

# HANSER



## Leseprobe

zu

## „SQL

## Der Grundkurs für Ausbildung und Praxis“

von Ralf Adams

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

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

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

Weitere Informationen und Bestellungen unter

<http://www.hanser-fachbuch.de/978-3-446-46110-9>

sowie im Buchhandel

© Carl Hanser Verlag, München

# Inhalt

Vorwort zur 3. Auflage .....	XVII
<b>Teil I Was man so wissen sollte .....</b>	<b>1</b>
<b>1 Datenbanksystem .....</b>	<b>3</b>
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
<b>2 Einführung in relationale Datenbanken .....</b>	<b>11</b>
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 <i>1:1</i> -Verknüpfung .....	24
2.2.2.1 Wann liegt eine <i>1:1</i> -Verknüpfung vor? .....	24
2.2.2.2 Wie kann ich eine <i>1:1</i> -Verknüpfung darstellen? .....	25
2.2.2.3 Kann man die Kardinalität genauer beschreiben? .....	26
2.2.3 <i>1:n</i> -Verknüpfung .....	27
2.2.3.1 Wann liegt eine <i>1:n</i> -Verknüpfung vor? .....	27
2.2.3.2 Wie kann ich eine <i>1:n</i> -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
<b>3</b>	<b>Unser Beispiel: Ein Online-Shop .....</b>	<b>41</b>
3.1	Kundenverwaltung .....	41
3.2	Artikelverwaltung .....	42
3.3	Bestellwesen .....	43
	<b>Teil II Datenbank aufbauen .....</b>	<b>45</b>
<b>4</b>	<b>Installation des Servers .....</b>	<b>47</b>
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
<b>5</b>	<b>Datenbank und Tabellen anlegen .....</b>	<b>63</b>
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
<b>6</b>	<b>Indizes anlegen .....</b>	<b>93</b>
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
<b>7</b>	<b>Werte in Tabellen einfügen.....</b>	<b>105</b>
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
<b>Teil III</b>	<b>Datenbank ändern .....</b>	<b>123</b>
<b>8</b>	<b>Datenbank und Tabellen umbauen.....</b>	<b>125</b>
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
<b>9</b>	<b>Werte in Tabellen verändern .....</b>	<b>141</b>
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
<b>Teil IV</b>	<b>Datenbank auswerten .....</b>	<b>155</b>
<b>10</b>	<b>Einfache Auswertungen .....</b>	<b>157</b>
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
<b>11</b>	<b>Tabellen verbinden .....</b>	<b>183</b>
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 Nichtschlüsselpalten .....	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
<b>12</b>	<b>Differenzierte Auswertungen .....</b>	<b>211</b>
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

<b>13</b>	<b>Auswertungen mit Unterabfragen</b> .....	<b>225</b>
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: IN() .....	232
13.2.2.2	Beispiel 2: ALL() .....	233
13.2.2.3	Beispiel 3: ALL() .....	234
13.2.2.4	Beispiel 4: ANY() .....	237
13.2.3	Unterschied zwischen IN(), ALL() und ANY() .....	238
13.2.4	Unterschied zwischen NOT IN() und <> ALL() .....	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
<b>14</b>	<b>Mengenoperationen</b> .....	<b>251</b>
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
<b>15</b>	<b>Bedingungslogik</b> .....	<b>261</b>
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

<b>16</b>	<b>Ansichtssache</b> .....	<b>273</b>
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
<b>17</b>	<b>Exkurs NoSQL</b> .....	<b>289</b>
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
<b>Teil V</b>	<b>Anweisungen kapseln</b> .....	<b>313</b>
<b>18</b>	<b>Locking</b> .....	<b>315</b>
<b>19</b>	<b>Transaktion</b> .....	<b>319</b>
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



<b>20</b>	<b>STORED PROCEDURE</b> .....	<b>333</b>
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
<b>21</b>	<b>Funktion</b> .....	<b>361</b>
<b>22</b>	<b>TRIGGER</b> .....	<b>363</b>
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
<b>23</b>	<b>EVENT</b> .....	<b>371</b>
23.1	Wie legt man ein Ereignis an? .....	371
23.2	Wie wird man ein Ereignis wieder los? .....	374
<b>Teil VI</b>	<b>Anhänge</b> .....	<b>375</b>
<b>24</b>	<b>Datenbank administrieren</b> .....	<b>377</b>
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

<b>25</b>	<b>Rund um den MySQL Client</b> .....	<b>389</b>
25.1	Aufruf(parameter) .....	389
25.2	Befehle .....	392
<b>26</b>	<b>SQL-Referenz</b> .....	<b>397</b>
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
<b>27</b>	<b>Ausgewählte Quelltexte</b> .....	<b>439</b>
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

<b>28 Rund ums Zeichen</b> .....	<b>457</b>
28.1 Für Deutsch relevante Zeichensätze .....	457
28.2 Für Deutsch relevante Sortierungen .....	458
<b>29 Quelltexte</b> .....	<b>461</b>
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
<b>Literatur</b> .....	<b>635</b>
<b>Stichwortverzeichnis</b> .....	<b>639</b>

# 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 Leser<sup>1</sup> 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](mailto: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.

---

<sup>1</sup> 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 Voraufgabe – 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

# 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 Nichtschlüsselspalten
  - EQUI JOIN
  - OUTER JOIN und referenzielle 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 erstellen<sup>1</sup>.



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).

<sup>1</sup> 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]]*];
```

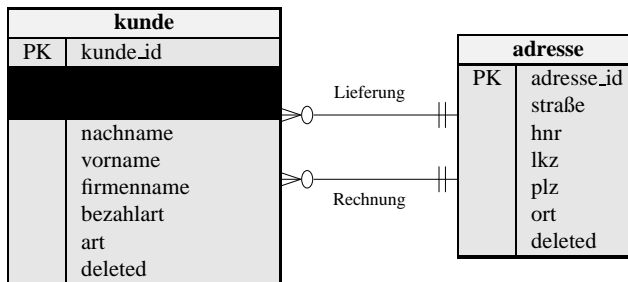


Bild 11.1 ER-Modell: kunde und adresse

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.<sup>2</sup>

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 +-----+-----+-----+-----+-----+
8 | kunde_id | nachname | vorname | rechnung_adresse_id | adresse_id |
9 |-----+-----+-----+-----+-----+
10 | 1 | Gamschie | Samweis | 1 | 1 |
11 | 2 | Beutlin | Frodo | 2 | 1 |
12 | 3 | Beutlin | Bilbo | 2 | 1 |
13 | 4 | Telcontar | Elessar | 3 | 1 |
14 | 5 | Earendilionn | Elrond | 4 | 1 |
15 | 6 | Eichenschild | Thorin | NULL | 1 |
16 | 1 | Gamschie | Samweis | 1 | 2 |
17 | 2 | Beutlin | Frodo | 2 | 2 |
```

<sup>2</sup> 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 | Earendilionn | Elrond     |      4 |      2 |
20 |      6 | Eichenschild | Thorin     | NULL   |      2 |
21 |      1 | Gamschie    | 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 | Gamschie    | 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 | Gamschie    | 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 | Gamschie    | 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 | Gamschie    | 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.

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 Produkt<sup>3</sup>, 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 | Gamschie | 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.

<sup>3</sup> 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 Unterabfragen<sup>4</sup>. Deshalb gibt es eine eigene Syntax für den *INNER JOIN*:



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

```
SELECT [DISTINCT]
  {*|spaltenliste|ausdruck}
FROM
  tabellefk INNER JOIN tabellepk
    ON tabellefk.fk = tabellepk.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           |
+-----+-----+-----+-----+-----+
| Gamdschie | 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   | Bruchthal   |
+-----+-----+-----+-----+-----+
```

<sup>4</sup> 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.

1. **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.
2. **Ermitteln der Primär- und Fremdschlüssel:** Schreiben Sie unter den Tabellennamen jeweils den Primärschlüssel und alle vorkommenden Fremdschlüssel.

<code>bankverbindung</code>	<code>bank</code>
<code>kunde_id</code>	<code>bank_id</code>
<code>bankverbindung_id</code>	
<code>bank_id</code>	

3. **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.

<code>bankverbindung</code>	<code>bank</code>
<code>kunde_id</code>	<code>bank_id</code>
<code>bankverbindung_id</code>	
<code>bank_id</code>	

4. **Primärschlüssel festlegen:** Jetzt markieren Sie in der gleichen Farbe in der anderen Tabelle den dazugehörigen Primärschlüssel. Hier ist es die Spalte `bank_id`

<code>bankverbindung</code>	<code>bank</code>
<code>kunde_id</code>	<code>bank_id</code>
<code>bankverbindung_id</code>	
<code>bank_id</code>	

5. **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
```

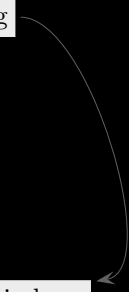
```
SELECT
kunde_id, kontonummer, blz, bankname
FROM
[ ] INNER JOIN [ ]
ON [ ] . [ ] - = [ ] . [ ] -
```

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
bankverbindung INNER JOIN [ ]
ON [ ] . [ ] - = [ ] . [ ] -
```

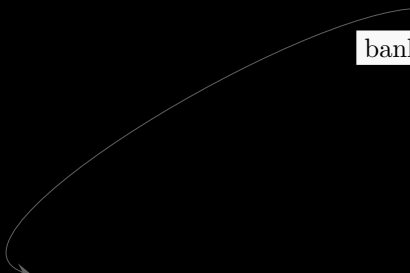


7. **Primärschlüsseltabelle eintragen:** Fügen Sie rechts vom INNER JOIN den Tabellennamen für die Tabelle mit dem markierten Primärschlüssel ein.

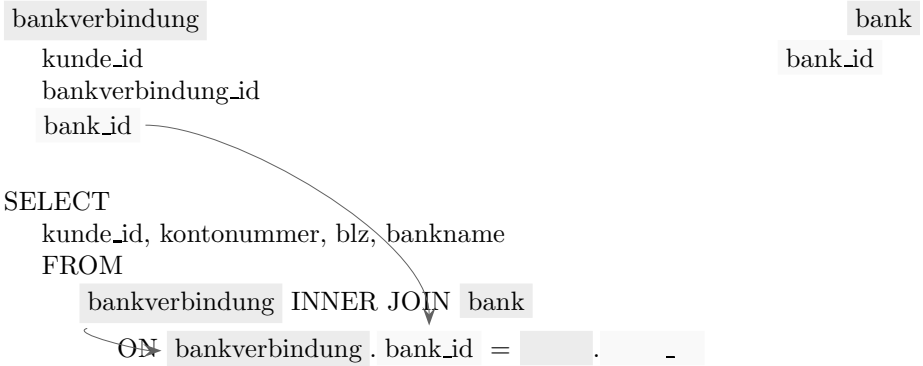
```
bankverbindung
kunde_id
bankverbindung_id
bank_id
```

```
bank
bank_id
```

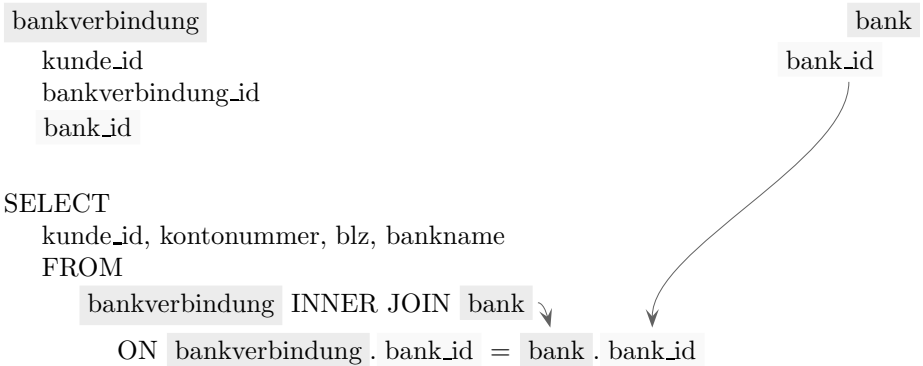
```
SELECT
kunde_id, kontonummer, blz, bankname
FROM
bankverbindung INNER JOIN bank
ON [ ] . [ ] - = [ ] . [ ] -
```



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 Primärschlüssel-tabelle, 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.

1. **Ermitteln der beteiligten Tabellen:** Es sind kunde und bankverbindung.
2. **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   | 333333331   |
10 | Beutlin    | Frodo   | 222222221   |
11 | Earendilionn | Elrond  | 555555551   |
12 | Gamdschie  | Samweis | 111111111   |
13 | Gamdschie  | Samweis | 111111112   |
14 | Telcontar  | Elessar | 444444441   |
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-/Primärschlüsselspalten 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 liefert<sup>5</sup>.

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.



### MySQL/MariaDB

```

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 |

```

<sup>5</sup> 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.

```

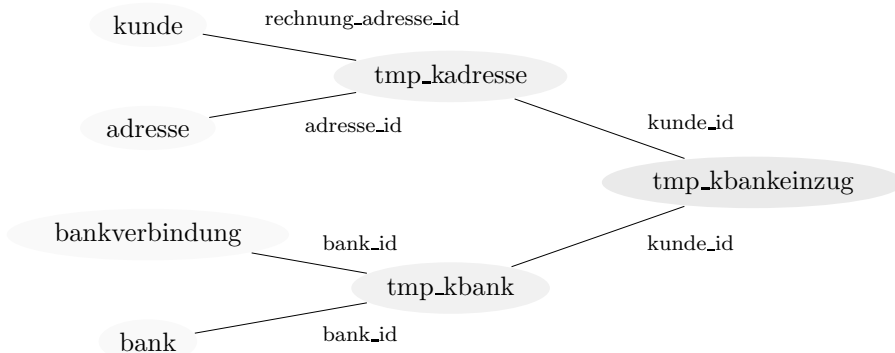
1  mysql> CREATE TEMPORARY TABLE tmp_kbank
2      -> SELECT
3      ->  bv.kunde_id, bv.bankverbindung_nr, bv.kontonummer,
4      ->  bv.iban, ba.blz, ba.bankname
5      -> FROM
6      ->  bankverbindung bv INNER JOIN bank ba USING (bank_id);
7
8  mysql> CREATE TEMPORARY TABLE tmp_kbankeinzug
9      -> 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 Blätter<sup>6</sup> sind die Ursprungstabellen. Zwei davon werden in je-

<sup>6</sup> 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 abzuwägen. 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   |     | 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
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 Nichtschlüsselspalten<sup>7</sup> 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 +-----+-----+-----+-----+-----+-----+
8 | kunde_id | vorname | strasse          | hnr | plz | ort          |
9 +-----+-----+-----+-----+-----+-----+
10 | 1 | Samweis | Beutelhaldenweg | 5 | 67676 | Hobbingen |
11 | 1 | Samweis | Beutelhaldenweg | 1 | 67676 | Hobbingen |
12 | 2 | Frodo   | Beutelhaldenweg | 1 | 67676 | Hobbingen |
13 | 3 | Bilbo   | Beutelhaldenweg | 1 | 67676 | Hobbingen |
14 | 1 | Samweis | Auf der Feste    | 1 | 54786 | Minas Tirith |
15 | 2 | Frodo   | Auf der Feste    | 1 | 54786 | Minas Tirith |
16 | 3 | Bilbo   | Auf der Feste    | 1 | 54786 | Minas Tirith |
17 | 4 | Elessar | Auf der Feste    | 1 | 54786 | Minas Tirith |
18 | 1 | Samweis | Letztes Haus     | 4 | 87567 | Bruchtal    |
19 | 2 | Frodo   | Letztes Haus     | 4 | 87567 | Bruchtal    |
20 | 3 | Bilbo   | Letztes Haus     | 4 | 87567 | Bruchtal    |
21 [...]
22 | 2 | Frodo   | Baradur          | 1 | 62519 | Lugburz    |
23 | 3 | Bilbo   | Baradur          | 1 | 62519 | Lugburz    |
24 | 4 | Elessar | Baradur          | 1 | 62519 | Lugburz    |
25 | 5 | Elrond  | Baradur          | 1 | 62519 | Lugburz    |
26 | 1 | Samweis | Hochstrasse      | 4a | 44879 | Bochum     |
27 | 2 | Frodo   | Hochstrasse      | 4a | 44879 | Bochum     |
28 | 3 | Bilbo   | Hochstrasse      | 4a | 44879 | Bochum     |
29 | 4 | Elessar | Hochstrasse      | 4a | 44879 | Bochum     |
30 | 5 | Elrond  | Hochstrasse      | 4a | 44879 | Bochum     |
31 | 1 | Samweis | Industriegebiet  | 8 | 44878 | Bochum     |
32 | 2 | Frodo   | Industriegebiet  | 8 | 44878 | Bochum     |
33 | 3 | Bilbo   | Industriegebiet  | 8 | 44878 | Bochum     |
34 | 4 | Elessar | Industriegebiet  | 8 | 44878 | Bochum     |

```

<sup>7</sup> 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. **Fremdschlüsseltable eintragen:** Fügen Sie links vom INNER JOIN den Tabellennamen für die Tabelle mit dem markierten Fremdschlüssel ein.
7. **Primärschlüsseltable 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.warengruppe_id
3      -> FROM
4      -> artikel_nm_warengruppe nm NATURAL JOIN artikel a;
5  +-----+-----+
6  | bezeichnung | warengruppe_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_warengruppe` und `warengruppe` führt zu einer zweiten Verknüpfung:

```

1  mysql> SELECT
2      -> w.bezeichnung, nm.artikel_id
3      -> FROM
4      -> artikel_nm_warengruppe nm NATURAL JOIN warengruppe 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 dienen 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
  tablename [INNER JOIN tablename
    ON tablename.spaltenname = tablename.spaltenname])*
[WHERE bedingung]
[ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
```

#### MySQL/MariaDB

```
SELECT [DISTINCT]
  {*|spaltenliste|ausdruck}
FROM
  tablename [INNER JOIN tablename
    {ON tablename.spaltenname = tablename.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?

Mchte 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 | Earendilioni | Elrond | NULL   |
13 |         6 | Eichenschild | Thorin | NULL   |
14 |         1 | Gamdschie | Samweis | 2012-03-24 17:41:00 |
15 |         4 | Telcontar | Elessar | NULL   |
16 +-----+-----+-----+-----+

```

Zuerst fllt auf, dass nicht zwei, sondern sechs Zeilen ausgegeben werden, denn jetzt werden auch die Kunden angezeigt, fr die keine Bestellungen vorliegen (Zeilen 10, 12, 13 und 15). Da SQL nicht wei, 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, fr welche keine passende 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 erklren als zu benutzen. Betrachten wir noch einmal obiges Beispiel. Der `INNER JOIN` liefert uns nur die Zeilen, fr welche ein passendes Primr-Fremdschlsselpaar 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, fr die keine passenden Primr-Fremdschlsselpaare gefunden werden. Und tatschlich, es sind dies die Kunden ohne Bestellung; fr 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
  tablename [[RIGHT OUTER|LEFT OUTER|FULL OUTER|INNER] JOIN tablename
    ON tablename.spaltenname = tablename.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
  tablename [[RIGHT [OUTER]]LEFT [OUTER]]INNER] JOIN tablename
    {ON tablename.spaltenname = tablename.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 wissen<sup>8</sup>.

```

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)

```

<sup>8</sup> 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 referenzielle 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

```

```

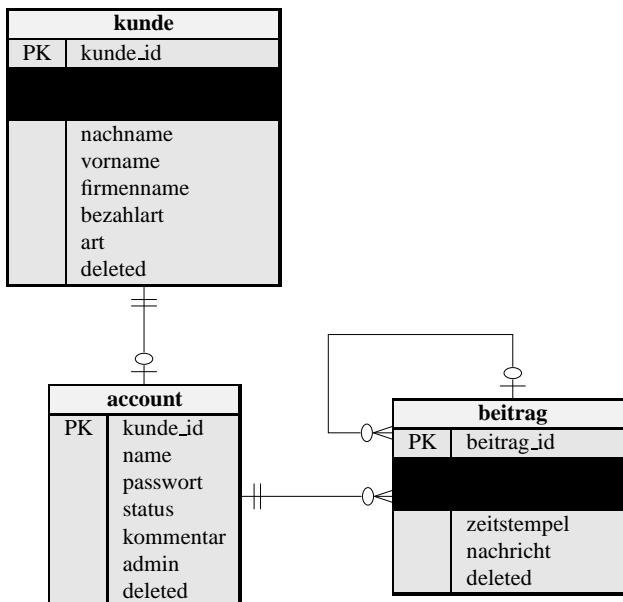
4 oshop-#  bestellung FULL OUTER JOIN kunde USING(kunde_id)
5 oshop-# ;
6
7 kunde_id | nachname | vorname | bestellung_id | datum
8 -----+-----+-----+-----+-----
9      1 | Gamschie | Samweis |      1 | 2012-03-24 17:41:00
10     2 | Beutlin  | Frodo   |      2 | 2012-03-23 16:11:00
11     5 | Earendil| 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 Verfügung<sup>9</sup>, 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;
```

<sup>9</sup> 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     -> beitrags_id, account_id, bezug_beitrags_id, LEFT(nachricht, 20)
16     -> FROM
17     -> beitrags;
18 +-----+-----+-----+-----+
19 | beitrags_id | account_id | bezug_beitrags_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 langs |
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     -> beitrags INNER JOIN beitrags
4     ->   ON bezug_beitrags_id = beitrags_id
5     ->   WHERE
6     ->     beitrags_id = 2;
7  ERROR 1066 (42000): Not unique table/alias: 'beitrags'

```

Die Fehlermeldung in Zeile 7 besagt, dass die Tabelle `beitrags` 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 `beitrags_id` jetzt die Spalte der linken oder rechten Tabelle `beitrags` 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 `beitrags` in der Rolle einer Antwort und rechts `orig` für `beitrags` 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     -> beitrags ant INNER JOIN beitrags orig
4     ->   ON ant.bezug_beitrags_id = orig.beitrags_id
5     ->   WHERE
6     ->     orig.beitrags_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*<sup>10</sup> 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 beitrags
6  -> WHERE beziehung_id = 2
7  -> UNION ALL
8  -> -- rekuriver Teil
9  -> SELECT ant.*
10 -> FROM
11 -> beitrags ant INNER JOIN ant_auf orig
12 -> ON ant.beziehung_id = orig.beitrags_id
13 -> )
14 -> SELECT beitrags_id, beziehung_id, nachricht FROM ant_auf;
15 +-----+-----+-----+
16 | beitrags_id | beziehung_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-

<sup>10</sup> Eine nähere Betrachtung von CTE muss hier leider aus Platzgründen entfallen. Eine gute Einführung finden Sie unter [Tut19].

ben – kann man übliche Verbindungen verbauen, 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 sein<sup>11</sup>. 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   | G0-54786 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?

<sup>11</sup> 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 erreichen<sup>12</sup>. Mithilfe eines Events<sup>13</sup> 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!

<sup>12</sup> Siehe Kapitel 22 auf Seite 363

<sup>13</sup> Siehe Kapitel 23 auf Seite 371

# Stichwortverzeichnis

- ||, 418
- () , 145
- \*, 162, 408
- +, 408
- , 408
- ~, 165
- .NET, 6
- /, 408
- ;;, 67
- <, 415
- <=, 415
- <=>, 414
- <>, 415
- =, 414
- >, 415
- >=, 415
- %, 408
- &&, 417
- \_id, 294
- !, 416
- !=, 415
  
- Abhängigkeit, mehrwertige, 25
- ABS(), 408
- Abweisende Schleife, 345
- ACID, 323
- ACOS(), 408
- AFTER, 131
- Aggregatfunktion, 211
- Alias, 162, 207
- ALL, 233, 234
- ALL(), 238
- ALTER DATABASE, 125, 419
- ALTER EVENT, 374, 419
- ALTER FUNCTION, 419
- ALTER INSTANCE, 420
- ALTER LOGFILE GROUP, 420
- ALTER PROCEDURE, 420
- ALTER SCHEMA, 125, 419
  
- ALTER SERVER, 420
- ALTER TABLE, 129, 420
  - ... ADD, 131, 149
  - ... ADD FOREIGN KEY, 176
  - ... ADD INDEX, 176
  - ... ADD PRIMARY KEY, 176
  - ... DISABLE KEYS, 104
  - ... DROP, 137
  - ... DROP FOREIGN KEY, 175
  - ... DROP INDEX, 175
  - ... DROP PRIMARY KEY, 175
  - ... ENABLE KEYS, 104
  - ... MODIFY, 132
  - ... RENAME, 130
- ALTER TABLESPACE, 422
- ALTER USER, 435
  - aktueller Benutzer, 436
  - Rolle, 436
- ALTER VIEW, 282, 422
- AND, 417
- Änderung, kaskadierende, 34, 86
- Änderungsweitergabe, 34, 86
- Annehmende Schleife, 345
- ANSI, 64
- Ansicht, 4, 273
  - Projektions-, 285
  - Selektions-, 283
  - veränderbare, 287
  - Verbund-, 284
- Antivalenz, 418
- ANY(), 237, 238
- ApacheFriends, 53
- API, 6
- ARCHIVE, 386
- Aria, 386
- Artikelverwaltung, 42
- AS IDENTITY, 75
- ASC, 164
- 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

IFDEF-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
- LOBLOB, 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
  - imperative, 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 Datenbanksystem, 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
- Vorbedingungen, 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