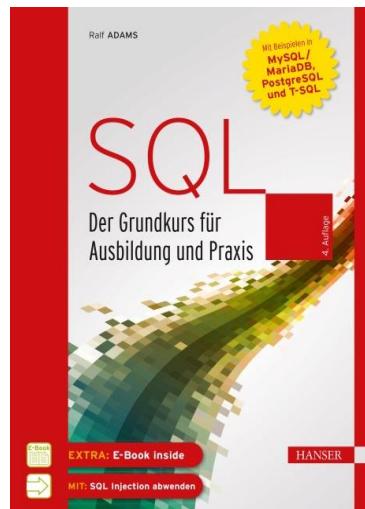


HANSER



Leseprobe

zu

SQL: Der Grundkurs für Ausbildung und Praxis

von Ralf Adams

Print-ISBN: 978-3-446-47168-9

E-Book-ISBN: 978-3-446-47220-4

Weitere Informationen und Bestellungen unter

<https://www.hanser-kundencenter.de/fachbuch/artikel/9783446471689>

sowie im Buchhandel

© Carl Hanser Verlag, München

Inhalt

Vorwort zur 4. 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 Relationale Datenbanken	11
2.1 Ein Einführung	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	21
2.2.1 Darstellung von Tabellen im ER-Modell	22
2.2.2 <i>1:1</i> -Verknüpfung	23
2.2.2.1 Wann liegt eine <i>1:1</i> -Verknüpfung vor?	23
2.2.2.2 Wie kann ich eine <i>1:1</i> -Verknüpfung darstellen?	25
2.2.2.3 Kann ich die Kardinalität genauer beschreiben?	25
2.2.3 <i>1:n</i> -Verknüpfung	26
2.2.3.1 Wann liegt eine <i>1:n</i> -Verknüpfung vor?	26
2.2.3.2 Wie kann ich eine <i>1:n</i> -Verknüpfung darstellen?	27

2.2.3.3	Kann ich die Kardinalität genauer beschreiben?	27
2.2.4	<i>n:m</i> -Verknüpfung	28
2.2.4.1	Wann liegt eine <i>n:m</i> -Verknüpfung vor?	28
2.2.4.2	Wie kann ich eine <i>n:m</i> -Verknüpfung darstellen?	29
2.2.4.3	Kann ich die Kardinalität genauer beschreiben?	30
2.2.5	Aufgaben zum ER-Modell	30
2.3	Referenzielle Integrität	31
2.3.1	Verletzung der referenziellen Integrität durch Löschen	32
2.3.2	Verletzung der referenziellen Integrität durch Änderungen	33
2.4	Normalformen	33
2.4.1	Normalform 1	34
2.4.2	Normalform 2	36
2.4.3	Normalform 3	37
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	58
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 rufe ich den MySQL Client auf?	65
5.2.2	Wie lege ich eine Datenbank an?	66
5.2.3	Wie lösche ich eine Datenbank?	68
5.2.4	Wie weise ich einen Zeichensatz zu?	68
5.2.5	Wie weise ich eine Sortierung zu?	70

5.3	Anlegen der Tabellen	72
5.3.1	Welche Datentypen gibt es?.....	73
5.3.2	Wie lege ich eine Tabelle an?.....	73
5.3.3	Wann eine Aufzählung (ENUM) und wann eine neue Tabelle?	76
5.3.4	Wann ein DECIMAL und wann ein DOUBLE?	78
5.3.5	Wann verwende ich NOT NULL?.....	80
5.3.6	Wie lege ich einen Fremdschlüssel fest?	82
5.3.7	Wie kann ich Tabellen aus anderen herleiten?	87
5.3.8	Ich brauche mal eben kurz 'ne Tabelle!	88
6	Indizes anlegen.....	91
6.1	Index für Anfänger	91
6.1.1	Wann wird ein Index automatisch erstellt?	93
6.1.2	Wie kann ich einen Index manuell erstellen?	95
6.2	Und jetzt etwas genauer	97
6.2.1	Wie kann ich die Schlüsseleigenschaft erzwingen?.....	97
6.2.2	Wie kann ich Dubletten verhindern?	98
6.2.3	Was bedeutet Indexselektivität?.....	100
6.2.4	Wie kann ich einen Index löschen?	102
7	Werte in Tabellen einfügen.....	103
7.1	Daten importieren	103
7.1.1	Das CSV-Format	103
7.1.2	LOAD DATA INFILE	105
7.1.3	Was ist, wenn ich geänderte Werte importieren will?	109
7.2	Daten anlegen	110
7.2.1	Wie lege ich mehrere Zeilen mit einem Befehl an?	111
7.2.2	Wie kann ich eine einzelne Zeile anlegen?.....	112
7.2.3	Vorsicht Constraints!.....	113
7.2.4	Einfügen von binären Daten über einen C#-Client.....	114
7.2.5	Einfügen von binären Daten LOAD FILE	117
7.3	Daten kopieren	118
Teil III	Datenbank ändern	121
8	Datenbank und Tabellen umbauen.....	123
8.1	Eine Datenbank ändern.....	123
8.2	Eine Datenbank löschen	125
8.3	Eine Tabelle ändern	127

8.3.1	Wie kann ich den Namen der Tabelle ändern?	127
8.3.2	Wie kann ich eine Spalte hinzufügen?.....	128
8.3.3	Wie kann ich die Spezifikation einer Spalte ändern?	130
8.3.4	Zeichenbasierte Spalten in der Länge verändern	131
8.3.5	Zeichensatz verändern	132
8.3.6	Zeichenbasierte Spalten in numerische Spalten verändern	132
8.3.7	Numerische Spalten im Wertebereich verändern	133
8.3.8	Datum- oder Zeitspalten verändern.....	133
8.3.9	Wie kann ich aus einer Tabelle Spalten entfernen?.....	135
8.4	Eine Tabelle löschen.....	136
8.4.1	Einfach löschen.....	136
8.4.2	Was bedeuten CASCADE und RESTRICT?	137
9	Werte in Tabellen verändern.....	139
9.1	WHERE-Klausel	139
9.1.1	Wie formuliere ich eine einfache Bedingung?.....	140
9.1.2	Wird zwischen Groß- und Kleinschreibung unterschieden?	141
9.1.3	Wie formuliere ich eine zusammengesetzte Bedingung?.....	143
9.2	Tabelleninhalte verändern	144
9.2.1	Szenario 1: Einfache Wertzuweisung	145
9.2.2	Szenario 2: Berechnete Werte	146
9.2.3	Szenario 3: Gebastelte Zeichenketten	147
9.2.4	Was bedeuten LOW_PRIORITY und IGNORE?	147
9.3	Tabelleninhalte löschen.....	148
9.3.1	Und was passiert bei Constraints?	149
9.3.2	Was passiert mit dem AUTO_INCREMENT?	149
9.3.3	Was bedeuten LOW_PRIORITY, QUICK und IGNORE?	150
9.3.4	Wie kann ich eine Tabelle komplett leeren?	151
Teil IV	Datenbank auswerten	153
10	Einfache Auswertungen	155
10.1	Ausdrücke.....	156
10.1.1	Konstanten	156
10.1.2	Wie kann ich Berechnungen vornehmen?	156
10.1.3	Wie ermittele ich Zufallszahlen?	157
10.1.4	Wie stecke ich das Berechnungsergebnis in eine Variable?	159
10.2	Zeilen- und Spaltenwahl.....	159

10.3	Sortierung	161
10.3.1	Was muss ich bei der Sortierung von Texten beachten?	163
10.3.2	Wird zwischen Groß- und Kleinschreibung unterschieden?	165
10.3.3	Wie werden Datums- und Uhrzeitwerte sortiert?	167
10.3.4	Wie kann ich das Sortieren beschleunigen?	168
10.4	Mehrfachausgaben unterbinden	171
10.4.1	Fallstudie: Datenimport von Bankdaten	172
10.4.2	Was muss ich beim DISTINCT bzgl. der Performance beachten?	175
10.5	Ergebnismenge ausschneiden	175
10.5.1	Wie kann ich die ersten n Datensätze ausschneiden?	175
10.5.2	Wie kann ich Teilmengen mittendrin ausschneiden?	176
10.6	Ergebnisse exportieren	177
10.6.1	Wie lege ich eine Exportdatei auf dem Server an?	178
10.6.2	Wie lege ich eine Exportdatei auf dem Client an?	178
10.6.3	Wie lese ich mithilfe eines C#-Client binäre Daten aus?	179
11	Tabellen verbinden	181
11.1	Heiße Liebe: Primär-Fremdschlüsselpaare	182
11.2	INNER JOIN zwischen zwei Tabellen	185
11.2.1	Bauanleitung für einen INNER JOIN	185
11.2.2	Abkürzende Schreibweisen	190
11.2.3	Als Datenquelle für temporäre Tabellen	190
11.2.4	JOIN über Nichtschlüssel Spalten	193
11.3	INNER JOIN über mehr als zwei Tabellen	195
11.4	Es muss nicht immer heiße Liebe sein: OUTER JOIN	198
11.5	Narzissmus pur: SELF JOIN	202
11.6	Eine Verknüpfung beschleunigen	206
12	Differenzierte Auswertungen	209
12.1	Statistisches mit Aggregatfunktionen	209
12.2	Tabelle in Gruppen zerlegen	212
12.3	Gruppenergebnisse filtern	216
12.4	Noch Fragen?	218
12.4.1	Kann ich nach Ausdrücken gruppieren?	218
12.4.2	Kann ich nach mehr als einer Spalte gruppieren?	218
12.4.3	Wie kann ich GROUP BY beschleunigen?	219
12.4.4	Parallele Bearbeitung – unterschiedliche Ergebnisse?	220
12.5	Aufgaben	221

13 Auswertungen mit Unterabfragen	223
13.1 Das Problem und die Lösung	223
13.2 Nicht korrelierende Unterabfrage	226
13.2.1 Skalarunterabfrage	226
13.2.1.1 Beispiel 1: Banken mit höchster BLZ	226
13.2.1.2 Beispiel 2: Überdurchschnittlich teure Artikel	227
13.2.1.3 Beispiel 3: Überdurchschnittlich wertvolle Bestellungen	228
13.2.2 Listenunterabfrage	230
13.2.2.1 Beispiel 1: IN()	230
13.2.2.2 Beispiel 2: ALL()	231
13.2.2.3 Beispiel 3: ALL()	232
13.2.2.4 Beispiel 4: ANY()	235
13.2.3 Unterschied zwischen IN(), ALL() und ANY()	236
13.2.4 Unterschied zwischen NOT IN() und <>> ALL()	237
13.2.5 Tabellenunterabfrage	237
13.3 Korrelierende Unterabfrage	238
13.3.1 Beispiel 1: Rechnungen mit vielen Positionen	238
13.3.2 Beispiel 2: EXISTS	239
13.4 Fallstudie Datenimport	240
13.5 Aufgaben	244
14 Mengenoperationen	245
14.1 Die Vereinigung mit UNION	245
14.2 Die Schnittmenge	248
14.2.1 Mit INTERSECT	248
14.2.2 Mit Unterabfragen	249
14.3 Die Differenzmenge	250
14.3.1 Mit EXCEPT	250
14.3.2 Mit Unterabfragen	251
14.4 UNION, INTERSECT und EXCEPT ... versteh' ich nicht!	252
15 Bedingungslogik	255
15.1 Warum ein CASE?	255
15.2 Einfacher CASE	257
15.3 Searched CASE	259
15.4 Fallbeispiele	261
15.4.1 Lagerbestand überprüfen	261
15.4.2 Kundengruppen ermitteln	262
15.4.3 Aktive Lieferanten ermitteln	265
15.4.4 Aufgaben	266

16 Ansichtssache	267
16.1 Was ist eine Ansicht?	267
16.1.1 Wie lege ich eine Ansicht an?	268
16.1.2 Wie wird eine Ansicht verarbeitet?.....	270
16.1.3 Wie lösche ich eine Ansicht?.....	273
16.1.4 Wie ändere ich eine Ansicht?	276
16.2 Anwendungsgebiet: Vereinfachung	276
16.3 Anwendungsgebiet: Datenschutz	279
16.4 Grenzen einer Ansicht	279
17 Exkurs NoSQL	283
17.1 Vorbereitung der MySQL-Shell	284
17.2 Datenmodellierung des Warenkorbs.....	285
17.2.1 JavaScript Object Notation (JSON)	285
17.2.2 Struktur unseres JSON-Dokuments	286
17.3 NoSQL: MySQL mit JavaScript-Client.....	288
17.3.1 Anlegen eines Warenkorbs.....	289
17.3.2 Inhalte des Warenkorbs anlegen	290
17.3.3 Inhalte des Warenkorbs auswerten	293
17.3.4 Inhalte des Warenkorbs verändern	296
17.4 NoSQL: klassisches SQL mit JSON-Funktionen	298
17.4.1 Anlegen eines Warenkorbs.....	298
17.4.2 Inhalte des Warenkorbs anlegen	299
17.4.3 Inhalte des Warenkorbs auswerten	301
17.4.4 Inhalte des Warenkorbs verändern	302
17.4.5 Inhalte des Warenkorbs löschen	305
Teil V Anweisungen kapseln	307
18 Locking	309
19 Transaktion	313
19.1 Das Problem	313
19.2 Was ist eine Transaktion?	315
19.3 Isolationsebenen.....	318
19.3.1 READ UNCOMMITTED	319
19.3.2 READ COMMITTED	320
19.3.3 REPEATABLE READ	321
19.3.4 SERIALIZABLE.....	322
19.4 Fallbeispiel in C#	323
19.5 Deadlock	325

20 STORED PROCEDURE	327
20.1 Einstieg und Variablen	328
20.2 Verzweigung	333
20.2.1 Einfache Verzweigung mit IF	333
20.2.2 Mehrfache Verzweigung mit CASE.....	336
20.3 Schleifen	339
20.3.1 LOOP-Schleife	340
20.3.2 WHILE-Schleife	342
20.3.3 REPEAT-Schleife	345
20.4 Transaktion innerhalb einer Prozedur	346
20.5 CURSOR.....	347
20.6 Aufgaben	354
21 Funktion	355
22 TRIGGER.....	357
22.1 Was ist das?	357
22.2 Ein Beispiel für einen INSERT-Trigger	359
22.3 Ein Beispiel für einen UPDATE-Trigger.....	360
22.4 Ein Beispiel für einen DELETE-Trigger	362
23 EVENT	365
23.1 Wie lege ich ein Ereignis an?	365
23.2 Wie werde ich ein Ereignis wieder los?.....	368
Teil VI Anhänge	369
24 Datenbank administrieren.....	371
24.1 Backup und Restore	371
24.1.1 Backup mit mysqldump.....	371
24.1.2 Restore mit mysqldump.....	373
24.2 Benutzerrechte	373
24.2.1 Benutzerrechte und Privilegien	373
24.2.2 Benutzer anlegen/Recht zuweisen	376
24.2.2.1 CREATE USER	376
24.2.2.2 GRANT	377
24.2.2.3 REVOKE	379
24.3 MySQL und MariaDB Engines	380

25 SQL Injection	383
25.1 Das Problem	383
25.2 Beispiel: Suchmaske	384
25.3 Gegenmaßnahmen	391
25.3.1 Never Trust an Unknown Input	391
25.3.2 Trennung von Anweisungen und Daten	392
25.3.2.1 Verwenden Sie Prozeduren oder Funktionen	392
25.3.2.2 Verwenden Sie Prepared Statements	392
25.3.3 Ich darf nur, was ich soll	393
25.3.4 Nichtssagende Fehlermeldungen	394
26 SQL-Referenz	395
26.1 Datentypen	395
26.1.1 Numerische Datentypen	395
26.1.1.1 Ganze Zahlen	395
26.1.1.2 Gebrochene Zahlen	396
26.1.2 Zeichen-Datentypen	397
26.1.3 Datums- und Zeit-Datentypen	398
26.1.4 Binäre Datentypen	401
26.1.5 JSON	401
26.1.6 Räumliche Datentypen	402
26.1.7 Standardwerte	403
26.1.8 Zusätze für Datentypen	404
26.2 Operatoren und Funktionen	405
26.2.1 Mathematische Operatoren	406
26.2.2 Mathematische Funktionen	406
26.2.3 Aggregatfunktionen	409
26.3 Bedingungen	412
26.3.1 Vergleichsoperatoren	412
26.3.2 Logikoperatoren	414
26.3.2.1 NOT, Negation, \neg	414
26.3.2.2 AND, Konjunktion, \wedge	415
26.3.2.3 OR, Disjunktion, \vee	416
26.3.2.4 XOR, Antivalenz, \otimes	416
26.4 Befehle	417
26.4.1 Data Definition Language	417
26.4.2 Data Manipulation Language	429
26.4.3 Benutzerverwaltung	433

27 Ausgewählte Quelltexte	437
27.1 DOUBLE versus DECIMAL	437
27.2 Rundungsfehler	441
27.3 NULL versus NOT NULL.....	442
27.4 Suchen mit und ohne Index	444
27.5 Messen der Performance der Einfügeoperation.....	448
27.6 Messen der Indexselektivität	451
27.7 Sortieren ohne und mit Index.....	453
28 Quelltexte	457
28.1 MySQL/MariaDB	457
28.1.1 Quelltexte zu Teil II	457
28.1.2 Quelltexte zu Teil III.....	469
28.1.3 Quelltexte zu Teil IV.....	473
28.1.4 Quelltexte zu Teil V.....	516
28.1.5 Quelltexte zu Teil VI.....	530
28.2 PostgreSQL.....	534
28.2.1 Quelltexte zu Teil II	534
28.2.2 Quelltexte zu Teil III.....	543
28.2.3 Quelltexte zu Teil IV.....	546
28.2.4 Quelltexte zu Teil V.....	577
28.3 Microsoft SQL Server	581
28.3.1 Quelltexte zu Teil II	581
28.3.2 Quelltexte zu Teil III.....	592
28.3.3 Quelltexte zu Teil IV.....	597
Literatur	631
Stichwortverzeichnis	637

Vorwort zur 4. 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, der dann durch berufliche Praxis ausgebaut werden kann. Trotzdem vertiefe ich an vielen Stellen im Buch den Einblick in SQL oder den MySQL Server – zum einen, um zu zeigen, dass ich auch ein bisschen was draufhabe, 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 so viel 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 Dialekten 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/UCu4ZybNXwIy4Rs4Mgx-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]).
- In Auflage 3 sind *Generierte Spalten*, die Option WITH ROLLUP bei Gruppierungen und *Common Table Expression* (einfach und rekursiv) hinzugekommen.
- Auch ist seit Auflage 3 ein Kapitel über NoSQL enthalten.
- In Auflage 4 habe ich das Thema SQL Injection neu hinzugefügt, da auch in der beruflichen Erstausbildung immer mehr Wert auf Kompetenzen im Bereich der Datensicherheit gelegt wird. Die Kapitel über den MySQL Client und über für die deutsche Sprache relevante Kodierungen und Sortierungen mussten dem weichen.

Die von plus.hanser-fachbuch.de downloadbaren Skripte, Beispiele und Musterlösungen wurden auf folgenden Servern getestet: *MySQL Community Server 8.0.23*, *MariaDB 10.5.9*, *PostgreSQL 13.2-1* und *MS SQL Server 15.0.4102.2*. Alle Server liefen in einer Dockerinstanz unter Debian 10 (Buster). 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 Dialekte oder Eigenheiten begründet sind, musste ich auf eine Transkription verzichten.



Ihr Plus – digitale Zusatzinhalte!

Auf unserem Download-Portal finden Sie zu diesem Titel kostenloses Zusatzmaterial.

Geben Sie auf plus.hanser-fachbuch.de einfach diesen Code ein:

plus-pk6rL-5dms4

Danksagung

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

Besonders will ich mich bei meinen Schülerinnen und Schülern der Technischen Beruflichen Schule 1 in Bochum (<http://www.tbs1.de>) bedanken. 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 ange спорnt, es noch verständlicher zu versuchen. Falls dieses Buch SQL gut vermittelt, ist das auch deren Verdienst.

Dass nun aber die 4. Auflage dieses Buchs erscheinen kann, ist in erster Linie Ihnen, liebe Leserinnen und Leser, zu verdanken; dafür ein herzliches *Dankeschön!*

Ralf Adams, September 2021



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 Nichtschlüssel Spalten
 - 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 müssen Sie 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 erstellen.¹



Die Quelltexte dieses Kapitels stehen in der Datei `listing08.sql` (siehe [Listing 28.8 auf Seite 477](#), [Listing 28.47 auf Seite 601](#) und [Listing 28.32 auf Seite 550](#)).

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]]*]
;
```

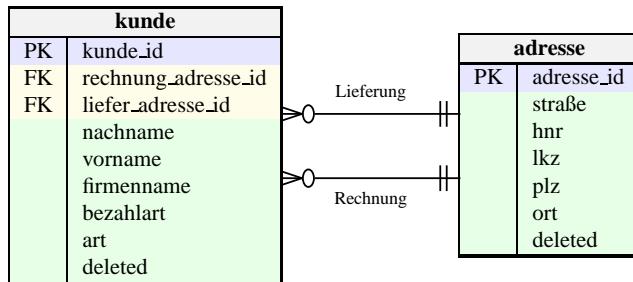


Bild 11.1 ER-Modell: kunde und adresse

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 | Earendilionn | 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 OO-Datenbanken oder NoSQL-Ansätzen.

```

17 |      3 | Beutlin     | Bilbo    |          2 |      2 |
18 |      4 | Telcontar   | Elessar   |          3 |      2 |
19 |      5 | Earendilionn | Elrond    |          4 |      2 |
20 |      6 | Eichenschild | 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 | Earendilionn | Elrond   |          4 |      3 |
26 |      6 | Eichenschild | 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 | Earendilionn | Elrond   |          4 |      4 |
32 |      6 | Eichenschild | 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 | Earendilionn | Elrond   |          4 |      5 |
38 |      6 | Eichenschild | 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 | Earendilionn | Elrond   |          4 |     10 |
44 |      6 | Eichenschild | 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 | Earendilionn | Elrond   |          4 |     11 |
50 |      6 | Eichenschild | 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.

Betrachten Sie nur die Werte von kunde_id und adresse_id, so erkennen Sie 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 können Sie 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 eine *coole SELECT-Erweiterung* 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
tabellenamefk INNER JOIN tabellenamepk
    ON tabellenamefk.fk = tabellenamepk.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 Befehl so um, dass folgende Ausgabe erzeugt wird:

nachname	vorname	strasse	hn	ort
Gamschie	Samweis	Beutelhaldenweg	5	Hobbingen
Beutlin	Frodo	Beutelhaldenweg	1	Hobbingen
Beutlin	Bilbo	Beutelhaldenweg	1	Hobbingen
Telcontar	Elessar	Auf der Feste	1	Minas Tirith
Earendilionn	Elrond	Letztes Haus	4	Bruchtal

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 Sie einen INNER

⁴ Siehe Kapitel 13 auf Seite 223.

JOIN zusammenbauen können. Die passende Aufgabe dazu: 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	
kunde_id	
bankverbindung_id	
bank_id	

	bank
	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	
kunde_id	
bankverbindung_id	
bank_id	

	bank
	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	
kunde_id	
bankverbindung_id	
bank_id	

	bank
	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
    INNER JOIN
ON . = .
```

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
    INNER JOIN
ON . = .
```

7. **Primärschlüsseltabelle eintragen:** Fügen Sie rechts vom INNER JOIN den Tabellennamen für die Tabelle mit dem markierten Primärschlüssel ein:

bankverbindung

kunde_id

bankverbindung_id

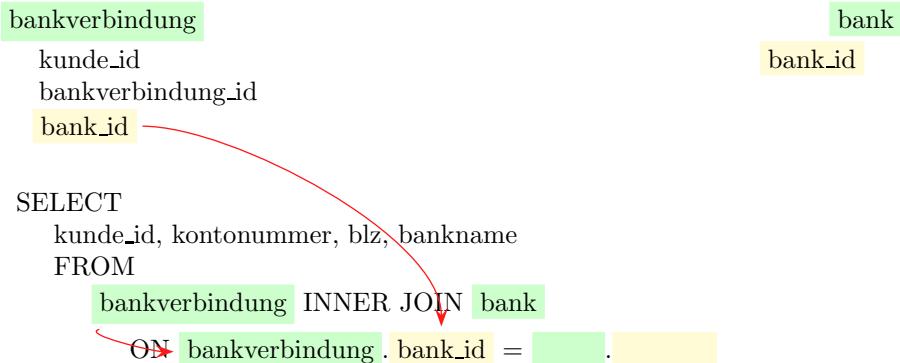
bank_id

bank

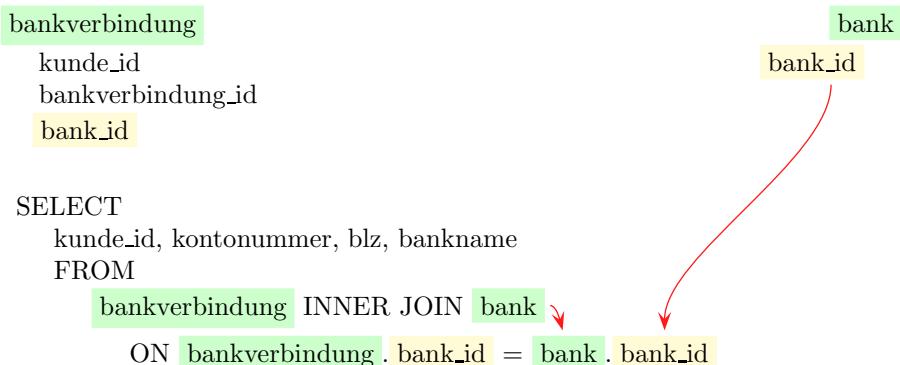
bank_id

```
SELECT
kunde_id, kontonummer, blz, bankname
FROM
    INNER JOIN
ON . = .
```

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 ON bankverbindung.bank_id = bank.bank_id
5     >     ORDER BY kunde_id, kontonummer;
6 +-----+-----+-----+-----+
7 | kunde_id | kontonummer | blz      | bankname   |
8 +-----+-----+-----+-----+
9 |      1 | 1111111111 | 10010010 | Postbank    |
10 |     1 | 1111111112 | 10010010 | Postbank    |
11 |     2 | 2222222221 | 10060198 | Pax-Bank    |
12 |     3 | 3333333331 | 10060198 | Pax-Bank    |
13 |     4 | 4444444441 | 12070000 | Deutsche Bank|
14 |     5 | 5555555551 | 12070000 | Deutsche Bank|
15 +-----+-----+-----+-----+

```

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 | Earendilonn | 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
        tabellenamefk INNER JOIN tabellenamepk
            [{ON tabellenamefk.fk = tabellenamepk.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, klinne Sie 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-/Primärschlüsselpalten 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 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 liefert.⁵

Nun können wir plausibel annehmen, dass sich die Kundenstammdaten (siehe [Definition 36](#)) für einen gewissen Auswertungszeitraum nicht ändern. Daher könnten wir 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.

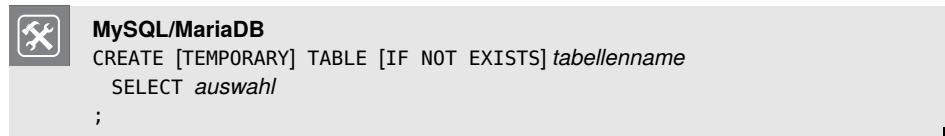
In beiden Fällen werden die Kundendaten mit Rechnungsanschrift benötigt. Das können wir schon:

```

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 a MySQL/MariaDB command-line interface. On the left, there's a small icon of a wrench and screwdriver. To its right, the text "MySQL/MariaDB" is displayed. Below that, a command is being entered into the terminal window:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tabellenname
    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 Der Query Cache ist seit der MySQL Version 8.0 abgeschaltet (siehe [[MyS21c](#)]).

```

19 |      5 | Earendillion | 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 | Earendillion | 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 jeweils

⁶ die Knoten ganz links

eine temporäre Tabelle mit INNER JOIN zusammengefasst. Die beiden neu erzeugten temporären Tabellen werden wiederum in eine temporäre Tabelle vereinigt.

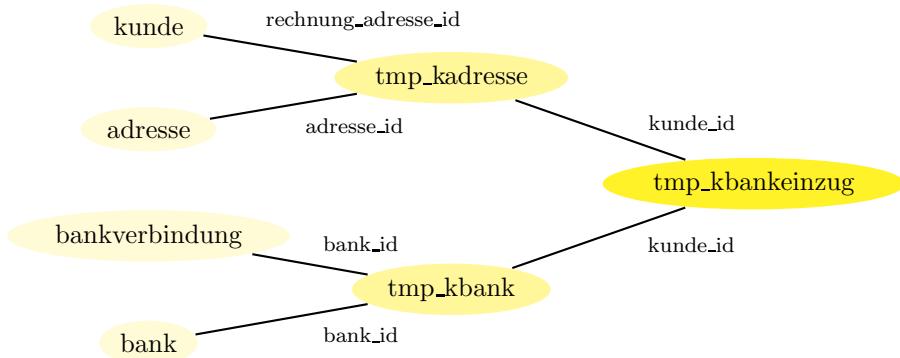


Bild 11.2 Temporäre Tabellen als Datenquelle

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 Sie mit vielen konkurrierenden Zugriffen auf die Tabellen `kunde` etc. rechnen.

Bei der Betrachtung von Transaktionen (siehe [Kapitel 19 auf Seite 313](#)) 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 abzuwägen.

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(11) unsigned | NO   | 0   | NULL    |
6 | nachname   | varchar(255)    | NO   |     | NULL    |
7 | vorname    | varchar(255)    | NO   |     | NULL    |
8 | strasse    | varchar(255)    | NO   |     | NULL    |
9 | hnr         | varchar(255)    | NO   |     | NULL    |
10 | plz        | char(9)          | NO   |     | NULL    |
11 | ort        | varchar(255)    | NO   |     | NULL    |
12 +-----+-----+-----+-----+-----+
13
14 mysql> DESCRIBE tmp_kbank;
```

15	+-----+-----+-----+-----+-----+					
16	Field	Type	Null	Key	Default	Extra
17	+-----+-----+-----+-----+-----+					
18	kunde_id	int unsigned	NO		NULL	NULL
19	bankverbindung_nr	int unsigned	NO		NULL	NULL
20	kontonummer	char(25)	NO			NULL
21	iban	char(34)	NO			NULL
22	blz	char(12)	NO			NULL
23	bankname	varchar(255)	NO			NULL
24	+-----+-----+-----+-----+-----+					

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 |
34 | 5 | Elrond | Industriegebiet | 8 | 44878 | Bochum |

```

⁷ Immerhin sind es Fremdschlüssel.

```
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 185](#) 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 191](#) 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 28](#)). 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.

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 INNER JOIN artikel a USING(artikel_id);
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 INNER JOIN warenguppe w USING(warenguppe_id);
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 können sehen, dass die [Zeile 4](#) selbst als Ergebnis eine neue Tabelle erzeugt, denn das Ergebnis besteht aus Zeilen und Spalten (siehe [Definition 7 auf Seite 16](#)). Und tatsächlich können wir diese Zeile anstelle einer einzelnen Tabelle in den ersten Befehl oben einsetzen:

```

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 ;

```

Wo vorher also nur `artikel_nm_warengruppe` als Datenquelle eines INNER JOIN stand, steht nun ein kompletter INNER JOIN – also die ganze [Zeile 4](#) – als Datenquelle. Dieser neue SELECT liefert das gewünschte Ergebnis:

Warengruppe	Artikel
Bürobedarf	Papier (100)
Bürobedarf	Tinte (gold)
Bürobedarf	Tinte (rot)
Bürobedarf	Tinte (blau)
Bürobedarf	Feder
Gartenbedarf	Silberwiebel
Gartenbedarf	Tulpenwiebel
Gartenbedarf	Schaufel
Gartenbedarf	Spaten
Pflanzen	Silberwiebel
Pflanzen	Tulpenwiebel
Werkzeug	Schaufel
Werkzeug	Spaten

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 *tablename* 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 Sie aber alle Kunden sehen, auch wenn diese keine Bestellungen aufgegeben haben, können Sie 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 +-----+-----+-----+

```

	kunde_id	nachname	vorname	datum
8				
9				
10	3	Beutlin	Bilbo	NULL
11	2	Beutlin	Frodo	2012-03-23 16:11:00
12	5	Earendilionn	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 Primärschlüsselwerte 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. Vertauschen Sie die beiden Tabellennamen und machen 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. Wollten Sie 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. 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 Lieferanten⁸ und ihre Artikel wissen.

```
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:

⁸ In der Datei listing08.sql werden diese angelegt.

```

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)
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.

Sie wollen wissen, welche Kunden bisher noch nichts bestellt haben? Kein Problem:

```

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 | Earendillion | Elrond   |
10 |      6 | Eichenschield | 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, müssen Sie 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 auf Seite 32](#)) 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   |           |
```

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 11.3 auf der nächsten Seite](#)).

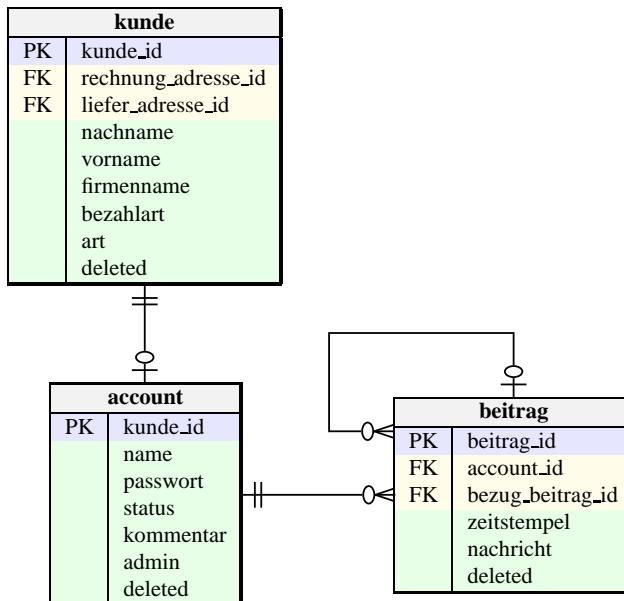


Bild 11.3 ER-Modell: Kundenforum

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 mit Primärschlüsselwerten der eigenen Tabelle, 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 11.3 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 ein Schlüssel sein.
- account: Der Status kann zwei Werte haben: aktiv und gesperrt.
- account: Der Kommentar muss einen langen Text aufnehmen können.
- account: Die Spalte admin ist vom Typ BOOL.
- 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 Platz für längere Texte haben.
- beitrag: Auf eine Thread-Verwaltung wird hier verzichtet.

Nun stehen uns Testdaten⁹ zur Verfügung, 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;
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 ON bezug_beitrag_id = beitrag_id

```

⁹ Die entsprechenden Befehle stehen in listing08.sql.

```

4      -> WHERE
5      -> beitrag_id = 2;
6 ERROR 1066 (42000): Not unique table/alias: 'beitrag'

```

Die Fehlermeldung in [Zeile 6](#) 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 3](#) 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. |

```

¹⁰ Eine nähere Betrachtung von CTE muss hier leider aus Platzgründen entfallen. Eine gute Einführung finden Sie unter [\[MyS19\]](#).

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 193](#). 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 beschrieben – können Sie 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 21](#) 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` sein.¹¹ 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),

```

¹¹ Eine von mir willkürlich gefällte Designentscheidung, um eine bessere Übersicht zu haben.

```

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 Tabellenname steht, eine Verknüpfung einsetzen.



Aufgabe 11.22: Überlegen Sie mal, lässt sich beim `UPDATE` eine geschickte `WHERE`-Klausel einbauen?

Jetzt wird jedes Mal, wenn sich eine Adresse innerhalb der Periode ändert, die Spalte `r_aktuell` oder `l_aktuell` auf `FALSE` gesetzt. Dies könnten Sie beispielsweise mit einem Trigger¹² erreichen. Mithilfe eines Events¹³ wird jetzt in periodischen Abständen nachgeschaut, ob sich die Adressdaten geändert haben. Falls ja, werden die redundanten Daten neu aufgebaut. Falls Sie die Änderung sofort in den redundanten Daten ändern wollen, können Sie 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 357](#).

¹³ Siehe [Kapitel 23 auf Seite 365](#).

Stichwortverzeichnis

|| 416
0 143
* 160, 406
+ 406
- 406
– 163
.NET 6
/ 406
; 67
< 413
<= 413
<=> 412
<> 412
<?php ... ?> 385
= 412
> 413
>= 413
% 406
&& 415
_id 288
! 414
!= 412

A
Abhängigkeit, mehrwertige 24
ABS() 406
Abweisende Schleife 339
ACID 317
ACOS() 406
AFTER 129
Aggregatfunktion 209
Akamai 383
Alias 160, 205
ALL 231, 232
ALL() 236
ALP 80

ALTER DATABASE 123, 417
ALTER EVENT 368, 417
ALTER FUNCTION 417
ALTER INSTANCE 418
ALTER LOGFILE GROUP 418
ALTER PROCEDURE 418
ALTER SCHEMA 123, 417
ALTER SERVER 418
ALTER TABLE 127, 418
– ... ADD 129, 147
– ... ADD FOREIGN KEY 174
– ... ADD INDEX 174
– ... ADD PRIMARY KEY 174
– ... DISABLE KEYS 102
– ... DROP 135
– ... DROP FOREIGN KEY 173
– ... DROP INDEX 174
– ... DROP PRIMARY KEY 174
– ... ENABLE KEYS 102
– ... MODIFY 130
– ... RENAME 128
ALTER TABLESPACE 420
ALTER USER 433
– aktueller Benutzer 434
– Rolle 434
ALTER VIEW 276, 420
AND 415
Änderung, kaskadierende 33, 85
Änderungsweitergabe 33, 85
Annehmende Schleife 339
ANSI 64
Ansicht 4, 267
– Projektions- 279
– Selektions- 276
– Verbund- 278
– veränderbare 281
Antivalenz 416

- ANY() 235, 236
 ApacheFriends 53
 API 6
 ARCHIVE 380
 Aria 380
 Artikelverwaltung 42
 AS IDENTITY 75
 ASC 161
 ASIN() 406
 ATAN() 406
 ATAN2() 406
 Atomare Tabelle 35
 Atomerität 317
 Atomicity 317
 Attribut 16
 Aufzählung 76
 AUTO_INCREMENT 74, 150, 404
 AUTOCOMMIT 318
 AVG() 210, 409
 AVG(DISTINCT) 409
- B**
 B-Baum 91
 Bank 41
 Bankverbindung 41
 Batchverarbeitung 388
 BCNF 39
 Bedingung 139
 BEGIN ... END 329
 BENCHMARK() 390
 Benutzerauthentifizierungsoption 435
 Benutzerrecht 373
 Benutzerspezifikation 435
 Bestellwesen 43
 BETWEEN 413
 Bewegungsdaten 144, 193
 BIGINT 395
 BINARY 141, 397, 404
 binding 392
 BIT 395
 BIT_AND() 409
 BIT_OR() 410
 BIT_XOR() 410
 BLACKHOLE 380
 blind injection 389
 BLOB 114, 401
 BOOL 395
 boolean attack 390
 Breitenabdeckung 397
 BSI 377
 Bücherei 30
 BULK INSERT 105, 108
- C**
 C 6
 C++ 6
 C# 6
 Cache 9
 CALL 429
 CASCADE 68, 85, 87, 137
 CASE 255, 336
 – einfach 257
 – searched 259
 Cassandra 380
 CD-Sammlung 31
 CEILING() 407
 CHAR 397
 CHARACTER SET 68, 164
 CHECK 404
 CHECK TABLE 274
 Chen, Jolly 9
 Chen-Notation 22
 CLOSE 347
 Clown-Agentur 30
 COBOL
 – Data Division 12
 – Satz kennzeichen 12
 – Stufenummer 12
 Codd, Dr. E. F. 11
 Codepage 850 69
 COLLATE 70
 collection 288
 Common Table Expression 205, 265, 302, 433
 Compilezeit 156
 CONCAT() 147
 Condition 139
 conditional comment 124
 CONNECT 380
 CONNECT_TIMEOUT 330
 Connection-Pool 8
 Connectionstring 385
 Connector 6
 Consistency 317
 CONSTRAINT 84
 constraint check 6
 CONV() 407
 COPY 105
 COS() 407
 COT() 407
 COUNT() 210, 410
 COUNT(*) 210, 410
 COUNT(DISTINCT) 210, 410
 cp850 69
 CRC32() 407
 CREATE DATABASE 66, 420

- CREATE EVENT 365, 420
CREATE FUNCTION 355, 421
CREATE INDEX 95, 421
CREATE LOGFILE GROUP 422
CREATE PROCEDURE 328, 422
CREATE ROLE 434
CREATE SCHEMA 66, 423
CREATE SERVER 423
CREATE SPATIAL REFERENCE SYSTEM 423
CREATE TABLE 73, 85, 87, 424
– ... LIKE 426
– ... SELECT 426
CREATE TABLESPACE 426
CREATE TEMPORARY TABLE 191, 224
CREATE TRIGGER 358, 427
CREATE USER 376, 393, 434
CREATE VIEW 427
CROSS JOIN 184
Crowfoot 22
CSV 380
CSV-Format 104
CURSOR 347
- D**
- Data Control Language 63
Data Definition Language 63
Data Manipulation Language 63
DATE 398
DATE() 135
DATE_FORMAT() 400
Dateisystem 9
Daten
– Bewegungs- 144, 193
– redundante 206
– Stamm- 144, 193
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, 279
Datensicherheit 6
Datentyp 73
Datenverzeichnis 57
- DATETIME 398
Dauerhaftigkeit 317
DBMS 5
DCL 63
DDL 63, 417
Deadlock 325, 326
DECIMAL 78, 396, 437
DECLARE 329
– ...FOR CURSOR 347
– CONTINUE 347
– EXIT 347
Dedicated Computer 50
DEFAULT 404
DEGREES() 407
Deklarative Programmiersprache 63
DELETE 148, 429
– ... IGNORE 150
– ... LOW_PRIORITY 150
– ... QUICK 150
deleted 32
DELIMITER 329
DESC 161
DESCRIBE 76, 129
Deutsche Brauereiverband 31
Deutscher Wetterdienst 24
Developer Server 47
Development Computer 49
Differenzmenge 250, 253
DIN 64
DIN 5007-1 71
DIN 66001 22
DIN 66261 22
dirty read 319
DISABLE KEYS 102
Disjunktion 416
DISTINCT 171
DIV 406
DML 63, 429
DO 430
Docker 58
– MariaDB 60
– MS SQL Server 62
– MySQL 58
– PostgreSQL 61
Domäne 15, 16
DOUBLE 78, 396, 437
DOUBLE PRECISION 396
Drei-Schichten-Architektur 15, 80
Dritte Normalform 38
DROP DATABASE 125, 427
DROP EVENT 368, 427
DROP FUNCTION 356, 427

DROP INDEX 102, 427
 DROP LOGFILE GROUP 428
 DROP PROCEDURE 332, 428
 DROP ROLE 434
 DROP SCHEMA 125, 428
 DROP SERVER 428
 DROP SPATIAL REFERENCE SYSTEM 428
 DROP TABLE 136, 428
 DROP TABLESPACE 428
 DROP TRIGGER 358, 428
 DROP USER 374, 377, 434
 DROP VIEW 273, 428
 DSGVO 33
 Dublette 98
 Durability 317

E

Eigenschaft 16
 1:1-Verknüpfung 24
 1:n-Verknüpfung
 – Definition 26
 – identifizierende 27
 ENABLE KEYS 102
 Engine 5, 9, 83
 – ARCHIVE 380
 – Aria 380
 – BLACKHOLE 380
 – Cassandra 380
 – CONNECT 380
 – CSV 380
 – EXAMPLE 380
 – FEDERATED 380
 – FederatedX 380
 – InnoDB 380
 – MEMORY 381, 438
 – MERGE 381
 – MyISAM 381
 – OQGRAPH 381
 – XtraDB 381
 Entity 16
 Entity Relationship Modell 22
 Entitytype 16
 Entität 16
 Entitätentyp 16
 – schwacher 18
 – starker 18
 ENUM 76, 395
 EQUI JOIN 195
 ER-Modell 22
 Eratosthenes 342
 Ereignis 4, 365
 Erlang 6

ERM 22
 error based attack 389
 Erste Normalform 36
 Escapen 391
 Event 365
 EXAMPLE 380
 Excel 6
 EXCEPT 250, 251, 253
 Existenzabhängige Tabelle 18
 Existenzunabhängige Tabelle 18
 EXISTS 239
 EXIT 66
 EXP() 407
 Experiment
 – DOUBLE vs. DECIMAL 437
 – Einfügen mit Index 448
 – Indexselektivität 451
 – NULL vs. NOT NULL 442
 – Rundungsfehler 441
 – Sortierung 453
 – Suchen 444
 EXPLAIN 168
 Exploit 389
 Exportieren
 – Binärdaten
 – C# 179
 – CSV-Daten 177

F

Fallunterscheidung 255, 336
 FEDERATED 380
 FederatedX 380
 Feld 16
 FETCH 347
 FIRST 129
 FLOAT 396
 FLOOR() 158, 407
 FOR ORDINALITY 301
 FOREIGN KEY 82, 404
 foreign key 19
 FORMAT() 258, 407
 Formatierungszeichen 400
 – Datum 399
 – Uhrzeit 400
 Fremdschlüssel 19, 23, 82
 – festlegen 82
 FULL OUTER JOIN 199
 Funktion 355

G

GENERATED 75
 GENERATED ...AS IDENTITY 404

- GENERATED ...AS PRIMARY KEY 404
generierte Spalten 405
GEOMETRY 403
GEOMETRYCOLLECTION 403
Geschlossene Schleife 339
GET_FORMAT() 400
GLOBAL 89
GRANT 377, 393, 434
 - ALL PRIVILEGES 378
 - CHARACTER SET 378
 - COLLATION 378
 - DOMAIN 378
 - FUNCTION 378
 - PROCEDURE 378
 - PROXY 434
 - Rolle 435
 - TABLE 378
 - TRANSLATION 378
 - WITH GRANT OPTION 378Grantoption 436
GROUP BY 212
GROUP BY ... WITH ROLLUP 215
GROUP_CONCAT() 293, 410
- H**
Hauptanweisung 225
HAVING 216, 217
Herz 155
HEX() 407
Hierarchische Datenbank 12
Hilfstabelle 29
htdocs 384
- I**
iconv 69
IDEF1X-Notation 22
Identifizierende 1:n-Verknüpfung 27
IF 334
 - IF EXISTS 68
 - IF NOT EXISTS 67
 - IGNORE 109Imperative Programmiersprache 63
Importieren
 - Binärdaten
 - C# 114
 - LOAD FILE 117
 - CSV-Daten 105
 - XML-Daten 348IN 328
in band injection 389
IN() 230, 236, 413
- Index 4, 91, 444, 448, 451, 453
 - anlegen 95
 - automatisch 93
 - Dublette 98
 - löschen 102
 - Schlüsseleigenschaft 97index scan
 - backward 170
 - forward 170Indexselektivität 100, 451
Inferential injection 389
Informix 9
INHERITS 88
Injection
 - in band 389
 - error based 389
 - union based 389
 - inferential 389
 - boolean 390
 - time based 390
 - out of band 390INNER JOIN 185
InnoDB 380
InnoDB Cluster 48
INOUT 328
INSERT 374
INSERT INTO
 - ... SELECT 118, 430
 - ... SET 112, 430
 - ... VALUES 111, 430INT 395
Integrität, referenzielle 32
Interpreter 5
INTERSECT 248, 252
INTO OUTFILE 390
IS FALSE 413
IS NOT NULL 414
IS NULL 414
IS TRUE 413
IS UNKNOWN 413
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 400Isolation 317
Item 16

- ITERATE 340
- J**
- JavaScript
 - *collection*
 - add() 289
 - arrayDelete() 297
 - arrayInsert() 290
 - find() 290
 - modify() 290
 - remove() 289
 - *object*
 - hasOwnProperty() 291
 - keys 291
 - *parameter*
 - bind() 290
 - *result*
 - fetchAll() 289, 294
 - fetchOne() 290
 - *schema*
 - dropCollection() 288
 - getCollection() 289
 - getCollections() 288
 - *session*
 - close() 288
 - getSchema() 288
 - sql() 290, 294
 - *statement*
 - execute() 290
 - *table*
 - select() 289
 - for...in 289
 - function 288
 - require() 288
 - return 288
 - JDBC 6
 - JOIN
 - ... ON 185
 - ... USING 190
 - CROSS 184
 - EQUI 195
 - INNER 185
 - NATURAL 190
 - OUTER 199
 - FULL 199, 202
 - LEFT 199
 - RIGHT 199
 - SELF 203
 - JSON 285, 401
 - JSON_ARRAY() 299
 - JSON_ARRAY_APPEND() 300
 - JSON_ARRAYAGG() 410
 - JSON_EXTRACT() 300
 - JSON_OBJECT() 298
 - JSON_OBJECTAGG() 411
 - JSON_PRETTY() 300
 - JSON_REMOVE() 306
 - JSON_REPLACE() 304, 306
 - JSON_SEARCH() 303
 - JSON_SET() 306
 - Jupyter 6
- K**
- Kalender
 - gregorianisch 398
 - julianisch 398
 - proleptischer gregorianischer 398
 - Kardinalität 24
 - Kartesisches Produkt 183
 - Kaskadierende Änderung 33, 85
 - Kaskadierendes Löschen 32, 85
 - key 17
 - Klammern 143
 - Klasse 16
 - Kommentar
 - bedingter 124
 - einzeilig 163
 - Konjunktion 415
 - Konnektor 6, 8
 - Konsistenz 317
 - Konstante 156
 - Kopf-/Fußgesteuerte Schleife 339
 - Korrelierende Unterabfrage 226
 - Kreuzprodukt 184
 - Krähenfuß-Notation 22
 - Kunde 41
 - Kundenverwaltung 41
- L**
- LAST_INSERT_ID() 314
 - latin1 69
 - Laufzeit 156
 - LEAVE 340
 - LEFT OUTER JOIN 199
 - LIKE 87, 414
 - LIMIT 175
 - LINESTRING 402
 - Listenunterabfragen 230
 - LN() 408
 - LOAD DATA INFILE 105, 107, 431
 - LOAD FILE 117
 - LOAD XML 349, 431
 - LOAD_FILE() 390
 - LOCAL 89

- lock [310](#)
- LOCK TABLES [311](#)
- locking [310](#)
- LOG() [408](#)
- LOG10() [408](#)
- LOG2() [408](#)
- lokale Variable [329](#)
- LONGBLOB [401](#)
- LONGTEXT [397](#)
- LOOP-Schleife [340](#)
- Löschen, kaskadierendes [32, 85](#)
- Löschkennzeichen [32](#)
- Löschweitergabe [32, 85](#)
- lost update [310](#)
- LPAD() [258](#)
- Löschkennzeichen [276](#)

- M**
- MariaDB Client [65](#)
- Martin-Notation [22](#)
- Maskieren [391](#)
- Matrix [16](#)
- MAX() [211, 411](#)
- MEDIUMBLOB [401](#)
- MEDIUMINT [395](#)
- MEDIUMTEXT [397](#)
- Mehrwertige Abhängigkeit [24](#)
- MEMORY [381, 438](#)
- Mengenverhältnis [24](#)
- MERGE [381](#)
- MERGED [270](#)
- MIN() [211, 411](#)
- Minimalität des Schlüssels [17](#)
- Minimumexistenz [163](#)
- MOD [406](#)
- MOD() [408](#)
- Modellierung [22](#)
- Monolithische Anwendung [80](#)
- MONTH() [219](#)
- MULTILINESTRING [403](#)
- MULTIPOINT [402](#)
- MULTIPOLYGON [403](#)
- MyISAM [381](#)
- MySQL [7](#)
- MySQL Client [65](#)
- MySQL Query Browser [65](#)
- MySQL Router [48](#)
- MySQL Workbench [23](#)
- mysqladmin [57](#)
- mysqldump [371](#)
- mysqli [385](#)
 - bind_param() [392](#)
- close() [386](#)
- connect_error [385](#)
- error [386](#)
- execute() [393](#)
- fetch_assoc() [386](#)
- get_result() [393](#)
- multi_query() [388](#)
- num_rows [386](#)
- prepare() [392](#)
- query() [386, 388](#)
- real_escape_string() [391](#)
- mysqlsh [284](#)

- N**
- n:m-Verknüpfung [28](#)
- n:m:k-Verknüpfung [29](#)
- Nassi-Shneidermann-Diagramm [22](#)
- NATURAL JOIN [190](#)
- Negation [414](#)
- NEW [358](#)
- Nicht korrelierende Unterfrage [226](#)
- NO ACTION [87](#)
- Node.js [6](#)
- Normalform [33](#)
 - Boyce-Codd [39](#)
 - erste [36](#)
 - zweite [37](#)
 - dritte [38](#)
 - vierte [39](#)
 - fünfte [39](#)
- Normalisierung [34, 36](#)
- NOT [414](#)
- NOT BETWEEN [413](#)
- NOT FOUND [347](#)
- NOT IN [413](#)
- NOT IN() [251](#)
- NOT LIKE [414](#)
- NOT NULL [80, 404, 442](#)
- Notation
 - Chen [22](#)
 - Crowfoot [22](#)
 - IDEF1X [22](#)
 - Krähenfuß [22](#)
 - Martin [22](#)
 - UML [22](#)
- NULL [80, 404, 442](#)
- NUMERIC [396](#)
- Nummernkreis [36, 76](#)

- O**
- Oberanweisung [225](#)
- Objectcode [392](#)

- Objekt 16
- Objektorientierte Datenbank 13
- Objekttyp 436
- ODBC 6
- Offene Schleife 339
- OGC 402
- OLD 358
- Online-Shop 41
 - Tabelle adresse 41
 - Tabelle artikel 42
 - Tabelle artikel_nm_lieferant 89
 - Tabelle artikel_nm_warengruppe 110
 - Tabelle bank 41
 - Tabelle bankverbindung 41
 - Tabelle bestellung 43
 - Tabelle bild 114
 - Tabelle kunde 41
 - Tabelle lagerbestand 212
 - Tabelle lieferant 42
 - Tabelle position_bestellung 43
 - Tabelle position_rechnung 43
 - Tabelle rechnung 43
 - Tabelle warengruppe 42
- OPEN 347
- OpenGIS 402
- Operatorenpriorität 157
- Operatorenrangfolge 157
- Optimierer 5, 9
- OQGRAPH 381
- OR 416
- Oracle 7
- ORDER BY 161
- Ordnung 163
- Orgenjoin 184
- OUT 328
- out of band injection 390
- OUTER JOIN 199
- OWASP 383

- P**
- page lock 310
- Parameterart
 - IN 328
 - INOUT 328
 - OUT 328
- Parameterbindung 290, 291, 392
- Parser 9
- PASSWORD() 376
- Passwortoption 436
- Penetrationstest 390
- Performancemessung 437, 444, 448, 453
- Perl 6
- PHP 6
 - die() 394
 - error_log() 394
 - MySQLi 385
 - PDO 385
 - preg_match() 391
 - Tag 385
- PI() 408
- Platzhalter 160
- Plausibilisierung 15
- Plausibilitätsüberprüfung 333
- POINT 402
- POLYGON 403
- PostgreSQL 9
- POSTQUEL 9
- POWER() 408
- Prepared Statement 392
- PRIMARY KEY 404
- primary key 18
- Primärschlüssel 18, 22
- Privileg 373
- Privilegtiefe 436
- Programmablaufplan 22
- Programmierschnittstelle 6
- Programmiersprache
 - deklarative 63
 - imperative 63
- Programmverzeichnis 55
- Projektionsansicht 279
- Property 16
- Prozedur 4
- Puh 128
- Python 6

- R**
- R 6
- RADIANS() 408
- RAND() 157, 408
- Randbedingungsprüfer 6
- RDBMS 6
- READ ONLY 123
- read only 405
- REAL 396
- real_escape_string() 391
- Record 16
- Recordset 16
- Redundante Daten 206
- Redundanz 21, 33
- REFERENCES 82, 83
- Referenz 21
- referenzielle Integrität 32, 41, 84
- reflection 291

- Regenbogentabelle 388
Regulärer Ausdruck 391
Relation 11, 16
Relationale Datenbank 11
relationales Datenbankmanagementsystem 6
RENAME TABLE 429
RENAME USER 435
REPLACE 109
Ressourceoption 436
RESTRICT 68, 86, 87, 137
REVOKE 374, 379, 435
 - ALL 435
 - ALL PRIVILEGES 380
 - PROXY 435
 - Rolle 435RIGHT OUTER JOIN 199
ROLLBACK 319
ROUND() 147, 408
row lock 310
Ruby 6
Rundungsfehler 78, 409, 441
räumliche Datentypen 402
- S**
- Sakila 7
Schema 16, 67
Schleife 339
 - abweisende 339
 - annehmende 339
 - fußgesteuerte 339
 - geschlossen 339
 - kopfgesteuerte 339
 - offene 339Schlüssel
 - Definition 17
 - Fremd- 19
 - Kandidat- 18
 - Minimalität des 17
 - Primär- 18
 - Sekundär- 18Schnittmenge 248, 252
Schwache Tabelle 18
Seitensperre 310
Sekundärschlüssel 18
SELECT 155, 431
 - ... INTO 159, 176, 331
 - ... INTO OUTFILE 178Selektionsansicht 276
SELF JOIN 202, 203
Semikolon 67
Seq_in_index 96
Server Computer 49
- serverglobal 330
session variable 330
SET 331, 395
SET DEFAULT 87
SET GLOBAL 367
SET NAMES 164
SET NULL 87
SET PASSWORD 435
SHOW
 - CHARACTER SET 69
 - COLLATION 70
 - CREATE DATABASE 124
 - CREATE SCHEMA 124
 - CREATE TABLE 83, 150
 - DATABASES 72
 - EVENTS 366
 - FULL TABLES 269
 - GRANTS FOR 376
 - INDEX 93
 - PROCEDURE STATUS 332
 - SCHEMAS 72
 - TABLES 75
 - TRIGGERS 363
 - VARIABLES 367
 - VARIABLES LIKE 66
 - VIEWS (work around) 269
 - WARNINGS 67, 102, 106Sieb des Eratosthenes 342
SIGN() 409
SIN 409
Single Responsibility Principle 80
Sitzung 8
sitzungsglobal 330
Sitzungsverwaltung 5
Skalarunterabfrage 226, 227
SLEEP() 390
SMALLINT 395
Sortierreihenfolge 70
Sortierung
 - zuweisen 70sp_rename 128
Spalte
 - * 160
 - Auswahl einer 160
 - Definition 15
 - Spezifikation einer 73spatial data types 402
Sperroption 436
Sprunglogik 415
SQL 63
SQL HANDLER 347
SQL Injection 383

SQL-Injection 116, 118, 178, 291
 SQL-Schnittstelle 9
 SQL_NO_CACHE 440
 SQLEXCEPTION 347
 SQLi 383
 SQLWARNING 347
 SQRT() 409
 SRP 80
 Stammdaten 144, 193
 Standardwert 403
 Starke Tabelle 18
 START TRANSACTION 317
 STD() 411
 STDDEV() 411
 STDDEV_POP() 411
 STDDEV_SAMP() 411
 Stonebraker, Michael 9
 Storage Engine 5, 9
 strict-Modus 374
 String 395
 Struktogramm 22
 Stundenplan-Software 31
 SUBSELECT 225
 SUBSTRING() 218
 Suchpfad 57
 SUM() 211, 411
 SUM(DISTINCT) 411
 Sun Microsystems 7
 Swift 6

T

Tabelle 4, 16
 – anlegen 72
 – atomar 35
 – existenzabhängige 18
 – existenzunabhängige 18
 – herleiten 87
 – schwach 18
 – starke 18
 – teilfunktional 37
 – temporär 88, 220
 – transitiv 38
 – vollfunktional 37
 – wiederholungsgruppenfrei 35
 Tabellenreferenzen 432
 Tabellensperre 310
 Tabellenunterabfragen 237
 table lock 310
 TAN() 409
 Tcl 6
 Teilfunktionale Tabelle 37
 TEMPORARY 89

Temporäre Tabelle 4, 88, 220
 TEMPTABLE 270
 TEXT 397
 Tiefenabdeckung 397
 TIME 398
 time based attack 390
 TIME() 167
 TIME_FORMAT() 400
 Timeout 5, 66
 TIMESTAMP 398
 TINYBLOB 401
 TINYINT 395
 TINYTEXT 397
 Transact-SQL 64
 Transaktion 316
 Transaktionsmanagement 6
 Transitiv Tabelle 38
 Transitivität 163
 Trennzeichen 104
 Trichotomie 163
 Trigger 4, 357
 TRUNCATE 151, 429
 TRUNCATE() 409
 Tupel 16
 Twebaze, Ambrose 7

U

Übergabeparameter 330
 UML-Notation 22
 UNDER 88
 Unicode 69
 UNION 245, 252, 387, 432
 union based attack 389
 UNIQUE 93, 404
 UNKNOWN 139
 UNLOCK TABLES 311
 UNSIGNED 74, 404
 Unterabfrage 225
 – korrelierende 226
 – Listen- 230
 – nicht korrelierende 226
 – skalar 226, 227
 – Tabellen- 237
 UPDATE 145, 433
 – ... IGNORE 147
 – ... LOW_PRIORITY 147
 USE 74
 USING 190
 utf8 69
 utf8mb4 69
 utf16 69
 utf32 69

- utf8 69
utf8mb3 70
utf8mb4 70
- V**
VAR_POP() 412
VAR_SAMP() 412
VARCHAR 75, 397
Variable 159, 176
 - global
 - server 330
 - sitzungs- 330
 - lokal 329
VARIANCE() 412
Verbinder 8
Verbindungsparameter 385
Verbundansicht 278
Vereinigung 245, 252
Vererbung 88
Verknüpfung 21
 - 1:1, Definition 24
 - 1:n
 - Definition 26
 - identifizierende 27
 - n:m, Definition 28
 - n:m:k 29
Verletzte referenzielle Integrität 32
Verschlüsselung 436
Verzeichnis
 - Daten 57
 - Programm 55
 - XAMPP 54
Verzweigung 333
Veränderbare Ansicht 281
VIEW 267
Vollfunktionale Tabelle 37
Vorbelegungen 403
- W**
Wartungsinstabilität 21, 23, 80
Wartungsstabilität 39, 276
Wertebereich 15
WHERE-Klausel 139, 140, 217
WHILE-Schleife 342
Widenius, Michael 7
Wiederholungsgruppe 29, 34
Wiederholungsgruppenfreiheit 35
Windows 10 53
Windows Service 51
WITH 433
WITH RECURSIVE 205
WITH ROLLUP 215
- X**
XAMPP 54, 384
XAMPP-Verzeichnis 54
XML-Format 13
XtraDB 381
- Y**
YEAR 398
YEAR() 219
Yu, Anrew 9
- Z**
Zeichenketten 112
Zeichensatz 68
 - zuweisen 68
Zeile
 - Auswahl einer 160
 - Definition 16
Zeilensperre 310
Zeilenumbruch 104
Zufallszahlen 157
Zweite Normalform 37
Zwischenspeicher 9