT 'Isolationsebenen (Seite 324)' Hinweis;
```

```
24 -- AUTOCOMMIT wird in PostgreSQL nicht länger unterstützt.  
25  
26 -- Start read uncommitted  
27 SELECT 'READ UNCOMMITTED (Seite 325)' Hinweis;  
28 -- Session 1  
29 START TRANSACTION ISOLATION  
30   LEVEL REPEATABLE READ;  
31 SELECT artikel_id, menge_aktuell  
32   FROM lagerbestand  
33  WHERE artikel_id = 9015  
34 ;  
35 UPDATE lagerbestand  
36   SET menge_aktuell = menge_aktuell - 1  
37  WHERE artikel_id = 9015  
38 ;  
39  
40 SELECT artikel_id, menge_aktuell  
41   FROM lagerbestand  
42  WHERE artikel_id = 9015  
43 ;  
44 ROLLBACK;  
45 SELECT artikel_id, menge_aktuell  
46   FROM lagerbestand  
47  WHERE artikel_id = 9015  
48 ;  
49 -- Session 2  
50 START TRANSACTION  
51   ISOLATION LEVEL READ UNCOMMITTED;  
52  
53  
54 SELECT artikel_id, menge_aktuell  
55   FROM lagerbestand  
56  WHERE artikel_id = 9015  
57 ;  
58 SELECT artikel_id, menge_aktuell  
59   FROM lagerbestand  
60  WHERE artikel_id = 9015  
61 ;  
62 SELECT artikel_id, menge_aktuell  
63   FROM lagerbestand  
64  WHERE artikel_id = 9015  
65 ;  
66 -- Ende read uncommitted  
67  
68 -- Start read committed  
69 SELECT 'READ COMMITTED (Seite 326)' Hinweis;  
70 -- Session 1  
71 SELECT artikel_id, menge_aktuell  
72   FROM lagerbestand  
73  WHERE artikel_id = 9015  
74 ;  
75 UPDATE lagerbestand  
76   SET menge_aktuell = menge_aktuell - 1  
77  WHERE artikel_id = 9015  
78 ;  
79 SELECT artikel_id, menge_aktuell  
80   FROM lagerbestand  
81  WHERE artikel_id = 9015  
82 ;
```

```
83 COMMIT;
84 SELECT artikel_id, menge_aktuell
85   FROM lagerbestand
86  WHERE artikel_id = 9015;
87 -- Session 2
88 SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
89 SELECT artikel_id, menge_aktuell
90   FROM lagerbestand
91  WHERE artikel_id = 9015
92 ;
93 SELECT artikel_id, menge_aktuell
94   FROM lagerbestand
95  WHERE artikel_id = 9015
96 ;
97 SELECT artikel_id, menge_aktuell
98   FROM lagerbestand
99  WHERE artikel_id = 9015
100 ;
101 -- Ende read committed
102
103 -- Start repeatable read
104 SELECT 'REPEATABLE READ (Seite 327)' Hinweis;
105 -- Session 1
106 SELECT artikel_id, menge_aktuell
107   FROM lagerbestand
108  WHERE artikel_id = 9015
109 ;
110 UPDATE lagerbestand
111   SET menge_aktuell = menge_aktuell - 1
112  WHERE artikel_id = 9015
113 ;
114 COMMIT;
115 UPDATE lagerbestand
116   SET menge_aktuell = menge_aktuell - 1
117  WHERE artikel_id = 9015
118 ;
119 COMMIT;
120 SELECT artikel_id, menge_aktuell
121   FROM lagerbestand
122  WHERE artikel_id = 9015
123 ;
124 -- Session 2
125 SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
126 START TRANSACTION;
127 SELECT artikel_id, menge_aktuell
128   FROM lagerbestand
129  WHERE artikel_id = 9015
130 ;
131 SELECT artikel_id, menge_aktuell
132   FROM lagerbestand
133  WHERE artikel_id = 9015
134 ;
135 COMMIT;
136 SELECT artikel_id, menge_aktuell
137   FROM lagerbestand
138  WHERE artikel_id = 9015
139 ;
140 -- Ende repeatable read
141
```

```
142 -- Start Serializable
143 SELECT 'SERIALIZABLE (Seite 328)' Hinweis;
144 -- Session 1
145 SELECT COUNT(*) FROM artikel;
146 INSERT INTO artikel VALUES (1, 'X', 0.0, 'EUR', 0);
147 COMMIT;
148
149 -- Session 2
150 SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
151 DELETE FROM artikel WHERE artikel_id = 1;
152 DELETE FROM artikel WHERE artikel_id = 1;
153 SELECT COUNT(*) FROM artikel;
154 -- Ende Serializable
155 SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
156 \echo 'Ende listing15.sql ///////////////////////////////'
```

■ 29.3 Microsoft SQL Server

29.3.1 Quelltexte zu Teil II

Listing 29.37 mssql/listing01.sql

```
1 -- Warum formatiere ich den Quelltext so ganz anders, als beispielsweise diese
   -- Beautyfier?
2 -- * http://www.dpriver.com/pp/sqlformat.htm
3 -- * https://codebeautify.org/sqlformatter
4 -- Ich finde die Arbeit dieser Programme sehr gut und auch die Ergebnisse sind
   -- ansprechend,
5 -- aber sie widersprechen meinem Anspruch an die leichte Verständlichkeit der
   -- Befehle.
6 -- Zwei Beispiele:
7 -- * Es werden keine Einrückungen eingefügt, die den inhaltlichen
   -- Zusammenhang
8 -- zwischen dem Befehl und seinen Parametern abbilden:
9 -- SELECT blz,
10 --       bankname
11 -- FROM  bank
12 -- ORDER BY blz DESC
13 -- LIMIT 1;
14 --
15 -- anstelle von
16 --
17 -- SELECT
18 --   blz, bankname
19 --   FROM bank
20 --   ORDER BY blz DESC
21 --   LIMIT 1
22 -- ;
23
24 -- * Hinter dem FROM wird ein Tabellenname gestellt, obwohl die Auswertung aus
   -- dem ganzen JOIN erfolgt:
25 -- SELECT k.kunde_id,
26 --       k.nachname,
27 --       k.vorname,
28 --       b.datum
29 -- FROM bestellung b
30 --   INNER JOIN kunde k USING (kunde_id)
31 -- ORDER BY k.nachname,
32 --           k.vorname;
33
34 -- anstelle von
35
36 -- SELECT
37 --   k.kunde_id, k.nachname, k.vorname, b.datum
38 --   FROM
39 --     bestellung b INNER JOIN kunde k USING (kunde_id)
40 --   ORDER BY
41 --     k.nachname, k.vorname
42 -- ;
43
44 -- Ich werde daher meine Darstellung beibehalten.
45
46
```

```

47 -- Anlegen der Datenbank auf der grünen Wiese
48 SELECT 'Start listing01.sql /////////////////////////////////';
49 SELECT 'Löschen einer ggf. vorhandenen Datenbank oshop (' Seite 68 ')';
50 DROP DATABASE IF EXISTS oshop;
51 GO
52
53 SELECT 'Ermitteln aller Collations und damit auch alle Zeichenkodierungen (' Seite 69 ')';
54 SELECT name FROM fn_helpcollations();
55 SELECT name FROM sys.fn_helpcollations() WHERE name LIKE '%latin%';
56 SELECT name FROM sys.fn_helpcollations() WHERE name LIKE '%german%';
57 GO
58
59 SELECT 'Anlegen der Datenbank mit Zeichensatz und Sortierung (' Seite 70 ')';
60 -- Phonebook: Sortierung nach DIN 5007-2
61 -- CI: case insensitive, ohne Berücksichtigung der Groß- und Kleinschreibung
62 -- AI: accent insensitive, ohne Berücksichtigung von Akzenten
63 -- (KS): kanatype sensitive: Es werden verschiedene japanische Schriftzeichen berücksichtigt.
64 -- (WS): width sensitive: Die Kodierungslänge wird berücksichtigt
65 CREATE DATABASE oshop
66   COLLATE German_PhoneBook_CI_AI
67 ;
68 GO
69
70 SELECT 'Anzeigen aller Datenbanken (' Seite 72 ')';
71 SELECT
72   LEFT(name, 10) AS dbname, collation_name
73   FROM sys.databases
74 ;
75 GO
76
77 SELECT 'Ende listing01.sql /////////////////////////////////';
78 GO

```

Listing 29.38 mssql/listing02.sql

```

1 -- Ausführen der vorherigen Befehle
2 :r listing01.sql
3
4 -- Hinweise: T-SQL kennt nicht:
5 -- UNSIGNED
6 -- ENUM
7 SELECT 'Start listing02.sql /////////////////////////////////';
8 GO
9 -- Erspare mir ein wenig Tipparbeit
10 USE oshop; -- Seite 75
11 GO
12 -- adresse
13 SELECT 'Tabelle adresse anlegen (' Seite 82 ')';
14 GO
15 CREATE TABLE adresse (
16   adresse_id          INT IDENTITY (1,1),
17   strasse              NVARCHAR(255) NOT NULL DEFAULT '',
18   hnr                  NVARCHAR(255) NOT NULL DEFAULT '',
19   lkz                  NCHAR(2) NOT NULL DEFAULT '',
20   plz                  NCHAR(9) NOT NULL DEFAULT '',
21   ort                  NVARCHAR(255) NOT NULL DEFAULT ''

```

```

22    deleted           TINYINT NOT NULL DEFAULT 0,
23    PRIMARY KEY(adresse_id)
24 );
25
26 -- kunde
27 SELECT 'Tabelle kunde mit Fremdschluessel anlegen (Seite 87)';
28 GO
29 CREATE TABLE kunde (
30   kunde_id          INT IDENTITY (1,1),
31   nachname          NVARCHAR(255) NOT NULL DEFAULT '',
32   vorname           NVARCHAR(255) NOT NULL DEFAULT '',
33   rechnung_adresse_id  INT,
34   liefer_adresse_id  INT,
35   bezahlart          INT NOT NULL DEFAULT 0,
36   art                INT NOT NULL DEFAULT 0,
37   deleted            TINYINT NOT NULL DEFAULT 0,
38   PRIMARY KEY(kunde_id),
39   FOREIGN KEY (rechnung_adresse_id)
40     REFERENCES adresse(adresse_id)
41     ON UPDATE CASCADE
42     ON DELETE NO ACTION,
43   FOREIGN KEY (liefer_adresse_id)
44     REFERENCES adresse(adresse_id)
45     ON UPDATE NO ACTION
46     ON DELETE NO ACTION
47 );
48 GO
49
50 -- bank
51 SELECT 'Tabelle bank anlegen (Seite 76)';
52 GO
53
54 CREATE TABLE bank (
55   bank_id           NCHAR(12),
56   bankname          NVARCHAR(255) NOT NULL DEFAULT '',
57   lkz               NCHAR(2) NOT NULL DEFAULT 'DE',
58   deleted            TINYINT NOT NULL DEFAULT 0,
59   PRIMARY KEY(bank_id)
60 );
61 GO
62
63 -- bankverbindung
64 SELECT 'Tabelle bankverbindung mit Fremdschluessel anlegen (Seite 76)';
65 CREATE TABLE bankverbindung (
66   kunde_id          INT,
67   bankverbindung_nr  INT,
68   bank_id            NCHAR(12) NOT NULL,
69   kontonummer        NCHAR(25) NOT NULL DEFAULT '',
70   iban               NCHAR(34) NOT NULL DEFAULT '',
71   deleted            TINYINT NOT NULL DEFAULT 0,
72   PRIMARY KEY(kunde_id,bankverbindung_nr),
73   FOREIGN KEY (kunde_id)
74     REFERENCES kunde(kunde_id)
75     ON UPDATE NO ACTION
76     ON DELETE NO ACTION,
77   FOREIGN KEY (bank_id)
78     REFERENCES bank(bank_id)
79     ON UPDATE NO ACTION
80     ON DELETE NO ACTION

```

```
81 );
82 GO
83
84 -- artikel
85 SELECT 'Tabelle artikel anlegen';
86 CREATE TABLE artikel (
87     artikel_id          INT IDENTITY (1,1),
88     bezeichnung         NVARCHAR(255) NOT NULL DEFAULT '',
89     einzelpreis        DECIMAL(14,6) NOT NULL DEFAULT 0.0, -- Wegen des SQL-
90                         Kompatibilitaet wird nicht MONEY verwendet.
91     waehrung           NCHAR(3) DEFAULT 'EUR',
92     deleted            TINYINT NOT NULL DEFAULT 0,
93     PRIMARY KEY(artikel_id)
94 );
95 GO
96
97 -- warengruppe
98 SELECT 'Tabelle warengruppe anlegen';
99 CREATE TABLE warengruppe (
100    warengruppe_id      INT IDENTITY (1,1),
101   bezeichnung         NVARCHAR(255) NOT NULL DEFAULT '',
102   deleted            TINYINT NOT NULL DEFAULT 0,
103  PRIMARY KEY(warengruppe_id)
104 );
105 GO
106
107 -- lieferant
108 SELECT 'Tabelle lieferant mit Fremdschlüssel anlegen';
109 CREATE TABLE lieferant (
110     lieferant_id        INT IDENTITY (1,1),
111     firmenname          NVARCHAR(255) NOT NULL DEFAULT '',
112     adresse_id          INT NOT NULL DEFAULT 0,
113     deleted             TINYINT NOT NULL DEFAULT 0,
114     PRIMARY KEY(lieferant_id),
115     FOREIGN KEY (adresse_id)
116         REFERENCES adresse(adresse_id)
117         ON UPDATE NO ACTION
118         ON DELETE NO ACTION
119 );
120 GO
121
122 -- Hilfstabelle artikel zu warengruppe
123 SELECT 'Hilfstabelle artikel_nm_warengruppe anlegen';
124 CREATE TABLE artikel_nm_warengruppe (
125     warengruppe_id      INT,
126     artikel_id          INT,
127     PRIMARY KEY(warengruppe_id, artikel_id),
128     FOREIGN KEY (warengruppe_id)
129         REFERENCES warengruppe(warengruppe_id)
130         ON UPDATE NO ACTION
131         ON DELETE NO ACTION,
132     FOREIGN KEY (artikel_id)
133         REFERENCES artikel(artikel_id)
134         ON UPDATE NO ACTION
135         ON DELETE NO ACTION
136 );
137 GO
138 -- Hilfstabelle artikel zu lieferant
```

```
139  SELECT 'Hilfstabelle artikel_nm_lieferant anlegen';
140  CREATE TABLE artikel_nm_lieferant (
141      lieferant_id      INT,
142      artikel_id        INT,
143      PRIMARY KEY(lieferant_id, artikel_id),
144      FOREIGN KEY (lieferant_id)
145          REFERENCES lieferant(lieferant_id)
146          ON UPDATE NO ACTION
147          ON DELETE NO ACTION,
148      FOREIGN KEY (artikel_id)
149          REFERENCES artikel(artikel_id)
150          ON UPDATE NO ACTION
151          ON DELETE NO ACTION
152  );
153  GO
154
155  -- bestellung
156  SELECT 'Tabelle bestellung mit Fremdschlüssel anlegen';
157  CREATE TABLE bestellung (
158      bestellung_id      INT IDENTITY (1,1),
159      kunde_id           INT NOT NULL DEFAULT 0,
160      adresse_id          INT NOT NULL DEFAULT 0,
161      datum               DATETIME NOT NULL DEFAULT 0,
162      status              INT NOT NULL DEFAULT 0,
163      deleted             TINYINT NOT NULL DEFAULT 0,
164      PRIMARY KEY(bestellung_id),
165      FOREIGN KEY (kunde_id)
166          REFERENCES kunde(kunde_id)
167          ON UPDATE NO ACTION
168          ON DELETE NO ACTION,
169      FOREIGN KEY (adresse_id)
170          REFERENCES adresse(adresse_id)
171          ON UPDATE NO ACTION
172          ON DELETE NO ACTION
173  );
174  GO
175
176  -- position der bestellung
177  SELECT 'Tabelle bestellung_position mit Fremdschlüssel anlegen';
178  CREATE TABLE bestellung_position (
179      bestellung_id      INT,
180      position_nr         INT,
181      artikel_id          INT NOT NULL DEFAULT 0,
182      menge                DECIMAL(14,6) NOT NULL DEFAULT 0.0,
183      deleted              TINYINT NOT NULL DEFAULT 0,
184      PRIMARY KEY(bestellung_id, position_nr),
185      FOREIGN KEY (bestellung_id)
186          REFERENCES bestellung(bestellung_id)
187          ON UPDATE NO ACTION
188          ON DELETE NO ACTION,
189      FOREIGN KEY (artikel_id)
190          REFERENCES artikel(artikel_id)
191          ON UPDATE NO ACTION
192          ON DELETE NO ACTION
193  );
194  GO
195
196  -- rechnung
197  SELECT 'Tabelle rechnung mit Fremdschlüssel anlegen';
```

```

198 CREATE TABLE rechnung (
199     rechnung_id      INT IDENTITY (1,1),
200     kunde_id        INT NOT NULL DEFAULT 0,
201     bestellung_id   INT NOT NULL DEFAULT 0,
202     adresse_id      INT NOT NULL DEFAULT 0,
203     bezahlart       INT NOT NULL DEFAULT 0,
204     datum           DATETIME NOT NULL DEFAULT 0,
205     status          INT NOT NULL DEFAULT 0,
206     rabatt          DECIMAL(6,2) NOT NULL DEFAULT 0.0,
207     skonto          DECIMAL(6,2) NOT NULL DEFAULT 0.0,
208     deleted         TINYINT NOT NULL DEFAULT 0,
209     PRIMARY KEY(rechnung_id),
210     FOREIGN KEY (kunde_id)
211         REFERENCES kunde(kunde_id)
212         ON UPDATE NO ACTION
213         ON DELETE NO ACTION,
214     FOREIGN KEY (adresse_id)
215         REFERENCES adresse(adresse_id)
216         ON UPDATE NO ACTION
217         ON DELETE NO ACTION
218 );
219 GO
220
221 -- position der rechnung
222 SELECT 'Tabelle rechnung_position mit Fremdschlüssel anlegen';
223 CREATE TABLE rechnung_position (
224     rechnung_id      INT,
225     position_nr     INT,
226     artikel_id      INT NOT NULL DEFAULT 0,
227     steuer          DECIMAL(14,6) NOT NULL DEFAULT 19.0,
228     menge           DECIMAL(14,6) NOT NULL DEFAULT 0.0,
229     deleted         TINYINT NOT NULL DEFAULT 0,
230     PRIMARY KEY(rechnung_id, position_nr),
231     FOREIGN KEY (rechnung_id)
232         REFERENCES rechnung(rechnung_id)
233         ON UPDATE NO ACTION
234         ON DELETE NO ACTION,
235     FOREIGN KEY (artikel_id)
236         REFERENCES artikel(artikel_id)
237         ON UPDATE NO ACTION
238         ON DELETE NO ACTION
239 );
240 GO
241
242 -- Zeige mir das Ergebnis
243 -- SHOW TABLES;
244 SELECT LEFT(name, 20) AS 'tablename', crdate FROM oshop.dbo.sysobjects WHERE
245     xtype='U' ORDER BY tablename
246 GO
247 SELECT 'Ende listing02.sql /////////////////////////////////';
248 GO

```

Listing 29.39 mssql/listing37.sql

```

1  -- Ausführen der vorherigen Befehle
2  :r listing02.sql
3
4  SELECT 'Start listing03.sql /////////////////////////////////';

```

```
5 GO
6
7 -- Seite 95
8 SELECT
9     LEFT(name, 30) AS index_name,
10    LEFT(type_desc, 15) As index_type,
11    is_unique,
12    LEFT(OBJECT_NAME(object_id), 30) As table_name
13 FROM
14    sys.indexes
15 WHERE
16    is_hypothetical = 0 AND
17    index_id != 0 AND
18    object_id = OBJECT_ID('kunde');
19 GO
20
21 -- Anlegen der Indizes nach Liste auf Seite 97
22 SELECT 'CREATE INDEX idx_kunde_nachname_vorname';
23 GO
24 CREATE INDEX idx_kunde_nachname_vorname
25     ON kunde (nachname, vorname)
26 ;
27 GO
28
29 SELECT 'CREATE INDEX idx_bank_bankname';
30 GO
31 CREATE INDEX idx_bank_bankname
32     ON bank (bankname)
33 ;
34 GO
35
36 SELECT 'CREATE INDEX idx_bankverbindung_bankid_kontonummer';
37 GO
38 CREATE INDEX idx_bankverbindung_bankid_kontonummer
39     ON bankverbindung (bank_id, kontonummer)
40 ;
41 GO
42
43 SELECT 'CREATE UNIQUE INDEX idx_bankverbindung_iban';
44 GO
45 CREATE UNIQUE INDEX idx_bankverbindung_iban
46     ON bankverbindung (iban)
47 ;
48 GO
49
50 SELECT 'CREATE INDEX idx_artikel_bezeichnung';
51 GO
52 CREATE INDEX idx_artikel_bezeichnung
53     ON artikel (bezeichnung)
54 ;
55 GO
56
57 SELECT 'CREATE INDEX idx_warengruppe_bezeichnung';
58 GO
59 CREATE INDEX idx_warengruppe_bezeichnung
60     ON warengruppe (bezeichnung)
61 ;
62 GO
63
```

```
64 SELECT 'CREATE INDEX idx_lieferant_firmenname';
65 GO
66 CREATE INDEX idx_lieferant_firmenname
67     ON lieferant (firmenname)
68 ;
69 GO
70
71 SELECT 'CREATE INDEX idx_bestellung_kundeid_datum';
72 GO
73 CREATE INDEX idx_bestellung_kundeid_datum
74     ON bestellung (kunde_id, datum)
75 ;
76 GO
77
78
79 SELECT 'CREATE UNIQUE INDEX idx_adresse_dublette (Seite 100)';
80 GO
81 CREATE UNIQUE INDEX idx_adresse_dublette
82     ON adresse (strasse, hnr, plz);
83 GO
84
85 -- Start Selektivitaet
86 SELECT 'Indexselektivitaet (Seite 103)';
87 SELECT 'CREATE TABLE #testdaten_bank';
88 GO
89
90 DROP TABLE IF EXISTS #testdaten_bank;
91 CREATE TABLE #testdaten_bank
92 (
93     blz      CHAR(8),
94     merkmal  CHAR(1),
95     bezeichnung VARCHAR(255),
96     plz      CHAR(5),
97     ort      VARCHAR(255),
98     kurz     VARCHAR(255),
99     pan      CHAR(5),
100    bic      VARCHAR(255),
101    sm       CHAR(2),
102    ds       CHAR(6),
103    kz       CHAR(1),
104    blzl     CHAR(1),
105    blzn     CHAR(8)
106 );
107 GO
108
109 SELECT 'BULK INSERT';
110 GO
111
112 SELECT 'BULK findet unter Linux Pfad nicht: Bug #9380471 for Microsoft-
113     internal reference.'
114 SELECT '      Ausf&uuml;rung unter Windows mit angepasstem Pfad klappt prima.'
115 GO
116 -- Hier Dateiname und Pfad einfuegen
117 -- BULK INSERT #testdaten_bank
118 --     FROM '/home/ [...] /src/mssql/blz_20120305.csv'
119 --     WITH (FIRSTROW = 2, FIELDTERMINATOR = ';', ROWTERMINATOR='0x0a', FORMAT='
120 --         CSV');
121 -- GO
122
```

```

121 SELECT '      Workaround zum Bug';
122 :r testdaten.blz.sql
123 GO
124
125
126 -- Workaround
127 -- TSQL kennt keine Einschänkung der Spaltenlänge
128 CREATE INDEX idx_testdaten_bank_bezeichnung
129     ON #testdaten_bank(bezeichnung)
130 ;
131 GO
132
133 SELECT
134     LEFT(name, 30) AS index_name,
135     LEFT(type_desc, 15) AS index_type,
136     is_unique,
137     LEFT(OBJECT_NAME(object_id), 30) AS table_name
138 FROM
139     sys.indexes
140 WHERE
141     is_hypothetical = 0 AND
142     index_id != 0 AND
143     object_id = OBJECT_ID('#testdaten_bank');
144 GO
145
146 -- Ende Selektivitaet
147
148 SELECT 'Ende listing03.sql /////////////////////////////////';
149 GO

```

Listing 29.40 mssql/listing04.sql

```

1  -- Ausführen der vorherigen Befehle
2  :r listing03.sql
3
4  SELECT 'Start listing04.sql /////////////////////////////////';
5  GO
6
7  SELECT 'BULK INSERT (Seite 108)';
8  GO
9
10 SELECT 'BULK findet unter Linux Pfad nicht: Bug #9380471 for Microsoft-
11     internal reference.'
12 SELECT '      Ausführung unter Windows mit angepasstem Pfad klappt prima.'
13 GO
14 -- Hier Dateiname und Pfad einfügen
15 -- BULK INSERT artikel
16 --     FROM '/home/ [...] /src/mssql/artikel02.csv'
17 --     WITH (FIRSTROW = 2, FORMATFILE = '/home/ [...] /src/mssql/artikel.fmt');
18 -- GO
19 SELECT '      Workaround zum Bug';
20 TRUNCATE TABLE artikel;
21 :r artikel02_insert.sql
22
23 SELECT * FROM artikel;
24 GO
25 -- Start INSERT

```

```
26
27 SELECT 'Anlegen mit INSERT INTO ... VALUES (Seite 113)';
28 SELECT 'Warenguppen anlegen';
29 GO
30 SET IDENTITY_INSERT warengruppe ON ;
31 INSERT INTO warengruppe (warengruppe_id, bezeichnung)
32     VALUES
33         (1, 'Bürobedarf')
34         ,(2, 'Pflanzen')
35         ,(3, 'Gartenbedarf')
36         ,(4, 'Werkzeug')
37 ;
38 SET IDENTITY_INSERT warengruppe OFF ;
39 SELECT * FROM warengruppe;
40 GO
41
42 -- T-SQL kennt die SET-Variante nicht.
43 SELECT 'Anlegen mit INSERT INTO ... SET (Seite 115)';
44 SELECT 'Warenguppen verbinden';
45 GO
46
47 INSERT INTO artikel_nm_warengruppe(artikel_id, warengruppe_id)
48     VALUES
49         (3001,1)
50         ,(3005,1)
51         ,(3006,1)
52         ,(3007,1)
53         ,(3010,1)
54         ,(7856,2)
55         ,(7856,3)
56         ,(7863,2)
57         ,(7863,3)
58         ,(9010,3)
59         ,(9010,4)
60         ,(9015,3)
61         ,(9015,4)
62 ;
63 GO
64 -- Ende INSERT
65
66 -- Start Kundendaten
67 SELECT 'Testdaten INSERT adresse';
68 GO
69 -- adresse
70 SET IDENTITY_INSERT adresse ON ;
71 GO
72 INSERT INTO adresse (adresse_id, strasse, hnr, lkz, plz, ort)
73     VALUES
74         (1, 'Beutelhaldenweg', '5', 'AL', '67676', 'Hobbingen')
75         ,(2, 'Beutelhaldenweg', '1', 'AL', '67676', 'Hobbingen')
76         ,(3, 'Auf der Feste', '1', 'GO', '54786', 'Minas Tirith')
77         ,(4, 'Letztes Haus', '4', 'ER', '87567', 'Bruchtal')
78         ,(5, 'Baradur', '1', 'MO', '62519', 'Lugburz')
79         ,(10, 'Hochstrasse', '4a', 'DE', '44879', 'Bochum')
80         ,(11, 'Industriegebiet', '8', 'DE', '44878', 'Bochum')
81 ;
82 GO
83 SET IDENTITY_INSERT adresse OFF ;
84 GO
```

```
85  SELECT 'Testdaten INSERT kunde';
86  GO
87  -- kunde
88  SET IDENTITY_INSERT kunde ON ;
89  GO
90  INSERT INTO kunde (kunde_id, rechnung_adresse_id, nachname, vorname, art)
91  VALUES
92      (1, 1, 'Gamdschie',    'Samweis', 1)
93      ,(2, 2, 'Beutlin',     'Frodo',   1)
94      ,(3, 2, 'Beutlin',     'Bilbo',   1)
95      ,(4, 3, 'Telcontar',   'Elessar', 1)
96      ,(5, 4, 'Earendilionn','Elrond', 3)
97  ;
98  GO
100
101 SET IDENTITY_INSERT kunde OFF ;
102 GO
103
104 SELECT 'Lieferadresse bei zwei Adressen ändern';
105 GO
106 UPDATE kunde SET liefer_adresse_id = 2 WHERE kunde_id = 1;
107 UPDATE kunde SET liefer_adresse_id = 4 WHERE kunde_id = 3;
108 GO
109 -- Ende Kundendaten
110
111 -- Start Aufbau von 2 Bestellungen
112 SELECT 'Aufbau von 2 Bestellungen: Bestelldaten';
113 GO
114
115 INSERT INTO bestellung (kunde_id, adresse_id, datum)
116 VALUES
117     (1, 1, '2012-03-24 17:41:00')
118     ,(2, 2, '2012-03-23 16:11:00')
119 ;
120 GO
121
122 SELECT 'Aufbau von 2 Bestellungen: Bestellpositionen';
123 GO
124
125 INSERT INTO bestellung_position (bestellung_id, position_nr, artikel_id, menge
126 )
126 VALUES
127     (1, 1, 7856, 30)
128     ,(1, 2, 7863, 50)
129     ,(1, 3, 9015, 1)
130     ,(2, 1, 7856, 10)
131     ,(2, 2, 9010, 5)
132 ;
133 GO
134
135 -- Ende Aufbau Bestellung
136
137 -- SELECT 'Drei Fehlermeldung wegen Constraints (Seite 116)!';
138 SELECT bestellung_id, kunde_id, adresse_id, datum FROM bestellung;
139 GO
140 -- INSERT INTO bestellung (datum) VALUES (NOW());
141 -- INSERT INTO bestellung (adresse_id, datum) VALUES (10, NOW());
142 -- INSERT INTO adresse (adresse_id, strasse) VALUES (10, 'Oberer Weg');
```

```

143
144 -- Bildtabelle
145 SELECT 'Bildtabelle anlegen (Seite 114)';
146 GO
147 CREATE TABLE bild
148 (
149     bild_id          INT IDENTITY (1,1) PRIMARY KEY,
150     bild            VARBINARY(max),
151     dateiname       NVARCHAR(255),
152     dateityp        NVARCHAR(255),
153     dateigroesse    BIGINT,
154     artikel_id      INT REFERENCES artikel(artikel_id)
155 );
156 GO
157
158 SELECT 'Daten kopieren (Seite 121)';
159 GO
160
161 -- T-SQL kennt diese Art, eine Tabelle zu erstellen nicht.
162 -- DROP TABLE IF EXISTS #kunde_beutlin;
163 -- CREATE TABLE #kunde_beutlin LIKE kunde;
164 -- INSERT INTO #kunde_beutlin
165 --     SELECT * FROM kunde WHERE nachname='Beutlin';
166 -- SELECT * FROM #kunde_beutlin;
167 -- GO
168
169 -- Das gleiche Ergebnis kann aber wie folgt erreicht werden.
170 SELECT *
171     INTO #kunde_beutlin
172     FROM kunde
173     WHERE nachname='Beutlin'
174 ;
175 SELECT * FROM #kunde_beutlin;
176 GO
177
178
179 SELECT 'Ende listing04.sql /////////////////////////////////';
180 GO

```

29.3.2 Quelltexte zu Teil III

Listing 29.41 mssql/listing05.sql

```

1  -- Ausführen der vorherigen Befehle
2  :r listing04.sql
3
4  SELECT 'Start listing05.sql /////////////////////////////////';
5  GO
6  SELECT 'Aktuelle Datenbankeinstellung (Seite 126)';
7  GO
8  SELECT
9      LEFT(name, 10) AS dbname, collation_name
10     FROM sys.databases
11 ;
12 GO

```

```
13
14  SELECT 'DATABASE löschen (Seite 127)';
15  GO
16  CREATE DATABASE wurstbrot;
17  SELECT
18    LEFT(name, 10) AS dbname, collation_name
19    FROM sys.databases
20  ;
21  GO
22
23  DROP DATABASE wurstbrot;
24  SELECT
25    LEFT(name, 10) AS dbname, collation_name
26    FROM sys.databases
27  ;
28  GO
29
30  -- Start Tabellen umbenennen
31  SELECT 'Tabellen umbenennen (Seite 130)';
32  USE oshop;
33  GO
34
35  DROP TABLE IF EXISTS wurstbrot;
36  SELECT LEFT(name, 20) AS 'tablename', crdate FROM oshop.dbo.sysobjects WHERE
37    xtype='U' ORDER BY tablename
38  GO
39
40  EXEC sp_rename adresse, wurstbrot;
41  SELECT LEFT(name, 20) AS 'tablename', crdate FROM oshop.dbo.sysobjects WHERE
42    xtype='U' ORDER BY tablename
43  GO
44
45  SELECT
46    ordinal_position,
47    LEFT(column_name, 20) AS 'column_name',
48    LEFT(data_type, 20) AS 'data_type',
49    character_maximum_length,
50    is_nullable
51    FROM information_schema.columns
52    WHERE table_name = 'kunde'
53  ;
54  GO
55
56  EXEC sp_rename wurstbrot, adresse;
57  GO
58  -- Ende Tabellen umbenennen
59
60  -- Start Spalten verändern
61  SELECT 'Spalte hinzufügen (Seite 131)';
62  GO
63  SELECT
64    ordinal_position,
65    LEFT(column_name, 20) AS 'column_name',
66    LEFT(data_type, 20) AS 'data_type',
67    character_maximum_length,
68    is_nullable
69    FROM information_schema.columns
70    WHERE table_name = 'kunde'
71  ;
```

```
70 GO
71 ALTER TABLE kunde
72   ADD firmenname VARCHAR(255) NOT NULL DEFAULT ''
73 ;
74 GO
75 SELECT
76   ordinal_position,
77   LEFT(column_name, 20) AS 'column_name',
78   LEFT(data_type, 20) AS 'data_type',
79   character_maximum_length,
80   is_nullable
81 FROM information_schema.columns
82 WHERE table_name = 'kunde'
83 ;
84 GO
85
86 SELECT 'Spalte Datentyp ändern (Seite 133)';
87 GO
88 SELECT 'Kein enum in TSQL'
89 GO
90 -- ALTER TABLE kunde
91 -- MODIFY
92 --   bezahlart ENUM('rechnung', 'bankeinzug', 'nachname') DEFAULT 'rechnung'
93 -- ;
94
95
96 SELECT 'Spaltenlänge ändern (Seite 133)';
97 GO
98
99 DROP TABLE IF EXISTS #bla ;
100 GO
101 SELECT * INTO #bla FROM artikel;
102 -- ALTER TABLE #bla
103 -- ALTER COLUMN bezeichnung CHAR(10);
104 SELECT MAX(LEN(bezeichnung)) AS 'maxlen' FROM artikel;
105 GO
106
107
108 SELECT 'Datum- und Zeitspalten ändern (Seite 134)';
109 GO
110 -- Die Beispiele lassen sich so nicht in TSQL nachbauen.
111 -- Das Abändern erfolgt über CAST- oder CONVERT-Funktionen
112 -- mit einem gleichzeitigen Neuaufbau der Tabellen.
113
114
115 DROP TABLE IF EXISTS wurstbrot ;
116 GO
117 SELECT * INTO wurstbrot FROM adresse;
118 GO
119 SELECT
120   ordinal_position,
121   LEFT(column_name, 20) AS 'column_name',
122   LEFT(data_type, 20) AS 'data_type',
123   character_maximum_length,
124   is_nullable
125 FROM information_schema.columns
126 WHERE table_name = 'wurstbrot'
127 ;
128 GO
```

```

129
130 SELECT 'Spalten löschen (Seite 137)';
131 ALTER TABLE wurstbrot
132   DROP COLUMN deleted
133 ;
134 ALTER TABLE wurstbrot
135   DROP COLUMN strasse
136 ;
137 ALTER TABLE wurstbrot
138   DROP COLUMN ort
139 ;
140
141 SELECT
142   ordinal_position,
143   LEFT(column_name, 20) AS 'column_name',
144   LEFT(data_type, 20) AS 'data_type',
145   character_maximum_length,
146   is_nullable
147   FROM information_schema.columns
148   WHERE table_name = 'wurstbrot'
149 ;
150 GO
151 SELECT 'Ende listing05.sql /////////////////////////////////';
152 GO

```

Listing 29.42 mssql/listing06.sql

```

1 -- Ausführen der vorherigen Befehle
2 :r listing05.sql
3
4 SELECT 'Start listing06.sql /////////////////////////////////';
5 GO
6 SELECT 'WHERE mit TRUE / FALSE (Seite 142)';
7 SELECT 'TSQL kennt keine boolschen Konstanten';
8 -- SELECT * FROM artikel WHERE TRUE;
9 -- SELECT * FROM artikel WHERE FALSE;
10
11 -- Start Groß- Kleinschreibung
12 SELECT 'Gross- Kleinschreibung (Seite 143)';
13 GO
14 DROP TABLE IF EXISTS #wurstbrot;
15 GO
16 CREATE TABLE #wurstbrot (
17   name NVARCHAR(15)
18 );
19 GO
20
21 INSERT INTO #wurstbrot
22   VALUES
23     ('Jaqueline')
24     ,('Kevin')
25     ,('kevin')
26     ,('jaqueline')
27
28 SELECT * FROM #wurstbrot WHERE name = 'kevin';
29 GO
30
31 DROP TABLE #wurstbrot;

```

```
32 GO
33
34 CREATE TABLE #wurstbrot (
35     name NVARCHAR(15) COLLATE SQL_Latin1_General_CI_AS
36 );
37 GO
38
39 INSERT INTO #wurstbrot
40     VALUES
41         ('Jaqueline')
42     ,('Kevin')
43     ,('kevin')
44     ,('jaqueline')
45
46 SELECT * FROM #wurstbrot WHERE name = 'kevin';
47 GO
48 -- Ende Groß- Kleinschreibung
49
50 -- Start Wertberechnung
51 SELECT 'Wertberechnung (Seite 148)' Hinweis;
52 SELECT * FROM artikel;
53 GO
54 UPDATE artikel SET einzelpreis = einzelpreis + einzelpreis / 100.0;
55 SELECT * FROM artikel;
56 GO
57 UPDATE artikel SET einzelpreis = ROUND(einzelpreis, 2);
58 SELECT * FROM artikel;
59 GO
60 -- Ende Wertberechnung
61
62 -- Start Gebastelte Zeichenketten
63 SELECT 'Gebastelte Zeichenketten: Anrede (Seite 149)' Hinweis;
64 GO
65 ALTER TABLE kunde ADD anrede NVARCHAR(255);
66 GO
67 UPDATE kunde
68     SET anrede = CONCAT('Sehr geehrte/r Frau/Herr ', nachname)
69     WHERE nachname <> ''
70 ;
71 SELECT * FROM kunde;
72 GO
73 -- Ende Gebastelte Zeichenketten
74
75 -- Start DELETE
76 SELECT 'Löschen mit DELETE (Seite 150)';
77 GO
78 SELECT * FROM bestellung_position;
79 GO
80 DELETE FROM bestellung_position WHERE bestellung_id = 1;
81 SELECT * FROM bestellung_position;
82 GO
83
84 -- Start Wiederherstellen der Daten
85 INSERT INTO bestellung_position (bestellung_id, position_nr, artikel_id, menge
86     )
86 VALUES
87     (1, 1, 7856, 30)
88     ,(1, 2, 7863, 50)
89     ,(1, 3, 9015, 1)
```

```

90 ;
91 GO
92 -- Ende Wiederherstellen der Daten
93
94 -- SELECT 'Löschversuch mit Constraint. Erwarte eine Fehlermeldung! ('  

-- Seite 152)';
95 -- DELETE FROM kunde WHERE kunde_id = 1;
96
97 SELECT 'Test IDENTITY(1,1) (Seite 152)';
98 GO
99 SELECT kunde_id, nachname FROM kunde;
100 GO
101 DELETE FROM kunde WHERE kunde_id IN (3,5);
102 GO
103 SELECT kunde_id, nachname FROM kunde;
104 GO
105 INSERT INTO kunde (nachname, vorname) VALUES ('Eichenschild', 'Thorin');
106 SELECT kunde_id, nachname FROM kunde;
107 GO
108 -- Start Wiederherstellen der Daten
109 SET IDENTITY_INSERT kunde ON ;
110 INSERT INTO kunde (kunde_id, rechnung_adresse_id, nachname, vorname, art)
111 VALUES
112     (3, 2, 'Beutlin',      'Bilbo',    1)
113     ,(5, 4, 'Earendilionn', 'Elrond',   3)
114 ;
115 SET IDENTITY_INSERT kunde OFF ;
116 SELECT kunde_id, nachname FROM kunde;
117 GO
118
119 SELECT 'Tabelle leeren (Seite 154)' Hinweis;
120 GO
121 -- DELETE FROM bestellung_position;
122 -- GO
123 -- TRUNCATE bestellung_position;
124 -- GO
125 -- Ende DELETE
126 SELECT 'Ende listing06.sql /////////////////////////////////';
127 GO

```

29.3.3 Quelltexte zu Teil IV

Listing 29.43 mssql/listing07.sql

```

1 -- Ausführen der vorherigen Befehle
2 :r listing06.sql
3
4 SELECT 'Start listing07.sql /////////////////////////////////';
5 GO
6 SELECT 'Ausdrücke (Seite 158)';
7 GO
8
9 SELECT 5, 'Beginn der Auswertung';
10 GO
11 SELECT -5, 9 + 4, 9 - 4, 9 * 4, 9 / 4, 9.0 / 4.0, 9 % 4;

```

```
12 GO
13 SELECT 3 * 4 / 3, (3 * 4) / 3, 3 * (4 / 3);
14 GO
15 SELECT 9 * NULL;
16 GO
17
18 SELECT 'Zufallszahlen (Seite 160)';
19 GO
20 SELECT RAND(1);
21 GO
22 SELECT RAND(1);
23 GO
24 SELECT RAND(CONVERT(VARBINARY, NEWID())) FROM artikel;
25 GO
26
27 SELECT 'Variablen (Seite 161)';
28 GO
29 DECLARE @x INT
30 SELECT POWER(2,8) INTO @x;
31 SELECT @x - 1;
32 GO
33
34 DECLARE @artnr INT
35 SELECT @artnr=3010;
36 SELECT * FROM artikel WHERE artikel_id=@artnr;
37 GO
38
39 SELECT 'Spalten- und Zeilenwahl (Seite 162)';
40 GO
41
42 SELECT * FROM artikel;
43 GO
44
45 SELECT
46 artikel_id, bezeichnung, einzelpreis, waehrung, deleted
47 FROM artikel
48 ;
49 GO
50
51 SELECT
52 artikel_id + 10000 AS Unsinn, deleted AS Löschkennzeichen
53 FROM artikel
54 ;
55 GO
56
57 SELECT
58 artikel_id, bezeichnung
59 FROM artikel
60 WHERE einzelpreis BETWEEN 10 AND 100
61 ;
62 GO
63
64 SELECT 'Sortierung (Seite 163)';
65 GO
66 SELECT
67 artikel_id, bezeichnung, einzelpreis
68 FROM artikel
69 ORDER BY einzelpreis ASC
70 ;
```

```
71 GO
72
73 SELECT
74     nachname, vorname
75     FROM kunde
76     ORDER BY nachname, vorname
77 ;
78 GO
79
80 SELECT
81     nachname, vorname
82     FROM kunde
83     ORDER BY nachname DESC, vorname ASC
84 ;
85 GO
86
87 SELECT 'TSQL liefert in diesem Fall die gleiche Sortierung' Hinweis;
88 GO
89 DROP TABLE IF EXISTS #wurstbrot;
90 CREATE TABLE #wurstbrot ( name VARCHAR(255) COLLATE SQL_AltDiction_CP850_CI_AI
91 );
92 INSERT INTO #wurstbrot
93     VALUES
94         ('winfried')
95         ,('achim')
96         ,('olga')
97         ,('zechine')
98         ,('ägidius')
99 ;
100 SELECT * FROM #wurstbrot ORDER BY name;
101 GO
102
103 DROP TABLE IF EXISTS #wurstbrot;
104 CREATE TABLE #wurstbrot ( name VARCHAR(255) COLLATE
105     SQL_Latin1_General_CP1_CS_AS);
106 INSERT INTO #wurstbrot
107     VALUES
108         ('winfried')
109         ,('achim')
110         ,('olga')
111         ,('zechine')
112         ,('ägidius')
113 ;
114 SELECT * FROM #wurstbrot ORDER BY name;
115 GO
116
117 DROP TABLE IF EXISTS #wurstbrot;
118 CREATE TABLE #wurstbrot (name VARCHAR(255) COLLATE
119     SQL_Latin1_General_CI_AS);
120 INSERT INTO #wurstbrot
121     VALUES
122         ('winfried')
123         ,('achim')
124         ,('olga')
125         ,('Olga')
126         ,('zechine')
127 ;
128 SELECT * FROM #wurstbrot ORDER BY name;
```

```
127 SELECT * FROM #wurstbrot ORDER BY name DESC;
128 GO
129
130 DROP TABLE IF EXISTS #wurstbrot;
131 GO
132 CREATE TABLE #wurstbrot (name VARCHAR(255) COLLATE
133           SQL_Latin1_General_CI_AS);
134 INSERT INTO #wurstbrot
135   VALUES
136     ('winfried')
137     ,('achim')
138     ,('olga')
139     ,('Olga')
140     ,('zechine')
141 ;
142 SELECT * FROM #wurstbrot ORDER BY name;
143 SELECT * FROM #wurstbrot ORDER BY name DESC;
144 GO
145
146 DROP TABLE IF EXISTS #wurstbrot;
147 GO
148
149 SELECT
150   bestellung_id, datum
151   FROM bestellung
152   ORDER BY datum;
153 GO
154
155 SELECT
156   bestellung_id, CONVERT(TIME, datum) Uhrzeit
157   FROM bestellung
158   ORDER BY uhrzeit;
159 GO
160 SET SHOWPLAN_ALL ON
161 GO
162 SELECT * FROM kunde ORDER BY nachname, vorname;
163 GO
164 SET SHOWPLAN_ALL OFF
165 GO
166
167 SELECT 'Mehrfachausgaben unterbinden (Seite 173)';
168 GO
169
170 SELECT ort FROM adresse ORDER BY ort;
171 SELECT DISTINCT ort FROM adresse ORDER BY ort;
172 SELECT DISTINCT nachname FROM kunde ORDER BY nachname;
173 GO
174
175
176 -- Das Fallbeispiel Bankimport müsste in TSQL ganz anders als
177 -- in MySQL oder MariaDB gelöst werden.
178 -- Daher hier nur der INSERT, damit die Bankdaten vorliegen.
179 SELECT 'Bankdaten importieren (Seite 174)';
180 GO
181 ALTER TABLE bank ADD blz NCHAR(12) NOT NULL DEFAULT '';
182 GO
183 :r blz_20120305.sql
184 GO
```

```
185
186 CREATE INDEX idx_bank_blzbankname
187   ON bank (blz, bankname)
188 ;
189 GO
190
191 -- Beginn Für die vorhandenen Kunden eine Bankverbindung bauen
192
193 INSERT INTO bankverbindung (kunde_id, bankverbindung_nr, bank_id, kontonummer,
194   iban)
195 VALUES
196   (1, 1, 2, '1111111111', '100100101111111111'),
197   (1, 2, 2, '1111111112', '100100101111111112'),
198   (2, 1, 35, '2222222221', '100601982222222221'),
199   (3, 1, 35, '3333333331', '100601983333333331'),
200   (4, 1, 90, '4444444441', '120700004444444441'),
201   (5, 1, 90, '5555555551', '120700005555555551')
202 ;
203 GO
204 -- Ende Für die vorhandenen Kunden eine Bankverbindung bauen
205
206 -- Ende Bankimport
207
208 SELECT 'Ausschneiden mit TOP (Seite 177)';
209 GO
210 -- SELECT * FROM bank;
211 SELECT TOP(3) blz, bankname FROM bank;
212 GO
213 SELECT TOP(1)
214   blz, bankname
215   FROM bank
216   ORDER BY blz DESC
217 ;
218 GO
219 DECLARE @blzmin BIGINT;
220 SELECT TOP(1) @blzmin =
221   blz
222   FROM bank
223   ORDER BY blz DESC
224 ;
225 SELECT
226   bankname
227   FROM bank
228   WHERE blz = @blzmin
229 ;
230 GO
231
232 SELECT blz, bankname FROM bank ORDER BY blz OFFSET 1 ROWS FETCH NEXT 2 ROWS
233   ONLY;
234 SELECT TOP(1) blz, bankname FROM bank ORDER BY blz ;
235 SELECT blz, bankname FROM bank ORDER BY blz OFFSET 0 ROWS FETCH NEXT 1 ROWS
236   ONLY;
237 GO
238 SELECT 'Ende listing07.sql /////////////////////////////////';
239 GO
```

Listing 29.44 mssql/listing08.sql

```
1 -- Ausführen der vorherigen Befehle
2 :r listing07.sql
3
4 SELECT 'Start listing08.sql //////////////////////////////';
5 GO
6
7 -- Start INNER JOIN
8 SELECT 'CROSS JOIN (Seite 184)';
9 GO
10 SELECT
11     kunde_id, nachname, vorname, rechnung_adresse_id, adresse_id
12     FROM
13         kunde, adresse
14 ;
15 GO
16
17 SELECT 'CROSS JOIN mit WHERE (Seite ??)';
18 GO
19 SELECT
20     kunde_id, nachname, vorname, rechnung_adresse_id, adresse_id
21     FROM
22         kunde, adresse
23     WHERE
24         rechnung_adresse_id = adresse_id
25 ;
26 GO
27
28 SELECT 'INNER JOIN (Seite 187)';
29 GO
30 SELECT
31     kunde_id, nachname, vorname, rechnung_adresse_id, adresse_id
32     FROM
33         kunde INNER JOIN adresse
34             ON rechnung_adresse_id = adresse_id
35 ;
36 GO
37
38 SELECT
39     kunde_id, kontonummer, blz, bankname
40     FROM
41         bankverbindung INNER JOIN bank
42             ON bankverbindung.bank_id = bank.bank_id
43 ;
44 GO
45 SELECT
46     nachname, vorname, kontonummer
47     FROM
48         bankverbindung INNER JOIN kunde ON bankverbindung.kunde_id = kunde.kunde_id
49 ;
50 GO
51
52 SELECT 'Abkürzung mit USING (Seite 192)';
53 SELECT 'TSQL kennt keine Abkürzung mit USING oder eine vergleichbare';
54 GO
55
56 SELECT 'Abkürzung mit NATURAL JOIN (Seite 192)';
57 SELECT 'TSQL kennt keine NATURAL JOIN';
```

```
58 GO
59 -- Ende INNER JOIN
60
61 -- Start Aufbau Lieferanten
62 SELECT 'Aufbau der Lieferanten';
63 GO
64 INSERT INTO lieferant (firmenname, adresse_id)
65 VALUES
66     ('Gartenbedarf AllesGrün', 10)
67     ,('Office International', 11)
68     ,('Bürohengst GmbH', 11)
69 ;
70
71 INSERT INTO artikel_nm_lieferant (lieferant_id, artikel_id)
72 VALUES
73     (1, 7856)
74     ,(1, 7863)
75     ,(1, 9010)
76     ,(1, 9015)
77     ,(3, 3001)
78     ,(3, 3005)
79     ,(3, 3006)
80     ,(3, 3007)
81     ,(3, 3010)
82 ;
83 GO
84 -- Ende Aufbau Lieferanten
85
86 -- Start OUTER JOIN
87 SELECT 'OUTER JOIN (Seite 200)';
88 GO
89 SELECT
90     k.kunde_id, k.nachname, k.vorname, b.datum
91     FROM
92         bestellung b INNER JOIN kunde k ON b.kunde_id = k.kunde_id
93     ORDER BY
94         k.nachname, k.vorname
95 ;
96 GO
97 SELECT
98     k.kunde_id, k.nachname, k.vorname, b.datum
99     FROM
100        bestellung b RIGHT OUTER JOIN kunde k ON b.kunde_id = k.kunde_id
101    ORDER BY
102        k.nachname, k.vorname
103 ;
104 GO
105 SELECT
106     k.kunde_id, k.nachname, k.vorname, b.datum
107     FROM
108         kunde k LEFT OUTER JOIN bestellung b ON k.kunde_id = b.kunde_id
109     ORDER BY
110         k.nachname, k.vorname
111 ;
112 GO
113 SELECT
114     l.lieferant_id, l.firmenname, a.artikel_id, a.bezeichnung
115     FROM
116         artikel_nm_lieferant nm INNER JOIN artikel a ON nm.artikel_id = a.artikel_id
```

```
117           INNER JOIN lieferant l ON nm.lieferant_id = l.
118           lieferant_id
119           ORDER BY firmenname
120       ;
121   GO
122   SELECT
123     l.lieferant_id, l.firmenname, a.artikel_id, a.bezeichnung
124   FROM
125     artikel_nm_lieferant nm INNER JOIN artikel a ON nm.artikel_id = a.artikel_id
126                               RIGHT JOIN lieferant l ON nm.lieferant_id = l.
127                               lieferant_id
128           ORDER BY firmenname
129       ;
130   GO
131   SELECT
132     l.firmenname
133   FROM
134     artikel_nm_lieferant nm INNER JOIN artikel a ON nm.artikel_id = a.artikel_id
135                               RIGHT JOIN lieferant l ON nm.lieferant_id = l.
136                               lieferant_id
137           WHERE a.artikel_id IS NULL
138           ORDER BY firmenname
139       ;
140   GO
141   SELECT k.kunde_id, k.nachname, k.vorname
142   FROM
143     bestellung b RIGHT JOIN kunde k ON k.kunde_id = b.kunde_id
144           WHERE b.bestellung_id IS NULL
145       ;
146   -- Ende OUTER JOIN
147   -- Start Aufbau Forum
148   SELECT 'SELF JOIN (Seite 205)';
149   GO
150   DROP TABLE IF EXISTS beitrag;
151   DROP TABLE IF EXISTS account;
152   GO
153   SELECT 'CREATE account';
154   GO
155   CREATE TABLE account (
156     kunde_id INT NOT NULL DEFAULT 0,
157     name NVARCHAR(255) NOT NULL DEFAULT '',
158     passwort NCHAR(34) NOT NULL DEFAULT '',
159     status INT NOT NULL DEFAULT '0',
160     kommentar TEXT,
161     admin TINYINT NOT NULL DEFAULT 0,
162     deleted TINYINT NOT NULL DEFAULT 0,
163     PRIMARY KEY(kunde_id),
164     FOREIGN KEY (kunde_id)
165       REFERENCES kunde(kunde_id)
166       ON UPDATE NO ACTION
167       ON DELETE NO ACTION
168   );
169   GO
170   SELECT 'CREATE beitrag';
171   GO
172   CREATE TABLE beitrag (
```

```

173    beitrag_id  INT NOT NULL IDENTITY(1,1),
174    account_id  INT NOT NULL,
175    bezug_beitrag_id INT NOT NULL DEFAULT 1,
176    zeitstempel  DATETIME NOT NULL DEFAULT '1000-01-01 00:00:00',
177    nachricht  TEXT ,
178    deleted      TINYINT NOT NULL DEFAULT 0,
179    PRIMARY KEY(beitrag_id),
180    FOREIGN KEY (account_id)
181        REFERENCES account(kunde_id)
182        ON UPDATE NO ACTION
183        ON DELETE NO ACTION,
184    FOREIGN KEY (bezug_beitrag_id)
185        REFERENCES beitrag(beitrag_id)
186        ON UPDATE NO ACTION
187        ON DELETE NO ACTION
188 );
189 GO
190
191 SELECT 'account daten';
192 GO
193 INSERT INTO account (kunde_id, name, passwort, admin)
194 VALUES
195     (1, 'admin', CONVERT(NCHAR(34), HASHBYTES('MD5', 'rosi'), 2), 1)
196     ,(2, 'frodo', CONVERT(NCHAR(34), HASHBYTES('MD5', 'elbereth'), 2), 0)
197     ,(3, 'bilbo', CONVERT(NCHAR(34), HASHBYTES('MD5', 'blitzerschlagen'), 2), 0)
198     ,(5, 'elle', CONVERT(NCHAR(34), HASHBYTES('MD5', 'feanor'), 2), 1)
199 ;
200 GO
201
202
203 SELECT 'beitrag daten';
204 SET IDENTITY_INSERT beitrag ON ;
205 INSERT INTO beitrag (beitrag_id, account_id, bezug_beitrag_id, zeitstempel,
206    nachricht)
207 VALUES
208     (1, 1, 1, CONVERT(DATETIME, '1753-01-01 00:00:00'), '')
209     ,(2, 2, 1, CONVERT(DATETIME, '2011-05-01 14:13:00'), 'Der Lieferservice ist
210         super.')
211     ,(3, 3, 2, CONVERT(DATETIME, '2011-05-02 11:45:00'), 'Das finde ich auch.')
212     ,(4, 5, 2, CONVERT(DATETIME, '2011-05-01 17:01:00'), 'Aber ein wenig langsam
213         .')
214     ,(5, 2, 4, CONVERT(DATETIME, '2011-05-01 17:15:00'), 'Finde ich nicht.')
215     ,(6, 5, 1, CONVERT(DATETIME, '2011-06-12 09:07:00'), 'Angebot könnte besser
216         sein.')
217 ;
218 SET IDENTITY_INSERT beitrag OFF ;
219 GO
220 -- Ende Aufbau Forum
221
222 -- Start SELF JOIN
223 SELECT 'SELF JOIN (Seite 206)';
224 GO
225 SELECT
226     kunde_id, name, status, admin
227     FROM
228         account
229 ;
230
231 SELECT

```

```
228     beitrag_id, account_id, bezug_beitrag_id, LEFT(CONVERT(NVARCHAR(255),
229         nachricht), 20) kurz
230   FROM
231   beitrag
232 ;
233 GO
234 -- SELECT 'Erwarte eine Fehlermeldung (Seite 207)';
235 --  SELECT nachricht
236 --   FROM
237 --     beitrag INNER JOIN beitrag ON bezug_beitrag_id = beitrag_id
238 --   WHERE beitrag.id = 2
239 -- ;
240 -- GO
241
242 SELECT ant.nachricht 'Antwort'
243   FROM
244     beitrag ant INNER JOIN beitrag orig ON ant.bezug_beitrag_id = orig.
245         beitrag_id
246 WHERE orig.beitrag_id = 2
247 ;
248 GO
249
250 WITH ant_auf AS
251 (
252   -- nicht rekuriver Teil
253   SELECT * FROM beitrag WHERE bezug_beitrag_id = 2
254 UNION ALL
255   -- rekuriver Teil
256   SELECT ant.*
257     FROM
258       beitrag ant INNER JOIN ant_auf orig
259         ON ant.bezug_beitrag_id = orig.beitrag_id
260 )
261 SELECT beitrag_id, bezug_beitrag_id, nachricht FROM ant_auf;
262 GO
263 -- Ende SELF JOIN
264
265 -- Start Redundant
266 SELECT 'Beschleunigen mit Redundanzen (Seite 209)';
267 GO
268 ALTER TABLE kunde
269   ADD r_strasse NVARCHAR(255)
270 ;
271 ALTER TABLE kunde
272   ADD r_ort NVARCHAR(255)
273 ;
274 ALTER TABLE kunde
275   ADD r_aktuell TINYINT NOT NULL DEFAULT 1
276 ;
277 ALTER TABLE kunde
278   ADD l_strasse NVARCHAR(255)
279 ;
280 ALTER TABLE kunde
281   ADD l_ort NVARCHAR(255)
282 ;
283 ALTER TABLE kunde
284   ADD l_aktuell TINYINT NOT NULL DEFAULT 1
```

```

285 ;
286 GO
287 UPDATE kunde
288 SET
289   r_strasse = CONCAT(strasse, ' ', hnr),
290   r_ort = CONCAT(lkz, '--', plz, ' ', ort),
291   r_aktuell = 1
292   FROM kunde INNER JOIN adresse ON rechnung_adresse_id = adresse_id
293 ;
294 GO
295 SELECT
296   kunde_id, r_strasse, r_ort, r_aktuell
297   FROM kunde
298 ;
299 GO
300 -- Ende Redundant

```

Listing 29.45 mssql/listing09.sql

```

1  -- Ausführen der vorherigen Befehle
2  :r listing08.sql
3
4  SELECT 'Start listing09.sql //////////////////////////////';
5  GO
6  SELECT 'Aggregatfunktionen (Seite 212)';
7  GO
8  SELECT AVG(einzelpreis) FROM artikel;
9  SELECT COUNT(*) FROM kunde;
10 SELECT COUNT(liefer_adresse_id) FROM kunde;
11 SELECT COUNT(DISTINCT(rechnung_adresse_id)) FROM kunde;
12 SELECT MAX(einzelpreis), MIN(einzelpreis) FROM artikel;
13 SELECT SUM(menge) FROM bestellung_position;
14 SELECT
15   bp.menge, a.einzelpreis
16   FROM
17     bestellung_position bp INNER JOIN artikel a ON bp.artikel_id = a.artikel_id
18 ;
19 SELECT
20   bp.menge * a.einzelpreis 'Positionswert'
21   FROM
22     bestellung_position bp INNER JOIN artikel a ON bp.artikel_id = a.artikel_id
23 ;
24 GO
25
26 -- Start Aufbau Lagerverwaltung
27 SELECT 'Aufbau Lagerbestand (Seite 214)';
28 SELECT 'CREATE lagerbestand';
29 GO
30 DROP TABLE IF EXISTS lagerbestand;
31 GO
32 CREATE TABLE lagerbestand (
33   artikel_id      INT NOT NULL DEFAULT 0,
34   menge_mindest    DECIMAL(14,6) NOT NULL DEFAULT 0.0,
35   menge_aktuell    DECIMAL(14,6) NOT NULL DEFAULT 0.0,
36   deleted          TINYINT NOT NULL DEFAULT 0,
37   PRIMARY KEY(artikel_id),
38   FOREIGN KEY (artikel_id)
39     REFERENCES artikel(artikel_id)

```

```
40      ON UPDATE NO ACTION
41      ON DELETE NO ACTION
42  );
43 GO
44 SELECT 'Lagerbestand festlegen';
45 GO
46 INSERT INTO lagerbestand (artikel_id, menge_aktuell, menge_mindest)
47 VALUES
48      (7856, 500, 100)
49      ,(7863, 400, 100)
50      ,(9010, 30, 10)
51      ,(9015, 25, 10)
52      ,(3001, 5000, 1000)
53      ,(3005, 250, 200)
54      ,(3006, 250, 200)
55      ,(3007, 149, 200)
56      ,(3010, 3, 100)
57 ;
58 GO
59 -- Ende Aufbau Lagerverwaltung
60
61 SELECT 'GROUP BY (Seite 215)';
62 GO
63 -- Start Group by
64 SELECT
65     COUNT(*)
66     FROM
67     bestellung_position
68 ;
69 GO
70 SELECT
71     bestellung_id Bestellnummer, COUNT(*) 'Anzahl der Positionen'
72     FROM
73     bestellung_position
74     GROUP BY
75     bestellung_id
76 ;
77 GO
78 SELECT
79     a.bezeichnung, bp.menge
80     FROM
81     bestellung_position bp INNER JOIN artikel a ON bp.artikel_id = a.artikel_id
82 ;
83 GO
84 SELECT
85     a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
86     FROM
87     bestellung_position bp INNER JOIN artikel a ON bp.artikel_id = a.artikel_id
88     GROUP BY
89     a.bezeichnung
90 ;
91 GO
92 SELECT
93     a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
94     FROM
95     bestellung_position bp RIGHT JOIN artikel a ON bp.artikel_id = a.artikel_id
96     GROUP BY
97     a.bezeichnung
98 ;
```

```
99  GO
100 SELECT
101  a.bezeichnung 'Artikelname',
102  CASE
103    WHEN SUM(bp.menge) IS NULL THEN 0
104    ELSE SUM(bp.menge)
105  END AS 'Anzahl bestellter Artikel'
106  FROM
107    bestellung_position bp RIGHT JOIN artikel a ON bp.artikel_id = a.artikel_id
108  GROUP BY
109    a.bezeichnung
110  ;
111 GO
112 -- SELECT 'Eine Fehlermeldung! (Seite 216)';
113 -- SELECT
114 -- a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
115 -- FROM
116 -- bestellung_position bp INNER JOIN artikel a USING (artikel_id)
117 -- WHERE
118 -- SUM(bp.menge) > 10
119 -- GROUP BY
120 -- artikel_id
121 -- ;
122
123 SELECT 'Gruppenergebnisse mit WITH ROLLUP summieren (Seite 217)';
124 GO
125 SELECT
126  artikel_id, SUM(bp.menge)
127  FROM
128    bestellung_position bp RIGHT JOIN artikel a ON bp.artikel_id = a.artikel_id
129  GROUP BY
130    artikel_id WITH ROLLUP
131  ;
132 GO
133
134 SELECT 'Gruppenergebnisse mit HAVING filtern (Seite 218)';
135 GO
136 SELECT
137  a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
138  FROM
139    bestellung_position bp INNER JOIN artikel a ON bp.artikel_id = a.artikel_id
140  GROUP BY
141    a.bezeichnung
142  HAVING
143    SUM(bp.menge) > 10
144  ;
145 GO
146 SELECT
147  a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
148  FROM
149    bestellung_position bp INNER JOIN artikel a ON bp.artikel_id = a.artikel_id
150  WHERE
151    a.bezeichnung LIKE 'Silber%'
152  GROUP BY
153    a.bezeichnung
154  HAVING
155    SUM(bp.menge) > 10
156  ;
157 GO
```

```

158
159 SELECT 'Fragen zu Gruppen (Seite 220)';
160 GO
161 SELECT
162   SUBSTRING(blz, 1, 1) 'Clearinggebiet', COUNT(*) 'Anzahl'
163   FROM
164     bank
165   GROUP BY
166     SUBSTRING(blz, 1, 1)
167   ORDER BY 'Anzahl' DESC
168 ;
169 GO
170
171 -- Start Bestellungen erweitern
172 SELECT 'Bestellungen erweitern';
173 GO
174 SET IDENTITY_INSERT bestellung ON ;
175 INSERT INTO bestellung (bestellung_id, kunde_id, adresse_id, datum)
176   VALUES
177     (3, 1, 1, '2011-01-15 16:43:00')
178     ,(4, 1, 1, '2011-01-16 09:15:00')
179     ,(5, 1, 1, '2011-01-16 09:16:00')
180     ,(6, 2, 2, '2012-04-01 13:11:00')
181 ;
182 SET IDENTITY_INSERT bestellung OFF ;
183 INSERT INTO bestellung_position (bestellung_id, position_nr, artikel_id, menge
184   )
185   VALUES
186     (3, 1, 7856, 10)
187     ,(3, 2, 7863, 10)
188     ,(4, 1, 3006, 1)
189     ,(4, 2, 3010, 4)
190     ,(5, 1, 3001, 100)
191     ,(5, 2, 3010, 5)
192     ,(5, 3, 3006, 1)
193     ,(5, 4, 3005, 4)
194 ;
195 GO
196 -- Ende Bestellungen erweitern
197
198 SELECT
199   LEFT(CONCAT(FORMAT(datum, 'yyyy', 'de-de'), '/', FORMAT(datum, 'MMMM', 'de-de'
200   ')), 20) AS 'Monat', COUNT(*) 'Anzahl'
201   FROM bestellung
202   GROUP BY
203     FORMAT(datum, 'yyyy', 'de-de'), FORMAT(datum, 'MMMM', 'de-de')
204   ORDER BY
205     FORMAT(datum, 'yyyy', 'de-de') DESC, FORMAT(datum, 'MMMM', 'de-de') DESC
206 ;
207 GO
208 -- Ende Group by
209 SELECT 'Ende listing09.sql /////////////////////////////////';
210 GO

```

Listing 29.46 mssql/listing10.sql

```

1 -- Ausführen der vorherigen Befehle
2 :r listing09.sql

```

```
3
4  SELECT 'Start listing10.sql //////////////////////////////';
5  GO
6
7  SELECT 'Start Aufbau Rechnungswesen';
8  GO
9  USE oshop;
10 GO
11
12 INSERT INTO rechnung
13   (kunde_id, bestellung_id, adresse_id, datum)
14   SELECT kunde_id, bestellung_id, adresse_id, DATEADD(DAY, 1, datum) FROM
15     bestellung;
16 GO
17
18 SET IDENTITY_INSERT rechnung ON ;
19 INSERT INTO rechnung (rechnung_id, kunde_id, bestellung_id, adresse_id, datum,
20   bezahlart, status)
21   VALUES
22     (7, 3, 0, 2, CONVERT(DATETIME, '2012-04-06 12:11:00'), 3, 2)
23   ,(8, 3, 0, 2, CONVERT(DATETIME, '2012-04-07 13:12:00'), 3, 3)
24   ,(9, 5, 0, 4, CONVERT(DATETIME, '2012-04-08 14:13:00'), 2, 5)
25   ,(10, 5, 0, 4, CONVERT(DATETIME, '2012-04-09 15:14:00'), 2, 6)
26   ,(11, 5, 0, 4, CONVERT(DATETIME, '2012-04-10 16:15:00'), 2, 6)
27 ;
28 SET IDENTITY_INSERT rechnung OFF ;
29 GO
30
31 UPDATE rechnung SET status = 4 WHERE rechnung_id = 2;
32 GO
33 INSERT INTO rechnung_position (rechnung_id, position_nr, artikel_id, menge)
34   SELECT bestellung_id, position_nr, artikel_id, menge FROM bestellung_position
35   ;
36 GO
37 INSERT INTO rechnung_position
38   (rechnung_id, position_nr, artikel_id, menge)
39   VALUES
40     (7, 1, 3001, 5)
41   ,(7, 2, 3005, 1)
42   ,(8, 1, 3006, 7)
43   ,(8, 2, 3007, 1)
44   ,(10, 1, 3010, 15)
45   ,(10, 2, 3001, 5)
46   ,(11, 1, 3005, 20)
47 ;
48 GO
49
50 UPDATE rechnung SET rabatt = 7 WHERE kunde_id = 1;
51 UPDATE rechnung SET skonto = 3 WHERE kunde_id = 5;
52 GO
53
54 SELECT 'Ende Aufbau Rechnungswesen';
55 GO
56 SELECT 'Problem und Lösung (Seite 225)';
57 GO
58 SELECT
59   bp.bestellung_id, SUM(bp.menge * a.einzelpreis) 'Bestellwert'
60   FROM
61   bestellung_position bp INNER JOIN artikel a ON bp.artikel_id = a.artikel_id
```

```
59     GROUP BY
60     bp.bestellung_id
61 ;
62 GO
63 SELECT
64     k.nachname, k.vorname, SUM(bp.menge * a.einzelpreis) 'Bestellwert'
65     FROM
66     bestellung_position bp INNER JOIN artikel a    ON bp.artikel_id = a.
67                         artikel_id
68                         INNER JOIN bestellung b ON bp.bestellung_id = b.
69                         bestellung_id
70                         INNER JOIN kunde k      ON b.kunde_id = k.kunde_id
71     GROUP BY
72     bp.bestellung_id, k.nachname, k.vorname
73 ;
74 GO
75 DROP TABLE IF EXISTS #tmp_bestellwert;
76 GO
77 SELECT
78     k.nachname, k.vorname, SUM(bp.menge * a.einzelpreis) 'bestellwert'
79     INTO #tmp_bestellwert
80     FROM
81     bestellung_position bp INNER JOIN artikel a    ON bp.artikel_id = a.
82                         artikel_id
83                         INNER JOIN bestellung b ON bp.bestellung_id = b.
84                         bestellung_id
85                         INNER JOIN kunde k      ON b.kunde_id = k.kunde_id
86     GROUP BY
87     bp.bestellung_id, k.nachname, k.vorname
88 ;
89 GO
90 SELECT * FROM #tmp_bestellwert;
91 GO
92 SELECT
93     nachname, vorname, SUM(bestellwert) 'Umsatz'
94     FROM #tmp_bestellwert
95     GROUP BY nachname, vorname
96 ;
97 GO
98 SELECT
99     bw.nachname, bw.vorname, SUM(bw.bestellwert) 'Umsatz'
100    FROM
101    (
102        SELECT
103            k.nachname, k.vorname, SUM(bp.menge * a.einzelpreis) 'bestellwert'
104            FROM
105            bestellung_position bp INNER JOIN artikel a    ON bp.artikel_id = a.
106                         artikel_id
107                         INNER JOIN bestellung b ON bp.bestellung_id = b.
108                         bestellung_id
109                         INNER JOIN kunde k      ON b.kunde_id = k.
110                         kunde_id
111     GROUP BY
112         bp.bestellung_id, k.nachname, k.vorname
113 ) AS bw
114     GROUP BY nachname, vorname
115 ;
116 GO
```

```
111 -- Start nicht korrelierende Unterabfrage
112 SELECT 'Nicht korrelierende Unterabfrage (Seite 228)';
113 GO
114 SELECT MAX(blz) FROM bank;
115 GO
116 SELECT bankname
117   FROM bank
118 WHERE blz = '37010050'
119 ;
120 GO
121 SELECT bankname
122   FROM bank
123 WHERE blz =
124 (
125   SELECT MAX(blz)
126     FROM bank
127 )
128 ;
129 GO
130 SELECT
131   AVG(einzelpreis) AS 'durchschnittspreis'
132   FROM artikel
133 ;
134 GO
135 SELECT *
136   FROM artikel
137 WHERE
138   einzelpreis > 30
139 ;
140 GO
141 SELECT *
142   FROM artikel
143 WHERE
144   einzelpreis >
145 (
146   SELECT
147     AVG(einzelpreis) AS 'durchschnittspreis'
148     FROM artikel
149 )
150 ;
151 GO
152 SELECT
153   bp.bestellung_id, SUM(bp.menge * a.einzelpreis) bestellwert
154   FROM
155     bestellung_position bp INNER JOIN artikel a ON bp.artikel_id = a.artikel_id
156   GROUP BY
157     bp.bestellung_id
158 ;
159 GO
160 -- Leider kann TSQL Alias nicht in WHERE-Klauseln wiedervewenden!!!
161 SELECT AVG(bw.bestellwert)
162   FROM (
163   SELECT
164     bp.bestellung_id, SUM(bp.menge * a.einzelpreis) bestellwert
165     FROM
166       bestellung_position bp INNER JOIN artikel a ON bp.artikel_id = a.
167           artikel_id
168     GROUP BY
169       bp.bestellung_id
```

```
169      ) AS bw
170 ;
171 GO
172 SELECT
173   bp1.bestellung_id, SUM(bp1.menge * a1.einzelpreis) bestellwert1
174   FROM
175     bestellung_position bp1 INNER JOIN artikel a1 ON bp1.artikel_id = a1.
176       artikel_id
177   GROUP BY
178     bp1.bestellung_id
179   HAVING SUM(bp1.menge * a1.einzelpreis) > 100
180 ;
181 GO
182 SELECT
183   bp1.bestellung_id, SUM(bp1.menge * a1.einzelpreis) bwert1
184   FROM
185     bestellung_position bp1 INNER JOIN artikel a1 ON bp1.artikel_id = a1.
186       artikel_id
187   GROUP BY
188     bp1.bestellung_id
189   HAVING
190     SUM(bp1.menge * a1.einzelpreis) >
191     (
192       SELECT AVG(bw.bwert2)
193       FROM
194       (
195         SELECT
196           bp2.bestellung_id, SUM(bp2.menge * a2.einzelpreis) bwert2
197           FROM
198             bestellung_position bp2 INNER JOIN artikel a2 ON bp2.artikel_id = a2.
199               artikel_id
200             GROUP BY
201               bp2.bestellung_id
202         ) AS bw
203     )
204 ;
205 GO
206 SELECT bestellung_id FROM bestellung;
207 GO
208 SELECT rechnung_id
209   FROM rechnung
210 WHERE bestellung_id IN
211   (
212     SELECT bestellung_id FROM bestellung
213   )
214 ;
215 GO
216 SELECT a.einzelpreis
217   FROM
218     artikel_nm_warengruppe nm INNER JOIN artikel a      ON nm.artikel_id = a.
219       artikel_id
220                           INNER JOIN warengruppe w ON nm.warengruppe_id = w.
221                             warengruppe_id
222 WHERE
223   w.bezeichnung = 'Gartenbedarf'
224 ;
225 GO
226 SELECT
227   a.bezeichnung, a.einzelpreis
```

```

223   FROM
224     artikel a
225   WHERE
226     a.einzelpreis > ALL(SELECT 10)
227 ;
228 GO
229 SELECT
230   a.bezeichnung, a.einzelpreis
231   FROM
232     artikel a
233   WHERE
234     a.einzelpreis > ALL
235   (
236     SELECT a.einzelpreis
237       FROM
238         artikel_nm_warengruppe nm INNER JOIN artikel a ON nm.artikel_id = a.
239           artikel_id
240           INNER JOIN warengruppe w ON nm.warengruppe_id
241             = w.warengruppe_id
242             WHERE
243               w.bezeichnung = 'Gartenbedarf'
244   )
245 ;
246 GO
247 SELECT
248   r.rechnung_id, r.kunde_id, SUM(a.einzelpreis*rp.menge) AS 'umsatz'
249   FROM
250     rechnung_position rp INNER JOIN artikel a ON rp.artikel_id = a.artikel_id
251           INNER JOIN rechnung r ON rp.rechnung_id = r.rechnung_id
252   GROUP BY
253     r.rechnung_id, r.kunde_id
254 ;
255 GO
256 SELECT SUM(umsatz) AS 'umsatzsumme'
257   FROM
258   (
259     SELECT
260       r.rechnung_id, r.kunde_id, SUM(a.einzelpreis*rp.menge) AS 'umsatz'
261       FROM
262         rechnung_position rp INNER JOIN artikel a ON rp.artikel_id = a.artikel_id
263           INNER JOIN rechnung r ON rp.rechnung_id = r.
264             rechnung_id
265           GROUP BY
266             r.rechnung_id, r.kunde_id
267   ) AS ksum
268   GROUP BY
269     ksum.kunde_id
270 ;
271 GO
272 SELECT SUM(umsatz) AS 'umsatzsumme'
273   FROM
274   (
275     SELECT
276       r.rechnung_id, k.kunde_id, SUM(a.einzelpreis*rp.menge) AS 'umsatz'
277       FROM
278         rechnung_position rp INNER JOIN artikel a ON rp.artikel_id = a.artikel_id
279           INNER JOIN rechnung r ON rp.rechnung_id = r.
280             rechnung_id
281           INNER JOIN kunde k ON r.kunde_id = k.kunde_id

```

```
278     WHERE nachname = 'beutlin'  
279     GROUP BY  
280         r.rechnung_id, k.kunde_id  
281 ) AS ksum  
282 GROUP BY  
283     ksum.kunde_id  
284 ;  
285 GO  
286 SELECT ksum.kunde_id, SUM(umsatz) AS 'umsatzsumme'  
287 FROM  
288 (  
289     SELECT  
290         r.rechnung_id, r.kunde_id, SUM(a.einzelpreis*rp.menge) AS 'umsatz'  
291     FROM  
292         rechnung_position rp INNER JOIN artikel a ON rp.artikel_id = a.artikel_id  
293                         INNER JOIN rechnung r ON rp.rechnung_id = r.  
294                             rechnung_id  
295                         GROUP BY  
296                             r.rechnung_id, r.kunde_id  
297 ) AS ksum  
298 GROUP BY  
299     ksum.kunde_id  
300 HAVING SUM(umsatz) > ALL (SELECT 100)  
301 ;  
302 GO  
303 SELECT k.kunde_id, k.nachname, k.vorname, SUM(umsatz) AS 'umsatzsumme'  
304 FROM  
305 (  
306     SELECT  
307         r.rechnung_id, r.kunde_id, SUM(a.einzelpreis*rp.menge) AS 'umsatz'  
308     FROM  
309         rechnung_position rp INNER JOIN artikel a ON rp.artikel_id = a.artikel_id  
310                         INNER JOIN rechnung r ON rp.rechnung_id = r.  
311                             rechnung_id  
312                         GROUP BY  
313                             r.rechnung_id, r.kunde_id  
314 ) AS ksum  
315 INNER JOIN kunde k ON ksum.kunde_id = k.kunde_id  
316 GROUP BY  
317     k.kunde_id, k.nachname, k.vorname  
318 HAVING SUM(umsatz) > ALL  
319 (  
320     SELECT SUM(umsatz) AS 'umsatzsumme'  
321     FROM  
322     (  
323         SELECT  
324             r.rechnung_id, r.kunde_id, SUM(a.einzelpreis*rp.menge) AS 'umsatz'  
325             FROM  
326                 rechnung_position rp INNER JOIN artikel a ON rp.artikel_id = a.  
327                     artikel_id  
328                         INNER JOIN rechnung r ON rp.rechnung_id = r.  
329                             rechnung_id  
330                         INNER JOIN kunde k ON r.kunde_id = k.kunde_id  
331 WHERE k.nachname = 'beutlin'  
332 GROUP BY  
333     r.rechnung_id, r.kunde_id  
334 ) AS ksum  
335 GROUP BY  
336     ksum.kunde_id
```

```
333      )
334  ;
335 GO
336 SELECT a.einzelpreis
337   FROM
338     artikel_nm_warengruppe nm INNER JOIN artikel a ON nm.artikel_id = a.
339           artikel_id
340           INNER JOIN warengruppe w ON nm.warengruppe_id = w.
341           warengruppe_id
342 WHERE
343   w.bezeichnung = 'Bürobedarf'
344 ;
345 GO
346 SELECT
347   a.bezeichnung, a.einzelpreis
348   FROM
349     artikel a
350 WHERE
351   a.einzelpreis > ANY(SELECT 10)
352 ;
353 GO
354 SELECT
355   a.bezeichnung, a.einzelpreis
356   FROM
357     artikel a
358 WHERE
359   a.einzelpreis > ANY
360   (
361     SELECT a.einzelpreis
362       FROM
363         artikel_nm_warengruppe nm INNER JOIN artikel a ON nm.artikel_id = a.
364             artikel_id
365             INNER JOIN warengruppe w ON nm.warengruppe_id =
366               w.warengruppe_id
367 WHERE
368   w.bezeichnung = 'Bürobedarf'
369 );
370 GO
371 SELECT
372   a.bezeichnung, a.einzelpreis
373   FROM
374     artikel a
375 WHERE
376   a.artikel_id NOT IN
377   (
378     SELECT
379       artikel_id
380       FROM
381         artikel_nm_warengruppe nm  INNER JOIN warengruppe w ON nm.warengruppe_id
382           = w.warengruppe_id
383 WHERE w.bezeichnung = 'Bürobedarf'
384 )
385 AND
386   a.einzelpreis > ANY
387   (
388     SELECT a.einzelpreis
389       FROM
```

```
386     artikel_nm_warengruppe nm INNER JOIN artikel a ON nm.artikel_id = a.
387             artikel_id
388
389             INNER JOIN warengruppe w ON nm.warengruppe_id =
390                     w.warengruppe_id
391
392         WHERE
393             w.bezeichnung = 'Bürobedarf'
394     )
395 ;
396 GO
397 -- Leider kann in TSQL kein Prädikat der Art (v1, v2) IN (tabspalte1,
398 -- tabspalte12) gemacht werden.
399
400 -- Ende nicht korrelierende Unterabfrage
401
402 -- Start korrelierende Unterabfrage
403 SELECT 'Korrelierende Unterabfrage (Seite 240)';
404 SELECT rechnung_id
405     FROM rechnung r
406     WHERE
407     (
408         SELECT
409             COUNT(position_nr)
410             FROM rechnung_position rp
411             WHERE
412                 rp.rechnung_id = r.rechnung_id
413             ) >= 3
414 ;
415 GO
416 SELECT rechnung_id
417     FROM rechnung r
418     WHERE
419         EXISTS
420         (
421             SELECT
422                 bestellung_id
423                 FROM bestellung b
424                 WHERE b.bestellung_id = r.bestellung_id
425         )
426 ;
427 GO
428 -- Ende korrelierende Unterabfrage
429
430 -- Start Fallstudie
431 SELECT 'Fallstudie Datenimport (Seite 242)';
432 GO
433 DROP TABLE IF EXISTS #tmp_import;
434 GO
435 CREATE TABLE #tmp_import
436 (
437     nachname NVARCHAR(255) COLLATE German_PhoneBook_CI_AI,
438     vorname NVARCHAR(255) COLLATE German_PhoneBook_CI_AI,
439     strasse NVARCHAR(255) COLLATE German_PhoneBook_CI_AI,
440     hnr NVARCHAR(255) COLLATE German_PhoneBook_CI_AI,
441     lkz NVARCHAR(255) COLLATE German_PhoneBook_CI_AI,
442     plz NVARCHAR(255) COLLATE German_PhoneBook_CI_AI,
443     ort NVARCHAR(255) COLLATE German_PhoneBook_CI_AI,
444     ktnr NVARCHAR(255) COLLATE German_PhoneBook_CI_AI,
445     blz NVARCHAR(255) COLLATE German_PhoneBook_CI_AI
446 );
447
```

```
442 /*
443  BULK INSERT #tmp_import
444  FROM 'X:\[...]\kunden01.csv'
445  WITH (FIRSTROW = 2, FIELDTERMINATOR = ';',      ROWTERMINATOR='0x0a', FORMAT='
446  CSV');
447 */
448 :r kunden01.sql
449 SELECT LEFT(nachname,10), LEFT(vorname,10), LEFT(strasse,10), LEFT(hnr,10),
450       LEFT(plz,10), LEFT(ort,10) FROM #tmp_import;
451
452 INSERT INTO kunde (nachname, vorname)
453 SELECT DISTINCT nachname, vorname
454   FROM #tmp_import
455 WHERE
456   CONCAT(TRIM(#tmp_import.vorname), TRIM(#tmp_import.nachname)) NOT IN
457   (
458     SELECT CONCAT(TRIM(vorname), TRIM(nachname)) FROM kunde
459   )
460 ;
461 GO
462
463 INSERT INTO adresse (strasse, hnr, lkz, plz, ort)
464 SELECT DISTINCT strasse, hnr, lkz, plz, ort
465   FROM #tmp_import tmp
466 WHERE
467   CONCAT(TRIM(tmp.strasse), TRIM(tmp.hnr), TRIM(tmp.lkz), TRIM(tmp.plz), TRIM
468         (tmp.ort)) NOT IN
469   (
470     SELECT CONCAT(TRIM(strasse), TRIM(hnr), TRIM(lkz), TRIM(plz), TRIM(ort))
471       FROM adresse
472   )
473 ;
474 GO
475
476 ALTER TABLE #tmp_import ADD kunde_id INT ;
477 ALTER TABLE #tmp_import ADD adresse_id INT ;
478 ALTER TABLE #tmp_import ADD bank_id NCHAR(12) COLLATE German_PhoneBook_CI_AI;
479 GO
480
481 SELECT 'UPDATE';
482 GO
483 UPDATE #tmp_import
484 SET
485   #tmp_import.kunde_id = k.kunde_id
486   FROM
487   (
488     SELECT kunde_id, nachname, vorname
489       FROM kunde
490   ) k
491 WHERE
492   CONCAT(TRIM(#tmp_import.vorname), TRIM(#tmp_import.nachname))
493   =
494   CONCAT(TRIM(k.vorname), TRIM(k.nachname))
495 ;
496 GO
497 UPDATE #tmp_import
498   SET #tmp_import.adresse_id = a.adresse_id
```

```
498 FROM
499 (
500     SELECT adresse_id, strasse, hnr, plz, ort
501         FROM adresse a
502    ) a
503 WHERE
504     CONCAT(TRIM(#tmp_import.strasse), TRIM(#tmp_import.hnr), TRIM(#tmp_import.
505             plz), TRIM(#tmp_import.ort))
506 =
507     CONCAT(TRIM(a.strasse), TRIM(a.hnr), TRIM(a.plz), TRIM(a.ort))
508 ;
509 -- SELECT k.kunde_id, a.adresse_id, CONCAT(TRIM(LEFT(k.vorname, 20)), TRIM(
510             LEFT(k.nachname, 20)), TRIM(LEFT(a.strasse, 20)), TRIM(LEFT(a.hnr, 20)))
511             FROM #tmp_import INNER JOIN kunde k ON #tmp_import.kunde_id = k.kunde_id
512             INNER JOIN adresse a ON #tmp_import.adresse_id = a.
513                 adresse_id ORDER BY kunde_id, adresse_id
514
515 GO
516 UPDATE #tmp_import
517     SET #tmp_import.bank_id = b.bank_id
518     FROM (SELECT bank_id, blz FROM bank) b
519     WHERE TRIM(#tmp_import.blz) = TRIM(b.blz)
520 ;
521 GO
522
523 INSERT INTO
524     bankverbindung (kunde_id, bankverbindung_nr, bank_id, kontonummer, iban)
525     SELECT DISTINCT kunde_id, 1, bank_id, ktNr, CONCAT(blz, ktNr)
526         FROM #tmp_import
527     WHERE 0 = (
528         SELECT COUNT(*)
529             FROM bankverbindung bv
530             WHERE
531                 #tmp_import.kunde_id = bv.kunde_id
532                 AND
533                 #tmp_import.bank_id = bv.bank_id
534                 AND
535                 #tmp_import.ktNr = bv.kontonummer
536         )
537 ;
538 GO
539
540 ALTER TABLE #tmp_import
541     ADD bankverbindung_nr INT NOT NULL DEFAULT 1
542 ;
543 GO
544
545 UPDATE kunde
546     SET
547         kunde.rechnung_adresse_id = #tmp_import.adresse_id
548     FROM kunde INNER JOIN #tmp_import ON #tmp_import.kunde_id = kunde.kunde_id
549     WHERE k.rechnung_adresse_id IS NULL
550 ;
551 GO
552
553 SELECT kunde_id, TRIM(LEFT(nachname, 20)), TRIM(LEFT(vorname, 20)), TRIM(LEFT(
554             strasse, 20)), TRIM(LEFT(ort, 20))
555     FROM
556     kunde LEFT JOIN adresse ON kunde.rechnung_adresse_id = adresse_id
557 ;
```

```
551 GO  
552  
553 SELECT 'Ende listing10.sql //////////////////////////////';  
554 GO
```

Listing 29.47 mssql/listing11.sql

```
1 -- Ausführen der vorherigen Befehle  
2 :r listing10.sql  
3  
4 SELECT 'Start listing11.sql //////////////////////////////';  
5 GO  
6  
7 SELECT 'UNION (Seite 251)';  
8 GO  
9 SELECT  
10    kunde_id, LEFT(vorname, 10), LEFT(strasse, 10), LEFT(hnr, 10), LEFT(lkz, 10),  
       LEFT(plz, 10), LEFT(ort, 10)  
11   FROM  
12     kunde INNER JOIN adresse ON rechnung_adresse_id = adresse_id  
13 ;  
14 GO  
15  
16 SELECT  
17   strasse, hnr, lkz, plz, ort  
18   FROM  
19     lieferant l INNER JOIN adresse a ON l.adresse_id = a.adresse_id  
20 ;  
21 GO  
22 SELECT  
23   strasse, hnr, lkz, plz, ort  
24   FROM  
25     kunde INNER JOIN adresse ON rechnung_adresse_id = adresse_id  
26 UNION  
27 SELECT  
28   strasse, hnr, lkz, plz, ort  
29   FROM  
30     lieferant l INNER JOIN adresse a ON l.adresse_id = a.adresse_id  
31 ;  
32 GO  
33 SELECT  
34   strasse, hnr, lkz, plz, ort  
35   FROM  
36     kunde INNER JOIN adresse ON rechnung_adresse_id = adresse_id  
37 UNION ALL  
38 SELECT  
39   strasse, hnr, lkz, plz, ort  
40   FROM  
41     lieferant l INNER JOIN adresse a ON l.adresse_id = a.adresse_id  
42 ;  
43 GO  
44  
45 SELECT 'INTERSECT (Seite 254)';  
46 GO  
47 SELECT 'Schnittmenge mit Unterabfrage (Seite 255)';  
48 GO  
49 SELECT kunde_id FROM rechnung WHERE YEAR(datum) = '2011';  
50 GO
```

```
51 SELECT DISTINCT kunde_id
52   FROM rechnung
53  WHERE YEAR(datum) = '2012' AND kunde_id IN (1, 2, 3)
54 ;
55 GO
56 SELECT DISTINCT kunde_id
57   FROM rechnung
58  WHERE
59    YEAR(datum) = '2012' AND kunde_id IN
60    (
61      SELECT kunde_id FROM rechnung WHERE YEAR(datum) = '2011'
62    )
63 ;
64 GO
65
66 SELECT 'EXCEPT (Seite 256)';
67 GO
68
69 SELECT 'Differenz mit Unterabfrage (Seite 256)';
70 SELECT
71   strasse, hnr, lkz, plz, ort
72  FROM
73   adresse
74 ;
75 GO
76 SELECT
77   strasse, hnr, lkz, plz, ort
78  FROM
79   kunde INNER JOIN adresse
80     ON liefer_adresse_id = adresse_id
81 ;
82 GO
83 SELECT kunde_id
84   FROM rechnung
85  WHERE YEAR(datum) <> '2011' AND YEAR(datum) = '2012'
86 ;
87 GO
88 SELECT kunde_id FROM rechnung WHERE YEAR(datum) = '2011';
89 GO
90 SELECT DISTINCT kunde_id
91   FROM rechnung
92  WHERE
93    YEAR(datum) = '2012' AND kunde_id NOT IN (1, 2, 3)
94 ;
95 GO
96 SELECT DISTINCT kunde_id
97   FROM rechnung
98  WHERE
99    YEAR(datum) = '2012' AND kunde_id NOT IN
100   (
101     SELECT kunde_id FROM rechnung WHERE YEAR(datum) = '2011'
102   )
103 ;
104 SELECT 'Ende listing11.sql /////////////////////////////////';
105 GO
```

```
1 -- Ausführen der vorherigen Befehle
2 :r listing11.sql
3
4 SELECT 'Start listing12.sql /////////////////////////////////';
5 GO
6
7 SELECT kunde_id, nachname, vorname, art FROM kunde;
8 GO
9
10 SELECT 'Start Aufbau einer temporären art-Tabelle (Seite 262)';
11 GO
12
13 DROP TABLE IF EXISTS #tmp_art;
14 CREATE TABLE #tmp_art
15 (
16     art INT,
17     k_art NVARCHAR(255)
18 )
19 ;
20 GO
21
22 INSERT INTO #tmp_art
23     VALUES
24         (1, 'Privatkunde')
25         ,(2, 'Geschäftskunde')
26         ,(0, 'Unbekannt')
27 ;
28
29 SELECT kunde_id, nachname, vorname, k_art
30     FROM
31         kunde INNER JOIN #tmp_art ON kunde.art = #tmp_art.art
32 ;
33 SELECT 'Ende Aufbau einer temporären art-Tabelle';
34 GO
35
36 SELECT 'Lösung mit UNIONs (Seite 262)';
37 GO
38
39 SELECT kunde_id, nachname, vorname, k_art
40     FROM
41         kunde INNER JOIN
42 (
43             SELECT 1 AS art, 'Privatkunde' AS k_art
44             UNION
45             SELECT 2 AS art, 'Geschäftskunde' AS k_art
46             UNION
47             SELECT 3 AS art, 'Unbekannt' AS k_art
48         ) t
49         ON kunde.art = t.art
50 ;
51 GO
52
53 SELECT 'Nö (Seite 263)';
54 SELECT kunde_id, nachname, vorname,
55 CASE art
56     WHEN 1 THEN 'Privatkunde'
57     WHEN 2 THEN 'Geschäftskunde'
58     ELSE 'Unbekannt'
59 END AS k_art
```

```
60  FROM kunde
61 ;
62 GO
63
64 -- Start Einfacher CASE
65 SELECT 'Einfacher CASE (Seite 264)';
66 SELECT artikel_id, bezeichnung,
67 CASE waehrung
68 WHEN 'EUR' THEN CONCAT(ROUND(einzelpreis, 2), ' €')
69 WHEN 'USD' THEN CONCAT(ROUND(einzelpreis, 2), ' $')
70 ELSE '?????'
71 END AS preis
72 FROM artikel
73 ;
74 GO
75
76 UPDATE artikel SET waehrung = 'XYZ' WHERE artikel_id = 3001;
77 SELECT artikel_id, bezeichnung,
78 CASE waehrung
79 WHEN 'EUR' THEN CONCAT(ROUND(einzelpreis, 2), ' €')
80 WHEN 'USD' THEN CONCAT(ROUND(einzelpreis, 2), ' $')
81 END AS preis
82 FROM artikel
83 ;
84 GO
85
86 UPDATE artikel SET waehrung = 'EUR' WHERE artikel_id = 3001;
87 GO
88 -- Ende Einfacher CASE
89
90 -- Start Searched CASE
91 SELECT 'Searched CASE (Seite 265)';
92 GO
93 SELECT beitrag_id,
94 CASE
95 WHEN bezug_beitrag_id > 1 THEN 'Antwort'
96 WHEN bezug_beitrag_id <= 1 THEN 'Keine Antwort'
97 ELSE '?????'
98 END AS Typ
99 FROM beitrag
100 ORDER BY beitrag_id
101 ;
102 GO
103
104 SELECT beitrag_id,
105 CASE
106 WHEN bezug_beitrag_id > 1
107 THEN CONCAT(nachricht, ' Antwort auf ', bezug_beitrag_id, ': >',
108 (
109     SELECT nachricht
110     FROM beitrag b2
111     WHERE b2.beitrag_id = b1.bezug_beitrag_id
112 )
113 ) -- Ende CONCAT
114 WHEN bezug_beitrag_id <= 1 THEN nachricht
115 ELSE '?????'
116 END AS Inhalt
117 FROM beitrag b1
118 WHERE beitrag_id > 1
```

```

119 ;
120 GO
121
122 SELECT beitrag_id,
123 CASE
124 WHEN bezug_beitrag_id >= 0 THEN 'Antwort'
125 WHEN bezug_beitrag_id > 1 THEN 'Keine Antwort'
126 ELSE '??????'
127 END AS Typ
128 FROM beitrag
129 ORDER BY beitrag_id
130 ;
131 GO
132 -- Ende Searched CASE
133
134 -- Start Fallbeispiele
135 SELECT 'Fallbeispiel Lagerbestand (Seite 267)';
136 GO
137 SELECT * FROM lagerbestand;
138 GO
139 SELECT a.bezeichnung,
140 CASE
141 WHEN l.menge_aktuell <= l.menge_mindest
142 THEN 'Artikel nachbestellen'
143 ELSE 'Bestand ausreichend'
144 END AS lagerstand
145 FROM lagerbestand l INNER JOIN artikel a ON l.artikel_id=a.artikel_id
146 ORDER BY l.menge_aktuell / l.menge_mindest
147 ;
148 GO
149
150 SELECT 'Fallbeispiel Kundengruppen (Seite 268)';
151 GO
152 SELECT DISTINCT k.kunde_id, LEFT(k.nachname, 20), LEFT(k.vorname, 20),
153 CASE
154 WHEN 3 <
155 (
156 SELECT COUNT(rechnung_id) FROM rechnung r
157 WHERE r.kunde_id = k.kunde_id
158 ) THEN 'Prämiumkunde'
159 WHEN 2 <
160 (
161 SELECT COUNT(rechnung_id) FROM rechnung r
162 WHERE r.kunde_id = k.kunde_id
163 ) THEN 'Guter Kunde'
164 ELSE 'Kleinkunde'
165 END kundenart
166 FROM
167 rechnung r RIGHT OUTER JOIN kunde k ON r.kunde_id = k.kunde_id
168 ;
169 GO
170 SELECT k.kunde_id, LEFT(k.nachname, 20), LEFT(k.vorname, 20),
171 (
172 SELECT SUM(a.einzelpreis * rp.menge)
173 FROM rechnung r INNER JOIN rechnung_position rp
174 ON rp.rechnung_id = r.rechnung_id
175 INNER JOIN artikel a
176 ON rp.artikel_id=a.artikel_id
177 WHERE r.kunde_id = k.kunde_id

```

```

178      ) rechnungswert,
179      CASE
180        WHEN
181          kunde_id NOT IN (SELECT kunde_id FROM rechnung)
182          THEN 'Kleinkunde'
183        WHEN
184        (
185          SELECT SUM(a.einzelpreis * rp.menge) rechnungswert
186            FROM rechnung r INNER JOIN rechnung_position rp
187              ON rp.rechnung_id = r.rechnung_id
188              INNER JOIN artikel a
189                ON rp.artikel_id=a.artikel_id
190              WHERE r.kunde_id = k.kunde_id
191        ) BETWEEN 0 AND 300 THEN 'Kleinkunde'
192      WHEN
193      (
194        SELECT SUM(a.einzelpreis * rp.menge) rechnungswert
195          FROM rechnung r INNER JOIN rechnung_position rp
196            ON rp.rechnung_id = r.rechnung_id
197            INNER JOIN artikel a
198              ON rp.artikel_id=a.artikel_id
199              WHERE r.kunde_id = k.kunde_id
200        ) BETWEEN 300 AND 1000 THEN 'Guter Kunde'
201      ELSE 'Prämiumkunde'
202    END AS kundenart
203  FROM kunde k
204  ORDER BY
205  (
206    SELECT SUM(a.einzelpreis * rp.menge) rechnungswert
207      FROM rechnung r INNER JOIN rechnung_position rp
208        ON rp.rechnung_id = r.rechnung_id
209        INNER JOIN artikel a
210          ON rp.artikel_id=a.artikel_id
211        WHERE r.kunde_id = k.kunde_id
212  ) DESC
213 ;
214 GO
215
216 DROP TABLE IF EXISTS #tmp_umsatz;
217 GO
218 SELECT kunde_id, SUM(a.einzelpreis * rp.menge) rechnungswert
219 INTO #tmp_umsatz
220   FROM rechnung r INNER JOIN rechnung_position rp ON rp.rechnung_id = r.
221               rechnung_id
222                 INNER JOIN artikel a ON rp.artikel_id = a.artikel_id
223   GROUP BY kunde_id
224 ;
225 GO
226 SELECT k.kunde_id, LEFT(k.nachname, 20), LEFT(k.vorname, 20), #tmp_umsatz.
227               rechnungswert,
228     CASE
229       WHEN
230         #tmp_umsatz.kunde_id IS NULL THEN 'Kleinkunde'
231       WHEN
232         rechnungswert BETWEEN 0 AND 300 THEN 'Kleinkunde'
233       WHEN
234         rechnungswert BETWEEN 300 AND 1000 THEN 'Guter Kunde'
235       ELSE 'Premiumkunde'
236     END AS kundenart

```

```

235 FROM kunde k LEFT JOIN #tmp_umsatz ON k.kunde_id = #tmp_umsatz.kunde_id
236 ORDER BY rechnungswert DESC
237 ;
238 GO
239
240 WITH tmp_umsatz (kunde_id, rechnungswert)
241 AS (
242     SELECT kunde_id, SUM(a.einzelpreis * rp.menge) rechnungswert
243         FROM rechnung r INNER JOIN rechnung_position rp ON rp.rechnung_id = r.
244             rechnung_id
245                 INNER JOIN artikel a ON rp.artikel_id = a.artikel_id
246             GROUP BY kunde_id
247 )
248     SELECT k.kunde_id, LEFT(k.nachname, 20), LEFT(k.vorname, 20), tmp_umsatz.
249         rechnungswert,
250     CASE
251     WHEN
252         tmp_umsatz.kunde_id IS NULL THEN 'Kleinkunde'
253     WHEN
254         rechnungswert BETWEEN 0 AND 300 THEN 'Kleinkunde'
255     WHEN
256         rechnungswert BETWEEN 300 AND 1000 THEN 'Guter Kunde'
257     ELSE 'Premiumkunde'
258 END AS kundenart
259     FROM kunde k LEFT JOIN tmp_umsatz ON k.kunde_id = tmp_umsatz.kunde_id
260 ORDER BY rechnungswert DESC
261 ;
262 GO
263
264 SELECT 'Fallbeispiel Lieferanten (Seite 271)';
265 GO
266 SELECT LEFT(l.firmenname, 20),
267     CASE
268     WHEN
269         l.lieferant_id IN
270             (
271                 SELECT lieferant_id FROM artikel_nm_lieferant
272             ) THEN 'aktiv'
273     ELSE 'inaktiv'
274 END AS status
275     FROM lieferant l
276 ORDER BY firmenname
277 ;
278 GO
279
280 SELECT LEFT(a.bezeichnung, 20),
281     CASE
282     WHEN l.menge_aktuell <= l.menge_mindest THEN 'Artikel sofort nachbestellen
283             ,
284             WHEN l.menge_aktuell <= l.menge_mindest + 0.3 * l.menge_mindest THEN '
285                 Artikel bald nachbestellen'
286             ELSE 'Bestand ausreichend' END AS lagerstand FROM lagerbestand l INNER
287                 JOIN artikel a ON l.artikel_id = a.artikel_id
288             ORDER BY l.menge_aktuell / l.menge_mindest
289 ;
290 -- Ende Fallbeispiele
291 SELECT 'Ende listing12.sql /////////////////////////////////';
292 GO
293

```

Listing 29.49 mssql/listing13.sql

```
1 -- Ausführen der vorherigen Befehle
2 -- :r listing12.sql
3
4 SELECT 'Start listing13.sql //////////////////////////////';
5 GO
6 SELECT 'Meine erste Ansicht (Seite 274)';
7 GO
8 CREATE VIEW
9     view_kundenrechnungsadresse (id, nachname, vorname, strasse, ort)
10    AS
11        SELECT
12            kunde_id, nachname, vorname,
13            CONCAT_WS(' ', strasse, hnr),
14            CONCAT_WS(' ', lkz, plz, ort)
15        FROM
16            kunde INNER JOIN adresse
17                ON rechnung_adresse_id = adresse_id
18        WHERE kunde.deleted = 0
19 ;
20 GO
21
22 SELECT * FROM view_kundenrechnungsadresse;
23 GO
24
25 SELECT 'VIEW-Verarbeitung (Seite 277)';
26 GO
27 CREATE VIEW view_artikel_aktiv
28    AS
29        SELECT * FROM artikel
30        WHERE deleted = 0
31 ;
32 GO
33 CREATE VIEW view_artikel_aktiv_tmp -- Nur um es später zu löschen!
34    AS
35        SELECT * FROM artikel
36        WHERE deleted = 0
37 ;
38 GO
39
40 SELECT FL00R(artikel_id / 1000) gruppe, AVG(einzelpreis)
41     FROM view_artikel_aktiv
42     GROUP BY FL00R(artikel_id / 1000)
43 ;
44 GO
45
46 -- EXPLAIN SELECT FL00R(artikel_id / 1000) gruppe, AVG(einzelpreis)
47 --   FROM view_artikel_aktiv
48 --   GROUP BY FL00R(artikel_id / 1000)
49 -- GO
50 -- ;
51
52
53 SELECT artikel_id, bezeichnung FROM
54     view_artikel_aktiv
55     WHERE
56         artikel_id BETWEEN 7000 AND 9000
57 ;
```

```
58 GO
59
60 -- EXPLAIN SELECT artikel_id, bezeichnung FROM
61 -- view_artikel_aktiv
62 -- WHERE
63 -- artikel_id BETWEEN 7000 AND 9000
64 -- GO
65
66
67 -- Start DROP VIEW
68 SELECT 'Ansicht loeschen (Seite 279)';
69 GO
70 DROP VIEW view_artikel_aktiv_tmp;
71 GO
72
73 DROP DATABASE IF EXISTS tmp1;
74 GO
75 CREATE DATABASE tmp1;
76 GO
77 USE tmp1;
78 GO
79 CREATE TABLE a (ai INT);
80 GO
81 CREATE TABLE b (bi INT);
82 GO
83 CREATE VIEW ab
84 AS
85     SELECT * FROM a INNER JOIN b ON ai = bi
86 ;
87 GO
88 DROP TABLE a;
89
90 -- SELECT 'Erwarte eine Fehlermeldung';
91 -- SELECT * FROM ab;
92 -- GO
93 -- Ende DROP VIEW
94 USE oshop;
95 GO
96 DROP DATABASE tmp1;
97 GO
98
99 SELECT 'Anwendungsgebiet: Vereinfachung (Seite 283)';
100 GO
101 CREATE OR ALTER
102     VIEW view_artikel_aktiv
103     AS
104         SELECT * FROM artikel
105             WHERE deleted = 0
106 ;
107 GO
108 CREATE OR ALTER VIEW view_artikel_verfuegbar
109 AS
110     SELECT a.artikel_id, bezeichnung
111         FROM artikel a INNER JOIN lagerbestand l ON a.artikel_id = l.artikel_id
112         WHERE menge_aktuell >= menge_mindest
113         -- ORDER BY a.artikel_id In T-SQL nicht möglich
114 ;
115 GO
116 SELECT * FROM view_artikel_verfuegbar;
```

```
117 GO
118 CREATE OR ALTER
119   VIEW view_lagerbestand_aktiv
120   AS
121     SELECT * FROM lagerbestand
122     WHERE deleted = 0
123 ;
124 GO
125 CREATE OR ALTER VIEW view_lagerbestand_artikelbezeichnung
126 AS
127   SELECT l.* , a.bezeichnung
128     FROM
129       view_lagerbestand_aktiv l INNER JOIN view_artikel_aktiv a
130       ON l.artikel_id = a.artikel_id
131 ;
132 GO
133 SELECT * FROM view_lagerbestand_artikelbezeichnung;
134 GO
135
136 SELECT 'Ende listing13.sql ///////////////////////////////';
```


Literatur

- [Ada14] ADAMS, Ralf: *Bug #71244 – Wrong result computation using ALL() and GROUP BY.* <http://bugs.mysql.com/bug.php?id=71244>, März 2014. – [Online: Stand 09.2019]
- [Arb17] ARBEZZANO, Gianluca: *Grundkurs Docker: Eine praktische Einführung in die Welt der Container.* <https://jaxenter.de/einfuehrung-docker-tutorial-container-61528>, September 2017. – [Online: Stand 07.2019]
- [Ass76] ASSOCIATION, American S.: American Standard Code for Information Interchange. In: ASA X3.4 (1976), Juni
- [Bar10] BARKOV, Alexander: *MySQL Collations by Character Set (as implemented in MySQL 6.0.4).* <http://collation-charts.org/mysql60/by-charset.shtml>, 2010. – [Online: Stand 09.2019]
- [Bun16] BUNDES BANK, Deutsche: *Bankleitzahlen.* http://www.bundesbank.de/Redaktion/DE/Standardartikel/Aufgaben/Unbarer_Zahlungsverkehr/bankleitzahlen_download.html, Januar 2016. – [Online: Stand 06.2016]
- [Che76] CHEN, Peter Pin-Shan: The Entity-Relationship Model – Toward a Unified View of Data. In: *ACM Transactions on Database Systems* 1 (1976), März, Nr. 1, S. 9–36
- [Cod70] CODD, Edgar F: A Relational Model of Data for Large Shared Data Banks. In: *Communications of the ACM* (1970), 13. Juni, S. 377–387. – ISSN 0001-0782
- [DE19] DB-ENGINES: *DB-Engines Ranking - Trend Popularity.* https://db-engines.com/en/ranking_trend, Juni 2019. – [Online: Stand 07.2019]
- [Doc17] DOCKERHUB: *Build and Ship any Application Anywhere.* <https://hub.docker.com>, Juni 2017. – [Online: Stand 07.2019]
- [EFH⁺] EDLICH, Stefan; FRIEDLAND, Achim; HAMPE, Jens; BRAUER, Benjamin; BRÜCKNER, Markus: *NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken.* 2. erweiterte Auflage. München: Carl Hanser Verlag. – ISBN 978-3-446-42855-3
- [Gar16] GARDEN, Euan: *SQL MythBusters – SQL Server is really a Sybase product not a Microsoft one.* <https://blogs.msdn.microsoft.com/euanga/2006/01/19/sql-mythbusters-sql-server-is-really-a-sybase-product-not-a-microsoft-one/>, Januar 2016. – [Online: Stand 07.2019]

- [Har] HARRISON, Guy: *Next Generation Databases: NoSQL And Big Data*. 1. Auflage. New York City, USA: Apress. – ISBN 978-1-4842-1330-8
- [Hel19] HELP, Software T.: *JSON Tutorial: Introduction and A Complete Guide for Beginners*. <https://www.softwaretestinghelp.com/json-tutorial/>, Juli 2019. – [Online: Stand 08.2019]
- [Inf19] INFORMATIONSTECHNIK, Bundesamt für Sicherheit in d.: *M 2.11 Regelung des Passwortgebrauchs*. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSIFB/sichere_passwoerter_faktenblatt.html, 2019. – [Online: Stand 09.2019]
- [Int13] INTERNATIONAL, ECMA: *The JSON Data Interchange Format*. <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>, 2013. – [Online: Stand 08.2019]
- [ISO85] ISO/TC97: *Information processing; Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts*. Februar 1985. – Entspricht der DIN 66001
- [JRH⁺04] Kapitel 3.4.7 Assoziationen. In: JECKLE, Mario; RUPP, Chris; HAHN, Jürgen; ZENGLER, Barbara ; QUEINS, Stefan: *UML 2 glasklar*. 1. Auflage. München: Carl Hanser Verlag, 2004. – ISBN 3-446-22575-7, S. 89 f.
- [KE01] KEMPER, Alfons; EICKLER, André: *Datenbanksysteme*. 4. Auflage. München: Oldenbourg, 2001. – ISBN 3-486-25706-4
- [Knu81] KNUTH, Donald E.: *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition*. Addison-Wesley, 1981. – ISBN 0-201-03822-6
- [Ltd10] LTD, Hughes Technologies P.: *Hughes Technologies*. <http://www.hughes.com.au>, August 2010. – [Online: Stand 09.2019]
- [Mar16] MARIADB: *MariaDB versus MySQL -- Kompatibilität*. <https://mariadb.com/kb/de/mariadb-vs-mysql-compatibility/#inkompatibilit%C3%A4ten-zwischen-mariadb-100-und-mysql-56>, Juni 2016. – [Online: Stand 09.2019]
- [Mar19] MARIADB: *Storage Engines*. <https://mariadb.com/kb/en/library/storage-engines/>, 2019. – [Online: Stand 07.2019]
- [Mik10] MIKIsoft: *MySQL Injection - Simple Load File and Into OutFile*. <https://www.exploit-db.com/papers/14635>, August 2010. – [Online: Stand 08.2019]
- [MOP06] MÖHRING, Rolf H.; OELLRICH, Martin ; PANKRATH, Robert: *Das Sieb des Eratosthenes, Wie schnell kann man alle Primzahlen bis eine Milliarde berechnen?* <http://www-i1.informatik.rwth-aachen.de/~algorithmus/algo25.php>, August 2006. – [Online: Stand 09.2019]
- [MyS18a] MySQL: *12.3.3 Logical Operators*. <https://dev.mysql.com/doc/refman/8.0/en/logical-operators.html>, 2018. – [Online: Stand 09.2019]

- [MyS18b] MySQL: *12.6.2 Mathematical Functions*.
<https://dev.mysql.com/doc/refman/8.0/en/mathematical-functions.html>, April 2018. – [Online: Stand 08.2019]
- [MyS18c] MySQL: *12.7 Date and Time Functions*.
<https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html>, April 2018. – [Online: Stand 08.2019]
- [MyS18d] MySQL: *13.2.11.4 Subqueries with ALL*.
<https://dev.mysql.com/doc/refman/8.0/en/all-subqueries.html>, April 2018. – [Online: Stand 08.2019]
- [MyS19a] MySQL: *Chapter 16 Alternative Storage Engines*.
<http://dev.mysql.com/doc/refman/8.9/en/storage-engines.html>, 2019. – [Online: Stand 07.2019]
- [MyS19b] MySQL: *Chapter 23 Writing a Custom Storage Engine*.
<http://dev.mysql.com/doc/internals/en/custom-engine.html>, 2019. – [Online: Stand 07.2019]
- [Nor85] NORMEN, Deutsches I.: *Sinnbilder für Struktogramme nach Nassi-Shneiderman*. November 1985. – auch: Nassi-Shneidermann-Diagramm
- [Oes06] Kapitel 4.4.2 Assoziationen. In: OESTEREICH, Bernd: *Analyse und Design mit UML 2.1*. 8. Auflage. Oldenbourg, 2006. – ISBN 978-3-486-57926-0, S. 271–282
- [Sei19] SEIDLER, Kai: *Apache Friends - Herunterladen*.
<http://www.apachefriends.org/de/xampp-windows.html>, Juni 2019. – [Online: Stand 07.2019]
- [Sha05] SHAFRANOVICH, Y.: *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. <http://tools.ietf.org/html/rfc4180>, Oktober 2005. – [Online: Stand 09.2019]
- [Sta18] STACKOVERFLOW: *Foreign key constraint may cause cycles or multiple cascade paths?* <https://stackoverflow.com/questions/851625/foreign-key-constraint-may-cause-cycles-or-multiple-cascade-paths>, Mai 2018. – [Online: Stand 07.2019]
- [SZT⁺09] Kapitel Optimale Datentypen auswählen. In: SCHWARZ, Baron; ZAITSEV, Peter; TKACHENKO, Vadim; ZAWDONY, Jeremy D.; LENTZ, Arjen ; BALLING, Derek J.: *High Performance MySQL*. 2. Auflage. Köln: O'Reilly, 2009. – ISBN 987-3-89721-889-5, S. 87 f.
- [The10] THE OPEN GEOSPATIAL CONSORTIUM: *OpenGIS Implementation Standard for Geographic Information – Simple feature access – Part 2: SQL option*.
<http://www.opengeospatial.org/standards/sfs>, 2010. – [Online: Stand 08.2019]
- [Tut19] TUTORIAL, MySQL: *An Introduction to MySQL CTE*.
<http://www.mysqltutorial.org/mysql-cte/>, August 2019. – [Online: Stand 08.2019]
- [Whe19] WHEAT, Jake: *Document*.
<https://jakewheat.github.io/sql-overview/sql-2016-foundation-grammar.html>, Juli 2019. – [Online: Stand 07.2019]
- [Wik16] WIKIPEDIA: *Unicode*. <http://de.wikipedia.org/wiki/Unicode#Gliederung>, April 2016. – [Online: Stand 09.2019]

- [Wik17] WIKIPEDIA: *IDEF1X*. <http://de.wikipedia.org/wiki/IDEF1X>, März 2017. – [Online: Stand 07.2019]
- [Wik18] WIKIPEDIA: *Chen-Notation*. <http://de.wikipedia.org/wiki/Chen-Notation>, Oktober 2018. – [Online: Stand 07.2019]
- [Wik19a] WIKIPEDIA: *B-Baum*. <http://de.wikipedia.org/wiki/B-Baum>, Juni 2019. – [Online: Stand 07.2019]
- [Wik19b] WIKIPEDIA: *Bankleitzahl*. <http://de.wikipedia.org/wiki/Bankleitzahl>, Juli 2019. – [Online: Stand 08.2019]
- [Wik19c] WIKIPEDIA: *Cobol*. <http://de.wikipedia.org/wiki/Cobol#Datendeklarationen>, Juni 2019. – [Online: Stand 07.2019]
- [Wik19d] WIKIPEDIA: *Data-Warehouse*. <http://de.wikipedia.org/wiki/Datawarehouse>, März 2019. – [Online: Stand 07.2019]
- [Wik19e] WIKIPEDIA: *Eierkennzeichnung*.
<https://de.wikipedia.org/wiki/Eierkennzeichnung>, Juni 2019. – [Online: Stand 07.2019]
- [Wik19f] WIKIPEDIA: *Entity-Relationship-Modell*.
<http://de.wikipedia.org/wiki/ER-Modell>, Juni 2019. – [Online: Stand 07.2019]
- [Wik19g] WIKIPEDIA: *Fahrzeug-Identifizierungsnummer*.
<http://de.wikipedia.org/wiki/Fahrzeug-Identifizierungsnummer>, Juli 2019. – [Online: Stand 07.2019]
- [Wik19h] WIKIPEDIA: *Include*. <http://de.wikipedia.org/wiki/Copybook>, Januar 2019. – [Online: Stand 07.2019]
- [Wik19i] WIKIPEDIA: *Internationale Bankkontonummer*.
https://de.wikipedia.org/wiki/Internationale_Bankkontonummer, Juni 2019. – [Online: Stand 07.2019]
- [Wik19j] WIKIPEDIA: *Internationale Standardbuchnummer*.
<http://de.wikipedia.org/wiki/ISBN>, Juli 2019. – [Online: Stand 07.2019]
- [Wik19k] WIKIPEDIA: *Kraftfahrzeugkennzeichen*.
<http://de.wikipedia.org/wiki/Kraftfahrzeugkennzeichen>, Mai 2019. – [Online: Stand 07.2019]
- [Wik19l] WIKIPEDIA: *Martin-Notation*. <https://de.wikipedia.org/wiki/Martin-Notation>, April 2019. – [Online: Stand 07.2019]
- [Wik19m] WIKIPEDIA: *Relationale Datenbank*.
http://de.wikipedia.org/wiki/Relationale_Datenbank, Mai 2019. – [Online: Stand 07.2019]
- [Wik19n] WIKIPEDIA: *Rufnummer*. <https://de.wikipedia.org/wiki/Rufnummer>, Juni 2019. – [Online: Stand 07.2019]
- [Wik19o] WIKIPEDIA: *SELECT (SQL)*.
https://en.wikipedia.org/wiki>Select_%28SQL%29#Result_limits, Februar 2019. – [Online: Stand 08.2019]
- [Wik19p] WIKIPEDIA: *SQL*. <http://de.wikipedia.org/wiki/SQL>, Juli 2019. – [Online: Stand 07.2019]

Stichwortverzeichnis

, 418	ALTER SERVER, 420
0, 145	ALTER TABLE, 129, 420
*, 162, 408	– ... ADD, 131, 149
+, 408	– ... ADD FOREIGN KEY, 176
-, 408	– ... ADD INDEX, 176
-, 165	– ... ADD PRIMARY KEY, 176
.NET, 6	– ... DISABLE KEYS, 104
/, 408	– ... DROP, 137
;, 67	– ... DROP FOREIGN KEY, 175
<, 415	– ... DROP INDEX, 175
<=, 415	– ... DROP PRIMARY KEY, 175
<=>, 414	– ... ENABLE KEYS, 104
<>, 415	– ... MODIFY, 132
=, 414	– ... RENAME, 130
>, 415	ALTER TABLESPACE, 422
>=, 415	ALTER USER, 435
% ¹ , 408	– aktueller Benutzer, 436
&&, 417	– Rolle, 436
_id, 294	ALTER VIEW, 282, 422
!, 416	AND, 417
!=, 415	Änderung, kaskadierende, 34, 86
	Änderungsweitergabe, 34, 86
Abhangigkeit, mehrwertige, 25	Annehmende Schleife, 345
ABS(), 408	ANSI, 64
Abweisende Schleife, 345	Ansicht, 4, 273
ACID, 323	– Projektions-, 285
ACOS(), 408	– Selektions-, 283
AFTER, 131	– veranderbare, 287
Aggregatfunktion, 211	– Verbund-, 284
Alias, 162, 207	Antivalenz, 418
ALL, 233, 234	ANY(), 237, 238
ALL(), 238	ApacheFriends, 53
ALTER DATABASE, 125, 419	API, 6
ALTER EVENT, 374, 419	ARCHIVE, 386
ALTER FUNCTION, 419	Aria, 386
ALTER INSTANCE, 420	Artikelverwaltung, 42
ALTER LOGFILE GROUP, 420	AS IDENTITY, 75
ALTER PROCEDURE, 420	ASC, 164
ALTER SCHEMA, 125, 419	ASIN(), 408

- ATAN(), 408
 ATAN2(), 408
 Atomare Tabelle, 36
 Atomarität, 323
 Atomicity, 323
 Attribut, 16
 Aufzählung, 77
 AUTO_INCREMENT, 75, 153, 406
 AUTOCOMMIT, 324
 AVG(), 212, 411
 AVG(DISTINCT), 411
- B-Baum, 93
 Bank, 41
 Bankverbindung, 41
 BCNF, 39
 Bedingung, 141
 BEGIN ... END, 335
 Benutzerauthentifizierungsoption, 437
 Benutzerrecht, 379
 Benutzerspezifikation, 437
 Bestellwesen, 43
 BETWEEN, 415
 Bewegungsdaten, 147, 195
 BIGINT, 397
 BINARY, 143, 399, 406
 BIT, 397
 BIT_AND(), 411
 BIT_OR(), 412
 BIT_XOR(), 412
 BLACKHOLE, 386
 BLOB, 116, 403
 BOOL, 397
 Breitenabdeckung, 399
 BSI, 383
 Bücherei, 31
 BULK INSERT, 107, 110
- C, 6
 C++, 6
 C#, 6
 Cache, 9
 CALL, 431
 CASCADE, 68, 85, 88, 140
 CASE, 261, 342
 Case insensitive, 457
 Case sensitive, 457
 Cassandra, 386
 CD-Sammlung, 32
 CEILING(), 409
 CHAR, 399
 CHARACTER SET, 68, 166
- CHECK, 406
 CHECK TABLE, 280
 Chen, Jolly, 9
 Chen-Notation, 22
 ci, 457
 CLOSE, 353
 Clown-Agentur, 31
 COBOL Data Division, 12
 Codd, Dr. E. F., 11
 Codepage 850, 69
 COLLATE, 70
 collection, 294
 Common Table Expression, 208, 435
 Compilezeit, 158
 CONCAT(), 149, 221
 Condition, 141
 conditional comment, 126
 CONNECT, 386
 CONNECT_TIMEOUT, 336
 Connection-Pool, 8
 Connector, 6
 Consistency, 323
 CONSTRAINT, 85
 constraint check, 6
 CONV(), 409
 COPY, 107
 COS(), 409
 COT(), 409
 COUNT(), 212, 412
 COUNT(*), 212, 412
 COUNT(DISTINCT), 212, 412
 cp850, 70
 CRC32(), 409
 CREATE DATABASE, 66, 422
 CREATE EVENT, 371, 422
 CREATE FUNCTION, 361, 423
 CREATE INDEX, 97, 423
 CREATE LOGFILE GROUP, 424
 CREATE PROCEDURE, 334, 424
 CREATE ROLE, 436
 CREATE SCHEMA, 66, 425
 CREATE SERVER, 425
 CREATE SPATIAL REFERENCE SYSTEM, 425
 CREATE TABLE, 74, 87, 89, 426
 – ... LIKE, 428
 – ... SELECT, 428
 CREATE TABLESPACE, 428
 CREATE TEMPORARY TABLE, 193, 226
 CREATE TRIGGER, 364, 429
 CREATE USER, 382, 436
 CREATE VIEW, 429
 CROSS JOIN, 186

- Crowfoot, 23
cs, 457
CSV, 386
CSV-Format, 106
CTE, 271, 308, 435
CURSOR, 353
- Data Control Language, 63
Data Definition Language, 63
Data Manipulation Language, 63
DATE, 400
DATE(), 137
DATE_FORMAT(), 402
Dateisystem, 9
Daten
– Bewegungs-, 147, 195
– redundante, 209
– Stamm-, 147, 195
Datenbank, 3, 5
– Abgrenzung, 11
– anlegen, 66
– hierarchische, 12
– löschen, 68
– objektorientierte, 13
– relationale, 11
Datenbankmanagementsystem, 5
Datenbanksystem, 3, 6
– Client-Server-, 8
Datenfeld, 16
Datenflussdiagramm, 22
Datensatz, 16
Datenschutz, 6, 285
Datensicherheit, 6
Datentyp, 73
Datenverzeichnis, 56
DATETIME, 400
Dauerhaftigkeit, 323
DBMS, 5
DCL, 63
DDL, 63, 419
Deadlock, 331, 332
DECIMAL, 78, 398, 439
DECLARE, 335
– ... FOR CURSOR, 353
– CONTINUE, 353
– EXIT, 353
Dedicated Computer, 50
DEFAULT, 406
DEGREES(), 409
Deklarative Programmiersprache, 63
DELETE, 151, 431
– ... IGNORE, 153
- ... LOW_PRIORITY, 153
– ... QUICK, 153
deleted, 33
DELIMITER, 335
DESC, 164
DESCRIBE, 77, 131
Deutscher Brauereiverband, 32
Deutscher Wetterdienst, 25
Developer Server, 47
Development Computer, 50
Differenzmenge, 256, 259
DIN, 64
DIN 5007-1, 71
DIN 5007-2, 71
DIN 66001, 22
DIN 66261, 22
dirty read, 324
DISABLE KEYS, 104
Disjunktion, 418
DISTINCT, 173
DIV, 408
DML, 63, 431
DO, 432
Docker, 57
– MariaDB, 60
– MS SQL Server, 62
– MySQL, 58
– PostgreSQL, 61
Domäne, 15, 16
DOUBLE, 78, 398, 439
DOUBLE PRECISION, 398
Drei-Schichten-Architektur, 16, 80
Dritte Normalform, 39
DROP DATABASE, 127, 429
DROP EVENT, 374, 429
DROP FUNCTION, 362, 429
DROP INDEX, 104, 429
DROP LOGFILE GROUP, 430
DROP PROCEDURE, 338, 430
DROP ROLE, 436
DROP SCHEMA, 127, 430
DROP SERVER, 430
DROP SPATIAL REFERENCE SYSTEM, 430
DROP TABLE, 138, 430
DROP TABLESPACE, 430
DROP TRIGGER, 364, 430
DROP USER, 380, 383, 436
DROP VIEW, 279, 430
DSGVO, 33
Dublette, 100
Durability, 323

- Eigenschaft, 16
- 1:1-Verknüpfung, 25
- 1:n-Verknüpfung
 - Definition, 27
 - identifizierende, 27
- ENABLE KEYS, 104
- Engine, 5, 9, 83
 - ARCHIVE, 386
 - Aria, 386
 - BLACKHOLE, 386
 - Cassandra, 386
 - CONNECT, 386
 - CSV, 386
 - EXAMPLE, 386
 - FEDERATED, 386
 - FederatedX, 386
 - InnoDB, 386
 - MEMORY, 387
 - MERGE, 387
 - MyISAM, 387
 - OQGRAPH, 387
 - XtraDB, 387
- Entität, 16
- Entitätentyp, 16
 - schwacher, 18
 - starker, 18
- Entity, 16
- Entity Relationship Model, 22
- Entitytype, 16
- ENUM, 77, 397
- EQUI JOIN, 197
- ER-Modell, 22
- Eratosthenes, 348
- Ereignis, 5, 371
- ERM, 22
- Erste Normalform, 36
- Event, 371
- EXAMPLE, 386
- EXCEPT, 256, 257, 259
- Existenzabhängige Tabelle, 18
- Existenzunabhängige Tabelle, 18
- EXISTS, 241
- EXIT, 66
- EXP(), 409
- Experiment
 - DOUBLE vs. DECIMAL, 439
 - Einfügen mit Index, 449
 - Indexselektivität, 452
 - NULL vs. NOT NULL, 444
 - Rundungsfehler, 443
 - Sortierung, 454
 - Suchen, 446
- EXPLAIN, 170
 - EXTENDED, 248
- Exportieren
 - Binärdaten
 - C#, 181
 - CSV-Daten, 179
- Fallunterscheidung, 261, 342
- FEDERATED, 386
- FederatedX, 386
- Feld, 16
- FETCH, 353
- FIRST, 131
- FLOAT, 398
- FLOOR(), 160, 409
- FOR ORDINALITY, 307
- FOREIGN KEY, 83, 406
- foreign key, 20
- FORMAT(), 264, 409
- Formatierungszeichen, 402
 - Datum, 401
 - Uhrzeit, 402
- Fremdschlüssel, 20, 23, 82
 - festlegen, 82
- FULL OUTER JOIN, 201
- Funktion, 362
- GENERATED, 75
- GENERATED ...AS IDENTITY, 406
- GENERATED ...AS PRIMARY KEY, 406
- generierte Spalten, 407
- GEOMETRY, 405
- GEOMETRYCOLLECTION, 405
- Geschlossene Schleife, 345
- GET_FORMAT(), 402
- GLOBAL, 91
- Globale Variable, 336
- GRANT, 383, 436
 - ALL PRIVILEGES, 384
 - CHARACTER SET, 384
 - COLLATION, 384
 - DOMAIN, 384
 - FUNCTION, 384
 - PROCEDURE, 384
 - PROXY, 436
 - Rolle, 437
 - TABLE, 384
 - TRANSLATION, 384
 - WITH GRANT OPTION, 384
- Grantoption, 438
- GROUP BY, 214
- GROUP BY ...WITH ROLLUP, 217

- GROUP_CONCAT(), 300, 412
Hauptanweisung, 228
HAVING, 218, 219
Herz, 157
HEX(), 410
Hierarchische Datenbank, 12
Hilfstabelle, 30

iconv, 70
IDEF1X-Notation, 22
Identifizierende 1:n-Verknüpfung, 27
IF, 340
IF EXISTS, 68
IF NOT EXISTS, 67
IGNORE, 111
Imperative Programmiersprache, 63
Importieren
– Binärdaten
– C#, 116
– LOAD FILE, 119
– CSV-Daten, 107
– XML-Daten, 354
IN(), 232, 238, 415
Index, 4, 93, 446, 449, 452, 454
– anlegen, 97
– automatisch, 95
– Dublette, 100
– löschen, 104
– Schlüsseleigenschaft, 99
Indexselektivität, 102, 452
Informix, 9
INHERITS, 90
INNER JOIN, 187
InnoDB, 386
InnoDB Cluster, 49
INSERT, 380
INSERT INTO
– ... SELECT, 121, 432
– ... SET, 114, 432
– ... VALUES, 113, 432
INT, 397
Integrität, referenzielle, 33
Interpreter, 5
INTERSECT, 254, 259
IS FALSE, 416
IS NOT NULL, 416
IS NULL, 416
IS TRUE, 416
IS UNKNOWN, 416
ISAM, 7
ISO, 64
ISO/IEC 5807, 22
ISO/IEC 8859-1, 69
ISO/IEC 8859-2, 69
ISO/IEC 8859-9, 69
ISO/IEC 8859-13, 69
ISO/IEC 9075:1999, 64
ISO/IEC 9075:2011, 64
ISO/IEC 9075, 402
Isolation, 323
Item, 16
ITERATE, 346

JavaScript
– *collection*
– add(), 295
– arrayDelete(), 303
– arrayInsert(), 296
– find(), 296
– modify(), 296
– remove(), 295
– *object*
– hasOwnProperty(), 298
– keys, 298
– *parameter*
– bind(), 296
– *result*
– fetchAll(), 295, 300
– fetchOne(), 296
– *schema*
– dropCollection(), 294
– getCollection(), 295
– getCollections(), 294
– *session*
– close(), 294
– getSchema(), 294
– sql(), 296, 300
– *statement*
– execute(), 296
– *table*
– select(), 295
– for ... in, 295
– function, 294
– require(), 294
– return, 294
JDBC, 6
JOIN
– ... ON, 187
– ... USING, 192
– CROSS, 186
– EQUI, 197
– INNER, 187
– NATURAL, 192

- OUTER, 201
 - FULL, 201, 204
 - LEFT, 201
 - RIGHT, 201
- SELF, 206
- JSON, 291, 404
- JSON_ARRAY(), 305
- JSON_ARRAY_APPEND(), 306
- JSON_ARRAYAGG(), 412
- JSON_EXTRACT(), 306
- JSON_OBJECT(), 305
- JSON_OBJECTAGG(), 413
- JSON_PRETTY(), 306
- JSON_REMOVE(), 312
- JSON_REPLACE(), 310, 312
- JSON_SEARCH(), 309
- JSON_SET(), 312

- Kalender
 - gregorianisch, 400
 - julianisch, 400
 - proleptischer gregorianischer, 400
- Kardinalität, 24
- Kartesisches Produkt, 185
- Kaskadierende Änderung, 34, 86
- Kaskadierendes Löschen, 33, 85
- key, 17
- Klammern, 145
- Klasse, 16
- Kommentar
 - bedingter, 126
 - einzeilig, 165
- Konjunktion, 417
- Konnektor, 6, 8
- Konsistenz, 323
- Konstante, 158
- Kopf-/Fußgesteuerte Schleife, 345
- Korrelierende Unterabfrage, 228
- Krähenfuß-Notation, 23
- Kreuzprodukt, 186
- Kunde, 41
- Kundenverwaltung, 41

- LAST_INSERT_ID(), 320
- latin1, 70
- Laufzeit, 158
- LE, 457
- LEAVE, 346
- LEFT OUTER JOIN, 201
- LIKE, 89, 416
- LIMIT, 177
- LINESTRING, 405

- Listenunterabfragen, 232
- little endian, 457
- LN(), 410
- LOAD DATA INFILE, 107, 109, 433
- LOAD FILE, 119
- LOAD XML, 355, 433
- LOCAL, 91
- lock, 316
- LOCK TABLES, 317
- locking, 316
- LOG(), 410
- LOG10(), 410
- LOG2(), 410
- lokale Variable, 335
- LONGBLOB, 403
- LONGTEXT, 399
- LOOP-Schleife, 346
- Löschen, kaskadierendes, 33, 85
- Löschkennzeichen, 33
- Löschweitergabe, 33, 85
- lost update, 316

- MariaDB Client, 65
- Martin-Notation, 23
- Matrix, 16
- MAX(), 213, 413
- MEDIUMBLOB, 403
- MEDIUMINT, 397
- MEDIUMTEXT, 399
- Mehrwertige Abhängigkeit, 25
- MEMORY, 387
- Mengenverhältnis, 24
- MERGE, 387
- MERGED, 276
- MIN(), 213, 413
- Minimalität des Schlüssels, 17
- Minimumexistenz, 165
- MOD, 408
- MOD(), 410
- Modellierung, 22
- Monolithische Anwendung, 80
- MONTH(), 221
- MONTHNAME(), 221
- MULTILINESTRING, 405
- MULTIPOINT, 404
- MULTIPOLYGON, 405
- MyISAM, 387
- MySQL, 7
- mysql -e, 180
- MySQL Client, 65
- MySQL Query Browser, 65
- MySQL Workbench, 23

- mysqladmin, 57
- mysqldump, 377
- mysqlsh, 290

- n:m-Verknüpfung, 29
- n:m:k-Verknüpfung, 30
- Nassi-Shneidermann-Diagramm, 22
- NATURAL JOIN, 192
- Negation, 416
- NEW, 364
- Nicht identifizierende 1:n-Verknüpfung, 27
- Nicht korrelierende Unterfrage, 228
- NO ACTION, 88
- Normalform, 34
 - Boyce-Codd, 39
 - erste, 36
 - zweite, 38
 - dritte, 39
 - vierte, 40
 - fünfte, 40
- Normalisierung, 35, 37
- NOT, 416
- NOT BETWEEN, 415
- NOT FOUND, 353
- NOT IN, 415
- NOT IN(), 257
- NOT LIKE, 416
- NOT NULL, 80, 406, 444
- Notation
 - Chen, 22
 - Crowfoot, 23
 - IDEF1X, 22
 - Krähenfuß, 23
 - Martin, 23
 - UML, 23
- NULL, 80, 406, 444
- NUMERIC, 398
- Nummernkreis, 77

- Oberanweisung, 228
- Objekt, 16
- Objektorientierte Datenbank, 13
- Objekttyp, 438
- ODBC, 6
- Offene Schleife, 345
- OGC, 404
- OLD, 364
- Online-Shop, 41
 - Tabelle adresse, 41
 - Tabelle artikel, 42
 - Tabelle artikel_nm_lieferant, 91
 - Tabelle artikel_nm_warengruppe, 112

- Tabelle bank, 41
- Tabelle bankverbindung, 41
- Tabelle bestellung, 43
- Tabelle bild, 116
- Tabelle kunde, 41
- Tabelle lagerbestand, 214
- Tabelle lieferant, 42
- Tabelle position_bestellung, 43
- Tabelle position_rechnung, 43
- Tabelle rechnung, 43
- Tabelle warengruppe, 42
- OPEN, 353
- OpenGIS, 404
- Operatorenpriorität, 159
- Operatorenrangfolge, 159
- Optimierer, 5, 9
- OQGRAPH, 387
- OR, 418
- Oracle, 7
- ORDER BY, 163
- Ordnung, 165
- Orgienjoin, 186
- OUTER JOIN, 201

- page lock, 316
- Parameterbindung, 296, 297
- Parser, 9
- PASSWORD(), 382
- Passwortoption, 438
- Performancemessung, 439, 446, 449, 454
- Perl, 6
- PHP, 6
- PI(), 410
- Platzhalter, 162
- Plausibilisierung, 16
- Plausibilitätsüberprüfung, 339
- POINT, 404
- POLYGON, 405
- PostgreSQL, 9
- POSTQUEL, 9
- POWER(), 410
- Primärschlüssel, 19, 23
- PRIMARY KEY, 406
- primary key, 19
- Privileg, 379
- Privilegtiefe, 438
- Programmablaufplan, 22
- Programmierschnittstelle, 6
- Programmiersprache
 - deklarative, 63
 - imperativ, 63
- Programmverzeichnis, 55

- Projektionsansicht, 285
- Property, 16
- Prozedur, 4
- Puh, 130
- Python, 6
- räumliche Datentypen, 404
- RADIANS(), 410
- RAND(), 160, 410
- Randbedingungsprüfer, 6
- RDBMS, 6
- read only, 407
- REAL, 398
- Record, 16
- Recordset, 16
- Redundante Daten, 209
- Redundanz, 20, 34
- REFERENCES, 83
- Referenz, 20
- referenzielle Integrität, 33, 41, 85
- reflection, 298
- Relation, 11, 16
- Relationale Datenbank, 11
- relationales Datenbankmanagementsystem, 6
- RENAME TABLE, 431
- RENAME USER, 437
- REPLACE, 111
- Ressourceoption, 438
- RESTRICT, 68, 87, 88, 140
- REVOKE, 380, 385, 437
 - ALL, 437
 - ALL PRIVILEGES, 386
 - PROXY, 437
 - Rolle, 437
- RIGHT OUTER JOIN, 201
- ROLLBACK, 325
- ROUND(), 149, 411
- row lock, 317
- Rundungsfehler, 79, 411, 443
- Sakila, 7
- Satzkennzeichen, 12
- Schema, 16, 67
- Schlüssel
 - Definition, 17
 - Fremd-, 20
 - Kandidat-, 19
 - Minimalität des, 17
 - Primär-, 19
 - Sekundär-, 19
 - Schleife, 345
 - abweisende, 345
 - annehmende, 345
 - fußgesteuerte, 345
 - geschlossen, 345
 - kopfgesteuerte, 345
 - offene, 345
- Schnittmenge, 254, 259
- Schwache Tabelle, 18
- Seitensperre, 316
- Sekundärschlüssel, 19
- SELECT, 157, 433
 - ... INTO, 161, 178, 337
 - ... INTO OUTFILE, 180
- Selektionsansicht, 283
- SELF JOIN, 205, 206
- Semikolon, 67
- Seq_in_index, 98
- Server Computer, 50
- session variable, 336
- SET, 337, 397
- SET DEFAULT, 88
- SET GLOBAL, 373
- SET NAMES, 166
- SET NULL, 88
- SET PASSWORD, 437
- SHOW
 - CHARACTER SET, 69
 - COLLATION, 70
 - CREATE DATABASE, 126
 - CREATE SCHEMA, 126
 - CREATE TABLE, 84, 153
 - DATABASES, 72
 - EVENTS, 372
 - FULL TABLES, 275
 - GRANTS FOR, 382
 - INDEX, 95
 - PROCEDURE STATUS, 338
 - SCHEMAS, 72
 - TABLES, 76
 - TRIGGERS, 369
 - VARIABLES, 373
 - VARIABLES LIKE, 66
 - VIEWS (work around), 275
 - WARNINGS, 67, 104, 108
- Sieb des Eratosthenes, 348
- SIGN(), 411
- SIN, 411
- Single Responsibility Principle, 81
- Sitzung, 8
- Sitzungsvariablen, 336
- Sitzungsverwaltung, 6
- Skalarunterabfrage, 228, 229
- SMALLINT, 397

- Sortierreihenfolge, 70
- Sortierung
 - deutsch, 458
 - zuweisen, 70
- sp_rename, 130
- Spalte
 - Auswahl einer, 162
 - Definition, 15
 - Spezifikation einer, 74
- spatial data types, 404
- Sperroption, 438
- Sprunglogik, 417
- SQL, 63
- SQL HANDLER, 353
- SQL-Injection, 118, 120, 180, 297
- SQL-Schnittstelle, 9
- SQL_NO_CACHE, 442
- SQLEXCEPTION, 353
- SQLWARNING, 353
- SQRT(), 411
- SRP, 81
- Stammdaten, 147, 195
- Standalone MySQL Server, 49
- Standardwert, 405
- Starke Tabelle, 18
- START TRANSACTION, 323
- STD(), 413
- STDDEV(), 413
- STDDEV_POP(), 413
- STDDEV_SAMP(), 413
- Stonebraker, Michael, 9
- Storage Engine, 5, 9
- strict-Modus, 380
- String, 397
- Struktogramm, 22
- Stundenplan-Software, 32
- SUBSELECT, 227
- SUBSTRING(), 220
- Suchpfad, 57
- SUM(), 213, 413
- SUM(DISTINCT), 414
- Sun Microsystems, 7

- Tabelle, 4, 16
 - anlegen, 72
 - atomar, 36
 - existenzabhängige, 18
 - existenzunabhängige, 18
 - herleiten, 89
 - schwach, 18
 - starke, 18
 - teilfunktional, 37
- temporär, 90, 222
- transitiv, 39
- vollfunktional, 37
- wiederholungsgruppenfrei, 35
- Tabellenreferenzen, 434
- Tabellensperre, 316
- Tabellenunterabfragen, 239
- table lock, 316
- TAN(), 411
- Tcl, 6
- Teilfunktionale Tabelle, 37
- Temporäre Tabelle, 4, 90, 222
- TEMPORARY, 90
- TEMPTABLE, 276
- TEXT, 399
- Tiefenabdeckung, 399
- TIME, 400
- TIME(), 169
- TIME_FORMAT(), 402
- Timeout, 6, 66
- TIMESTAMP, 400
- TINYBLOB, 403
- TINYINT, 397
- TINYTEXT, 399
- Transact-SQL, 64
- Transaktion, 323
- Transaktionsmanagement, 6
- Transitive Tabelle, 39
- Transitivität, 165
- Trennzeichen, 106
- Trichotomie, 165
- Trigger, 5, 363
- TRUNCATE, 154, 431
- TRUNCATE(), 411
- Tupel, 16
- Twebaze, Ambrose, 8

- Übergabeparameter, 336
- UML-Notation, 23
- UNDER, 89
- Unicode, 69
- UNION, 251, 258, 434
- UNIQUE, 406
- UNKNOWN, 141
- UNLOCK TABLES, 317
- UNSIGNED, 75, 406
- Unterabfrage, 227
 - korrelierende, 228
 - Listen-, 232
 - nicht korrelierende, 228
 - skalar, 228, 229
 - Tabellen-, 239

- UPDATE, 147, 435
 - ... IGNORE, 150
 - ... LOW_PRIORITY, 150
- USE, 75
- USING, 192
- utf8, 69, 70
- utf16, 69, 70
- UTF16LE, 457
- utf32, 69, 70
- VAR_POP(), 414
- VAR_SAMP(), 414
- VARCHAR, 75, 399
- Variable, 161, 178
 - global, 336
 - lokal, 335
 - Sitzungs-, 336
- VARIANCE(), 414
- Veränderbare Ansicht, 287
- Verbinder, 8
- Verbundansicht, 284
- Vereinigung, 251, 258
- Vererbung, 89
- Verknüpfung, 20
 - 1:1, Definition, 25
 - 1:n
 - Definition, 27
 - identifizierende, 27
 - n:m, Definition, 29
 - n:m:k, 30
- Verletzte referenzielle Integrität, 33
- Verschlüsselung, 438
- Verzeichnis
 - Daten, 56
 - Programm, 55
 - XAMPP, 55
- Verzweigung, 339
- VIEW, 273
- Vollfunktionale Tabelle, 37
- Vorbelegungen, 405
- Wartungsinstabilität, 81
- Wartungsstabilität, 24
- Wertebereich, 15
- WHERE-Klausel, 141, 142, 219
- WHILE-Schleife, 348
- Widenius, Michael, 7
- Wiederholungsgruppe, 30, 35
- Wiederholungsgruppenfreiheit, 35
- Windows 10, 53
- Windows Service, 51
- WITH, 435
 - WITH RECURSIVE, 208
 - WITH ROLLUP, 217
- XAMPP, 54
- XAMPP-Verzeichnis, 55
- XML-Format, 13
- XtraDB, 387
- YEAR, 400
- YEAR(), 221
- Yu, Anrew, 9
- Zeichenketten, 114
- Zeichensatz, 69
 - deutsch, 457
 - zuweisen, 68
- Zeile
 - Auswahl einer, 163
 - Definition, 16
- Zeilensperre, 317
- Zeilenumbruch, 106
- Zufallszahlen, 160
- Zweite Normalform, 38
- Zwischenspeicher, 9

SQL //

- Lernen Sie SQL ohne Vorwissen über Datenbanken.
- Erfahren Sie, wie Datenbanken geplant und beschrieben werden.
- Programmieren Sie alle Beispiele selbst nach.
- Alle Beispiele sind für MySQL/MariaDB, PostgreSQL und T-SQL getestet.
- Im Internet: Die Beispieldatenbank und die Lösungen zu den Übungen des Buches



Ralf ADAMS,
Diplom-Informatiker, ist Lehrer
für Informatik
(Schwerpunkt
Anwendungs-
entwicklung) am

Technischen Berufskolleg 1
der Stadt Bochum. Von 2003 bis
2016 war er Mitglied des IHK-
Prüfungsausschusses für Fach-
informatiker/Anwendungsent-
wicklung in Bochum. Seine Erfah-
rungen im Schulbetrieb und
im Prüfungsausschuss sind in
das Buch mit eingeflossen.

Wer in seiner Ausbildung oder im beruflichen Alltag mit Datenbanken zu tun hat, braucht auch Kenntnisse in der Datenbanksprache SQL. Einsteiger werden in diesem Buch Schritt für Schritt an die Arbeit mit SQL herangeführt. Vom Aufbau über das Ändern einer Datenbank und die Auswertung der Daten bis hin zur Administration lernen Sie alle wesentlichen Aufgabenstellungen kennen.

In dieser aktualisierten Neuauflage geht der Autor neben MySQL und MariaDB auch auf PostgreSQL und T-SQL ein. Anhand einer Beispieldatenbank erfahren Sie, wie Sie SQL sinnvoll anwenden. Ebenfalls neu ist ein einführendes Kapitel zu den NoSQL-Themen JSON und JavaScript-Client-Programmierung in der erweiterten MySQL-Shell.

Zudem erhalten Sie eine kurze Einführung in die wichtigsten Grundbegriffe und Designregeln für relationale Datenbanken wie ER-Modell, Schlüssel, referenzielle Integrität und Normalformen.

Jedes Kapitel enthält Übungen, mit denen Sie lernen, Ihr frisch erworbene Wissen umzusetzen. Die Lösungen finden Sie im Internet. Wenn Sie SQL-Befehle einfach nachschlagen wollen, hilft Ihnen der MySQL-Befehlsindex am Ende des Buches.

Das E-Book enthält zusätzlich alle MySQL/MariaDB-, PostgreSQL- und T-SQL-Skripte zum schnellen Nachschlagen und Vergleichen.

AUS DEM INHALT //

- Einführung in die Grundbegriffe der relationalen Datenbank und der Theorie (ER-Modell, Normalformen etc.)
- Modellierung eines Online-Shops
- Anlegen/Ändern und Löschen von Datenbanken, Tabellen und Indizes
- Importieren/Exportieren von CSV- und Binärdaten
- Auswertungen mit SELECT: Von einfach bis schwer
- NoSQL: JSON und JavaScript-Client-Programmierung
- Locking und Transaktionen
- Prozeduren und Cursor
- Trigger und Events
- Benutzer und Benutzerrechte
- Daten sichern und wiederherstellen
- Der MySQL-Client
- Referenz der SQL-Befehle, Datentypen, Operatoren und einiger Funktionen

HANSER

