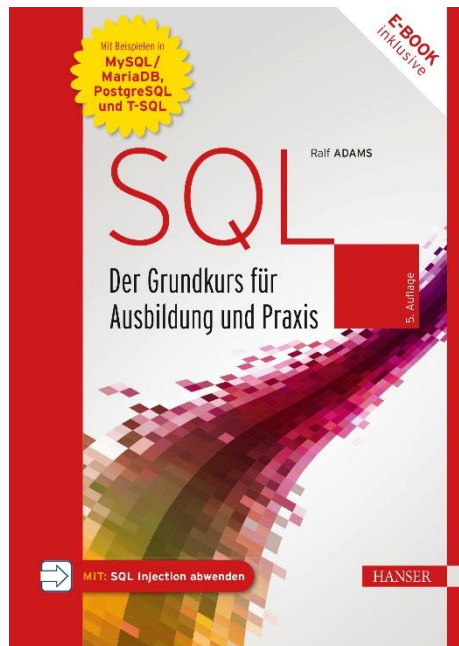


HANSER



Leseprobe

zu

SQL

von Ralf Adams

Print-ISBN: 978-3-446-47913-5

E-Book-ISBN: 978-3-446-47919-7

E-Pub-ISBN: 978-3-446-48028-5

Weitere Informationen und Bestellungen unter

<https://www.hanser-kundencenter.de/fachbuch/artikel/9783446479135>

sowie im Buchhandel

© Carl Hanser Verlag, München

Inhalt

Vorwort zur 5. Auflage	XVII
------------------------------	------

Teil I Was man so wissen sollte	1
---------------------------------------	---

1 Datenbanksystem	3
-------------------------	---

1.1 Aufgaben und Komponenten	3
1.1.1 Datenbank	3
1.1.2 Datenbankmanagementsystem	5
1.2 Im Buch verwendete Server	7
1.2.1 MySQL und MariaDB	7
1.2.2 PostgreSQL	9
1.2.3 Microsoft SQL Server	10

2 Relationale Datenbanken	11
---------------------------------	----

2.1 Einführung	11
2.1.1 Abgrenzung zu anderen Datenbanken	11
2.1.2 Tabelle, Zeile und Spalte	13
2.1.3 Schlüssel, Primärschlüssel und Fremdschlüssel	16
2.2 Kardinalitäten und ER-Modell	21
2.2.1 Darstellung von Tabellen im ER-Modell	22
2.2.2 1:1-Verknüpfung	23
2.2.2.1 Wann liegt eine 1:1-Verknüpfung vor?	23
2.2.2.2 Wie kann ich eine 1:1-Verknüpfung darstellen?	25
2.2.2.3 Kann ich die Kardinalität genauer beschreiben?	25
2.2.3 1:n-Verknüpfung	26
2.2.3.1 Wann liegt eine 1:n-Verknüpfung vor?	26
2.2.3.2 Wie kann ich eine 1:n-Verknüpfung darstellen?	27
2.2.3.3 Kann ich die Kardinalität genauer beschreiben?	27

2.2.4	<i>n:m</i> -Verknüpfung	28
2.2.4.1	Wann liegt eine <i>n:m</i> -Verknüpfung vor?	28
2.2.4.2	Wie kann ich eine <i>n:m</i> -Verknüpfung darstellen?	30
2.2.4.3	Kann ich die Kardinalität genauer beschreiben?	30
2.2.5	Aufgaben zum ER-Modell	30
2.3	Referenzielle Integrität	31
2.3.1	Verletzung der referenziellen Integrität durch Löschen	32
2.3.2	Verletzung der referenziellen Integrität durch Änderungen	33
2.4	Normalformen	33
2.4.1	Normalform 1	34
2.4.2	Normalform 2	36
2.4.3	Normalform 3	37
2.4.4	Normalform Rest	39
3	Unser Beispiel: Ein Online-Shop	41
3.1	Kundenverwaltung	41
3.2	Artikelverwaltung	42
3.3	Bestellwesen	43
Teil II	Datenbank aufbauen	45
4	Installation des Servers	47
4.1	MySQL unter Windows 11	47
4.2	MariaDB unter Windows 11	51
4.3	Andere Installationen mit Docker	55
4.3.1	MySQL	56
4.3.2	MariaDB	58
4.3.3	PostgreSQL	59
4.3.4	Microsoft SQL Server	60
5	Datenbank und Tabellen anlegen	61
5.1	Die Programmiersprache SQL	61
5.2	Anlegen der Datenbank	62
5.2.1	Wie rufe ich den MySQL Client auf?	63
5.2.2	Wie lege ich eine Datenbank an?	64
5.2.3	Wie lösche ich eine Datenbank?	65
5.2.4	Wie weise ich einen Zeichensatz zu?	66
5.2.5	Wie weise ich eine Sortierung zu?	68

5.3	Anlegen der Tabellen	70
5.3.1	Welche Datentypen gibt es?	70
5.3.2	Wie lege ich eine Tabelle an?	71
5.3.3	Wann eine Aufzählung (ENUM) und wann eine neue Tabelle?	74
5.3.4	Wann ein DECIMAL und wann ein DOUBLE?	76
5.3.5	Wann verwende ich NOT NULL?	77
5.3.6	Wie lege ich einen Fremdschlüssel fest?	80
5.3.7	Wie kann ich Tabellen aus anderen herleiten?	85
5.3.8	Ich brauche mal eben kurz 'ne Tabelle!	86
6	Indizes anlegen	89
6.1	Index für Anfänger	89
6.1.1	Wann wird ein Index automatisch erstellt?	90
6.1.2	Wie kann ich einen Index manuell erstellen?	93
6.2	Und jetzt etwas genauer	95
6.2.1	Wie kann ich die Schlüsseleigenschaft erzwingen?	95
6.2.2	Wie kann ich Dubletten verhindern?	96
6.2.3	Was bedeutet Indexselektivität?	98
6.2.4	Wie kann ich einen Index löschen?	100
7	Werte in Tabellen einfügen	101
7.1	Daten importieren	101
7.1.1	Das CSV-Format	101
7.1.2	CSV-Daten laden mit LOAD DATA INFILE	103
7.1.3	Was ist, wenn ich geänderte Werte importieren will?	107
7.2	Daten anlegen	108
7.2.1	Wie lege ich mehrere Zeilen mit einem Befehl an?	109
7.2.2	Wie kann ich eine einzelne Zeile anlegen?	110
7.2.3	Vorsicht Constraints!	111
7.2.4	Einfügen von binären Daten über einen C#-Client	112
7.2.5	Einfügen von binären Daten LOAD FILE	115
7.3	Daten kopieren	116
Teil III	Datenbank ändern	119
8	Datenbank und Tabellen umbauen	121
8.1	Eine Datenbank ändern	121
8.2	Eine Datenbank löschen	123

8.3	Eine Tabelle ändern	125
8.3.1	Wie kann ich den Namen der Tabelle ändern?	125
8.3.2	Wie kann ich eine Spalte hinzufügen?	126
8.3.3	Wie kann ich die Spezifikation einer Spalte ändern?	128
8.3.4	Zeichenbasierte Spalten in der Länge verändern	129
8.3.5	Zeichensatz verändern	130
8.3.6	Zeichenbasierte Spalten in numerische Spalten verändern	130
8.3.7	Numerische Spalten im Wertebereich verändern	131
8.3.8	Datum- oder Zeitspalten verändern	131
8.3.9	Wie kann ich aus einer Tabelle Spalten entfernen?	133
8.4	Eine Tabelle löschen	134
8.4.1	Einfach löschen	134
8.4.2	Was bedeuten CASCADE und RESTRICT?	135
9	Werte in Tabellen verändern	137
9.1	WHERE-Klausel	137
9.1.1	Wie formuliere ich eine einfache Bedingung?	138
9.1.2	Wird zwischen Groß- und Kleinschreibung unterschieden?	139
9.1.3	Wie formuliere ich eine zusammengesetzte Bedingung?	140
9.2	Tabelleninhalte verändern	142
9.2.1	Szenario 1: Einfache Wertzuweisung	143
9.2.2	Szenario 2: Berechnete Werte	144
9.2.3	Szenario 3: Gebastelte Zeichenketten	145
9.2.4	Was bedeuten LOW_PRIORITY und IGNORE?	145
9.3	Tabelleninhalte löschen	146
9.3.1	Und was passiert bei Constraints?	147
9.3.2	Was passiert mit dem AUTO_INCREMENT?	148
9.3.3	Was bedeuten LOW_PRIORITY, QUICK und IGNORE?	149
9.3.4	Wie kann ich eine Tabelle komplett leeren?	149
Teil IV	Datenbank auswerten	151
10	Einfache Auswertungen	153
10.1	Ausdrücke	154
10.1.1	Konstanten	154
10.1.2	Wie kann ich Berechnungen vornehmen?	154
10.1.3	Wie ermittle ich Zufallszahlen?	155
10.1.4	Wie stecke ich das Berechnungsergebnis in eine Variable?	157

10.2	Zeilen- und Spaltenwahl	158
10.3	Sortierung	159
10.3.1	Was muss ich bei der Sortierung von Texten beachten?	161
10.3.2	Wird zwischen Groß- und Kleinschreibung unterschieden?	163
10.3.3	Wie werden Datums- und Uhrzeitwerte sortiert?	165
10.3.4	Wie kann ich das Sortieren beschleunigen?	166
10.4	Mehrfachausgaben unterbinden	169
10.4.1	Fallstudie: Datenimport von Bankdaten	170
10.4.2	Was muss ich beim DISTINCT bzgl. der Performance beachten?	173
10.5	Ergebnismenge ausschneiden	173
10.5.1	Wie kann ich die ersten n Datensätze ausschneiden?	173
10.5.2	Wie kann ich Teilmengen mittendrin ausschneiden?	174
10.6	Ergebnisse exportieren	176
10.6.1	Wie lege ich eine Exportdatei auf dem Server an?	176
10.6.2	Wie lege ich eine Exportdatei auf dem Client an?	177
10.6.3	Wie lese ich mithilfe eines C#-Client binäre Daten aus?	177
11	Tabellen verbinden	179
11.1	Heiße Liebe: Primär-Fremdschlüsselpaare	180
11.2	INNER JOIN zwischen zwei Tabellen	183
11.2.1	Bauanleitung für einen INNER JOIN	184
11.2.2	Abkürzende Schreibweisen	188
11.2.3	Als Datenquelle für temporäre Tabellen	189
11.2.4	JOIN über Nichtschlüsselspalten	191
11.3	INNER JOIN über mehr als zwei Tabellen	193
11.4	Es muss nicht immer heiße Liebe sein: OUTER JOIN	196
11.5	Narzissmus pur: Self Join	201
11.6	Eine Verknüpfung beschleunigen	205
12	Differenzierte Auswertungen	207
12.1	Statistisches mit Aggregatfunktionen	207
12.2	Tabelle in Gruppen zerlegen	210
12.3	Gruppenergebnisse filtern	214
12.4	Noch Fragen?	216
12.4.1	Kann ich nach Ausdrücken gruppieren?	216
12.4.2	Kann ich nach mehr als einer Spalte gruppieren?	217
12.4.3	Wie kann ich GROUP BY beschleunigen?	217
12.4.4	Parallele Bearbeitung – unterschiedliche Ergebnisse?	219
12.5	Aufgaben	219

13	Auswertungen mit Unterabfragen	221
13.1	Das Problem und die Lösung	221
13.2	Nicht korrelierende Unterabfrage	224
13.2.1	Skalarunterabfrage	224
13.2.1.1	Beispiel 1: Banken mit höchster BLZ	224
13.2.1.2	Beispiel 2: Überdurchschnittlich teure Artikel	225
13.2.1.3	Beispiel 3: Überdurchschnittlich wertvolle Bestellungen	226
13.2.2	Listenunterabfrage	227
13.2.2.1	Beispiel 1: IN ()	227
13.2.2.2	Beispiel 2: ALL ()	229
13.2.2.3	Beispiel 3: ALL ()	230
13.2.2.4	Beispiel 4: ANY ()	232
13.2.2.5	Unterschied zwischen IN (), ALL () und ANY ()	234
13.2.2.6	Unterschied zwischen NOT IN () und <> ALL ()	235
13.2.3	Tabellenunterabfrage	235
13.3	Korrelierende Unterabfrage	236
13.3.1	Beispiel 1: Rechnungen mit vielen Positionen	236
13.3.2	Beispiel 2: EXISTS	237
13.4	Common Table Expression versus Unterabfrage	237
13.5	Fallstudie Datenimport	239
13.6	Wie ticken Unterabfragen intern?	243
13.7	Aufgaben	244
14	Mengenoperationen	245
14.1	Die Vereinigung mit UNION	245
14.2	Die Schnittmenge	248
14.2.1	Mit INTERSECT	248
14.2.2	Mit Unterabfragen	249
14.3	Die Differenzmenge	250
14.3.1	Mit EXCEPT	250
14.3.2	Mit Unterabfragen	251
14.4	UNION, INTERSECT und EXCEPT ... versteh' ich nicht!	252
15	Bedingungslogik	255
15.1	Warum ein CASE?	255
15.2	Einfacher CASE	257
15.3	Searched CASE	259
15.4	Fallbeispiele	261
15.4.1	Lagerbestand überprüfen	261

15.4.2	Kundengruppen ermitteln	262
15.4.3	Aktive Lieferanten ermitteln	265
15.4.4	Aufgaben	266
16	Ansichtssache	267
16.1	Was ist eine Ansicht?	267
16.1.1	Wie lege ich eine Ansicht an?	268
16.1.2	Wie wird eine Ansicht verarbeitet?	270
16.1.3	Wie lösche ich eine Ansicht?	273
16.1.4	Wie ändere ich eine Ansicht?	276
16.2	Anwendungsgebiet: Vereinfachung	276
16.3	Anwendungsgebiet: Datenschutz	279
16.4	Grenzen einer Ansicht	279
17	Exkurs NoSQL	283
17.1	Vorbereitung der MySQL-Shell	283
17.2	Datenmodellierung des Warenkorbs	285
17.2.1	JavaScript Object Notation (JSON)	285
17.2.2	Struktur unseres JSON-Dokuments	286
17.3	NoSQL: MySQL mit JavaScript-Client	287
17.3.1	Anlegen eines Warenkorbs	288
17.3.2	Inhalte des Warenkorbs anlegen	289
17.3.3	Inhalte des Warenkorbs auswerten	293
17.3.4	Inhalte des Warenkorbs verändern	296
17.4	NoSQL: klassisches SQL mit JSON-Funktionen	297
17.4.1	Anlegen eines Warenkorbs	298
17.4.2	Inhalte des Warenkorbs anlegen	298
17.4.3	Inhalte des Warenkorbs auswerten	300
17.4.4	Inhalte des Warenkorbs verändern	302
17.4.5	Inhalte des Warenkorbs löschen	305
Teil V	Anweisungen kapseln	307
18	Locking	309
19	Transaktionen	313
19.1	Das Problem	313
19.2	Was ist eine Transaktion?	315
19.3	Isolationsebenen	318

19.3.1	READ UNCOMMITTED	319
19.3.2	READ COMMITTED	320
19.3.3	REPEATABLE READ	321
19.3.4	SERIALIZABLE	322
19.4	Fallbeispiel in C#	323
19.5	Deadlock	325
20	Prozeduren	327
20.1	Einstieg und Variablen	328
20.2	Verzweigung	333
20.2.1	Einfache Verzweigung mit IF	333
20.2.2	Mehrfache Verzweigung mit CASE	336
20.3	Schleifen	339
20.3.1	LOOP-Schleife	340
20.3.2	WHILE-Schleife	342
20.3.3	REPEAT-Schleife	345
20.4	Transaktion innerhalb einer Prozedur	346
20.5	Cursor	347
20.6	Aufgaben	354
21	Funktionen	355
22	Auslöser	357
22.1	Was ist das?	357
22.2	Ein Beispiel für einen INSERT-Trigger	359
22.3	Ein Beispiel für einen UPDATE-Trigger	360
22.4	Ein Beispiel für einen DELETE-Trigger	362
23	Ereignisse	365
23.1	Wie lege ich ein Ereignis an?	365
23.2	Wie werde ich Ereignisse wieder los?	368
Teil VI	Anhänge	369
24	Datenbank administrieren	371
24.1	Daten sichern und wiederherstellen	371
24.1.1	Backup	371
24.1.2	Restore	373
24.2	Benutzerrechte	373
24.2.1	Benutzerrechte und Privilegien	373

24.2.2	Benutzer anlegen und Rechte zuweisen	376
24.2.2.1	CREATE USER bzw. CREATE ROLE	376
24.2.2.2	Rechte vergeben mit GRANT	378
24.2.2.3	Rechte entziehen mit REVOKE	379
24.3	MySQL und MariaDB Engines	380
25	SQL-Injection	383
25.1	Das Problem	383
25.2	Beispiel: Suchmaske	384
25.3	Gegenmaßnahmen	391
25.3.1	Never Trust an Unknown Input	391
25.3.2	Trennung von Anweisungen und Daten	392
25.3.2.1	Verwenden Sie Prozeduren oder Funktionen	392
25.3.2.2	Verwenden Sie Prepared Statements	392
25.3.3	Ich darf nur, was ich soll	393
25.3.4	Nichtssagende Fehlermeldungen	394
26	SQL-Referenz	395
26.1	Datentypen	395
26.1.1	Numerische Datentypen	395
26.1.1.1	Ganze Zahlen	395
26.1.1.2	Gebrochene Zahlen	396
26.1.2	Zeichen-Datentypen	397
26.1.3	Datums- und Zeit-Datentypen	398
26.1.4	Binäre Datentypen	401
26.1.5	JSON	401
26.1.6	Räumliche Datentypen	402
26.1.7	Standardwerte	403
26.1.8	Zusätze für Datentypen	404
26.2	Operatoren und Funktionen	406
26.2.1	Mathematische Operatoren	406
26.2.2	Mathematische Funktionen	406
26.2.3	Aggregatfunktionen	409
26.3	Bedingungen	412
26.3.1	Vergleichsoperatoren	412
26.3.2	Logikoperatoren	415
26.3.2.1	NOT, Negation, $\neg$	415
26.3.2.2	AND, Konjunktion, $\wedge$	415
26.3.2.3	OR, Disjunktion, $\vee$	416

26.3.2.4	XOR, Antivalenz, $\otimes$	416
26.3.2.5	Verallgemeinerung auf mehr als zwei Operanden	416
26.4	Befehle	417
26.4.1	Data Definition Language	417
26.4.2	Data Manipulation Language	429
26.4.3	Benutzerverwaltung	433
27	Ausgewählte Quelltexte	437
27.1	DOUBLE versus DECIMAL	437
27.2	Rundungsfehler	441
27.3	NULL versus NOT NULL	441
27.4	Suchen mit und ohne Index	444
27.5	Messen der Performance der Einfügeoperation	447
27.6	Messen der Indexselektivität	450
27.7	Sortieren ohne und mit Index	452
28	Quelltexte	457
28.1	MySQL/MariaDB	457
28.1.1	Quelltexte zu Teil II	457
28.1.2	Quelltexte zu Teil III	469
28.1.3	Quelltexte zu Teil IV	473
28.1.4	Quelltexte zu Teil V	517
28.1.5	Quelltexte zu Teil VI	531
28.2	PostgreSQL	536
28.2.1	Quelltexte zu Teil II	536
28.2.2	Quelltexte zu Teil III	545
28.2.3	Quelltexte zu Teil IV	549
28.2.4	Quelltexte zu Teil V	579
28.3	Microsoft SQL Server	584
28.3.1	Quelltexte zu Teil II	584
28.3.2	Quelltexte zu Teil III	595
28.3.3	Quelltexte zu Teil IV	600
Literatur		635
Stichwortverzeichnis		641

Vorwort zur 5. Auflage

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

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

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

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

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

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

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

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

- Ich habe angefangen, für die Aufgaben Musterlösungen bei YouTube (<http://www.youtube.com/channel/UCu4ZybNXw1y4Rs4Mgx-4HKw>) einzustellen. In diesen Videos kann ich einfach besser erklären, worauf es bei den Lösungen ankommt.

Wenn Sie auf

plus.hanser-fachbuch.de

den Code

eingeben, können Sie Skripte, Beispiele und Musterlösungen downloaden. Diese wurden auf folgenden Servern getestet: *MySQL Community Server 8.0.33*, *MariaDB 10.6.14*, *MariaDB 11.0.2*, *PostgreSQL 15.3* und *MS SQL Server 2022 16.x*. Alle Server liefen in einer Dockerinstanz unter Debian 12 (Bookworm). Für alle Quelltexte, die bis einschließlich Kapitel 16 vorgestellt werden, gibt es Varianten in MySQL/MariaDB, PostgreSQL und T-SQL. Nur bei ganz wenigen Ausnahmen, die durch die Dialekte oder Eigenheiten begründet sind, musste ich auf eine Transkription verzichten.

Seit kurzem gibt es außerdem im Hanser Verlag den Hanser eCampus, das adaptive Kursangebot für die Hochschullehre und die Unternehmensweiterbildung. Hier ist der E-Learning-Kurs *Datenbankgrundlagen* erschienen, der auf Teil I dieses Buches basiert. Weitere Infos zum Kurs und eine Demoversion finden Sie hier:

<https://www.hanser-ecampus.de/kurse/informatik/datenbankgrundlagen>.

Danksagung

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

Besonders will ich mich bei meinen Schülerinnen und Schülern der Technischen Beruflichen Schule 1 in Bochum (<http://www.tbs1.de>) bedanken. Die hier vorgestellten Beispiele und Konzepte sind in großen Teilen durch ihre schonungslose Kritik an bestehenden Lehrmaterialien entstanden. Das penetrante *Kapier ich nicht!* hat mich immer weiter angespornt, es noch verständlicher zu versuchen. Falls dieses Buch SQL gut vermittelt, ist das auch deren Verdienst.

Dass nun aber die 5. Auflage dieses Buchs erscheinen kann, ist in erster Linie Ihnen, liebe Leserinnen und Leser, zu verdanken; dafür ein herzliches *Dankeschön!*

Ralf Adams, September 2023



Auswertungen mithilfe von Aggregatfunktionen erstellen.

- Grundkurs
 - Einfache Statistik mit `MIN()`, `MAX()`, `SUM()`, `COUNT()` und `AVG()`
 - Tabellen mit `GROUP BY` in Gruppen zerlegen
 - Aggregatfunktionen auf Gruppen anwenden
 - Gruppenergebnisse mit `HAVING` aussortieren
- Vertiefendes
 - Aggregatfunktionen
 - Summenbildung mit `WITH ROLLUP`
 - Gruppieren nach Ausdrücken
 - Gruppieren nach mehr als einer Spalte
 - Einfluss von Indizes auf Gruppierungen



Die Quelltexte dieses Kapitels stehen in der Datei `listing09.sql` (siehe [Listing 28.9](#), [Listing 28.49](#) und [Listing 28.34](#)).

12.1 Statistisches mit Aggregatfunktionen

Die Daten, die bisher aus einem `SELECT` kamen, sind immer Originaldaten gewesen. Soll heißen, es wurden nur Texte oder Zahlen angezeigt, die in den Tabellen so abgelegt waren. Mithilfe von Aggregatfunktionen werden nun Auswertungen über die Daten erstellt. Eine Übersicht der verfügbaren Aggregatfunktionen finden Sie in [Abschnitt 26.2.3](#). Der *ausdruck* in [Abschnitt 26.2.3](#) ist meist ein Spaltenname oder eine mathematische Kombination aus Spaltennamen.

Die wichtigsten Aggregatfunktionen sind: `MIN()`, `MAX()`, `SUM()`, `COUNT()` und `AVG()`. Mit `MIN()` und `MAX()` lassen sich der minimale und maximale Wert einer Liste ermitteln. `SUM()` addiert die Werte einer Liste auf, und `COUNT()` zählt die Anzahl von Werten. Mit `AVG()` wird das arithmetische Mittel einer Werteliste berechnet.

Diese Funktionen lassen sich überall da einbauen, wo man mit Werten arbeitet. Beim SELECT beispielsweise in der Spaltenliste oder beim WHERE in den Vergleichen usw. Wir wollen mal die meisten Aggregatfunktionen an unseren Daten ausprobieren:

Was ist der durchschnittliche Preis unserer Artikel?

Der Artikelpreis steht in der Spalte `einzelpreis`. Da dieser mit der Option `NOT NULL` erstellt wurde, erwarten wir keine Schwierigkeiten.

```

1 SELECT AVG(einzelpreis) FROM artikel;
2 -----+
3 | AVG(einzelpreis) |
4 +-----+
5 |      12.577777778 |
6 +-----+
```

Wie viele Zeilen hat eine Tabelle?

Wir verwenden hier die Tabelle `kunde`, es könnte aber auf diese Art und Weise von jeder beliebigen Tabelle die Anzahl der Zeilen ermittelt werden.

```

1 SELECT COUNT(*) FROM kunde;
2 +-----+
3 | COUNT(*) |
4 +-----+
5 |          6 |
6 +-----+
```

Wie viele Kunden haben eine eigene Lieferadresse?

Da der Inhalt des Fremdschlüssels `liefer_adresse_id` den Wert `NULL` hat, wenn keine eigene Lieferadresse erfasst ist, kann über diese Spalte die Anzahl ermittelt werden.

```

1 SELECT COUNT(liefer_adresse_id) FROM kunde;
2 +-----+
3 | COUNT(liefer_adresse_id) |
4 +-----+
5 |                          1 |
6 +-----+
```

Wie viele unterschiedliche Rechnungsadressen gibt es?

Die Tabelle `kunde` hat in der Spalte `rechnung_adresse_id` die Fremdschlüsselwerte zu den Rechnungsadressen abgelegt. Die Anzahl unterschiedlicher Werte liefert mir das gewünschte Ergebnis.

```

1 SELECT COUNT(DISTINCT(rechnung_adresse_id)) FROM kunde;
2 +-----+
3 | COUNT(DISTINCT(rechnung_adresse_id)) |
4 +-----+
5 |                                  4 |
6 +-----+
```

Was ist unser teuerster und unser billigster Einzelpreis?

Die Tabelle artikel enthält in der Spalte einzelpreis unsere Preise.

```

1  SELECT MAX(einzelpreis), MIN(einzelpreis) FROM artikel;
2  +-----+-----+
3  | MAX(einzelpreis) | MIN(einzelpreis) |
4  +-----+-----+
5  |          56.260000 |          0.520000 |
6  +-----+-----+
```

Wie viele Einzelartikel sind bestellt worden?

Die Bestellmenge steht in der Tabelle bestellung_position in der Spalte menge.

```

1  SELECT SUM(menge) FROM bestellung_position;
2  +-----+
3  | SUM(menge) |
4  +-----+
5  |  96.000000 |
6  +-----+
```

Wie hoch ist das Bestellvolumen?

Hier werden zwei Tabellen zur Auswertung benötigt: bestellung_position und artikel. Bilden Sie zuerst den INNER JOIN wie in [Abschnitt 11.2.1](#) beschrieben.

```

1  mysql> SELECT
2      ->  bp.menge, a.einzelpreis
3      ->  FROM
4      ->  bestellung_position bp INNER JOIN artikel a USING(artikel_id);
5  +-----+-----+
6  | menge   | einzelpreis |
7  +-----+-----+
8  | 30.000000 | 0.520000 |
9  | 50.000000 | 3.420000 |
10 | 1.000000 | 20.100000 |
11 | 10.000000 | 0.520000 |
12 | 5.000000 | 15.100000 |
13 +-----+-----+
```

Der Wert einer Position lässt sich nun einfach dadurch ermitteln, dass die beiden Werte menge und einzelpreis miteinander multipliziert werden:

```

1  mysql> SELECT
2      ->  bp.menge * a.einzelpreis 'Positionswert'
3      ->  FROM
4      ->  bestellung_position bp INNER JOIN artikel a USING(artikel_id);
5  +-----+
6  | Positionswert |
7  +-----+
8  | 15.600000000000 |
9  | 171.000000000000 |
10 | 20.100000000000 |
11 | 5.200000000000 |
12 | 75.500000000000 |
13 +-----+
```

Zum Schluss werden die Positionswerte summiert.

```

1 SELECT
2   SUM(bp.menge * a.einzelpreis)
3 FROM
4   bestellung_position bp INNER JOIN artikel a USING(artikel_id);
5 +-----+
6 | SUM(bp.menge * a.einzelpreis) |
7 +-----+
8 |                287.400000000000 |
9 +-----+

```

Wir haben hier ein Beispiel dafür, dass als Parameter einer Aggregatfunktion nicht nur Spaltennamen, sondern auch Ausdrücke vorkommen können.



Aufgabe 12.1: Hier ein paar Übungsaufgaben. Für die Übungsaufgaben habe ich in listing09.sql noch einen Lagerbestand aufgebaut.

- Ergänzen Sie das ER-Modell in [Bild 3.2](#) um die Lagerverwaltung in listing09.sql.
- Ermitteln Sie die durchschnittliche Bestellmenge.
- Ermitteln Sie die Anzahl der Artikel mit der Währung USD.
- Ermitteln Sie die Anzahl der Privatkunden.
- Ermitteln Sie die Anzahl der Kunden, die noch keine Bestellung aufgegeben haben.
- Was ist unsere kleinste und was unsere größte Bestellmenge?
- Ermitteln Sie, wie viele ml Tinte noch auf Lager sind.
- Ermitteln Sie den Wert des Lagers.

■ 12.2 Tabelle in Gruppen zerlegen

Die Aggregatfunktionen liefern uns bis jetzt ihre Ergebnisse bezogen auf die ganze Tabelle, also *eine* Zahl. Es kommt aber viel häufiger vor, dass man die Auswertung pro Kunde, pro Postleitzahl, pro Bestellung, pro Artikel etc. vornehmen möchte, z.B. Anzahl der Bestellpositionen pro Artikel. Wir brauchen ein Werkzeug, was die Anwendung von Aggregatfunktionen auf Teiltabellen (Gruppen) ermöglicht.



SQL:2023, MySQL/MariaDB, PostgreSQL, T-SQL

```

SELECT [DISTINCT]
  {*|spaltenliste|ausdruck}
FROM
  from_ausdruck
[WHERE bedingung]
[GROUP BY spaltenliste|ausdruck]
[ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
[LIMIT [offset ,] anzahl]
[INTO OUTFILE 'dateiname' exportoptionen]
;

```

Wie viele Positionen pro Bestellungen gibt es?

Da es irgendwas mit *Anzahl* zu tun hat, ist COUNT () unser Mann.

```

1  mysql> SELECT COUNT(*)
2      -> FROM bestellung_position
3      -> ;
4  +-----+
5  | COUNT(*) |
6  +-----+
7  |         5 |
8  +-----+
```

Jetzt wissen wir die Anzahl der Positionen überhaupt. Wir wollen aber wissen, wie viele es pro Bestellung sind. Wir müssen also ein GROUP BY einfügen. Aber was steht in *spaltenliste* hinter dem GROUP BY? Die Spalte, die bestimmt, zu welcher Gruppe die Zeile gehört. In unserem Fall der Fremdschlüssel auf die Tabelle bestellung.

```

1  mysql> SELECT
2      -> bestellung_id Bestellnummer, COUNT(*) 'Anzahl der Positionen'
3      -> FROM
4      -> bestellung_position
5      -> GROUP BY
6      -> bestellung_id
7      -> ;
8  +-----+-----+
9  | Bestellnummer | Anzahl der Positionen |
10 +-----+-----+
11 |             1 |                      3 |
12 |             2 |                      2 |
13 +-----+-----+
```

Noch mal: Schauen wir uns die erste Version an, so wird die Aggregatfunktion COUNT(*) auf die gesamte Tabelle angewendet. In der zweiten Version weisen wir an, dass die Tabelle in Gruppen zerlegt wird. Als Unterscheidungsmerkmal wird in [Zeile 6](#) die Spalte bestellung_id angegeben. Das bedeutet, dass alle Zeilen mit dem gleichen Spaltenwert zur gleichen Gruppe gehören. Daher gibt es jetzt zwei Gruppen: für jede bestellung_id eine.

Ist eine Tabelle in Gruppen zerlegt worden, so werden die Aggregatfunktionen immer pro Gruppe ausgeführt. In unserem Fall bedeutet das, dass pro Bestellung COUNT(*) die Zeilen in bestellung_position zählt.



Hinweis: In [Zeile 2](#) wird ein Alias mit Leerzeichen verwendet. Deshalb muss um den Alias eine Stringbegrenzung erfolgen. Die *normalen* Hochkomma ' oder Gänsefüßchen " können in MySQL/MariaDB nicht verwendet werden, da an dieser Stelle auch Stringkonstanten stehen könnten. Als neues Zeichen wird auch ein Hochkomma ' verwendet, aber das auf der Taste links neben dem Backspace.

Wir wollen wissen, wie oft ein Artikel bestellt wurde

Dabei soll der Artikelname mit ausgegeben werden. Zuerst wenden wir bzgl. der beiden Tabellen bestellung_position und artikel das in [Abschnitt 11.2.1](#) beschriebene Verfahren an, um den Artikelname zu erfahren.

```

1 mysql> SELECT a.bezeichnung, bp.menge
2       -> FROM
3       -> bestellung_position bp INNER JOIN artikel a USING(artikel_id)
4       -> ;
5 +-----+-----+
6 | bezeichnung | menge |
7 +-----+-----+
8 | Silberzwiebel | 30.000000 |
9 | Tulpenzwiebel | 50.000000 |
10 | Spaten       | 1.000000 |
11 | Silberzwiebel | 10.000000 |
12 | Schaufel      | 5.000000 |
13 +-----+-----+

```

Jetzt heißt es herauszufinden, nach welcher Spalte die Gruppen gebildet werden sollen. Da hilft ein Tipp: Wenn die Aufgabenstellung Formulierungen wie *pro Artikel* oder *für jeden Kunden* enthält, dann sind dies in der Regel die Gruppierungsmerkmale. In unserem Beispiel wird demnach nach der Spalte `artikel_id` gruppiert. Die Aggregatfunktion, um die Anzahl der Artikel zu ermitteln, ist hier `SUM()` und nicht `COUNT()`, da pro Zeile mehr als ein Artikel verkauft wird.

```

1 mysql> SELECT
2       -> a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
3       -> FROM
4       -> bestellung_position bp INNER JOIN artikel a USING(artikel_id)
5       -> GROUP BY
6       -> artikel_id
7       -> ;
8 +-----+-----+
9 | Artikelname | Anzahl bestellter Artikel |
10 +-----+-----+
11 | Silberzwiebel | 40.000000 |
12 | Tulpenzwiebel | 50.000000 |
13 | Schaufel      | 5.000000 |
14 | Spaten       | 1.000000 |
15 +-----+-----+

```

Jetzt ist es aber so, dass hier die Artikel, die in keiner Bestellung vorkommen, gar nicht aufgeführt werden. Ein `RIGHT OUTER JOIN` könnte helfen:

```

1 mysql> SELECT
2       -> a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
3       -> FROM
4       -> bestellung_position bp RIGHT JOIN artikel a USING(artikel_id)
5       -> GROUP BY
6       -> artikel_id;
7 +-----+-----+
8 | Artikelname | Anzahl bestellter Artikel |
9 +-----+-----+
10 | Papier (100) | NULL |
11 | Tinte (gold) | NULL |
12 | Tinte (rot)  | NULL |
13 | Tinte (blau) | NULL |
14 | Feder       | NULL |
15 | Silberzwiebel | 40.000000 |
16 | Tulpenzwiebel | 50.000000 |
17 | Schaufel      | 5.000000 |
18 | Spaten       | 1.000000 |
19 +-----+-----+

```

Wie Sie die Ausgabe NULL in eine schöne Ausgabe verwandeln können, erfahren Sie in [Kapitel 15](#). Ach was, ich kann mich nicht beherrschen:

```

1  mysql> SELECT
2      -> a.bezeichnung 'Artikelname',
3      -> CASE
4      ->   WHEN SUM(bp.menge) IS NULL THEN 0
5      ->   ELSE SUM(bp.menge)
6      -> END AS 'Anzahl bestellter Artikel'
7      -> FROM
8      -> bestellung_position bp RIGHT JOIN artikel a USING(artikel_id)
9      -> GROUP BY
10     -> artikel_id
11     -> ;
12
13  +-----+-----+
14  | Artikelname | Anzahl bestellter Artikel |
15  +-----+-----+
16  | Papier (100) | 0 |
17  | Tinte (gold) | 0 |
18  | Tinte (rot) | 0 |
19  | Tinte (blau) | 0 |
20  | Feder | 0 |
21  | Silberzwiebel | 40.000000 |
22  | Tulpenzwiebel | 50.000000 |
23  | Schaufel | 5.000000 |
24  | Spaten | 1.000000 |
25  +-----+-----+

```

Mit dem CASE können Sie eine Fallunterscheidung auf Werte vornehmen und die Ausgabe danach anpassen. Den Rest erzähle ich Ihnen wirklich erst später ;-).



Hinweis: Im SQL:2023-Standard muss das Gruppierungsmerkmal in der Spaltenliste hinter dem SELECT auftauchen (siehe Zeile 10). PostgreSQL und T-SQL setzen diese Forderung um.

```

1  oshop=# SELECT
2  oshop=#   a.bezeichnung Artikelname,
3  oshop=#   CASE
4  oshop=#     WHEN SUM(bp.menge) IS NULL THEN 0
5  oshop=#     ELSE SUM(bp.menge)
6  oshop=#   END AS Anzahl_bestellter_Artikel
7  oshop=# FROM
8  oshop=#   bestellung_position bp RIGHT JOIN artikel a USING(artikel_id)
9  oshop=# GROUP BY
10 oshop=#   a.bezeichnung
11 oshop=# ;

```

Ein schönes Feature ist die Summenbildung bei Aggregatfunktionen mit `WITH ROLLUP`. Dabei wird am Ende der Auswertung eine zusätzliche Zeile eingefügt, welche die summierten Auswertungen enthält. Die Gruppierungsspalte (hier `artikel_id`) bekommt den Wert NULL zugewiesen.

In unserem Beispiel wird in [Zeile 10](#) die Option angehängt. Dadurch werden in der letzten Zeile des Ergebnisses ([Zeile 24](#)) die Summe aller Werte der Spalte `Anzahl bestellter Artikel` ausgegeben.

```

1  mysql> SELECT
2      -> artikel_id,
3      -> CASE
4      ->   WHEN SUM(bp.menge) IS NULL THEN 0
5      ->   ELSE SUM(bp.menge)
6      -> END AS 'Anzahl bestellter Artikel'
7      -> FROM
8      ->   bestellung_position bp RIGHT JOIN artikel a USING(artikel_id)
9      -> GROUP BY
10     ->   artikel_id WITH ROLLUP
11     -> ;
12 +-----+-----+
13 | artikel_id | Anzahl bestellter Artikel |
14 +-----+-----+
15 |      3001 |                0 |
16 |      3005 |                0 |
17 |      3006 |                0 |
18 |      3007 |                0 |
19 |      3010 |                0 |
20 |      7856 |            40.000000 |
21 |      7863 |            50.000000 |
22 |      9010 |            5.000000 |
23 |      9015 |            1.000000 |
24 |      NULL |            96.000000 | ← Hier steht die Summe!
25 +-----+-----+

```

Werden mehr als ein Gruppierungselement benannt, werden Zwischenzeilen mit Zwischensummen ausgegeben.

■ 12.3 Gruppenergebnisse filtern



SQL:2023, MySQL/MariaDB, PostgreSQL, T-SQL

```

SELECT [DISTINCT]
  {|spaltenliste|ausdruck}
FROM
  from_ausdruck
[WHERE bedingung]
[[GROUP BY spaltenliste|ausdruck]
 [HAVING bedingung]]
[ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
[LIMIT [offset ,] anzahl]
[INTO OUTFILE 'dateiname' exportoptionen]
;

```

Wir wollen nur solche Artikel sehen, die mehr als zehn Mal verkauft wurden. Intuitiv wollen wir dazu die WHERE-Klausel verwenden, müssen aber feststellen, dass das nicht geht.

```

1  mysql> SELECT
2      -> a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
3      -> FROM
4      ->   bestellung_position bp INNER JOIN artikel a USING(artikel_id)
5      -> WHERE

```

```

6      -> SUM(bp.menge) > 10
7      -> GROUP BY
8      -> artikel_id
9      -> ;
10 ERROR 1111 (HY000): Invalid use of group function

```

Die [Definition 35](#) sagt aus, dass die Operation auf die Zeilen eingeschränkt wird, für welche die Bedingung der WHERE-Klausel TRUE ergibt. Das Problem ist aber, dass wir gar nicht die Zeilen beschränken wollen, die in die Berechnungen einfließen, sondern die Ergebnisse, die aus den Berechnungen herausfließen. Dazu wird ein HAVING gebraucht.



Definition 48: HAVING

Durch ein *HAVING* werden die Ergebnisse eines GROUP BY verworfen, für welche die Bedingung nicht TRUE ergibt.

```

1  mysql> SELECT
2      -> a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
3      -> FROM
4      -> bestellung_position bp INNER JOIN artikel a USING(artikel_id)
5      -> GROUP BY
6      -> artikel_id
7      -> HAVING
8      -> SUM(bp.menge) > 10
9      -> ;
10 +-----+-----+
11 | Artikelname | Anzahl bestellter Artikel |
12 +-----+-----+
13 | Silberzwiebel | 40.000000 |
14 | Tulpenzwiebel | 50.000000 |
15 +-----+-----+

```

In [Zeile 7](#) wird der HAVING verwendet. Der Bau der Bedingung kann die gleichen Vergleichsoperatoren (siehe [Abschnitt 26.3.1](#)) und Verknüpfungsoperatoren (siehe [Abschnitt 26.3.2](#)) verwenden wie das WHERE.

Um den Unterschied zwischen WHERE und HAVING zu verdeutlichen, wird die obige Auswertung um solche Zeilen eingeschränkt, deren Artikelname mit Silber beginnen.

```

1  mysql> SELECT
2      -> a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
3      -> FROM
4      -> bestellung_position bp INNER JOIN artikel a USING(artikel_id)
5      -> WHERE
6      -> a.bezeichnung LIKE 'Silber%'
7      -> GROUP BY
8      -> artikel_id
9      -> HAVING
10     -> SUM(bp.menge) > 10
11     -> ;
12 +-----+-----+
13 | Artikelname | Anzahl bestellter Artikel |
14 +-----+-----+
15 | Silberzwiebel | 40.000000 |
16 +-----+-----+

```

Die WHERE-Klausel lässt nur Artikel zu, die das Präfix Silber haben. Deshalb steht das WHERE auch vor dem GROUP BY. Erst nach diesem Filter werden die restlichen Daten grup-

piert und ausgewertet. Die Berechnungsergebnisse selbst werden dann noch mit dem `HAVING` weiter gefiltert.

■ 12.4 Noch Fragen?

12.4.1 Kann ich nach Ausdrücken gruppieren?

In der syntaktischen Beschreibung von `GROUP BY` in [Abschnitt 12.3](#) steht *spaltenliste*, aber es können auch Ausdrücke verwendet werden.

Beispiel: Nach [\[Wik23b\]](#) steht die erste Ziffer der Bankleitzahl für das Clearinggebiet. Wir wollen wissen, wie viele Bankleitzahlen pro Clearinggebiet in der Tabelle `bank` sind.

```

1  mysql> SELECT
2      -> SUBSTRING(blz, 1, 1) 'Clearinggebiet', COUNT(*) 'Anzahl'
3      -> FROM
4      -> bank
5      -> GROUP BY
6      -> 'Clearinggebiet'
7      -> ORDER BY 'Anzahl' DESC
8      -> ;
9
10 +-----+-----+
11 | Clearinggebiet | Anzahl |
12 +-----+-----+
13 | 7              | 1249  |
14 | 5              | 1193  |
15 | 6              | 1100  |
16 | 2              | 989   |
17 | 4              | 556   |
18 | 3              | 527   |
19 | 8              | 287   |
20 | 1              | 205   |
21 +-----+-----+
```

Um das Clearinggebiet zu ermitteln, verwende ich in [Zeile 2](#) die Funktion `SUBSTRING()`. Diese schneidet mir aus einer Zeichenkette einen Abschnitt heraus. Der erste Buchstabe der Zeichenkette hat den Index 1. Als letzter Parameter der Funktion wird die Länge des Abschnitts festgelegt, hier ebenfalls 1.

Dadurch, dass ich den Alias `Clearinggebiet` vergeben habe, kann ich überall dort, wo der Ausdruck `SUBSTRING(blz, 1, 1)` gebraucht wird, den Alias verwenden. So auch in [Zeile 6](#) beim `GROUP BY`.



Aufgabe 12.2: Wie viele Banken gibt es pro Bankengruppe (4. Ziffer der Bankleitzahl)?

12.4.2 Kann ich nach mehr als einer Spalte gruppieren?

In der syntaktischen Beschreibung von GROUP BY in [Kapitel 12.3](#) steht hinter dem GROUP BY das Wort *spaltenliste*. Wir können Gruppen also auch über mehrere Elemente definieren. Bisher waren es einfache Spalten wie die `artikel_id`.

Beispiel: Wir wollen die Anzahl der Bestellungen pro Jahr und Monat wissen¹:

```

1  mysql> SELECT
2      -> YEAR(datum) 'Jahr', MONTH(datum) 'Monat', COUNT(*) 'Anzahl'
3      -> FROM bestellung
4      -> GROUP BY
5      -> YEAR(datum), MONTH(datum)
6      -> ORDER BY
7      -> YEAR(datum) DESC, MONTH(datum) DESC
8      -> ;
9  +-----+-----+-----+
10 | Jahr | Monat | Anzahl |
11 +-----+-----+-----+
12 | 2012 |      4 |      1 |
13 | 2012 |      3 |      2 |
14 | 2011 |      1 |      3 |
15 +-----+-----+-----+
```

Der entscheidende Teil ist die [Zeile 5](#). Die Gruppierung wird zuerst nach dem Jahr vorgenommen, anschließend nach dem Monat. Die Funktion `YEAR()` liefert mir aus einer Datumsangabe die vierstellige Darstellung des Jahrs als Zahl; `MONTH()` liefert die ein- oder zweistellige Darstellung des Monats als Zahl. Beide Funktionen verwende ich in [Zeile 7](#), um die Ausgabe nach dem Datum sinnvoll zu ordnen. Eine Übersicht mit vielen guten Beispielen finden Sie unter [\[MyS21a\]](#).



Aufgabe 12.3: Wie viele Banken gibt es pro Clearinggebiet und Bankengruppe (4. Stelle der Bankleitzahl)?

12.4.3 Wie kann ich GROUP BY beschleunigen?

Bei der Ausführung eines GROUP BY muss für jeden Datensatz entschieden werden, zu welcher Gruppe er gehört. Dazu müssen die Spalten und Ausdrücke ausgewertet werden. Wie bei der Sortierung (siehe [Abschnitt 10.3](#)) können Indizes diesen Vorgang erheblich beschleunigen.

Umgekehrt sind GROUP BY-Operationen, die keine Indexunterstützung haben, recht teuer. Besonders, wenn die Gruppierung über Ausdrücke erfolgt, muss der Ausdruck ausgewertet werden, was oft mit erheblicher Rechenleistung verbunden ist, da die Tabelle sequenziell durchlaufen werden muss. Schauen wir uns beispielsweise diesen Befehl an:

```

1  mysql> EXPLAIN
2      -> SELECT
3      -> SUBSTRING(blz, 1, 1) 'Clearinggebiet', COUNT(*) 'Anzahl'
4      -> FROM
```

¹ Ich habe dazu in `listing09.sql` die Bestellungen um Daten erweitert.

```

5      ->  bank
6      ->  GROUP BY
7      ->  'Clearinggebiet'
8      ->  ORDER BY
9      ->  'Anzahl' DESC
10     ->  \G
11  ***** 1. row *****
12      id: 1
13      select_type: SIMPLE
14      table: bank
15      partitions: NULL
16      type: index
17      possible_keys: idx_bank_blzbankname
18      key: idx_bank_blzbankname
19      key_len: 1070
20      ref: NULL
21      rows: 6066
22      filtered: 100.00
23      Extra: Using index; Using temporary; Using filesort

```

Unter der Überschrift **Extra** in [Zeile 23](#) wird angegeben, wie der Befehl ausgeführt wird. Da ich auf die Bankleitzahl einen Index gesetzt hatte, wird dieser genutzt. Nun verwende ich ein Gruppierungsmerkmal, welches zugegeben sinnlos ist, aber keinen Index nutzen kann:

```

1  mysql> EXPLAIN
2      ->  SELECT
3      ->  CONCAT(lkz, SUBSTRING(blz, 6, 3)) 'Sinnlos', COUNT(*) 'Anzahl'
4      ->  FROM
5      ->  bank
6      ->  GROUP BY
7      ->  'Sinnlos'
8      ->  ORDER BY
9      ->  'Sinnlos'
10     ->  \G
11  ***** 1. row *****
12      id: 1
13      select_type: SIMPLE
14      table: bank
15      partitions: NULL
16      type: ALL
17      possible_keys: NULL
18      key: NULL
19      key_len: NULL
20      ref: NULL
21      rows: 6066
22      filtered: 100.00
23      Extra: Using temporary; Using filesort

```

Hier teilt mir MySQL mit, dass es eine temporäre Tabelle aufbauen muss und deren Inhalt mit Filesort sortiert. Keine Rede mehr davon, dass irgendwelche Indizes verwendet werden könnten.

Werden Gruppierungen nicht über Indizes unterstützt und Sie erwarten einen häufigen Zugriff auf nicht kleine Mengen, so sollten – falls möglich – entsprechende Indizes eingerichtet werden. Die Argumente für oder gegen das Anlegen von Indizes sind in [Tabelle 6.1](#) zusammengefasst.

Falls die Gruppierungen nicht permanent gebraucht werden und zum Zeitpunkt der Auswertung nicht oder nur mit wenigen Änderungen der Daten zu rechnen ist, bietet sich das Anlegen einer (temporären) Tabelle an.

12.4.4 Parallele Bearbeitung – unterschiedliche Ergebnisse?

Bei nicht transaktionsfähigen Engines wie MyISAM wird die Anzahl der Zeilen in den Infos über die Tabelle vom System mitgepflegt. Es müssen also nicht alle Zeilen gezählt werden, um die Anzahl zu ermitteln. Bei transaktionsfähigen Engines wie InnoDB können in offenen Transaktionen unterschiedlich viele Zeilen der Tabelle parallel vorhanden sein (Näheres siehe [Kapitel 19](#)).

Die Anzahl der Zeilen muss daher aktiv ermittelt werden und kann in jeder Sitzung unterschiedlich sein, weil jede Sitzung Zeilen hinzugefügt oder gelöscht haben könnte, die in anderen Sitzungen noch nicht sichtbar sind.

■ 12.5 Haben Sie keine Aufgaben für mich?



Aufgabe 12.4: Klar, hab' ich:

- a) Ermitteln Sie pro Bankleitzahl die Anzahl der Banknamen. Sortieren Sie das Ergebnis nach Bankleitzahl.
- b) Ermitteln Sie pro Adresse die Anzahl der Verwendungen als Rechnungsanschrift. Die nicht verwendeten Adressen sollen auch angezeigt werden. Diese haben die Anzahl 0. Sortieren Sie nach der Anzahl absteigend.
- c) Geben Sie pro Warengruppe die Anzahl der Artikel aus. Es sollen auch die Warengruppen angezeigt werden, denen keine Artikel angehören.
- d) Geben Sie pro Lieferant die Anzahl der gelieferten Artikel aus. Lieferanten, die keine Artikel liefern, sollen ebenfalls angezeigt werden.
- e) Geben Sie den Lagerbestand pro Warengruppe aus. Sortieren Sie die Ausgabe nach Lagerbestand absteigend.
- f) Geben Sie die Kundennamen mit mehr als einer Bestellung aus.
- g) Geben Sie für jeden Kundennamen die durchschnittliche Anzahl der Bestellungen aus.
- h) Geben Sie pro Kunde die durchschnittliche Menge an bestellten Artikeln aus. Die Ausgabe soll nach dem Durchschnitt aufsteigend sortiert werden.
- i) Geben Sie den Wert der teuersten Einzelposition aus.
- j) Geben Sie den durchschnittlichen Wert einer Einzelposition aus.
- k) Ermitteln Sie, welche Artikel bestellt sind, deren Mindestmenge im Lager unterschritten ist.

Stichwortverzeichnis

-> 300
->> 300
<< 406
>> 406
|| 415
|, bitweise 406
() 141
* 158, 406
+ 406
- 406
- 161
.NET 6
/ 406
; 65
< 412
<= 413
<=> 412
<> 412
<?php ... ?> 385
= 412
> 413
>= 413
% 406
&& 415
&, bitweise 406
&, bitweise 406
^, bitweise 406
_id 287
! 415
!= 412

A

ABS() 406
Abweisende Schleife 339
ACID 317
ACOS() 406
AFTER 127
Aggregatfunktion 207
Akamai 383

Alias 158, 203
ALL 229, 230
ALL() 234
ALP 78
ALTER DATABASE 121, 417
ALTER EVENT 368, 417
ALTER FUNCTION 417
ALTER INSTANCE 417
ALTER LOGFILE GROUP 418
ALTER PROCEDURE 418
ALTER SCHEMA 121, 417
ALTER SERVER 418
ALTER TABLE 125, 418
- ... ADD 127, 145
- ... ADD FOREIGN KEY 172
- ... ADD INDEX 172
- ... ADD PRIMARY KEY 172
- ... DISABLE KEYS 100
- ... DROP 133
- ... DROP FOREIGN KEY 171
- ... DROP INDEX 172
- ... DROP PRIMARY KEY 172
- ... ENABLE KEYS 100
- ... MODIFY 128
- ... RENAME 126
ALTER TABLESPACE 420
ALTER USER 433
- aktueller Benutzer 433
- Rolle 433
ALTER VIEW 276, 420
AND
- bitweise 406
- Logikoperation 415
Änderungsweitergabe 33, 83
Annehmende Schleife 339
ANSI 61

Ansicht [4](#), [267](#)
 – Projektions- [279](#)
 – Selektions- [276](#)
 – veränderbare [281](#)
 – Verbund- [278](#)
 Antivalenz [416](#)
 ANY() [232](#), [234](#)
 ApacheFriends [51](#)
 API [6](#)
 ARCHIVE [380](#)
 Aria [380](#)
 Artikelverwaltung [42](#)
 ASC [160](#)
 ASIN() [406](#)
 ATAN() [407](#)
 ATAN2() [407](#)
 Atomare Tabelle [35](#)
 Atomarität [317](#)
 Atomicity [317](#)
 Attribut [16](#)
 Aufzählung [74](#)
 Auslöser [357](#)
 AUTO_INCREMENT [72](#), [148](#), [404](#)
 AUTOCOMMIT [315](#), [318](#)
 AVG() [208](#), [409](#)
 AVG(DISTINCT) [409](#)

B

B-Baum [89](#)
 Bank [41](#)
 Bankverbindung [41](#)
 Batchverarbeitung [388](#)
 BCNF [39](#)
 Bedingung [137](#)
 BEGIN ... END [328](#)
 BENCHMARK() [390](#)
 Benutzerauthentifizierungsoption [435](#)
 Benutzerrecht [373](#)
 Benutzerspezifikation [435](#)
 Bestellwesen [43](#)
 BETWEEN [413](#)
 Bewegungsdaten [142](#), [191](#)
 BIGINT [395](#)
 BINARY [139](#), [397](#), [404](#)
 binding [392](#)
 BIT [395](#)
 BIT_AND() [409](#)
 BIT_OR() [409](#)
 BIT_XOR() [409](#)
 Bitshift
 – nach links [406](#)
 – nach rechts [406](#)

BLACKHOLE [380](#)
 Blind Injection [390](#)
 BLOB [112](#), [401](#)
 BOOL [395](#)
 Boolean Attack [390](#)
 Breitenabdeckung [397](#)
 BSI [377](#)
 Bücherei [30](#)
 BULK INSERT [103](#), [106](#)

C

C [6](#)
 C++ [6](#)
 C# [6](#)
 Cache [9](#)
 CALL [429](#)
 CASCADE [66](#), [82](#), [85](#), [135](#)
 CASE [255](#), [336](#)
 – einfach [257](#)
 – searched [259](#)
 Cassandra [380](#)
 CD-Sammlung [31](#)
 CEIL() [407](#)
 CEILING() [407](#)
 CHAR [397](#)
 CHARACTER SET [66](#), [162](#)
 CHECK [404](#)
 CHECK TABLE [274](#)
 Chen, Jolly [10](#)
 Chen-Notation [22](#)
 CLOSE [347](#)
 Clown-Agentur [30](#)
 COBOL
 – Data Division [12](#)
 – Satzkennzeichen [12](#)
 – Stufennummer [12](#)
 Codd, Dr. E. F. [11](#)
 Codepage 850 [66](#)
 COLLATE [68](#), [162](#)
 Collection [287](#)
 COMMIT [315](#)
 Common Table Expression [204](#), [237](#), [265](#), [301](#),
[433](#)
 Compilezeit [154](#)
 CONCAT() [145](#)
 Condition [137](#)
 Conditional Comment [122](#)
 CONNECT [380](#)
 CONNECT_TIMEOUT [330](#)
 Connection-Pool [8](#)
 Connectionstring [385](#)
 Connector [6](#)

- Consistency 317
- CONSTRAINT 81
- Constraint Check 6
- CONV() 407
- COPY 103
- COS() 407
- COT() 407
- COUNT() 208, 410
- COUNT(*) 208, 410
- COUNT(DISTINCT) 208, 410
- cp850 67
- CRC32() 407
- CREATE DATABASE 64, 420
- CREATE EVENT 365, 420
- CREATE FUNCTION 355, 421
- CREATE INDEX 93, 421
- CREATE LOGFILE GROUP 422
- CREATE PROCEDURE 328, 422
- CREATE ROLE 376, 434
- CREATE SCHEMA 64, 422
- CREATE SERVER 423
- CREATE SPATIAL REFERENCE SYSTEM 423
- CREATE TABLE 71, 83, 85, 424
 - ...LIKE 426
 - ...SELECT 426
- CREATE TABLESPACE 426
- CREATE TEMPORARY TABLE 189, 222
- CREATE TRIGGER 358, 427
- CREATE USER 376, 393, 434
- CREATE VIEW 427
- CROSS JOIN 182
- Crowfoot 22
- CSV 380
- CSV-Format 102
- CURSOR 347
- D**
- Data Administration Language 61
- Data Control Language 61
- Data Definition Language 61
- Data Manipulation Language 61
- DATE 398
- DATE() 133
- DATE_FORMAT() 400
- Dateisystem 9
- Daten
 - Bewegungs- 142, 191
 - redundante 205
 - Stamm- 142, 191
- Datenbank 3, 5
 - Abgrenzung 11
 - anlegen 64
 - hierarchische 12
 - löschen 65
 - objektorientierte 13
 - relationale 11
- Datenbankmanagementsystem 5
 - objektorientiertes 6
 - relationales 6
- Datenbanksystem 3, 6
- Datenfeld 16
- Datenflussdiagramm 22
- Datensatz 16
- Datenschutz 6, 279
- Datensicherheit 6
- Datentyp 70
- Datenverzeichnis 53
- DATEIME 398
- Dauerhaftigkeit 317
- DBMS 5
- DCL 61
- DDL 61, 417
- Deadlock 325, 326
- DECIMAL 76, 396, 437
- DECLARE 329
 - ...FOR CURSOR 347
 - CONTINUE 347
 - EXIT 347
- Dedicated Computer 49
- DEFAULT 404
- DEGREES() 407
- Deklarative Programmiersprache 61
- DELETE 146, 429
 - ... IGNORE 149
 - ... LOW_PRIORITY 149
 - ... QUICK 149
- deleted 32
- DELIMITER 329
- DESC 160
- DESCRIBE 74, 109, 127
- Deutscher Brauereiverband 31
- Deutscher Wetterdienst 25
- Developer Server 47
- Development Computer 48
- Differenzmenge 250, 253
- DIN 61
 - 5007-1 68
 - 66001 22
 - 66261 22
- dirty read 319
- DISABLE KEYS 100
- Disjunktion 416
- DISTINCT 170
- DIV 406

DML 61, 429
 DO 429
 Docker 55
 – MariaDB 58
 – MS SQL Server 60
 – MySQL 56
 – PostgreSQL 59
 Domäne 15, 16
 DOUBLE 76, 396, 437
 DOUBLE PRECISION 396
 Drei-Schichten-Architektur 15, 78
 Dritte Normalform 38
 DROP DATABASE 123, 427
 DROP EVENT 368, 427
 DROP FUNCTION 356, 427
 DROP INDEX 100, 427
 DROP LOGFILE GROUP 427
 DROP PROCEDURE 332, 428
 DROP ROLE 434
 DROP SCHEMA 123, 428
 DROP SERVER 428
 DROP SPATIAL REFERENCE SYSTEM 428
 DROP TABLE 134, 428
 DROP TABLESPACE 428
 DROP TRIGGER 358, 428
 DROP USER 374, 377, 434
 DROP VIEW 273, 428
 DSGVO 33
 Dublette 96
 Durability 317

E

Eigenschaft 16
 1:1-Verknüpfung 24
 1:n-Verknüpfung
 – Definition 26
 – identifizierende 27
 ENABLE KEYS 100
 Endlosschleife 340
 Engine 5, 9, 80
 – ARCHIVE 380
 – Aria 380
 – BLACKHOLE 380
 – Cassandra 380
 – CONNECT 380
 – CSV 380
 – EXAMPLE 380
 – FEDERATED 380
 – FederatedX 380
 – InnoDB 380
 – MEMORY 381, 438
 – MERGE 381

 – MyISAM 381
 – OQGRAPH 381
 – XtraDB 381
 Entity 16
 Entity Relationship Modell 22
 Entitytype 16
 Entität 16
 Entitätentyp 16
 – schwacher 18
 – starker 18
 ENUM 74, 395
 Equi Join 193
 ER-Modell 22
 Eratosthenes 342
 Ereignis 4, 365
 Erlang 6
 ERM 22
 Error Based Attack 389
 Erste Normalform 36
 Escapen 391
 Event 365
 EXAMPLE 380
 Excel 6
 EXCEPT 250, 251, 253
 Existenzabhängige Tabelle 18
 Existenzunabhängige Tabelle 18
 EXISTS 237
 EXIT 63
 EXP() 407
 Experiment
 – DOUBLE vs. DECIMAL 437
 – Einfügen mit Index 447
 – Indexselektivität 450
 – NULL vs. NOT NULL 441
 – Rundungsfehler 441
 – Sortierung 452
 – Suchen 444
 EXPLAIN 166
 Exploit 389
 Exportieren
 – Binärdaten
 – C# 177
 – CSV-Daten 176

F

Fallunterscheidung 255, 336
 FEDERATED 380
 FederatedX 380
 Feld 16
 FETCH 347
 FIRST 127
 FLOAT 396

FLOOR() 156, 407
FOR ORDINALITY 301
FOREIGN KEY 80, 404
foreign key 19
FORMAT() 258, 407
Formatierungszeichen 400

- Datum 399
- Uhrzeit 400

Fremdschlüssel 19, 23, 80

- festlegen 80

FULL OUTER JOIN 197
Funktion 355

G

GENERATED

- ...AS IDENTITY 72, 404
- ...AS PRIMARY KEY 404

Generierte Spalten 404
GEOMETRY 403
GEOMETRYCOLLECTION 403
Geschlossene Schleife 339
GET_FORMAT() 400
GLOBAL 87
GRANT 378, 393, 434

- ALL PRIVILEGES 378
- CHARACTER SET 378
- COLLATION 378
- DOMAIN 378
- FUNCTION 378
- PROCEDURE 378
- PROXY 434
- Rolle 434
- TABLE 378
- TRANSLATION 378
- WITH GRANT OPTION 378

Grantoption 436
GROUP BY 210

- ...WITH ROLLUP 213

GROUP_CONCAT() 293, 410

H

Hauptanweisung 223
HAVING 214, 215
Herz 153
HEX() 407
Hierarchische Datenbank 12
Hilfstabelle 29
htdocs 384

I

iconv 67
IFDEF-Notation 22

Identifizierende 1:n-Verknüpfung 27
IF 334
IF EXISTS 66
IF NOT EXISTS 65
IGNORE 107
Imperative Programmiersprache 61
Importieren

- Binärdaten
 - C# 112
 - LOAD FILE 115
- CSV-Daten 103
- XML-Daten 348

IN 328
In Band Injection 389
IN() 227, 234, 413
Index 4, 89, 444, 447, 450, 452

- anlegen 93
- automatisch 90
- Dublette 96
- löschen 100
- Schlüsseleigenschaft 95

Index Scan

- backward 168
- forward 168

Indexselektivität 98, 450
Inferential Injection 390
Informix 9
INHERITS 86
Injection

- in band 389
 - error based 389
 - union based 389
- inferential 390
 - boolean 390
 - time based 390
- out of band 390

INNER JOIN 183
InnoDB 380
InnoDB Cluster 48
INOUT 328
INSERT 374
INSERT INTO

- ...SELECT 116, 430
- ...SET 110, 430
- ...VALUES 109, 430

INT 395
Integrität, referenzielle 32
Interpreter 5
INTERSECT 248, 253
INTO OUTFILE 390
IS FALSE 413
IS NOT NULL 414

IS NULL 413
 IS TRUE 413
 IS UNKNOWN 413
 ISAM 7
 ISO 61
 ISO/IEC
 – 8859-1 66
 – 8859-2 67
 – 8859-9 67
 – 8859-13 67
 – 9075:1999 62
 – 9075:2011 62
 – 5807 22
 – 9075 400
 Isolation 317
 Item 16
 ITERATE 340

J

JavaScript
 – *collection*
 – add() 289
 – arrayDelete() 297
 – arrayInsert() 290
 – find() 290
 – modify() 290
 – remove() 289
 – *object*
 – hasOwnProperty() 291
 – keys 291
 – *parameter*
 – bind() 290
 – *result*
 – fetchAll() 289, 293
 – fetchOne() 290
 – *schema*
 – dropCollection() 288
 – getCollection() 289
 – getCollections() 288
 – *session*
 – close() 288
 – getSchema() 288
 – sql() 290, 293
 – *statement*
 – execute() 290
 – *table*
 – select() 289
 – for...in 289
 – function 288
 – require() 287
 – return 288
 JDBC 6

JOIN
 – ... ON 183
 – ... USING 188
 – CROSS 182
 – Equi 193
 – INNER 183
 – NATURAL 188
 – OUTER 197
 – FULL 197, 200
 – LEFT 197
 – RIGHT 197
 – Self 202
 JSON 285, 401
 JSON-Operator
 – – > 4 300
 – – >> 300
 JSON_ARRAY() 298
 JSON_ARRAY_APPEND() 300
 JSON_ARRAYAGG() 410
 JSON_EXTRACT() 300
 JSON_OBJECT() 298
 JSON_OBJECTAGG() 411
 JSON_PRETTY() 300
 JSON_REMOVE() 306
 JSON_REPLACE() 303
 JSON_SEARCH() 303
 JSON_SET() 306
 Jupyter 6

K

Kalender
 – gregorianisch 398
 – julianisch 398
 – proleptischer gregorianischer 398
 Kardinalität 24
 Kartesisches Produkt 181
 Kaskadierend
 – Löschen 32, 82
 – Änderung 33, 83
 key 17
 Klammern 141
 Klasse 16
 Kommentar
 – bedingter 122
 – einzeliger 161
 Konjunktion 415
 Konnektor 6, 8
 Konsistenz 317
 Konstante 154
 Kopf-/Fußgesteuerte Schleife 339
 Korrelierende Unterabfrage 224
 Kreuzprodukt 182

Krähenfuß-Notation 22

Kunde 41

Kundenverwaltung 41

L

LAST_INSERT_ID() 314, 329

latin1 67

Laufzeit 154

LEAVE 340

LEFT OUTER JOIN 197

LIKE 85, 414

LIMIT 173, 345

LINESTRING 402

Listenunterabfragen 227

LN() 408

LOAD DATA INFILE 103, 105, 430

LOAD FILE 115

LOAD XML 349, 431

LOAD_FILE() 390

LOCAL 87

Lock 310

LOCK TABLES 311

Locking 310

LOG() 408

LOG2() 408

LOG10() 408

Lokale Variable 329

LONGBLOB 401

LONGTEXT 397

LOOP-Schleife 340

Löschkennzeichen 32

Löschweitergabe 32, 82

Lost Update 310

LPAD() 258

Löschkennzeichen 276

M

MariaDB

– Client 63

– Server 51

Martin-Notation 22

Maskieren 391

Matrix 16

MAX() 209, 411

MEDIUMBLOB 401

MEDIUMINT 395

MEDIUMTEXT 397

Mehrwertige Abhängigkeit 24

MEMBER OF 414

MEMORY 381, 438

Mengenverhältnis 24

MERGE 381

MERGED 270

MIN() 209, 411

Minimalität des Schlüssels 17

Minimumexistenz 161

MOD 406

MOD() 408

Modellierung 22

Monolithische Anwendung 78

MONTH() 217

MULTILINESTRING 403

MULTIPOINT 402

MULTIPOLYGON 403

MyISAM 381

MySQL 7

– Client 63

– Connector 48

– Query Browser 63

– Router 48

– Server 48

– Shell 48

– Workbench 23, 48, 63

mysqladmin 55

mysqldump 371

mysql 385

– bind_param() 392

– close() 386

– connect_error 385

– error 386

– execute() 393

– fetch_assoc() 386

– get_result() 393

– multi_query() 388

– num_rows 386

– prepare() 392

– query() 386, 388

– real_escape_string() 391

mysqlsh 283

N

n:m-Verknüpfung 28

n:m:k-Verknüpfung 29

Nassi-Shneidermann-Diagramm 22

NATURAL JOIN 188

Negation 415

NEW 358

Nicht korrelierende Unterfrage 224

NO ACTION 85

Node.js 6

Normalform 33

– Boyce-Codd 39

– erste 36

– zweite 37

- dritte 38
- vierte 39
- fünfte 39
- Normalisierung 34, 36
- NOT 415
- NOT BETWEEN 413
- NOT FOUND 347
- NOT IN 413
- NOT IN() 251
- NOT LIKE 414
- NOT NULL 77, 404, 441
- NOT REGEXP 414
- NOT RLIKE 414
- Notation
 - Chen 22
 - Crowfoot 22
 - IDEFIX 22
 - Krähenfuß 22
 - Martin 22
 - UML 22
- NULL 77, 404, 441
- NUMERIC 396
- Nummernkreis 36, 74

O

- Oberanweisung 223
- Objectcode 392
- Objekt 16
- Objektorientierte Datenbank 13
- Objektyp 436
- ODBC 6
- Offene Schleife 339
- OGC 402
- OLD 358
- Online-Shop 41
 - Tabelle adresse 41
 - Tabelle artikel 42
 - Tabelle artikel_nm_lieferant 87
 - Tabelle artikel_nm_warengruppe 108
 - Tabelle bank 41
 - Tabelle bankverbindung 41
 - Tabelle bestellung 43
 - Tabelle bild 112
 - Tabelle kunde 41
 - Tabelle lagerbestand 210
 - Tabelle lieferant 42
 - Tabelle position_bestellung 43
 - Tabelle position_rechnung 43
 - Tabelle rechnung 43
 - Tabelle warengruppe 42
- OODBMS 6
- OPEN 347

- OpenGIS 402
- Operatorenpriorität 155
- Operatorenrangfolge 155
- Optimierer 5, 9
- OQGRAPH 381
- OR
 - bitweise 406
 - Logikoperation 415
- Oracle 7
- ORDER BY 159
- Ordnung 161
- Orgienjoin 182
- OUT 328
- Out of Band Injection 390
- OUTER JOIN 197
- OWASP 383

P

- Page Lock 310
- Parameterart
 - IN 328
 - INOUT 328
 - OUT 328
- Parameterbindung 289–291, 392
- Parser 9
- PASSWORD() 376
- Passwortoption 436
- Penetrationstest 390
- Performancemessung 437, 444, 447, 452
- Perl 6
- PHP 6
 - die() 394
 - error_log() 394
 - MySQLi 385
 - PDO 385
 - preg_match() 391
 - Tag 385
- PI() 408
- Platzhalter 158
- Plausibilisierung 15
- Plausibilitätsüberprüfung 333
- POINT 402
- POLYGON 403
- PostgreSQL 9
- POSTQUEL 10
- POW() 408
- POWER() 408
- Prepared Statement 392
- PRIMARY KEY 404
- Primärschlüssel 18, 23
- Privileg 373
- Privilegtiefe 436

Programmablaufplan 22
Programmierschnittstelle 6
Programmiersprache
– deklarative 61
– imperative 61
Programmverzeichnis 53
Projektionsansicht 279
Property 16
Prozedur 4
Puh 126
Python 6

R

R 6
RADIANS() 408
RAND() 155, 408
Randbedingungsprüfer 6
RDBMS 6
READ ONLY 121
Read Only 405
REAL 396
real_escape_string() 391
Record 16
Recordset 16
Redundante Daten 205
Redundanz 21, 33
REFERENCES 80
Referenz 21
Referenzielle Integrität 32, 41, 82
Reflection 291
Regenbogentabelle 388
REGEXP 414
Regulärer Ausdruck 391, 414
Relation 11, 16
Relationale Datenbank 11
RENAME TABLE 428
RENAME USER 434
REPLACE 107
Ressourceoption 436
RESTRICT 66, 84, 85, 135
REVOKE 374, 379, 435
– ALL 435
– ALL PRIVILEGES 380
– PROXY 435
– Rolle 435
RIGHT OUTER JOIN 197
RLIKE 414
ROLLBACK 315, 320
ROUND() 144, 408
Row Lock 310
Ruby 6
Rundungsfehler 76, 409, 441

Räumliche Datentypen 402

S

Sakila 8
Schema 16, 64
Schleife 339
– abweisende 339
– annehmende 339
– fußgesteuerte 339
– geschlossen 339
– kopfgesteuerte 339
– offene 339
Schlüssel
– Definition 17
– Fremd- 19
– Kandidat- 18
– Minimalität des 17
– Primär- 18
– Sekundär- 18
Schnittmenge 248, 253
Schwache Tabelle 18
Seitensperre 310
Sekundärschlüssel 18
SELECT 153, 431
– ... INTO 157, 174, 331
– ... INTO OUTFILE 176
Selektionsansicht 276
Self Join 201, 202
Semikolon 65
Seq_in_index 94
Server Computer 49
serverglobal 330
Session Variable 330
SET 331, 395
SET DEFAULT 85
SET GLOBAL 367
SET NAMES 162
SET NULL 85
SET PASSWORD 435
SET TRANSACTION ISOLATION LEVEL
– GLOBAL 319
– READ COMMITTED 320
– READ UNCOMMITTED 319
– REPEATABLE READ 321
– SERIALIZABLE 322
– SESSION 319
SHOW
– CHARACTER SET 67
– COLLATION 68
– CREATE DATABASE 122
– CREATE SCHEMA 122
– CREATE TABLE 81, 109, 148

- DATABASES 69
- EVENTS 366
- FULL TABLES 269
- GRANTS FOR 376
- INDEX 91
- PROCEDURE STATUS 332
- SCHEMAS 69
- TABLES 73
- TRIGGERS 363
- VARIABLES 367
- VARIABLES LIKE 64, 318
- VIEWS (work around) 269
- WARNINGS 65, 100, 104
- Sieb des Eratosthenes 342
- SIGN() 408
- SIN() 409
- Single Responsibility Principle 78
- Sitzung 8
- Sitzungsglobal 330
- Sitzungsverwaltung 5
- Skalarunterabfrage 224, 225
- SLEEP() 390
- SMALLINT 395
- Sortierreihenfolge 68
- SOUNDS LIKE 414
- sp_rename 126
- Spalte
 - * 158
 - Auswahl einer 158
 - Definition 15
 - Spezifikation einer 71
- Spatial Data Types 402
- Sperroption 436
- Sprunglogik 415
- SQL 61
- SQL HANDLER 347
- SQL-Injection 114, 116, 176, 291, 383
- SQL_NO_CACHE 440
- SQL-Schnittstelle 9
- SQLEXCEPTION 347
- SQLi 383
- SQLWARNING 347
- SQRT() 409
- SRP 78
- Stammdaten 142, 191
- Standardwert 403
- Starke Tabelle 18
- START TRANSACTION 315, 318
- STD() 411
- STDDEV() 411
- STDDEV_POP() 411
- STDDEV_SAMP() 411

- Stonebraker, Michael 9
- Storage Engine 5, 9
- strict-Modus 374
- String 395
- Struktogramm 22
- Stundenplan-Software 31
- SUBSELECT 223
- SUBSTRING() 216
- Suchpfad 55
- SUM() 209, 411
- SUM(DISTINCT) 411
- Sun Microsystems 7
- Swift 6

T

- Tabelle 3, 16
 - anlegen 70
 - atomare 35
 - existenzabhängige 18
 - existenzunabhängige 18
 - herleiten 85
 - schwache 18
 - starke 18
 - teilfunktionale 37
 - temporäre 86, 219
 - transitive 38
 - vollfunktionale 37
 - wiederholungsgruppenfreie 35
- Tabellenreferenzen 432
- Tabellensperre 310
- Tabellenunterabfragen 235
- Table Lock 310
- TAN() 409
- Tcl 6
- Teilfunktionale Tabelle 37
- TEMPORARY 86
- Temporäre Tabelle 4, 86, 219
- TEMPTABLE 270
- TEXT 397
- Tiefenabdeckung 397
- TIME 398
- Time Based Attack 390
- TIME() 165
- TIME_FORMAT() 400
- Timeout 5, 64
- TIMESTAMP 398
- TINYBLOB 401
- TINYINT 395
- TINYTEXT 397
- Transact-SQL (T-SQL, TSQL) 62
- Transaktion 317
- Transaktionsmanagement 6

Transitive Tabelle 38
Transitivität 161
Trennzeichen 102
Trichotomie 161
Trigger 4, 357
TRUNCATE 149, 429
TRUNCATE() 409
Tupel 16
Twebaze, Ambrose 8

U

Übergabeparameter 330
Uhrzeit 398
UML-Notation 22
UNDER 86
Unicode 66
UNION 245, 252, 387, 432
Union Based Attack 389
UNIQUE 91, 404
UNKNOWN 137
UNLOCK TABLES 311
UNSIGNED 72, 404
Unterabfrage 223
– korrelierende 224
– Listen- 227
– nicht korrelierende 224
– Skalar- 224, 225
– Tabellen- 235
UPDATE 142, 433
– ... IGNORE 145
– ... LOW_PRIORITY 145
USE 72
USING 188
utf8 66
utf8mb4 67
utf16 66, 67
utf32 66
utf8 67
utf8mb3 67
utf8mb4 67

V

VAR_POP() 411
VAR_SAMP() 412
VARBINARY 397
VARCHAR 72, 397
Variable 157, 174
– global
– server 330
– sitzungs- 330
– lokal 329
VARIANCE() 411

Verbinder 8
Verbindungsparameter 385
Verbundansicht 278
Vereinigung 245, 252
Vererbung 86
Verknüpfung 21
– 1:1, Definition 24
– 1:n
– Definition 26
– identifizierende 27
– n:m, Definition 28
– n:m:k 29
Verletzte referenzielle Integrität 32
Verschlüsselung 436
Verzeichnis
– Daten 53
– Programm 53
– XAMPP 53
Verzweigung 333
Veränderbare Ansicht 281
VIEW 267
Vollfunktionale Tabelle 37
Vorbelegungen 403

W

Wartungsinstabilität 21, 24, 78
Wartungsstabilität 39, 276
Wertebereich 15
WHERE-Klausel 137, 138, 215
WHILE-Schleife 342
Widenius, Michael 7
Wiederholungsgruppe 29, 34
Wiederholungsgruppenfreiheit 35
Windows 11 51
Windows Service 50
WITH 204, 433
WITH RECURSIVE 204
WITH ROLLUP 213

X

XAMPP 51, 384
XAMPP-Verzeichnis 53
XML-Format 13
XOR
– bitweise 406
– Logikoperation 415
XtraDB 381

Y

YEAR 398
YEAR() 217
Yu, Anrew 10

Z

Zeichenketten [110](#)

Zeichensatz [66](#)

– zuweisen [66](#)

Zeile

– Auswahl einer [159](#)

– Definition [16](#)

Zeilensperre [310](#)

Zeilenumbruch [102](#)

Zeitdifferenz [399](#)

Zufallszahlen [155](#)

Zweite Normalform [37](#)

Zwischenspeicher [9](#)