

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

SQL

Ralf ADAMS

Der Grundkurs für
Ausbildung und Praxis

5. Auflage



MIT: SQL Injection abwenden

HANSER

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

SQL

Ralf ADAMS

Der Grundkurs für
Ausbildung und Praxis

5. Auflage



MIT: SQL Injection abwenden

HANSER

HANSER

Ralf Adams

SQL

Der Grundkurs für Ausbildung und Praxis

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

5. Auflage

Ihr Plus – digitale Zusatzinhalte!

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

Geben Sie auf [plus.hanser-
fachbuch.de](https://plus.hanser-fachbuch.de) einfach diesen
Code ein:

plus-6qrL3-2dsm7

Der Autor:

Ralf Adams, Bochum

Kontakt: sqlbuch@ralfadams.de

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

Die endgültige Entscheidung über die Eignung der Informationen für die vorgesehene Verwendung in einer bestimmten Anwendung liegt in der alleinigen Verantwortung des Nutzers.

Aus Gründen der besseren Lesbarkeit wird auf die gleichzeitige Verwendung der Sprachformen männlich, weiblich und divers (m/w/d) verzichtet. Sämtliche Personenbezeichnungen gelten gleichermaßen für alle Geschlechter.

Bibliografische Information der Deutschen Nationalbibliothek:

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

Dieses Werk ist urheberrechtlich geschützt.

Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren), auch nicht für Zwecke der Unterrichtsgestaltung – mit Ausnahme der in den §§ 53, 54 URG genannten Sonderfälle –, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

© 2024 Carl Hanser Verlag München, <http://www.hanser-fachbuch.de>

Lektorat: Sylvia Hasselbach

Copy editing: Sandra Gottmann, Wasserburg

Layout: der Autor mit LaTeX

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

Covergestaltung und Titelmotiv: Tom West, unter Verwendung von Grafiken von

© shutterstock.com/Viktorus

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

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

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

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

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

Inhalt

Titelei

Impressum

Inhalt

Vorwort zur 5. Auflage

Teil I Was man so wissen sollte

1 Datenbanksystem

1.1 Aufgaben und Komponenten

1.1.1 Datenbank

1.1.2 Datenbankmanagementsystem

1.2 Im Buch verwendete Server

1.2.1 MySQL und MariaDB

1.2.2 PostgreSQL

1.2.3 Microsoft SQL Server

2 Relationale Datenbanken

2.1 Einführung

2.1.1 Abgrenzung zu anderen Datenbanken

2.1.2 Tabelle, Zeile und Spalte

2.1.3 Schlüssel, Primärschlüssel und Fremdschlüssel

2.2 Kardinalitäten und ER-Modell

2.2.1 Darstellung von Tabellen im ER-Modell

2.2.2 1:1-Verknüpfung

2.2.2.1 Wann liegt eine 1:1-Verknüpfung vor?

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

2.2.2.3 Kann ich die Kardinalität genauer beschreiben?

2.2.3 1:n-Verknüpfung

2.2.3.1 Wann liegt eine 1:n-Verknüpfung vor?

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

2.2.3.3 Kann ich die Kardinalität genauer beschreiben?

2.2.4 n:m-Verknüpfung

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

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

2.2.4.3 Kann ich die Kardinalität genauer beschreiben?

2.2.5 Aufgaben zum ER-Modell

2.3 Referenzielle Integrität

2.3.1 Verletzung der referenziellen Integrität durch Löschen

2.3.2 Verletzung der referenziellen Integrität durch Änderungen

2.4 Normalformen

2.4.1 Normalform 1

2.4.2 Normalform 2

2.4.3 Normalform 3

2.4.4 Normalform Rest

3 Unser Beispiel: Ein Online-Shop

3.1 Kundenverwaltung

3.2 Artikelverwaltung

3.3 Bestellwesen

Teil II Datenbank aufbauen

4 Installation des Servers

4.1 MySQL unter Windows 11

4.2 MariaDB unter Windows 11

4.3 Andere Installationen mit Docker

4.3.1 MySQL

4.3.2 MariaDB

4.3.3 PostgreSQL

4.3.4 Microsoft SQL Server

5 Datenbank und Tabellen anlegen

5.1 Die Programmiersprache SQL

5.2 Anlegen der Datenbank

5.2.1 Wie rufe ich den MySQL Client auf?

5.2.2 Wie lege ich eine Datenbank an?

5.2.3 Wie lösche ich eine Datenbank?

5.2.4 Wie weise ich einen Zeichensatz zu?

5.2.5 Wie weise ich eine Sortierung zu?

5.3 Anlegen der Tabellen

5.3.1 Welche Datentypen gibt es?

5.3.2 Wie lege ich eine Tabelle an?

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

5.3.4 Wann ein DECIMAL und wann ein DOUBLE?

5.3.5 Wann verwende ich NOT NULL?

5.3.6 Wie lege ich einen Fremdschlüssel fest?

5.3.7 Wie kann ich Tabellen aus anderen herleiten?

5.3.8 Ich brauche mal eben kurz 'ne Tabelle!

6 Indizes anlegen

6.1 Index für Anfänger

6.1.1 Wann wird ein Index automatisch erstellt?

6.1.2 Wie kann ich einen Index manuell erstellen?

6.2 Und jetzt etwas genauer

6.2.1 Wie kann ich die Schlüsseleigenschaft erzwingen?

6.2.2 Wie kann ich Dubletten verhindern?

6.2.3 Was bedeutet Indexselektivität?

6.2.4 Wie kann ich einen Index löschen?

7 Werte in Tabellen einfügen

7.1 Daten importieren

7.1.1 Das CSV-Format

7.1.2 CSV-Daten laden mit LOAD DATA INFILE

7.1.3 Was ist, wenn ich geänderte Werte importieren will?

7.2 Daten anlegen

7.2.1 Wie lege ich mehrere Zeilen mit einem Befehl an?

7.2.2 Wie kann ich eine einzelne Zeile anlegen?

7.2.3 Vorsicht Constraints!

7.2.4 Einfügen von binären Daten über einen C#-Client

7.2.5 Einfügen von binären Daten LOAD FILE

7.3 Daten kopieren

Teil III Datenbank ändern

8 Datenbank und Tabellen umbauen

8.1 Eine Datenbank ändern

8.2 Eine Datenbank löschen

8.3 Eine Tabelle ändern

8.3.1 Wie kann ich den Namen der Tabelle ändern?

8.3.2 Wie kann ich eine Spalte hinzufügen?

8.3.3 Wie kann ich die Spezifikation einer Spalte ändern?

8.3.4 Zeichenbasierte Spalten in der Länge verändern

8.3.5 Zeichensatz verändern

8.3.6 Zeichenbasierte Spalten in numerische Spalten verändern

8.3.7 Numerische Spalten im Wertebereich verändern

8.3.8 Datum- oder Zeitspalten verändern

8.3.9 Wie kann ich aus einer Tabelle Spalten entfernen?

8.4 Eine Tabelle löschen

8.4.1 Einfach löschen

8.4.2 Was bedeuten CASCADE und RESTRICT?

9 Werte in Tabellen verändern

9.1 WHERE-Klausel

9.1.1 Wie formuliere ich eine einfache Bedingung?

9.1.2 Wird zwischen Groß- und Kleinschreibung unterschieden?

9.1.3 Wie formuliere ich eine zusammengesetzte Bedingung?

9.2 Tabelleninhalte verändern

9.2.1 Szenario 1: Einfache Wertzuweisung

9.2.2 Szenario 2: Berechnete Werte

9.2.3 Szenario 3: Gebastelte Zeichenketten

9.2.4 Was bedeuten LOW_PRIORITY und IGNORE?

9.3 Tabelleninhalte löschen

9.3.1 Und was passiert bei Constraints?

9.3.2 Was passiert mit dem AUTO_INCREMENT?

9.3.3 Was bedeuten LOW_PRIORITY, QUICK und IGNORE?

9.3.4 Wie kann ich eine Tabelle komplett leeren?

Teil IV Datenbank auswerten

10 Einfache Auswertungen

10.1 Ausdrücke

10.1.1 Konstanten

10.1.2 Wie kann ich Berechnungen vornehmen?

10.1.3 Wie ermittle ich Zufallszahlen?

10.1.4 Wie stecke ich das Berechnungsergebnis in eine Variable?

10.2 Zeilen- und Spaltenwahl

10.3 Sortierung

10.3.1 Was muss ich bei der Sortierung von Texten beachten?

10.3.2 Wird zwischen Groß- und Kleinschreibung unterschieden?

10.3.3 Wie werden Datums- und Uhrzeitwerte sortiert?

10.3.4 Wie kann ich das Sortieren beschleunigen?

10.4 Mehrfachausgaben unterbinden

10.4.1 Fallstudie: Datenimport von Bankdaten

10.4.2 Was muss ich beim DISTINCT bzgl. der

Performance beachten?

10.5 Ergebnismenge ausschneiden

10.5.1 Wie kann ich die ersten n Datensätze ausschneiden?

10.5.2 Wie kann ich Teilmengen mittendrin ausschneiden?

10.6 Ergebnisse exportieren

10.6.1 Wie lege ich eine Exportdatei auf dem Server an?

10.6.2 Wie lege ich eine Exportdatei auf dem Client an?

10.6.3 Wie lese ich mithilfe eines C#-Client binäre Daten aus?

11 Tabellen verbinden

11.1 Heiße Liebe: Primär-Fremdschlüsselpaare

11.2 INNER JOIN zwischen zwei Tabellen

11.2.1 Bauanleitung für einen INNER JOIN

11.2.2 Abkürzende Schreibweisen

11.2.3 Als Datenquelle für temporäre Tabellen

11.2.4 JOIN über Nichtschlüsselspalten

11.3 INNER JOIN über mehr als zwei Tabellen

11.4 Es muss nicht immer heiße Liebe sein: OUTER JOIN

11.5 Narzissmus pur: Self Join

11.6 Eine Verknüpfung beschleunigen

12 Differenzierte Auswertungen

12.1 Statistisches mit Aggregatfunktionen

12.2 Tabelle in Gruppen zerlegen

12.3 Gruppenergebnisse filtern

12.4 Noch Fragen?

12.4.1 Kann ich nach Ausdrücken gruppieren?

12.4.2 Kann ich nach mehr als einer Spalte gruppieren?

12.4.3 Wie kann ich GROUP BY beschleunigen?

12.4.4 Parallele Bearbeitung – unterschiedliche
Ergebnisse?

12.5 Aufgaben

13 Auswertungen mit Unterabfragen

13.1 Das Problem und die Lösung

13.2 Nicht korrelierende Unterabfrage

13.2.1 Skalarunterabfrage

13.2.1.1 Beispiel 1: Banken mit höchster BLZ

13.2.1.2 Beispiel 2: Überdurchschnittlich teure Artikel

13.2.1.3 Beispiel 3: Überdurchschnittlich wertvolle Bestellungen

13.2.2 Listenunterabfrage

13.2.2.1 Beispiel 1: IN()

13.2.2.2 Beispiel 2: ALL()

13.2.2.3 Beispiel 3: ALL()

13.2.2.4 Beispiel 4: ANY()

13.2.2.5 Unterschied zwischen IN(), ALL() und ANY()

13.2.2.6 Unterschied zwischen NOT IN() und <> ALL()

13.2.3 Tabellenunterabfrage

13.3 Korrelierende Unterabfrage

13.3.1 Beispiel 1: Rechnungen mit vielen Positionen

13.3.2 Beispiel 2: EXISTS

13.4 Common Table Expression versus Unterabfrage

13.5 Fallstudie Datenimport

13.6 Wie ticken Unterabfragen intern?

13.7 Aufgaben

14 Mengenoperationen

14.1 Die Vereinigung mit UNION

14.2 Die Schnittmenge

14.2.1 Mit INTERSECT

14.2.2 Mit Unterabfragen

14.3 Die Differenzmenge

14.3.1 Mit EXCEPT

14.3.2 Mit Unterabfragen

14.4 UNION, INTERSECT und EXCEPT ... versteh' ich nicht!

15 Bedingungslogik

15.1 Warum ein CASE?

15.2 Einfacher CASE

15.3 Searched CASE

15.4 Fallbeispiele

15.4.1 Lagerbestand überprüfen

15.4.2 Kundengruppen ermitteln

15.4.3 Aktive Lieferanten ermitteln

15.4.4 Aufgaben

16 Ansichtssache

16.1 Was ist eine Ansicht?

16.1.1 Wie lege ich eine Ansicht an?

16.1.2 Wie wird eine Ansicht verarbeitet?

16.1.3 Wie lösche ich eine Ansicht?

16.1.4 Wie ändere ich eine Ansicht?

16.2 Anwendungsgebiet: Vereinfachung

16.3 Anwendungsgebiet: Datenschutz

16.4 Grenzen einer Ansicht

17 Exkurs NoSQL

17.1 Vorbereitung der MySQL-Shell

17.2 Datenmodellierung des Warenkorbs

17.2.1 JavaScript Object Notation (JSON)

17.2.2 Struktur unseres JSON-Dokuments

17.3 NoSQL:MySQLmit JavaScript-Client

17.3.1 Anlegen eines Warenkorbs

17.3.2 Inhalte des Warenkorbs anlegen

17.3.3 Inhalte des Warenkorbs auswerten

17.3.4 Inhalte des Warenkorbs verändern

17.4 NoSQL: klassisches SQL mit JSON-Funktionen

17.4.1 Anlegen eines Warenkorbs

17.4.2 Inhalte des Warenkorbs anlegen

17.4.3 Inhalte des Warenkorbs auswerten

17.4.4 Inhalte des Warenkorbs verändern

17.4.5 Inhalte des Warenkorbs löschen

Teil V Anweisungen kapseln

18 Locking

19 Transaktionen

19.1 Das Problem

19.2 Was ist eine Transaktion?

19.3 Isolationsebenen

19.3.1 READ UNCOMMITTED

19.3.2 READ COMMITTED

19.3.3 REPEATABLE READ

19.3.4 SERIALIZABLE

19.4 Fallbeispiel in C#

19.5 Deadlock

20 Prozeduren

20.1 Einstieg und Variablen

20.2 Verzweigung

20.2.1 Einfache Verzweigung mit IF

20.2.2 Mehrfache Verzweigung mit CASE

20.3 Schleifen

20.3.1 LOOP-Schleife

20.3.2 WHILE-Schleife

20.3.3 REPEAT-Schleife

20.4 Transaktion innerhalb einer Prozedur

20.5 Cursor

20.6 Aufgaben

21 Funktionen

22 Auslöser

22.1 Was ist das?

22.2 Ein Beispiel für einen INSERT-Trigger

22.3 Ein Beispiel für einen UPDATE-Trigger

22.4 Ein Beispiel für einen DELETE-Trigger

23 Ereignisse

23.1 Wie lege ich ein Ereignis an?

23.2 Wie werde ich Ereignisse wieder los?

Teil VI Anhänge

24 Datenbank administrieren

24.1 Daten sichern und wiederherstellen

24.1.1 Backup

24.1.2 Restore

24.2 Benutzerrechte

24.2.1 Benutzerrechte und Privilegien

24.2.2 Benutzer anlegen und Rechte zuweisen

24.2.2.1 CREATE USER bzw. CREATE ROLE

24.2.2.2 Rechte vergeben mit GRANT

24.2.2.3 Rechte entziehen mit REVOKE

24.3 MySQL und MariaDB Engines

25 SQL-Injection

25.1 Das Problem

25.2 Beispiel: Suchmaske

25.3 Gegenmaßnahmen

25.3.1 Never Trust an Unknown Input

25.3.2 Trennung von Anweisungen und Daten

25.3.2.1 Verwenden Sie Prozeduren oder Funktionen

25.3.2.2 Verwenden Sie Prepared Statements

25.3.3 Ich darf nur, was ich soll

25.3.4 Nichtssagende Fehlermeldungen

26 SQL-Referenz

26.1 Datentypen

26.1.1 Numerische Datentypen

26.1.1.1 Ganze Zahlen

26.1.1.2 Gebrochene Zahlen

26.1.2 Zeichen-Datentypen

26.1.3 Datums- und Zeit-Datentypen

26.1.4 Binäre Datentypen

26.1.5 JSON

26.1.6 Räumliche Datentypen

26.1.7 Standardwerte

26.1.8 Zusätze für Datentypen

26.2 Operatoren und Funktionen

26.2.1 Mathematische Operatoren

26.2.2 Mathematische Funktionen

26.2.3 Aggregatfunktionen

26.3 Bedingungen

26.3.1 Vergleichsoperatoren

26.3.2 Logikoperatoren

26.3.2.1 NOT, Negation, $\neg$

26.3.2.2 AND, Konjunktion, $\wedge$

26.3.2.3 OR, Disjunktion, $\vee$

26.3.2.4 XOR, Antivalenz, $\otimes$

26.3.2.5 Verallgemeinerung auf mehr als zwei Operanden

26.4 Befehle

26.4.1 Data Definition Language

26.4.2 DataManipulation Language

26.4.3 Benutzerverwaltung

27 Ausgewählte Quelltexte

27.1 DOUBLE versus DECIMAL

27.2 Rundungsfehler

27.3 NULL versus NOT NULL

27.4 Suchen mit und ohne Index

27.5 Messen der Performance der Einfügeoperation

27.6 Messen der Indexselektivität

27.7 Sortieren ohne und mit Index

28 Quelltexte

28.1 MySQL/MariaDB

28.1.1 Quelltexte zu Teil II

28.1.2 Quelltexte zu Teil III

28.1.3 Quelltexte zu Teil IV

28.1.4 Quelltexte zu Teil V

28.1.5 Quelltexte zu Teil VI

28.2 PostgreSQL

28.2.1 Quelltexte zu Teil II

28.2.2 Quelltexte zu Teil III

28.2.3 Quelltexte zu Teil IV

28.2.4 Quelltexte zu Teil V

28.3 Microsoft SQL Server

28.3.1 Quelltexte zu Teil II

28.3.2 Quelltexte zu Teil III

28.3.3 Quelltexte zu Teil IV

Literatur

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 Beispielen 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.
- Ich habe angefangen, für die Aufgaben Musterlösungen bei YouTube (<http://www.youtube.com/channel/UCu4ZybNXw1y4Rs4Mg.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

`plus-6qrL3-2dsm7`

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 Weihart 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!*

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.

TEIL I

Was man so wissen sollte

1 Datenbanksystem

1.1 Aufgaben und Komponenten



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

- Grundkurs
 - Datenbank
 - Datenbankmanagementsystem
 - Datenbanksystem

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

In [Bild 1.1](#) ist der Aufbau eines Datenbanksystems schematisch dargestellt. Die Datenbanken enthalten die eigentlichen Daten und unmittelbar damit verknüpfte Datenobjekte wie z.B. eine Ansicht (siehe [Kapitel 16](#)). Über eine

Kommunikationsschnittstelle werden diese Datenobjekte vom DBMS verwaltet. Das DBMS selbst besteht wiederum aus vielen kleinen Komponenten, die jeweils auf eine Aufgabe spezialisiert sind.

1.1.1 Datenbank

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

Üblicherweise werden in einer Datenbank folgende Datenobjekte vorkommen¹:

- *Tabellen:* Bei einer relationalen Datenbank werden die Daten in Tabellen organisiert (Kundentabelle, Artikeltabelle, Filmtabelle usw.). Deshalb sind die Tabellen das Herzstück einer Datenbank. Alle anderen Datenbankobjekte sind aus diesen Tabellen abgeleitet oder verwenden diese.

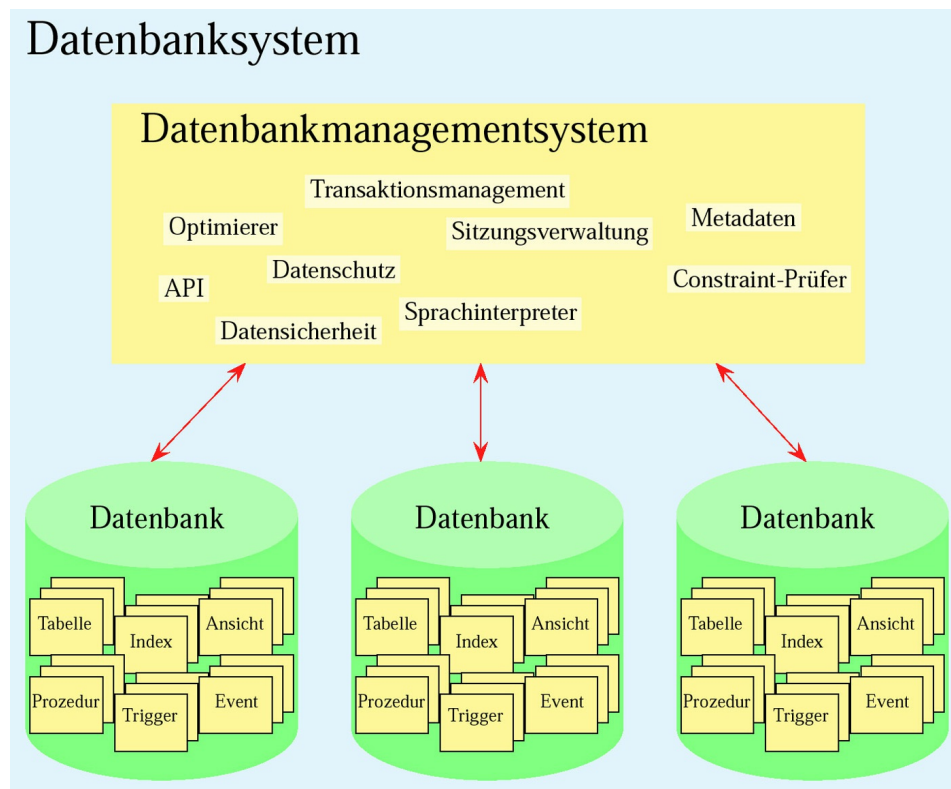


Bild 1.1 Aufbau eines Datenbanksystems

- *Temporäre Tabellen:* Sie werden explizit vom Programmierer oder implizit vom Optimierer angelegt, um Zwischenergebnisse wiederverwendbar zu machen. In der Regel werden diese automatisch nach Beendigung einer Sitzung gelöscht.
- *Indizes:* Diese erlauben eine erhebliche Beschleunigung bestimmter Auswertungen. Die Daten aus den Tabellen werden dabei in frei wählbaren, aber festgelegten Reihenfolgen sortiert.
- *Ansichten:* Auf Vorrat gebastelte Auswertungen, die wie Tabellen verwendet werden können, ohne dabei einen eigenen Datenbestand aufzubauen.
- *Prozeduren:* Kleine, selbst geschriebene SQL-Programme, die auf dem Server ausgeführt werden.

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



Definition 1: Datenbank (eng)

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

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



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

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

1.1.2 Datenbankmanagementsystem

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



Definition 2: Datenbankmanagementsystem (DBMS)

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

Dies sind die Aufgaben eines DBMS²:

- *Sprachinterpreter*: Herzstück des DBMS ist der Interpreter³. Dieser übersetzt die Befehle in einen ausführbaren Code. Die Sprache, die wir hier im Buch verwenden werden, ist SQL. Es gibt und gab aber auch andere Datenbankabfragesprachen wie zum Beispiel dBase, VB für MS-Access, OO-SQL, Sequel usw.
- *Optimierer*: Die Ausführung eines SQL-Befehls kann oft auf verschiedene Art und Weise passieren. Der Optimierer versucht, anhand von Schätzungen und Algorithmen einen Plan für die Ausführung anzulegen, der möglichst schnell abgearbeitet werden kann.
- *Sitzungsverwaltung*: Wann immer ein Befehl an den Server gesendet werden soll, muss man sich in einer Sitzung (engl. *session*) befinden. Dazu muss zuerst eine Sitzung geöffnet werden. Jetzt können beliebig viele SQL-Befehle gesendet und Daten empfangen werden. Zum Schluss wird die Sitzung serverseitig – z.B. durch einen Timeout – oder clientseitig beendet.

- *Randbedingungsprüfer*: Für Tabellen können Randbedingungen (engl. *constraints*) formuliert werden, die immer gelten müssen. Würde die Ausführung eines Befehls dazu führen, dass diese Randbedingungen nicht erfüllt sind, wird die Ausführung des Befehls in der Regel verweigert.
- *Datenschutz*: Durch die Vergabe von Zugriffsrechten kann das Recht auf lesende und schreibende Zugriffe so wie auf das Ausführen von Prozeduren passgenau zugeschnitten werden.
- *Datensicherheit*: Der Verlust von Daten ist der GAU⁴ schlechthin. Das DBMS muss sicherstellen, dass nicht durch Serverabsturz oder Ähnliches Daten verloren gehen.
- *Transaktionsmanagement*: Transaktionen ermöglichen parallelen Zugriff und eine Art *undo* im Fehlerfall. Das zu gewährleisten, erfordert eine Menge Mühe. Die Qualität des Transaktionsmanagements ist oft ein entscheidendes Merkmal eines DMBS.
- *API*⁵: Die Daten werden in der Regel durch eine oder mehrere Anwendungen (Clients) bearbeitet. Damit die Anwendung auf das DBMS zugreifen kann, braucht es eine Schnittstelle, über die es zu den Daten gelangt. Diese APIs werden in der MySQL-/MariaDB-Welt *Konnektoren* genannt. Der MySQL oder MariaDB Server bietet beispielsweise APIs für C, C++, C#/.NET, Node.js PHP, Perl, Python, Ruby und Tcl. Auch stehen APIs für JDBS⁶ und ODBC⁷ zur Verfügung. MariaDB kennt zusätzlich noch Schnittstellen zu Erlang, Jupyter, Excel, Swift und R.
- *Metadaten*: Verwaltungsinformationen, Statistiken etc., eben der ganze Rest.

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

Sind die Datenbanken eines Datenbankmanagementsystems in Form von Tabellen organisiert, so handelt es sich um ein *relationales Datenbankmanagementsystem (RDBMS)*; bei objektorientierten Datenbanken spricht man analog von *objektorientierten Datenbankmanagementsystemen (OODBMS)*.

Und noch der Vollständigkeit halber:



Definition 3: Datenbanksystem

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

1.2 Im Buch verwendete Server



Kurzporträts der verwendeten SQL Server: Hersteller und Geschichte

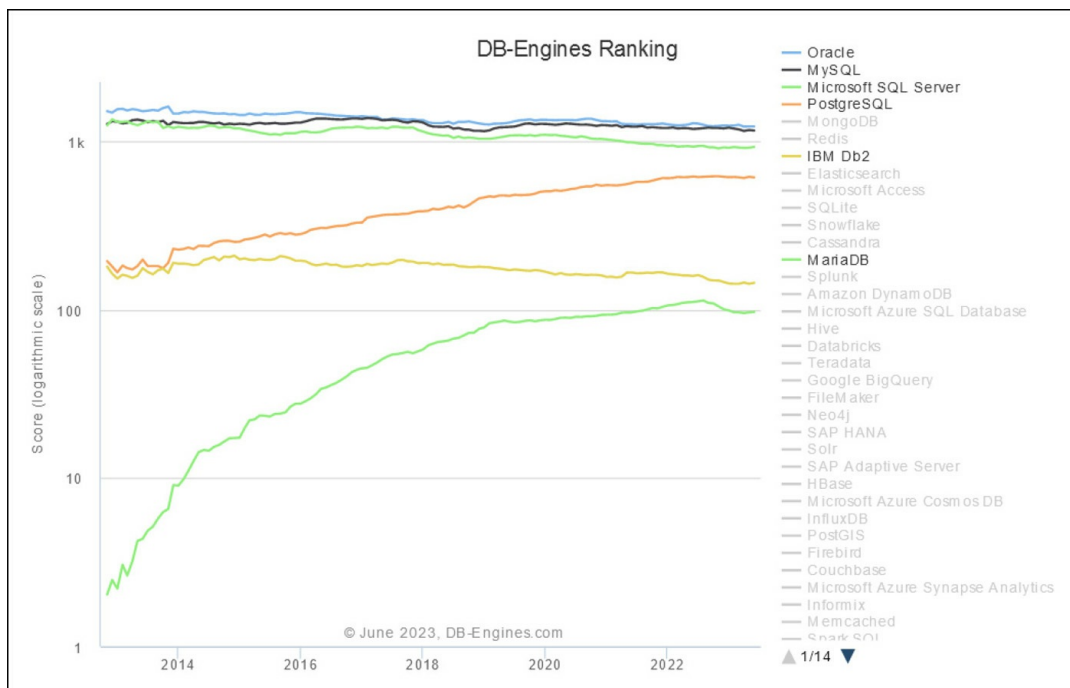


Bild 1.2 Ranking einiger RDBMSen, Quelle [DE23]

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

Wollen Sie auf die Installation von Servern verzichten, können Sie auch online diverse Server verwenden, um SQL-Code auszuprobieren. Natürlich gibt es dabei Einschränkungen. Ich empfehle: <http://sqlfiddle.com/> und <https://dbfiddle.uk/>.

1.2.1 MySQL und MariaDB

Die schwedische Firma MySQL AB hat MySQL von 1994 bis 2008

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

Der Name *MySQL* kommt nicht vom englischen *my* (mein). Einer der Firmengründer, Michael Widenius, hat sympathischerweise den Vornamen *My* seiner Tochter verwendet. Der Name des Delphins im Logo ist Sakila. Er wurde in einem Wettbewerb ermittelt, den der Open Source-Entwickler Ambrose Twebaze aus Uganda gewann. Sakila ist ein Mädchenname in der Sprache *siSwati* und auch der Name einer Stadt in Tansania.

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

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

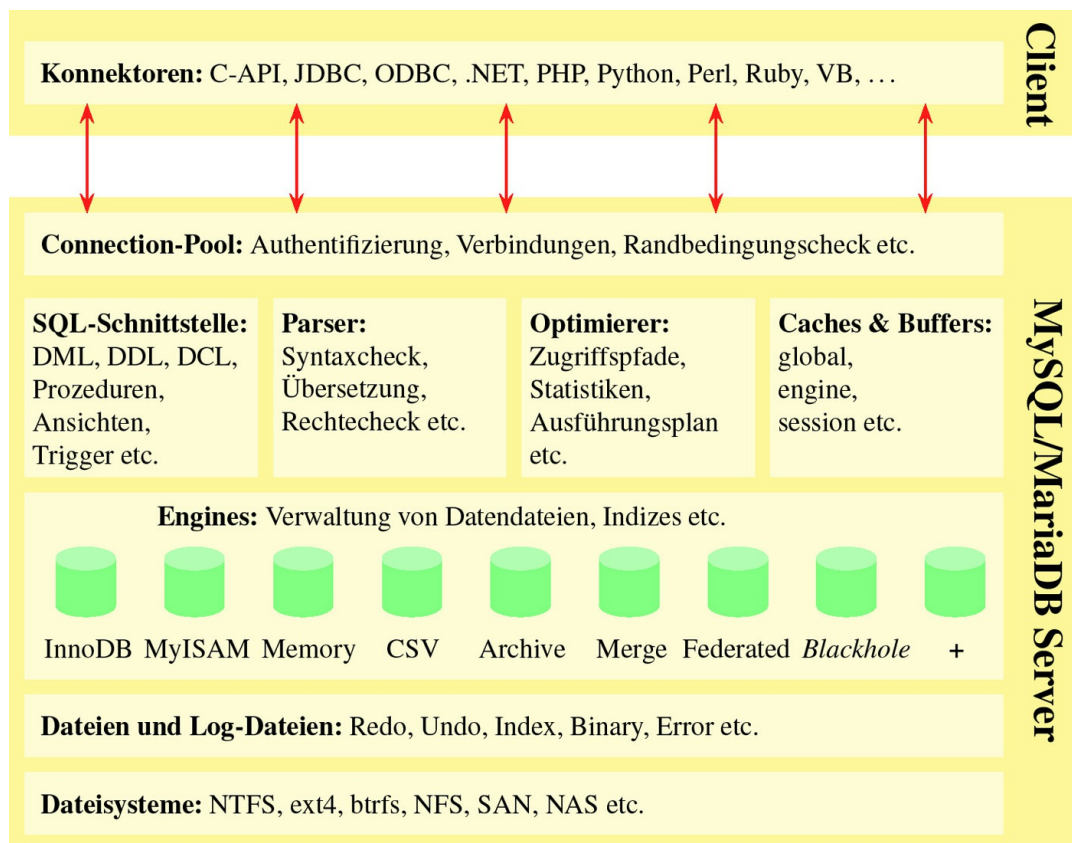


Bild 1.3 Komponenten von MySQL und MariaDB

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

entgegengenommen. Sie werden dann zum Parser weitergereicht.

- *Parser*: Der Parser überprüft die Syntax eines Befehls und ob man die Ausführungsrechte für diesen Befehl hat.
- *Optimizer* (Optimierer): Anhand von Schätzungen, Statistiken und Algorithmen wird für nicht triviale Befehle ein Ausführungsplan erstellt. Dieser Ausführungsplan berücksichtigt ggf. im Cache vorhandene Ergebnisse.
- *Caches und Buffers* (Zwischenspeicher): Daten und Ergebnisse können in Zwischenspeichern aufgehoben werden.⁸
- *Storage Engines*: Die Motoren des Servers. Hier werden die Daten tatsächlich verarbeitet. Jede Engine ist dabei für bestimmte Aufgabenstellungen besonders gut geeignet. Der große Vorteil von MySQL und MariaDB ist, dass jeder mit einem besonderen Bedarf eine Engine bauen kann. Er muss *nur* die Schnittstellen beachten (siehe [[MyS21e](#)]) und kann dann seine Speziallösungen anbieten.
- *File System* (Dateisystem): Je nach Betriebssystem werden hier Daten in unterschiedlichen Dateisystemen (NTFS, BTRFS, EXT4 etc.) abgelegt. Bis auf die Frage, ob das Dateisystem zwischen Groß- und Kleinschreibung unterscheidet, spielt dieses für die SQL-Programmierung keine Rolle.
- *Management Services & Utilities*: Parallel dazu gibt es die vielen kleinen Helferlein, ohne die nichts geht: für das Sichern und Wiederherstellen von Daten, Datenreplikation, Administration und Konfiguration, Datenmigration und Metadaten.

Zur Geschichte von MySQL gibt es einen netten Absatz in Wikipedia (<https://de.wikipedia.org/wiki/MySQL#Geschichte>)

und Dokumentation und Downloads finden Sie auf <http://www.mysql.com>. Die Homepage von MariaDB ist unter <https://mariadb.org> zu finden. Eine kurze, aber gute Übersicht bezüglich der aktuellen Unterschiede zwischen MariaDB und MySQL, finden Sie unter [[Ahm22](#)].

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

1.2.2 PostgreSQL

Geboren wurde PostgreSQL 1986 als Projekt an der Universität von Kalifornien, Berkeley. Professor Michael Stonebraker⁹ begann das Projekt als Nachfolgeprojekt der Ingres-Datenbank, welche ebenfalls noch heute verwendet wird. Daher leitet sich auch der Name her: *post ingres*. In den nächsten acht Jahren wurde Postgres von Prof. Stonebraker und seinen Studenten immer weiter entwickelt. Bis 1995 verstand Postgres allerdings kein SQL, sondern nur eine eigene Sprache namens POSTQUEL. Die beiden Doktoranden Andrew Yu und Jolly Chen haben Postgres um SQL erweitert und als Postgres95 veröffentlicht.

1996 wurde Postgres95 als Open-Source-Projekt der Netzgemeinde zur Verfügung gestellt und wird seitdem sehr stark von Unterstützern weiterentwickelt und gefördert. Ebenso wurde das Erscheinungsjahr aus dem Produktnamen entfernt und die Datenbank heißt seitdem PostgreSQL.

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

nachlesen. Von <http://www.postgresql.org/download/> können Sie die aktuelle PostgreSQL-Version für Linux-Distributionen und Windows herunterladen. Eine sehr ausführliche Online-Dokumentation steht unter <http://www.postgresql.org/docs/> zur Verfügung.

1.2.3 Microsoft SQL Server

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

In Kooperation mit Sybase brachte Microsoft 1989 die erste Version seines Servers auf den Markt. Zielplattform war das Betriebssystem OS/2. Die Kooperation sah so aus, dass beide gemeinsam am Sybase Server arbeiteten und Sybase das Produkt unter seinem Namen auf UNIX-basierenden Systemen anbot und Microsoft auf OS/2. Spätestens seit 1992 speist sich der Quelltext beider Server-Produkte (Sybase 4.0 und MS SQL Server 4.2) gleichwertig aus beiden Firmen. Ab 1993 ist nicht mehr OS/2, sondern Windows NT die Zielplattform des MS SQL Servers. Mit der Portierung auf Windows NT wurden erhebliche Anpassungen notwendig, die nicht mehr für das Sybase-Produkt verwendet werden konnten; die beiden Systeme begannen sich voneinander zu entfernen. Besonders der Wunsch von Sybase, dass der Quelltext möglichst plattformneutral bleiben sollte, widersprach den strategischen Zielen Microsofts, die eine volle Unterstützung der damals wirklich bedeutsamen Neuerungen von Windows NT anstrebten.

Microsoft entschied, dass der SQL Server ein zentrales Produkt für die Windows NT Strategie werden sollte. Daher wurden von

anderen Datenbankherstellern erfahrene Entwickler angeworben. Dieses geballte Know-how mündete 1998 in die Serverversion 7.0 mit dem Arbeitstitel *Sphinx*. Mit dieser völlig überarbeiteten Version wurde auch die Zusammenarbeit mit Sybase überflüssig und beide Produkte sind seitdem voneinander unabhängig. Es kamen Sprachelemente hinzu, die Sicherheit wurde verbessert, die Performance gesteigert, die Stabilität erhöht, andere Betriebssysteme werden unterstützt usw. Besonders das grafische Frontend und die hohe Integration in die Visual Studio Entwicklungsumgebung werden von vielen Entwicklern geschätzt.

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

Der MS SQL Server kann in verschiedenen Varianten von <https://www.microsoft.com/de-de/sql-server/sql-server-downloads> bezogen werden; die Dokumentation liegt in <https://docs.microsoft.com/de-de/sql/?view=sql-server-ver15>.

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

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

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

⁴ Abkürzung für: Größter anzunehmender Unfall

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

⁶ Abkürzung für: Java Database Connectivity: Programmierschnittstelle für JAVA

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

⁸ MySQL hat ab Version 8.0 den Cache abgeschafft ([[MyS18c](#)]); MariaDB verwendet ihn weiterhin ([[Mar23b](#)]).

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

2 Relationale Datenbanken

2.1 Einführung



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

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

- Starke und schwache Entitätentypen

2.1.1 Abgrenzung zu anderen Datenbanken

Als *Erfinder* relationaler Datenbanken gilt Dr. E. F. Codd. Dieser hatte in im Paper [[Cod70](#)] den Begriff eingeführt und die Vorteile seiner Lösung beschrieben.

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

Relation ist ein wohldefinierter Fachbegriff und kann umgangssprachlich als *Tabelle* übersetzt werden (siehe [Tabelle 2.1](#) in [Abschnitt 2.1](#)). Relationale Datenbanken sind also Datenbanken, die in Tabellen organisiert sind. Soll heißen: Die Daten werden in Spalten und Zeilen strukturiert. Das hört sich so selbstverständlich an, dass man sich klar machen muss, dass es auch Alternativen gibt.

Als Beispiel sei hier die Ablage von Bestelldaten eines Kunden in einem Format, wie es beispielsweise in einer COBOL Data Division¹ verwendet wird, angegeben:

```
01 12345 Gamdschie      Samweis  Beutelhaldenwe
g 5   67676  Hobbingen   Privatkunde
05 00001 20120512 12:30:00 bezahlt
10 7856 Silberzwiebel  15   6.15 €
10 7863 Tulpenzwiebel  10  32.90 €
10 9015 Spaten         1  19.90 €
05 00002 20120530 17:15:00 offen
```

10	9010	Schaufel	1	15.00	\$
10	9015	Spaten	1	19.90	€
01	12346	Beutlin	Frodo	Beutelhaldenwe	
g 1	67676	Hobbingen	Privatkunde		
05	00001	20111110	11:15:00	bezahlt	
10	3001	Papier (100)	10	23.00	€
10	3005	Tinte (gold)	1	55.70	€
10	3006	Tinte (rot)	1	6.20	€
10	3010	Feder	5	25.00	€



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

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

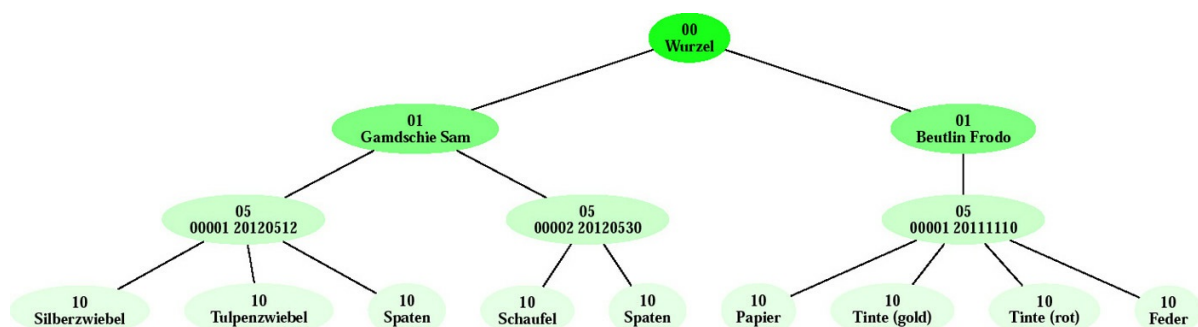


Bild 2.1 Hierarchische Darstellung der Bestelldaten

Die ersten beiden Ziffern sind die sogenannte *Satzkennzeichen*, welches dem verarbeitenden Programm mitteilt, was für eine

Art von Information in der Zeile steht. Irgendwo im System müssen diese Satzzeichen z.B. in Copybooks (siehe [\[Wik23l\]](#)) spezifiziert sein.

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

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

```
1  <?xml version="1.0" encoding="iso-8859-15"
?>
2  <Rechnung>
3    <Kunde>
4      <Kundennummer>12345</Kundennummer>
5      <Nachname>Gamdschie</Nachname>
6      <Vorname>Sam</Vorname>
7      <Strasse>Beutelhaldenweg</Strasse>
8      <Hausnummer>5</Hausnummer>
9      <Postleitzahl>67676</Postleitzahl>
10     <Ort>Hobbingen</Ort>
11     <Kundenart>Privatkunde</Kundenart>
12     <Bestellung>
13       <Bestellnummer>00001</Bestellnummer>
14       <Bestelldatum>20120512 12:30:00</Bes
telldatum>
15       <Status>bezahlt</Status>
16       <Position>
17         <Artikelnummer>7856</Artikelnummer
>
18         <Artikelname>Silberzwiebel</Artike
lname>
19         <Menge>15</Menge>
20         <Preis>6.13</Preis>
21         <Währung>EUR</Währung>
```

```
22         </Position>
23         <Position>
24             <Artikelnummer>7863</Artikelnummer>
25             <Artikelname>Tulpenzwiebel</Artikelname>
26             <Menge>10</Menge>
27             <Preis>32.90</Preis>
28             <Währung>EUR</Währung>
29         </Position>
30     [...]
31 </Rechnung>
```

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

Erinnern Sie sich noch an die Definition einer Datenbank (siehe [Definition 1](#) in [Abschnitt 1.1.1](#))? Dort wird lediglich darauf abgestellt, dass die Daten eine Struktur haben. Haben Daten eine effiziente Struktur, können diese leicht von Programmen ausgelesen und editiert werden. Der Begriff *Datenbank* ist somit unabhängig von der Art, wie die Daten strukturiert sind.

Eine andere *moderne* Art der Datenhaltung sind objektorientierte Datenbanken. Diese verwalten alle Daten, die zu einem Objekt gehören. So richtig durchgesetzt haben sich diese Datenbanken aber noch nicht, auch wenn Produkte wie MongoDB mittlerweile weit verbreitet sind (siehe [\[DE23\]](#)).

2.1.2 Tabelle, Zeile und Spalte

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

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

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

Bild 2.2 Kunde in Tabellenform

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

Bild 2.3 Bestellung in Tabellenform

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

Bild 2.5 Artikel in Tabellenform

Sie erkennen sicherlich einen entscheidenden Vorteil der tabellarischen Darstellung: Die Daten sind inhaltlich konsistent angeordnet. Sowohl Mensch als auch Maschine haben eine eindeutige Zuordnung des Informationstyps zur Tabelle. Bei anderen Darstellungen ist diese Eindeutigkeit nicht gegeben. Beispielsweise in einer XML-Datei werden alle Informationen,

die zu einem Geschäftsprozess gehören, zusammengefasst; auch das kann vorteilhaft sein.

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

Bild 2.4 Bestellposition in Tabellenform

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

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

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



Definition 4: Spalte

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

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

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

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

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

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



Definition 5: Wertebereich

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

Den Wertebereich festzulegen, ist eine Aufgabe, die nicht zwingend dem Datenbankdesign zugeordnet werden muss. Meist werden die Wertebereiche bei der Formulierung des Pflichtenhefts ([[Nor09](#)]) mit der Spezifikation der Eingabemasken festgelegt. Dies ist deshalb sinnvoll, weil bei der Dateneingabe in der Regel die fachliche und formale Plausibilisierung der Daten erfolgt (siehe Drei-Schichten-Architektur auf [Bild 5.2](#) in [Abschnitt 5.3.5](#)).



Definition 6: Zeile

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

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



Definition 7: Tabelle

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

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

Tabelle 2.1 Namensübersicht

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

Ich will nicht in Abrede stellen, dass so tolle Namen wie *Tupel* oder *Entitätentyp* irgendwo notwendig sind, aber zum Verständnis und zur Programmierung von Datenbanken sind einfache intuitive Namen völlig ausreichend. Bei Wikipedia ([\[Wik23r\]](#)) können weitere Begriffe nachgelesen werden.

2.1.3 Schlüssel, Primärschlüssel und Fremdschlüssel

Zwar sind die Daten nun auf Tabellen verteilt (siehe [Bild 2.2](#) in [Abschnitt 2.1.2](#)), aber der Zusammenhang zwischen den Informationen ist verloren gegangen!

Die Bestellungen haben die Information verloren, zu welchem Kunden sie gehören. Die Positionen haben die Information verloren, zu welcher Bestellung sie gehören. Lediglich die Artikeldaten in der Tabelle `position` sind erhalten geblieben, kommen aber doppelt vor.

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

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



Definition 8: Schlüssel

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

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

Was bedeutet das? Nehmen wir die Tabelle `position` in [Bild 2.4](#) in [Abschnitt 2.1.2](#). Betrachten Sie jede Zeile in der Tabelle.

Es fällt auf, dass die Position mit dem Spaten zweimal vorkommt. Warum auch nicht? Beide Positionen wurden in unterschiedlichen Bestellungen aufgegeben. Somit machen die Spalten `artikel_id`, `bezeichnung`, `menge`, `preis` und `währung` die Zeile nicht eindeutig, d.h. unterscheidbar von allen anderen. Wir brauchen eine Spalte, die unabhängig von den anderen Spalteninhalten jede Zeile einmalig, also eindeutig macht.

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

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

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

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

Nun haben wir einen Schlüssel im Sinne der [Definition 8](#) aus [Abschnitt 2.1.2](#):

- Ein *Schlüssel* ist mindestens eine Spalte: `bestellung_id` + `positionsnr`.
- Jede Zeile ist konzeptionell eindeutig: Da die Bestellnummer in `bestellung_id` pro Bestellung und die Positionsnummer in `positionsnr` innerhalb der Bestellung eindeutig sind, ist die Kombination von beiden innerhalb der Tabelle `position` eindeutig.

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

Bild 2.7 Tabelle `position`, 2. Stufe des Umbaus

- Er ist minimal: Nimmt man die Bestellnummer weg, ist die Zeile nicht mehr konzeptionell eindeutig. Nimmt man die Positionsnummer weg, ist die Zeile nicht mehr konzeptionell eindeutig. Da der Schlüssel aus keinen weiteren Bestandteilen zusammengesetzt ist, ist er insgesamt minimal.

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



Definition 9: Starke Tabelle

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

Definition 10: Schwache Tabelle

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

Was auch immer man mit *stark* oder *schwach* ausdrücken wollte: Starke Tabellen sind häufig solche, deren Existenz nicht von anderen abhängig ist. Schwache Tabellen sind hingegen oft von der anderen Tabelle existenzabhängig. An unserem Beispiel wird dies leicht deutlich: Ohne eine Bestellung gibt es keine Bestellpositionen. Löscht man eine Bestellung, werden zwangsläufig auch die damit verbundenen Positionen gelöscht².



Definition 11: Primärschlüssel

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

Die [Definition 11](#) kommt ein bisschen seltsam daher, ist aber für die Praxis sehr wichtig. Schlüssel werden bei relationalen Datenbanken sehr oft und intensiv verwendet, wie wir noch sehen werden. Deshalb muss festgelegt werden, welcher der

Schlüssel zur Weiterverarbeitung als Primärschlüssel verwendet wird und welcher nicht.

Beispiel: Nehmen wir eine Tabelle, in der PKW-Daten für eine Autovermietung abgespeichert sind. Neben vielen anderen Spalten wird es sicherlich eine Spalte kennzeichen³ geben. Schnell haben Sie überprüft, dass diese Spalte ein Schlüssel ist. Jetzt gibt es aber auch die Spalte fahrgestellnummer⁴. Auch diese ist für sich alleine schon ein Schlüssel der Tabelle. Beide Spalten würden jede Zeile der Tabelle unabhängig voneinander konzeptionell eindeutig machen. Es stellt sich nun die Frage, welcher der beiden Schlüssel für die Programmierung verwendet werden soll, welcher also meine Nummer 1 ist.

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



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

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

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

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

Bild 2.8 Tabelle bestellung



Definition 12: Fremdschlüssel

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

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

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

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

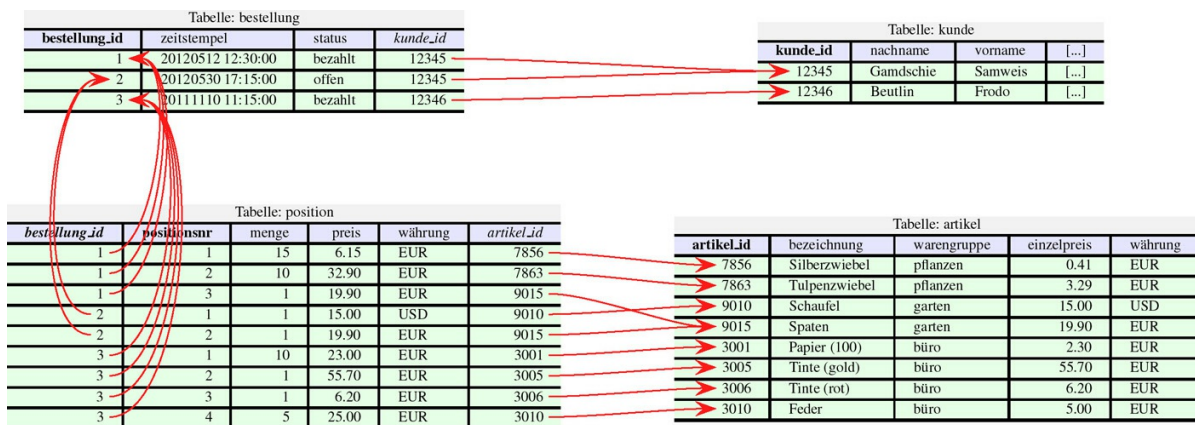


Bild 2.9 Bestellung: Primär-Fremdschlüsselbeziehungen



Definition 13: Redundanz

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

Die Nachteile der Redundanz sind offensichtlich: Zumeinen verschwendet man Speicherplatz, und zum anderen müssen Änderungen – hier beispielsweise an der Artikelbezeichnung – mehrfach vorgenommen werden (*Wartungsinstabilität*). Auch können unter bestimmten Umständen Performanceverluste auftreten, da mehr Datenmenge verarbeitet werden muss. Wir werden später aber noch sehen, dass Redundanzen auch gewollt und sinnvoll sein können⁵.



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

Wie Primär- und Fremdschlüssel zusammen Informationen in

Tabellen zu einem System verknüpfen, können Sie in [Bild 2.9](#) sehen.

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



Definition 14: Verknüpfung

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

2.2 Kardinalitäten und ER-Modell



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

- Grundkurs
 - Darstellung einer Tabelle
 - *1:1*-Verknüpfung, *1:n*-Verknüpfung und *n:m*-Verknüpfung
 - Umsetzung der Kardinalitäten in den Tabellen
- Vertiefendes
 - Verschiedene Notationen
 - Varianten der *1:1*-Verknüpfung, *1:n*-Verknüpfung und *n:m*-Verknüpfung
 - Identifizierende *1:n*-Verknüpfung

- Selbstreferenzen bei verschiedenen Kardinalitäten

Die Zeilen der verknüpften Tabellen stehen in einem gewissen Mengenverhältnis zueinander. Im Prinzip stehen davon drei Arten zur Verfügung, von denen es aber Varianten gibt. Bevor wir uns diese genauer anschauen, möchte ich eine grafische Notation vorstellen, die uns bei der Darstellung der Tabellen helfen soll.

2.2.1 Darstellung von Tabellen im ER-Modell

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



Definition 15: Entity Relationship Model

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

Die Tabellen unseres Online-Shops sollen in einer einfachen und übersichtlichen Form dargestellt werden. Ein kurzer Blick in [\[Wik23i\]](#) liefert folgende Notationen (Auszug):

- **Chen (modifiziert):** Hier werden die Tabellen als Rechtecke dargestellt. Die Spalten der Tabellen werden als Blasen um die Tabelle herum notiert. In die Blase schreibt man den Spaltennamen. Der Name des Primärschlüssels wird dabei unterstrichen. Genaueres unter [[Wik23c](#)] und [[Che76](#)].
- **IDEF1X:** Die Tabellen werden auch hier als Rechtecke (ggf. mit abgerundeten Ecken) dargestellt. Die Spalten werden aber innerhalb des Rechtecks notiert. Die Primärschlüsselspalten werden dabei durch eine Linie von den anderen Spalten abgetrennt. Diese Notation ist der Quasistandard US-amerikanischer Behörden. Genaueres unter [[Wik23k](#)].
- **UML-Klassendiagramm:** Die Tabellen werden dabei wie Klassen in der UML-Notation dargestellt. Die Verknüpfung ist dabei vom Typ *Assoziation*. Genaueres unter [[Oes06](#)] oder [[JRH+04](#)].
- **Krähenfuß (Martin):** Die Tabellen werden in Rechtecken dargestellt. Diese enthalten die Spaltennamen. Vor dem Spaltennamen ist Platz für eine weitere Spezifikation der Spalte (z.B. als Fremdschlüssel). Auch dazu gibt es einen Wikipedia-Eintrag: [[Wik23p](#)].

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

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

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

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

In [Bild 2.10](#) wird beispielhaft die Tabelle `bestellung` dargestellt. Die erste Zeile des Rechtecks enthält den Tabellennamen. Dieser wird – wie auch die Spaltennamen – per Konvention immer klein und im Singular geschrieben. Alle nachfolgenden Zeilen sind in zwei Spalten aufgeteilt. Die linke Spalte kann dazu verwendet werden, eine Spalte bzgl. ihrer Besonderheiten auszuweisen. Meist durch zwei Abkürzungen: PK für den Primärschlüssel (*primary key*) und FK für den Fremdschlüssel (*foreign key*). Daneben stehen die Spaltennamen.

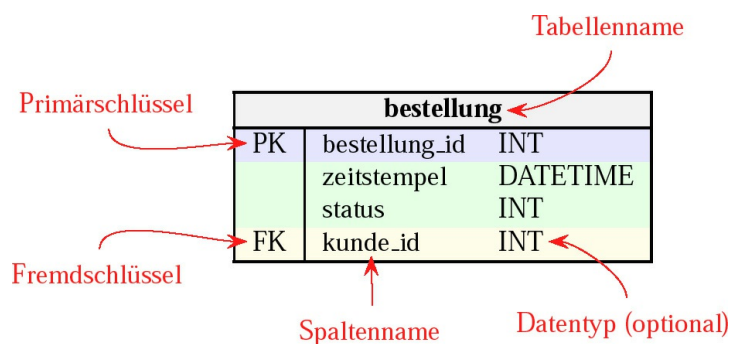


Bild 2.10 Darstellung einer Tabelle in der Krähenfuß-Notation

Wird die Darstellung in CASE-Tools⁶ wie MySQL Workbench erstellt, kann man oft den Datentyp als dritte Spalte oder durch ein Trennzeichen hinter dem Namen angeben. In Anlehnung an das UML-Klassendiagramm geschieht dies immer häufiger in der Notation *name: datentyp*.



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

2.2.2 1:1-Verknüpfung

2.2.2.1 Wann liegt eine 1:1-Verknüpfung vor?

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

Im Prinzip stehen zwei Möglichkeiten zur Verfügung:

1. Die Tabelle kunde wird um die Spalten produktqualität, service und internetauftritt erweitert.
2. Es wird eine zusätzliche Tabelle umfrage eingeführt. Um sicherzustellen, dass die Antworten – also die Zeilen der Tabelle umfrage – dem richtigen Kunden zugewiesen werden, haben die Antworten die Kundennummer kunde_id als Primärschlüssel.

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

Für Variante 2 spricht, dass diese Lösung langfristig *wartungsstabiler* ist. Die Tabelle kunde ist sicherlich eine der

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



Definition 16: Kardinalität

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

Die [Definition 16](#) schließt $A = B$ nicht aus, da es vorkommen kann, dass eine Tabelle mit sich selbst verknüpft ist. Denken Sie beispielsweise an eine Personentabelle, die für jede Person auch den Ehepartner dokumentiert⁷.



Definition 17: 1:1-Verknüpfung

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

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

Um sicherzustellen, dass ein Kunde nur einen Antwortsatz abgeben kann, müssen wir in die Datenbank einbauen, dass das Mengenverhältnis zwischen den Kunden und den Antworten immer passt. Haben zwei Tabellen den gleichen Primärschlüssel, muss eine *1:1*-Verknüpfung vorliegen. In unserem Beispiel würde man den Primärschlüssel der Tabelle kunde auch zum Primärschlüssel der Tabelle umfrage machen. Es gibt zwar auch andere Wege, eine *1:1*-Verknüpfung herzustellen, aber dieser Weg ist der einfachste⁸.



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

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

Beispiel: Wenn Sie Wetterdaten einer Wetterstation z.B. vom Deutschen Wetterdienst bekommen, finden Sie pro Wetterstation in einer Zeile *zig* Angaben. Diese kann man inhaltlich aufteilen und in *1:1*-verknüpfte Tabellen auslagern. Eine Tabelle enthält alle Daten zur Luft (Temperatur,

Windgeschwindigkeit und -richtung usw.) und eine andere die Daten zum Niederschlag (Menge, Art usw.). Hintergrund dieser Aufteilung ist, dass man für eine Auswertung meist nur die inhaltlich zusammenhängenden Informationen benötigt und die anderen Daten die Auswertung – selbst wenn man sie aktiv ausblendet – nur belasten würden.

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

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

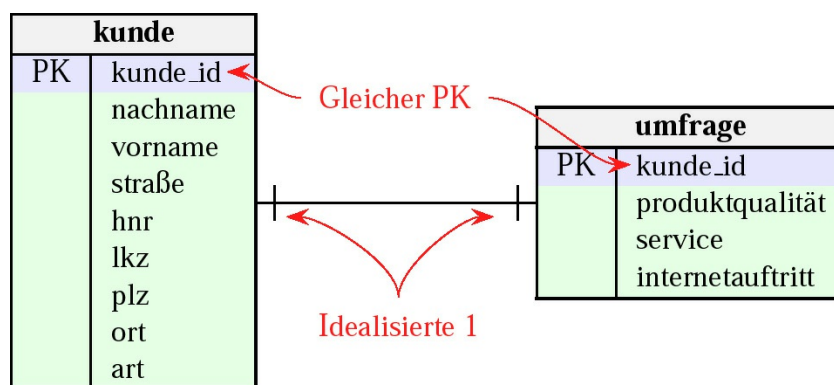


Bild 2.11 Einfache 1:1-Darstellung in Krähfuß-Notation

2.2.2.3 Kann ich die Kardinalität genauer beschreiben?

Diese Information ist zwar richtig, aber wir können es noch besser: Ein Kunde muss sich nicht an der Umfrage beteiligt

haben, aber eine Umfrage muss zu einem Kunden gehören. Dazu kann man die Notation wie in [Bild 2.12](#) erweitern.

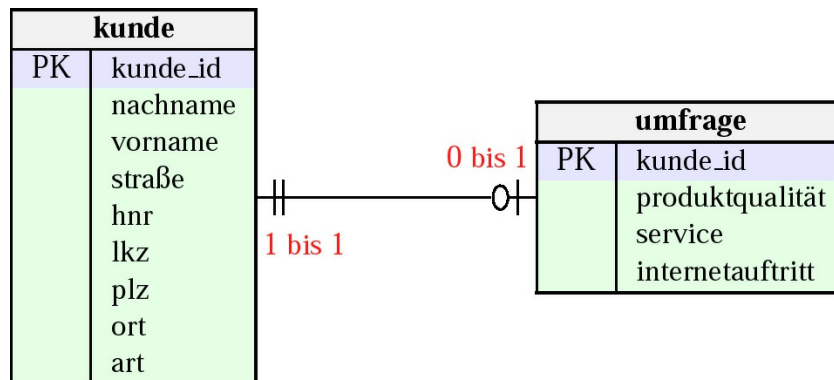


Bild 2.12 Erweiterte 1:1-Darstellung in Krähenfuß-Notation

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

Wann verwende ich die einfache und wann die erweiterte Notation?

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

Fall ist die verfeinerte Notation als Ergebnisdokumentation sinnvoll.

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

2.2.3 1:n-Verknüpfung

2.2.3.1 Wann liegt eine 1:n-Verknüpfung vor?

Betrachten wir eine Zeile in der Tabelle `bestellung` in [Bild 2.9](#) in [Abschnitt 2.1.3](#), so ist diese mit einem Kunden verknüpft; eine Bestellung kann nur von einem Kunden aufgegeben werden. Eine Zeile aus der Tabelle `kunde` hingegen kann zu mehreren Zeilen in `bestellung` gehören, da ein Kunde mehrere Bestellungen aufgeben kann. Somit haben wir eine 1:n-Verknüpfung mit dem n auf der Seite von `bestellung`.



Definition 18: 1:n-Verknüpfung

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



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

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

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



Aufgabe 2.7: Stellen Sie fest, ob die Verknüpfungen in [Bild 2.9](#) in [Abschnitt 2.1.3](#) die Kardinalität $1:n$ haben. Notieren Sie sich, wo das n steht.

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



Definition 19: Identifizierende $1:n$ -Verknüpfung

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

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

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

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

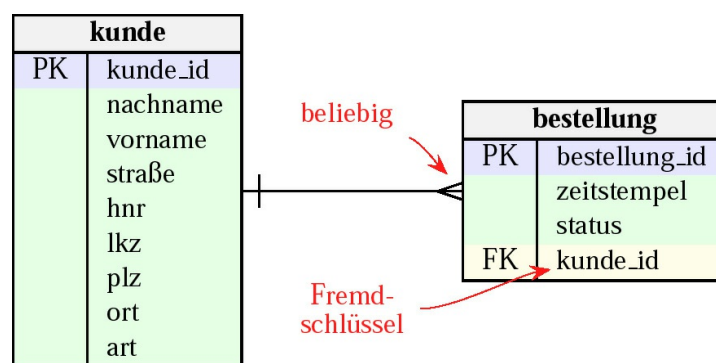


Bild 2.13 Einfache Darstellung der 1:n-Verknüpfung in Krähfuß-Notation

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

2.2.3.3 Kann ich die Kardinalität genauer beschreiben?

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

Würde man verlangen, dass ein Kunde immer mindestens auch eine Bestellung aufgegeben haben muss⁹, würde anstelle der idealisierten 0 wieder ein senkrechter Strich stehen.

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

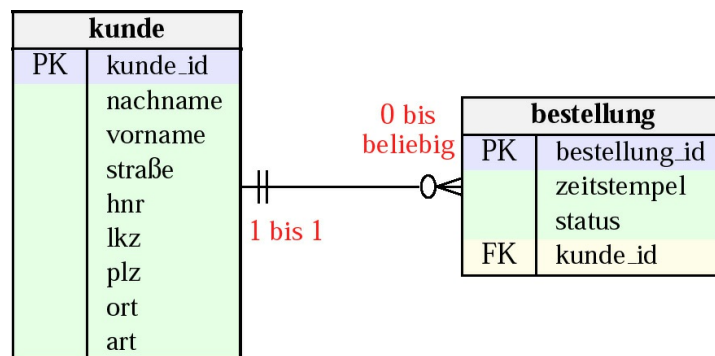


Bild 2.14 Erweiterte Darstellung der 1:n-Verknüpfung in Krähenfuß-Notation

2.2.4 $n:m$ -Verknüpfung

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

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



Definition 20: $n:m$ -Verknüpfung

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



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

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

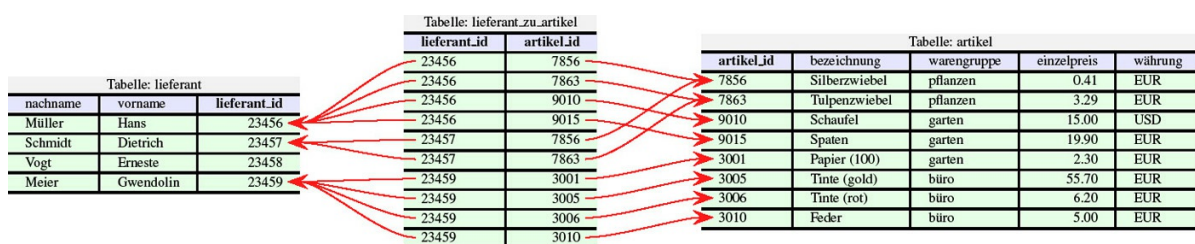


Bild 2.15 Aufbau einer $n:m$ -Beziehung

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

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

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



Definition 21: Hilfstabelle

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

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

Man könnte die $n:m$ -Verknüpfung auch dadurch erreichen, indem in den Spalten nicht nur *einzelne* Werte abgespeichert werden können. Besonders die Darstellung von Listen (Wiederholungsgruppen) bietet sich hier an (siehe [Definition 23](#)). In der Tabelle `artikel` gäbe es eine Spalte

`lieferant_id`, und diese würde alle Primärschlüsselwerte der entsprechenden Lieferanten enthalten. Analoges würde in der Tabelle `lieferant` passieren. Die damit verbundenen Nachteile sind aber gravierend.

Nimmt beispielsweise ein Lieferant einen Artikel aus seinem Angebot, somüsste man komplizierte Dinge tun: Die Liste in `lieferant_id` in `artikel` müsste in seine einzelnen Werte aufgeteilt werden. Anschließend muss dieser Lieferant aus der Liste gelöscht und die Liste wieder zusammenmontiert zurückgeschrieben werden. Analoges müsste mit der Liste `artikel_id` in `lieferant` geschehen¹⁰.

Bei unserer Lösung mit der Hilfstabelle muss man nur die entsprechende Zeile mit den passenden `lieferant_id` und `artikel_id` löschen. Eine solche Operation ist hinsichtlich der Performance ungleich billiger¹¹.



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

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

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

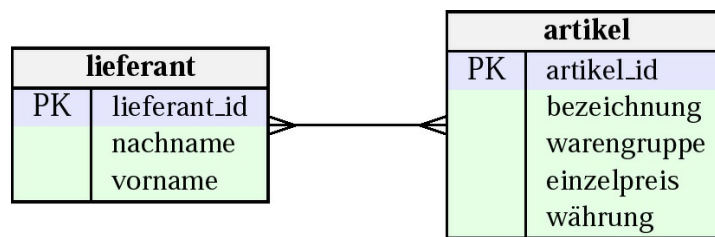


Bild 2.16 Einfache Darstellung der $n:m$ -Verknüpfung in Krähenfuß-Notation

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



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

2.2.4.3 Kann ich die Kardinalität genauer beschreiben?

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

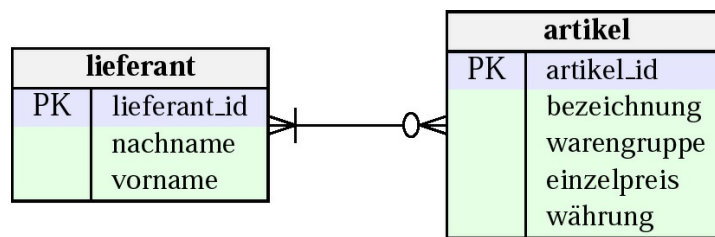


Bild 2.17 Erweiterte Darstellung der $n:m$ -Verknüpfung in Krähenfuß-Notation

2.2.5 Aufgaben zum ER-Modell



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

Aufgabe 2.11: Eine Clown-Agentur beauftragt Sie mit der Erstellung einer Clown-Datenbank. Diese soll Informationen über Clowns, ihre Verträge etc. enthalten. Erstellen Sie ein ER-Modell für die Tabellen *clown*, *vertrag*, *veranstalter* und *programm*. In *clown* sind die Basisdaten des Künstlers erfasst, *veranstalter* analog. In *vertrag* werden Ort und Termin eines Auftritts festgelegt. Jeder Clown bietet mehrere Programme an. Bei einigen Programmen arbeiten mehrere Clowns zusammen. Es kann im Vertrag festgelegt werden, welche der angebotenen Programme bei der Veranstaltung vorgeführt werden sollen!

Aufgabe 2.12: Der Deutsche Brauereiverband will online ein Bierarchiv zu Werbezwecken zur Verfügung stellen. Dieses Bierarchiv soll über alle deutschen Biere

Auskunft geben. Es sollen zu einem Bier alle interessanten Informationen abgelegt werden können: Name, Emblem (auch historische), Brauerei, Biergattung, Bierart, Biersorte, Brauart, Reinheitsgebot, Alkoholgehalt und bei Mischgetränken die anderen Getränke.

Zu den Brauereien werden die üblichen Stammdaten erfasst. Zu den Brauereien sollen auch Bilder und Embleme erfasst werden können. Ebenso wird abgebildet, ob eine Brauerei eine Tochtergesellschaft einer anderen Brauerei ist.

Der Besucher der Web-Seite soll seine Postleitzahl eingeben können, damit die Seite ihm die Händler in einem frei wählbaren Umkreis ausgibt, bei denen das Bier gekauft werden kann. Händler führen in der Regel mehrere Biere im Sortiment!

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

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

2.3 Referenzielle Integrität



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

- Grundkurs

- Wie Sie eine verletzte referenzielle Integrität erkennen können
- Wie die Verletzung entstehen kann
- Vorteil einer laufenden Nummer als Primärschlüssel

- Vertiefendes

- Löschweitergabe
- Löschkennzeichen
- Änderungsweitergabe
- Duplizieren

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

Tabelle: bestellung		
bestellung_id	[...]	kunde_id
1	[...]	12345
2	[...]	12345
3	[...]	12347
4	[...]	12346

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

Bild 2.18 Vollständiger Datenauszug von kunde und bestellung



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



Definition 22: Referenzielle Integrität

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

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

2.3.1 Verletzung der referenziellen Integrität durch Löschen

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

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

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

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

Oftmals wird überhaupt nicht gelöscht! Man fügt den Tabellen standardmäßig eine Spalte mit dem Namen `aktiv` oder `deleted` hinzu. Diese Spalte dient als Löschkennzeichen. Will man einen Datensatz löschen, wird dieses Kennzeichen beispielsweise auf 1 gesetzt.

Bei allen weiteren Aktivitäten wird immer darauf geachtet, dass man nur Zeilen mit einem Löschkennzeichen 0 verarbeitet¹².

2.3.2 Verletzung der referenziellen Integrität durch Änderungen

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

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

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

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

Lässt sich das nicht vermeiden, dupliziert man die Zeile und ändert den Primärschlüsselwert in der neuen Zeile; die alte bleibt unverändert und wird *versiegelt*. Für alle zeitlich nachfolgenden Ereignisse wird die neue Zeile mit dem geänderten Primärschlüsselwert verwendet. Die alten Referenzen bleiben dabei erhalten und zeigen auf die nun veraltete, aber intakte Zeile¹³.

2.4 Normalformen



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

- Grundkurs
 - Wiederholungsgruppen und Atomarität

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

Der inhaltliche Aufbau von Tabellen erfolgt oft sehr intuitiv. Dies hat zur Folge, dass viele Datenbanken im Laufe der Zeit sehr langsam im Zugriff werden, die Redundanz zu- und die Konsistenz abnimmt. Man sucht Wege, Datenbanken grundsätzlich so zu gestalten, dass diese Probleme erst gar nicht entstehen. Das Ergebnis ist die Formulierung von Normalformen. Die Anpassung einer bestehenden Datenbank an die Normalformen nennt man Normalisierung.

2.4.1 Normalform 1

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

Tabelle: warenkorb		
warenkorb_id	artikel	kunde_id
1	7856 30;7863 50;9015 1	12345
2	3006 1;3010 4	12346

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

Bild 2.19 Tabelle warenkorb vor der Normalisierung



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

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

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



Definition 23: Wiederholungsgruppe

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

In der erste Zeile der Tabelle `warenkorb` haben wir eine Wiederholungsgruppe mit drei Elementen; die Listenelemente der Wiederholungsgruppe werden durch ein Semikolon

getrennt. Jedes Listen- oder besser Gruppenelement enthält die Informationen zu einer Bestellposition: Artikelnummer und Anzahl.



Definition 24: Wiederholungsgruppenfreiheit

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

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

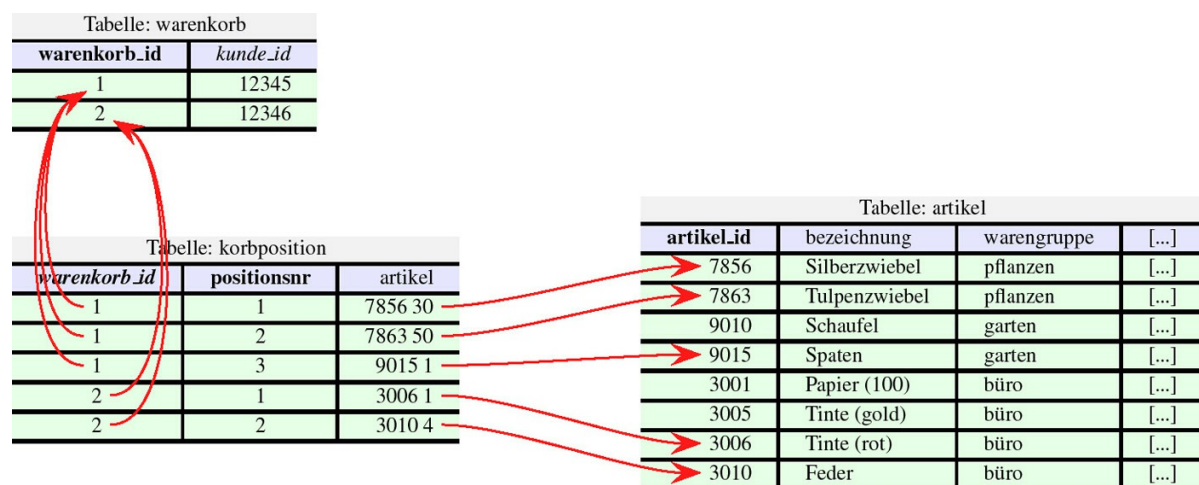


Bild 2.20 Tabelle warenkorb ohne Wiederholungsgruppen

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



Definition 25: Atomar

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

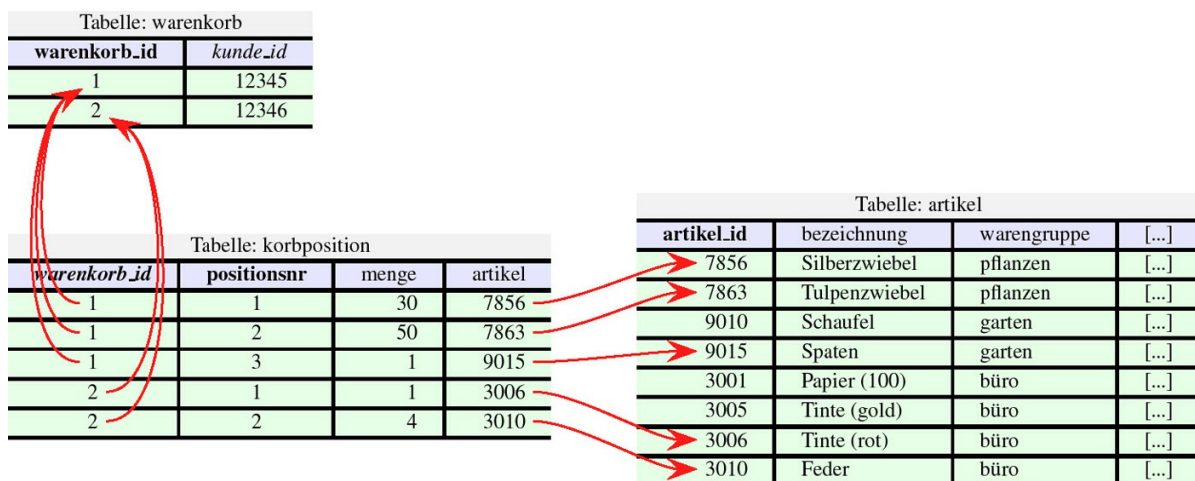


Bild 2.21 Tabelle warenkorb in der 1. Normalform

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



Definition 26: Erste Normalform

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

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



Aufgabe 2.17: Überführen Sie [Bild 2.21](#) in ein ER-Modell in Krähenfuß-Notation. Tipp: Es kommen nur $1:n$ -Verknüpfungen vor.

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

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

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

Die Aufbereitung und Darstellung der Daten wird durch Skriptsprachen wie PHP oder JavaScript geleistet. Diese können die Daten des Warenkorbs leicht dekodieren und ggf. wieder zusammengepackt an den Datenbankserver zurückschicken, da für den einzelnen Vorgang nur geringe Datenmengen verarbeitet werden müssen. Meist werden die

Daten dazu im JSON-Format bereitgestellt. Mehr dazu in [Kapitel 17](#).

2.4.2 Normalform 2

Betrachten Sie die Inhalte der Tabelle `artikel` in [Bild 2.21](#).



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

Nur weiterlesen, wenn Sie sich vorher selbst etwas überlegt haben: Die erste Ziffer hat einen Bezug zur Warengruppe. Man spricht bei solchen Ziffern von *Nummernkreisen*.



Definition 27: Voll- und teilfunktional

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

Teilfunktionalität tritt nur bei zusammengesetzten Schlüsseln auf (siehe [Definition 8](#) in [Abschnitt 2.1.3](#)). Laufende Zähler sind – außer bei *bösartig* konstruierten Gegenbeispielen – davon nicht betroffen. Vermeiden Sie daher informationstragende Schlüssel und verwenden Sie laufende Zähler.



Definition 28: Zweite Normalform

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

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

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

- Eine Tabelle buch mit dem Primärschlüssel isbn und einer Spalte verlagsnamen. Der Name des Verlags ergibt sich aus einer der Zifferngruppen der ISBN (siehe [\[Wik23n\]](#)).
- Die Telefonnummer setzt sich aus mehreren unterschiedlichen Teilnummern zusammen, wobei zwischen Festnetz- und Mobilfunknummern unterschieden wird. Kommt jetzt in der Tabelle noch eine Spalte ort bzw. anbieter vor, wäre diese Spalte nicht vom gesamten Schlüssel, sondern nur von einem Teil abhängig (siehe [\[Wik21\]](#)).
- Die IBAN ist ebenfalls eine Komposition aus verschiedenen Teilinformationen. Sie enthält neben anderen Informationen auch die Bankleitzahl. Kommt nun in der Tabelle mit dem Primärschlüssel iban die Spalte bankname vor, wäre diese nicht vom gesamten Primärschlüssel, sondern nur von einem Teil der IBAN abhängig (siehe [\[Wik23m\]](#)).
- Die Kennzeichnung von Eiern setzt sich aus den

Informationen Haltungsform, Herkunftsland und Betriebsnummer zusammen. In einer Tabelle mit dem Primärschlüssel erzeugercode wäre die Spalte herkunftsland somit teilfunktional (siehe [Wik23g]).



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

2.4.3 Normalform 3

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

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

Bild 2.22 Tabelle bankverbindung mit transitiven Informationen



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

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



Definition 29: Transitiv

Eine Nichtschlüsselspalte ist *transitiv*, wenn sie sich aus anderen Nichtschlüsselspalten herleiten lässt.

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



Definition 30: Dritte Normalform

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

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



Aufgabe 2.21: Erstellen Sie zu [Bild 2.23](#) ein ER-Modell in Krähenfuß-Notation.

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

Bild 2.23 Tabelle bankverbindung und bank ohne transitive Informationen



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

2.4.4 Normalform Rest

Es gibt noch die Boyce-Codd¹⁴-Normalform(BCNF), die 4. und die 5. Normalform, auf deren genaue Darstellung ich hier verzichten möchte. Die BCNF ist in diesem Buch in die 3. Normalform eingeflossen. Die Unterscheidung zwischen der 3. Normalform laut Literatur¹⁵ und der BCNF wird in der Praxis als akademisch empfunden. Schließlich sind beide einschränkende Vorschriften zum Thema transitive Informationen. Nichts spricht dagegen, gleich die schärfere BCNF als 3. Normalform zu verwenden.

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

Die 5. Normalform ist so was wie "*Jetzt hören wir aber auf mit dem Normalisieren*". Diese verlangt, dass Tabellen, deren Inhalte sich durch Verbundoperationen (JOIN, UNION, INTERSECT etc.) herstellen lassen, aufgelöst werden. Hier geht es wieder darum, Redundanzen (siehe [Definition 13](#) in [Abschnitt 2.1.3](#)) zu vermeiden und wartungsstabiler zu werden. Ist nämlich eine Tabelle aus anderen herleitbar, muss die Information ja – direkt oder indirekt – mehrfach abgelegt werden.

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

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

¹ Das Format ist hier vereinfacht und mischt Stufennummern mit Satzzeichen. Es lohnt sich, mal einen Blick in die Spezifikation zu werfen: [[Wik23d](#)].

² Eine analoge Unterscheidung findet Sie auch in den OO-Begriffen *Aggregation* und *Komposition*.

³ Siehe [[Wik23o](#)].

⁴ Siehe [[Wik23j](#)].

⁵ Z.B. Vermeidung der Änderungsweitergabe in [Abschnitt 2.3.2](#) oder NoSQL-Dokumente in [Abschnitt 17.2.2](#).

⁶ Abkürzung für: Computer Aided Software Engineering; engl. für rechnerunterstützte Softwareentwicklung.

⁷ Mehr unter dem Stichwort *Self Join* in [Abschnitt 11.5](#).

⁸ In [Abschnitt 6.2.1](#) wird erklärt, wie man so einen Index erstellt.

⁹ Im Ansatz sehr bedenklich: Plausibilitätsprüfungen sind Aufgabe der Funktionsschicht: siehe [Abschnitt 5.2](#).

¹⁰ Objektorientierte Datenbanken oder auch schon objektrelationale Datenbanken gehen genauso vor. Dies muss technisch erheblich unterstützt werden, damit die Zugriffe einigermaßen performant sind.

¹¹ Um ehrlich zu sein, nur bei den Operationen INSERT und DELETE. Auswertungen durch SELECT können sehr wohl durch die Hilfstabelle verlangsamt werden.

¹² Ob dabei datenschutzrechtliche Bestimmungen wie das informelle Selbstbestimmungsrecht oder die Datenschutzgrundverordnung (DSGVO) berührt werden, muss von Ihnen überprüft werden! Eine nähere Betrachtung der Löschprobleme, die durch die DSGVO ausgelöst werden, kann hier nicht erfolgen.

¹³ Hier haben wir ein gutes Beispiel dafür, dass Redundanzen sinnvoll sein können.

¹⁴ Derselbe Codd, der die relationale Datenbank *erfunden* hat.

¹⁵ Siehe [[KE01](#)]

3 Unser Beispiel: Ein Online-Shop



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

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



Hinweis: Alle Tabellen enthalten die Spalte `deleted`. Diese Spalte markiert, ob eine Zeile als gelöscht markiert ist (=1) oder nicht (=0). Hintergrund ist die Erhaltung der referenziellen Integrität bei Löschoperationen (siehe Hinweis zum Löschkennzeichen in [Abschnitt 2.3.1](#)).

3.1 Kundenverwaltung

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

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

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

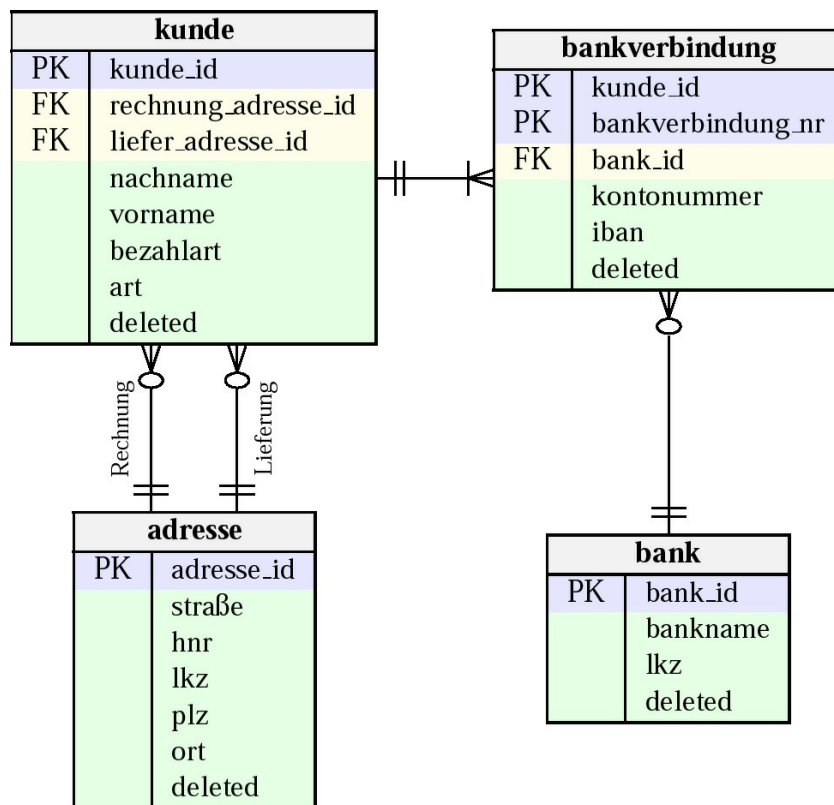


Bild 3.1 ER-Modell: Kundenverwaltung

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

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

3.2 Artikelverwaltung

Auch hier tragen wir erst mal zusammen, was bisher schon bezüglich der Artikel gesagt wurde: Wir haben die Tabellen

artikel, warengruppe und lieferant.

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

Da schon eine Tabelle adresse vorhanden ist, sollten wir für den Lieferant die Adressdaten nicht in der Tabelle lieferant speichern. Da ein Lieferant mindestens eine Adresse haben muss und eine Adresse zu keinem oder einem Lieferanten gehört, handelt es sich hier um eine 1:1-Verknüpfung, die nicht wie oben in [Abschnitt 2.2.2.1](#) beschrieben, durch einen gemeinsamen Primärschlüssel gelöst wird.

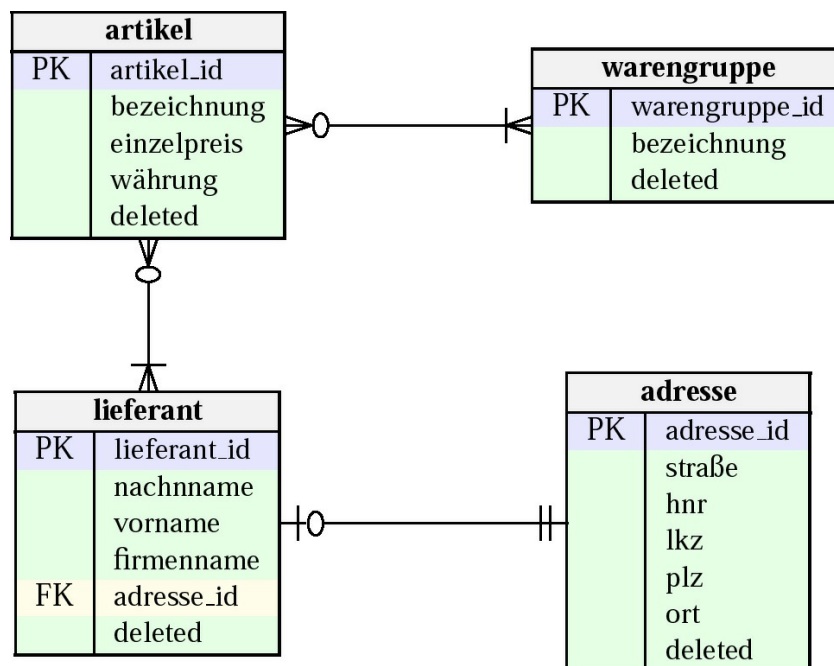


Bild 3.2 ER-Modell: Artikelverwaltung

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

3.3 Bestellwesen

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

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

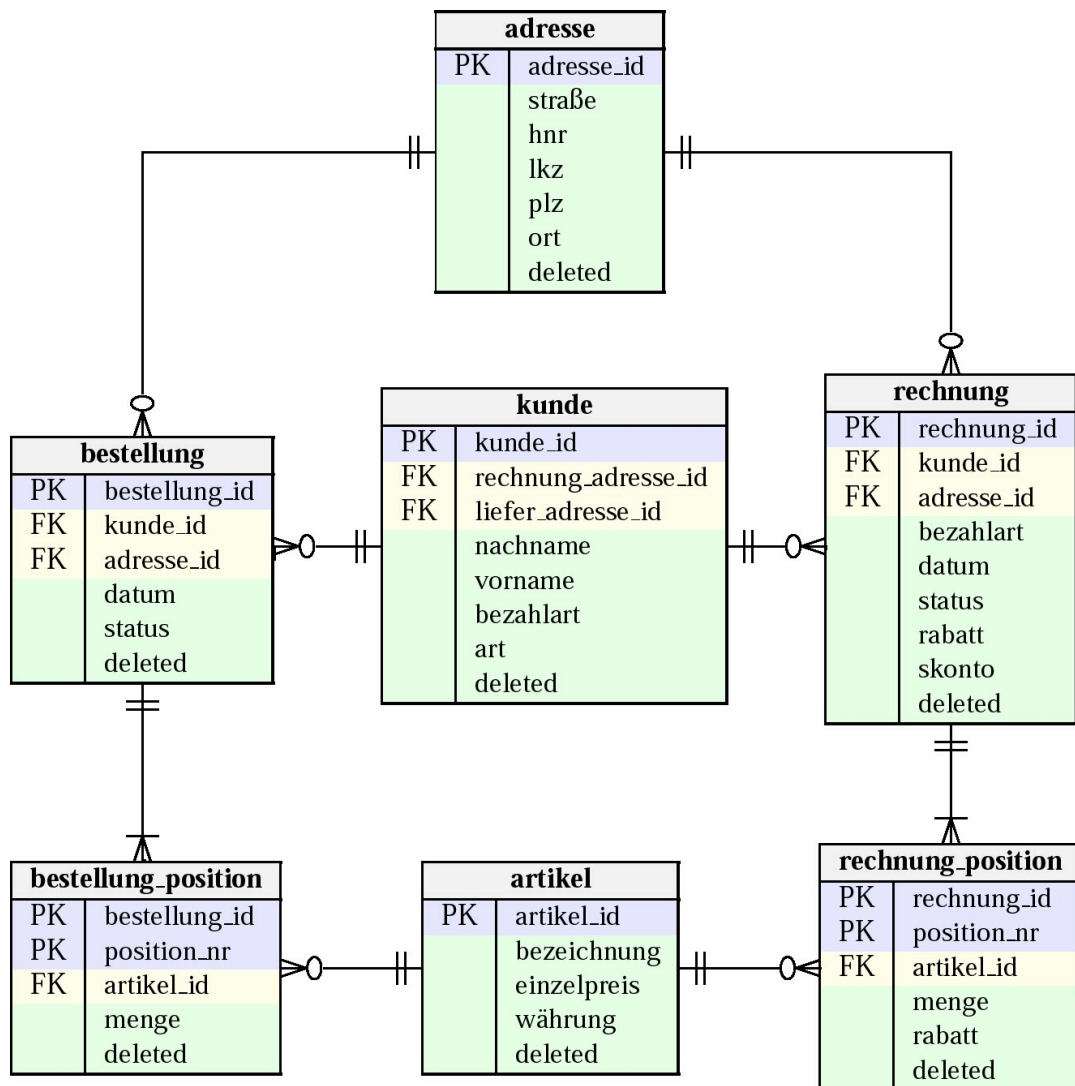


Bild 3.3 ER-Modell: Bestellwesen

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

Die Tabellen *rechnung* und *bestellung* sind jeweils mit der Tabelle *adresse* $1:n$ verknüpft. Die Frage ist, ob diese Information nicht aus der Tabelle *kunde* ermittelt werden kann. Dort gibt es doch eine Liefer- und eine Rechnungsadresse. Dies hängt von der Implementierung ab.

Der Kunde kann, muss aber nicht unterschiedliche Adressen angeben; dies soll auf der Oberfläche vom Kunden ausgewählt werden. Diese getroffene Auswahl wird als Fremdschlüsselwert in die jeweilige Tabelle geschrieben.

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

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

TEIL II

Datenbank aufbauen

4 Installation des Servers

4.1 MySQL unter Windows 11



Installation des MySQL Servers als Entwicklerserver

- Grundkurs
 - Installationsquelle
 - Installation unter Windows 11

Der MySQL Server und eine Reihe von weiteren Tools können direkt mithilfe eines Installers unter Windows 11 eingerichtet werden. Gehen Sie dazu nach <http://dev.mysql.com/downloads/file/?id=518834> und wählen Sie den *MySQL-Installer for Windows* aus. Anschließend können Sie sich registrieren oder direkt zur Download-Seite weiterspringen.



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

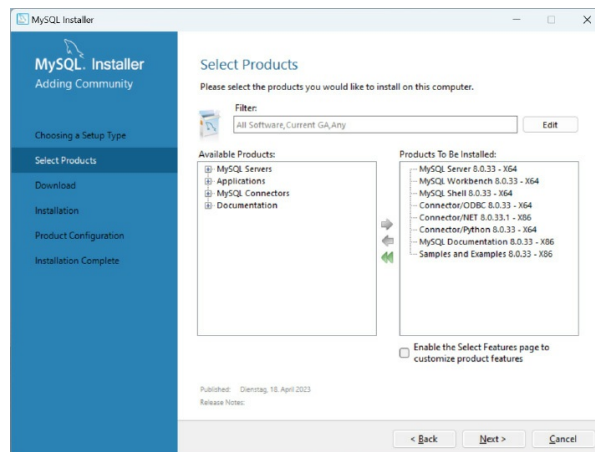


Bild 4.1 Schritt 2: Produktauswahl

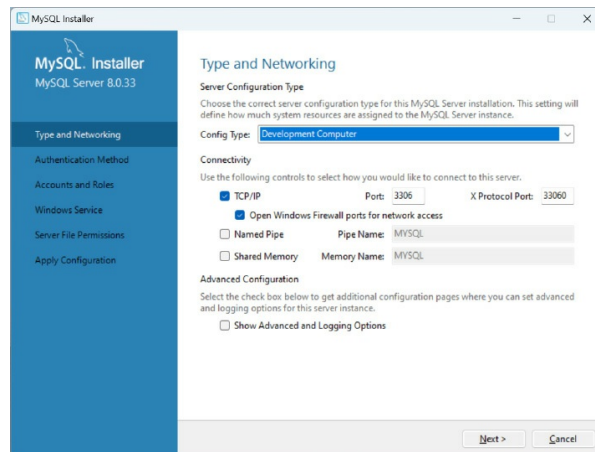


Bild 4.2 Schritt 4: Servertyp und Netzwerk

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

- **MySQL Server:** Community Server
- **MySQL Workbench:** Sie ist in jedem Fall ein nützliches

Werkzeug und daher zu empfehlen. Sie können damit über eine grafische Oberfläche Datenbanken planen, anlegen und bearbeiten.

- **MySQL Shell:** Konsolenbasiertes Kommandofenster
- **Connectoren:** Es werden die APIs für ODBC, Python und .NET installiert. Falls Sie eine Anwendung in einer der Sprachen entwickeln und auf den MySQL Server zugreifen wollen, ist es genau das, was Sie brauchen.
- **Beispiele:** Beispieldatenbanken und Skripte
- **Dokumentation:** Handbuch und How-tos

Was wir hier nicht installieren werden, ist der *MySQL Router*. Diese Komponente wird zur Implementierung von InnoDB-Cluster verwendet. Mithilfe eines solchen Clusters kann man Architekturen aufsetzen, die dafür konzipiert sind, Daten auf mehrere Serverinstanzen zu verteilen. Wir werden diese Technologie – so spannend sie auch ist – in diesem Buch nicht behandeln.²



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

Klicken Sie nun auf NEXT >. Es kann sein, dass der Installer merkt, dass bestimmte Komponenten fehlen, z.B. *MSVisual C++ 2015-2019 Redistributable (X64)*. Klicken Sie auf EXECUTE und folgen den Anweisungen. Irgendwann klicken Sie dann wieder auf NEXT >.

Falls Sie jetzt merken, dass Sie bestimmte Komponenten nicht installieren möchten, gehen Sie mit < BACK zurück und wählen

Sie CUSTOM. Deselektieren Sie dann die nicht erwünschte Komponente. Klicken Sie auf EXECUTE und beobachten Sie dabei die Fortschrittsanzeige. Ich empfehle Ihnen, vor Ort zu bleiben, um Fehlermeldungen gleich bearbeiten zu können.

Zuerst werden die Komponenten mit TO BE DOWNLOADED heruntergeladen. Lief was schief, klicken Sie auf TRY AGAIN. Falls das nichts fruchtet: Einfach die Rückfrage, ob man schon mal die erfolgreich heruntergeladenen Produkte installieren möchte, mit JA quittieren.

Im nächsten Schritt klicken Sie auf NEXT > und es erscheint [Bild 4.2](#). Unter *Config Type* kann die Verwendung des Servers festgelegt werden:

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

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

- **TCP/IP:** Wenn Sie diese Option auswählen (Default), kann der Server außerhalb Ihres Rechners über eine IP-Adresse angesprochen werden. Falls Sie die Option deselektieren, kann der Server nur über `127.0.0.1` oder `localhost` erreicht werden. Die vorgeschlagene Portnummer 3306 sollte nur verändert werden, wenn Sie genau wissen, warum. Der Client – eine eigene Anwendung oder der MySQL Client – muss dann über die andere Portnummer informiert werden, da er sonst versucht, den Standardport zu verwenden.
- **Open Firewall port for network access:** Die Firewall hält normalerweise die Ports geschlossen, um unerwünschten Besuchern das Leben etwas zu erschweren. Damit aber der Server für einen Client erreichbar ist, muss die Portnummer in der Firewall freigeschaltet werden. Lassen Sie diese Checkbox also selektiert.

Nach NEXT > müssen Sie festlegen, ob Sie sich über die moderne Variante der Authentifizierung oder über die der Version MySQL 5.x beim Server ausweisen (siehe [Bild 4.3](#)). Ich empfehle Ihnen hier und später in Produktion immer die moderne, da diese erheblich sicherer ist; diese Option ist vorausgewählt.

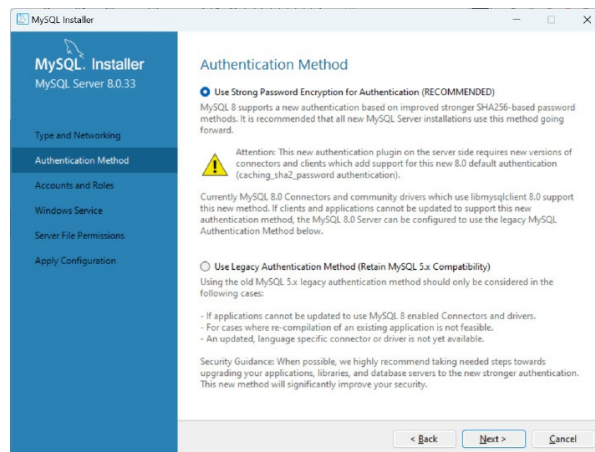


Bild 4.3 Schritt 5: Authentifizierungsmethode

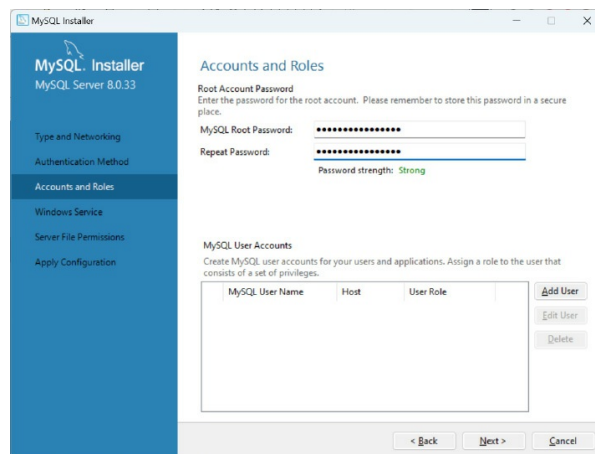


Bild 4.4 Schritt 6: root-Passwort festlegen

In Schritt 6 (siehe [Bild 4.4](#)) legen Sie das root-Passwort fest. Merken Sie sich dieses unbedingt. Mir ist kein Verfahren bekannt, ein einmal vergessenes root-Passwort wieder zurückzusetzen. Es ist daher sinnvoll, einen zweiten Administrator einzurichten, der dann das root-Passwort neu setzen kann. Das Passwort selbst sollte mindestens acht Stellen haben sowie Groß- und Kleinbuchstaben, Zahlen und Sonderzeichen enthalten. Bewährt haben sich Eselsbrücken. Beispielsweise wird *Wer 1mal lügt, dem glaubt man nicht*,

selbst wenn er nun die Wahrheit spricht! zu W11, dgm, swendWs ! ;-)

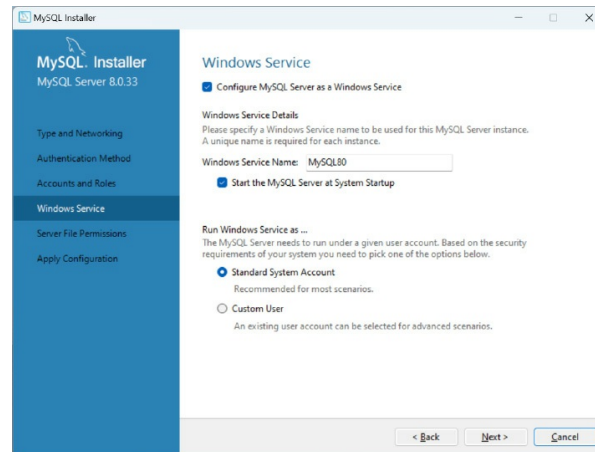


Bild 4.5 Schritt 7: Service konfigurieren

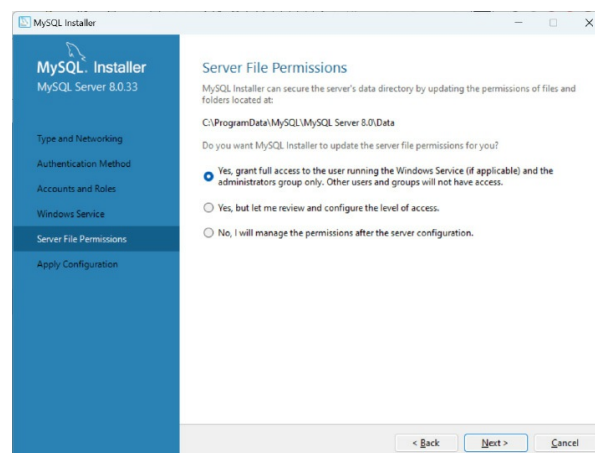


Bild 4.6 Schritt 8: Zugriffsrechte konfigurieren

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

haben, sollten Sie den Namen einfach stehen lassen. Alternativ könnte man die Server durchnummerieren³.

Wenn Sie wollen, können Sie auch den Service manuell starten und nicht automatisch beim Systemstart. Dann sollten Sie die Checkbox *Start the MySQL Server at System Startup* deselektieren. Danach werden Sie gebeten, die Zugriffsrechte für das Datenverzeichnis – also das Verzeichnis, wo nachher tatsächlich die Tabellen abgespeichert werden – festzulegen ([Bild 4.6](#)). Ich empfehle die Vorbelegung beizubehalten. Drücken Sie nun auf NEXT >.

Ihnen wird jetzt eine Liste der auszuführenden *Configuration Steps* angezeigt. Klicken Sie einfach auf EXECUTE. Hat alles geklappt, drücken Sie auf FINISH.

Klicken Sie nun nochmalig auf NEXT >, damit die Beispieldatenbanken und Skripte installiert werden. Sie müssen nun das root-Passwort eingeben, was Sie in der Maske von [Bild 4.4](#) vergeben haben. Sie können erst fortfahren, wenn Sie mit CHECK die Zugangsdaten erfolgreich überprüft haben. Ein letztmaliges NEXT > und EXECUTE und Sie sind fertig. Auf der letzten Maske deselektieren Sie nun die beiden Checkboxes und klicken Sie auf FINISH.

Mithilfe der Workbench oder des MySQL Clients kann nun überprüft werden, ob der Server läuft. Rufen Sie unter START → MYSQL 8.0 COMMAND CLIENT - UNICODE den MySQL Client auf. Alternativ aber auch direkt, wie hier abgebildet:

```
1 C:\>"c:\Program Files\MySQL\MySQL Server 8
.0\bin\mysql" -uroot -p
2 Enter password: *****
3 Welcome to the MySQL monitor. Commands end
with ; or \g.
4 Your MySQL connection id is 21
5 Server version: 8.0.33 MySQL Community Ser
```

```

ver - GPL
6  [...]
7  mysql> SHOW DATABASES;
8  +-----+
9  | Database          |
10 +-----+
11 | information_schema |
12 | mysql              |
13 | performance_schema |
14 | sakila             |
15 | sys                |
16 | world              |
17 +-----+
18 6 rows in set (0.00 sec)
19
20 mysql> exit
21 Bye

```

Prima, der Client findet den Server.

4.2 MariaDB unter Windows 11



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

- Grundkurs
 - Installationsquelle
 - Installation unter Windows 11
- Vertiefendes
 - Dateistruktur

- Windows-Suchpfad anpassen

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

Für Linux und Windows ist XAMPP⁴ eine weit verbreitete Entwicklungs- und Testumgebung für Datenbank- und Webentwickler. Seit dem Release v5.5.30 bzw. v5.6.14 wird anstelle von MySQL MariaDB verwendet. Die Gründe dafür können Sie unter <https://www.quora.com/Why-did-XAMPP-switch-to-MariaDB-from-MySQL> nachverfolgen. Die Installationspakete sowohl für Windows als auch für verschiedene Linux-Distributionen können unter <https://www.apachefriends.org/de/download.html> heruntergeladen werden. Installationsanweisungen und Dokus zu XAMPP sind in Form von FAQs vorhanden:

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

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

Die hier verwendete Konfiguration ist XAMPP 8.2.4 mit MariaDB 10.4.28.⁵ Falls Sie eine ältere Version von XAMPP installiert haben, sollten Sie diese zuerst deinstallieren: START→

SYSTEMSTEUERUNG→PROGRAMME UND FUNKTIONEN→XAMPP DEINSTALLIEREN. Laden Sie anschließend von [[Fri23](#)] die Windows-Version mit Installer herunter.

Vielleicht bekommen Sie eine Warnung, wie in [Bild 4.7](#) zu sehen. Wir wollen uns den Ratschlag zu Herzen nehmen und später nicht ins Programmverzeichnis installieren.

Nun startet die Installation mit einer Begrüßung, und nach einem Klick auf NEXT > können Sie auswählen (siehe [Bild 4.8](#)), welche Komponenten installiert werden sollen.

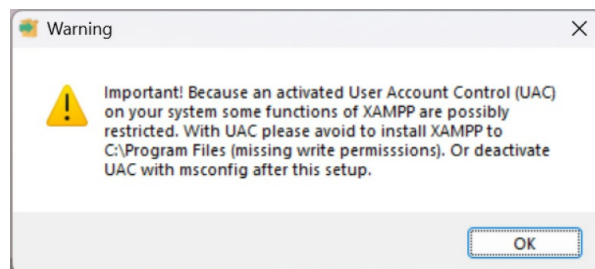


Bild 4.7 Warnung wegen UAC

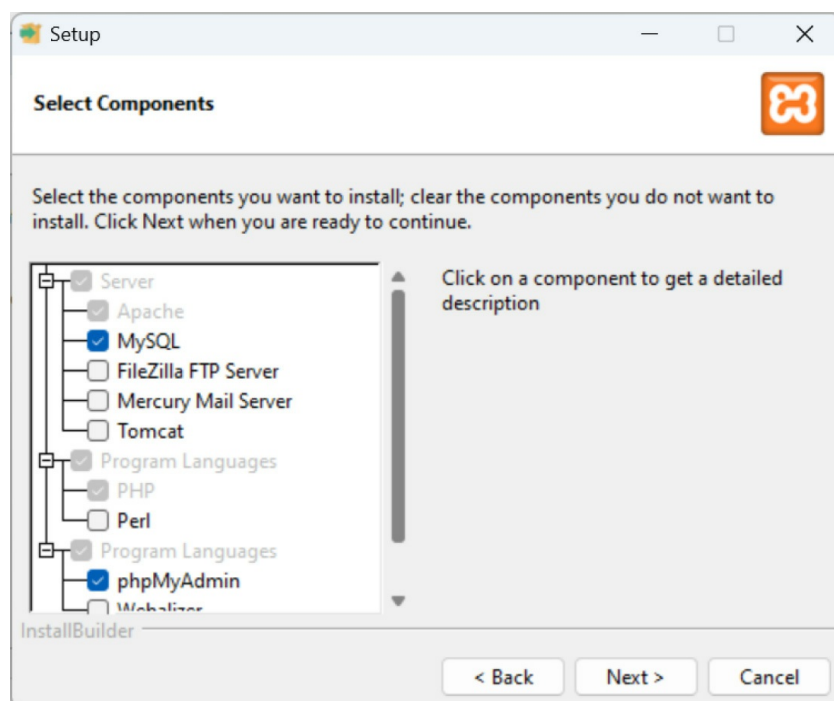


Bild 4.8 Auswahl der Komponenten

Sie können bedenkenlos alles ausgewählt lassen; ich selektiere nur die, die für dieses Buch gebraucht werden: MySQL und phpMyAdmin; Apache und PHP lassen sich nicht abwählen.

Im nächsten Schritt wählen Sie das Zielverzeichnis aus (siehe [Bild 4.9](#)). Wenn Sie die obige Warnung erhalten haben, sollten Sie tatsächlich die Daten unter `c:\xampp` oder einem vergleichbaren Pfad ablegen.

Sie können nun die Sprache auswählen (siehe [Bild 4.10](#)) und nach weiteren Klicks startet die Installation. Die dauert einige Zeit, aber bei Weitem nicht mehr so lange, wie ich es noch für die erste Auflage dieses Buches berichten musste :-). Wenn Sie nun auf FINISH klicken, achten Sie darauf, dass der Start des Control Panels aktiviert ist.

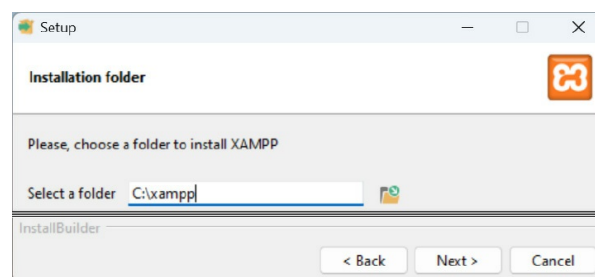


Bild 4.9 Verzeichnis festlegen

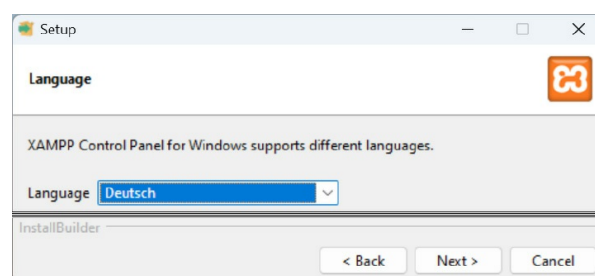


Bild 4.10 Sprache festlegen

Es erscheint das Control Panel ([Bild 4.11](#)). Starten Sie nun Apache und MySQL – wobei tatsächlich MariaDB gestartet wird. Wenn Sie in der MySQL-Zeile auf ADMIN klicken, wird *phpMyAdmin* geöffnet. Dieses sehr bequeme Tool dient der eingeschränkten Administration des Servers und der Datenabfrage.

Das MariaDB Server-Programm liegt in einem XAMPP-Pfad: `c:\xampp\mysql\bin`. Hier sind die Programme und – *sehr wichtig!* – die `my.ini` (siehe [Zeile 11](#)) abgelegt. Der MariaDB Client ist in [Zeile 15](#) zu finden, der Server selbst in [Zeile 19](#).

```
1  c:\>cd c:\xampp\mysql\bin
2
3  c:\xampp\mysql\bin>dir
4  Verzeichnis von c:\xampp\mysql\bin
5
6  26.06.2023  16:50      <DIR>          .
7  26.06.2023  16:46      <DIR>          ..
8  15.07.2022  18:07                21.984  api-ms
-win-core-console-l1-1-0.dll
9  [...]
10 24.01.2023  10:37                31.568  msvcp1
40_codecvt_ids.dll
11 26.06.2023  16:50                5.776  my.ini
12 30.01.2023  17:46            3.735.976  myisam
chk.exe
13 [...]
14 30.01.2023  17:36            25.746  myrock
s_hotbackup
15 30.01.2023  17:45            3.780.520  mysql.
exe
16 30.01.2023  17:45            3.684.776  mysqla
dmin.exe
17 30.01.2023  17:45            3.818.408  mysqlb
inlog.exe
18 30.01.2023  17:45            3.696.040  mysqlc
heck.exe
```


19	30.01.2023	17:46	16.549.288	mysqld
	.exe			
20	30.01.2023	17:45	3.770.280	mysqld
	ump.exe			
21	[...]			
22	30.05.2022	12:57	44.320	vcrunt
	ime140_1.dll			
23		90 Datei(en),	145.484.830 B	
		ytes		
24		2 Verzeichnis(se),	60.242.505	
	.728 Bytes frei			

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

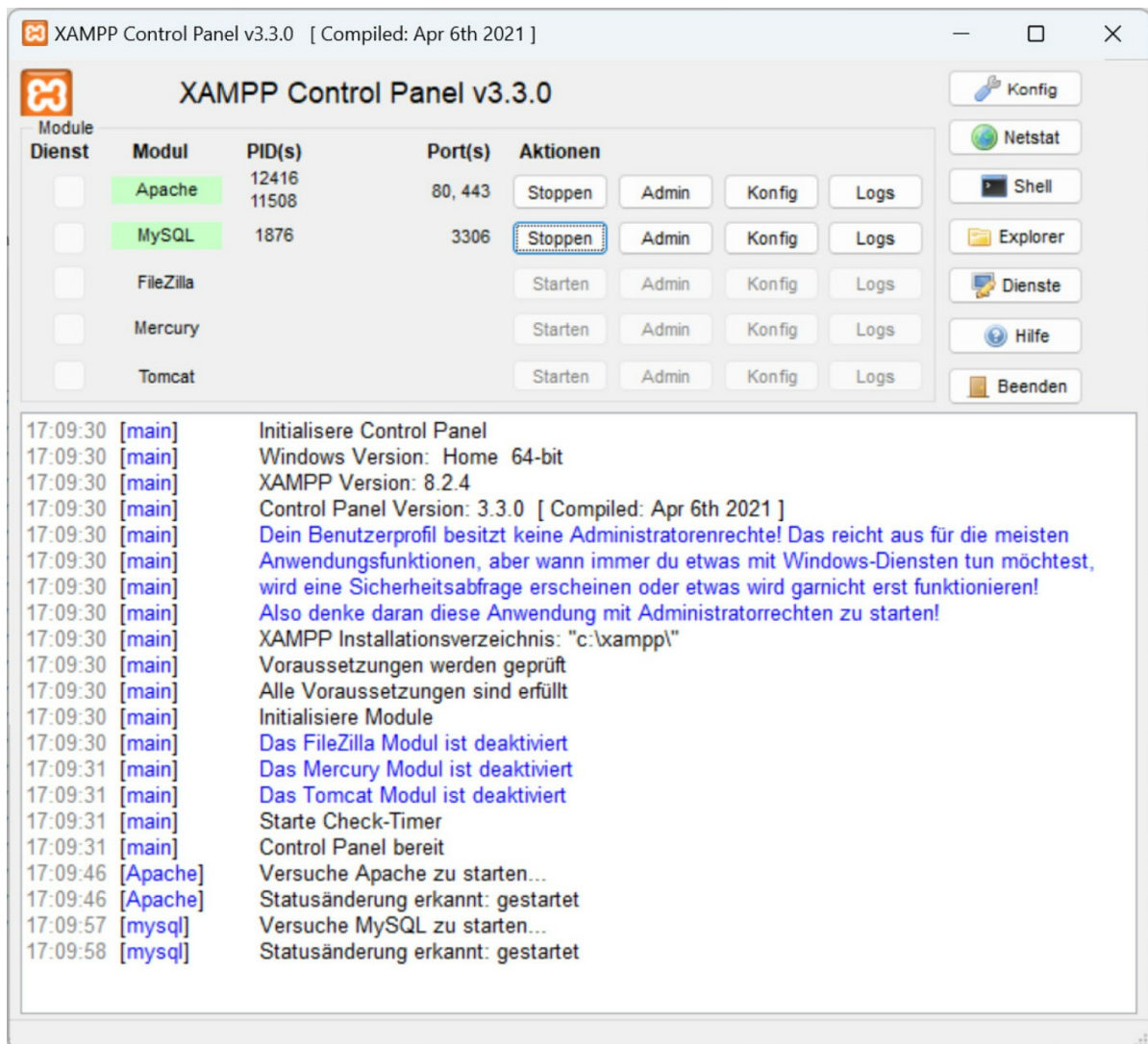


Bild 4.11 Das XAMPP-Panel

Dieses Verzeichnis sollte regelmäßig gesichert werden. Man kann hier die schon installierten Datenbanken erkennen: mysql ([Zeile 17](#)), performance_schema ([Zeile 20](#)), phpmyadmin ([Zeile 21](#)) und test ([Zeile 22](#)).

```

1  c:\xampp\mysql\bin>cd ../data
2
3  c:\xampp\mysql\data>dir
4  Verzeichnis von c:\xampp\mysql\data
5
6  26.06.2023 17:09 <DIR> .

```

```

7  26.06.2023 16:46 <DIR> ..
8  26.06.2023 17:06 24.576 aria_log.00000001
9  26.06.2023 17:06 52 aria_log_control
10 26.06.2023 17:06 12.582.912 ibdata1
11 26.06.2023 17:09 12.582.912 ibtmp1
12 26.06.2023 17:06 1.555 ib_buffer_pool
13 26.06.2023 17:09 5.242.880 ib_logfile0
14 21.10.2019 14:19 5.242.880 ib_logfile1
15 21.10.2019 14:19 0 multi-master.info
16 21.10.2019 14:17 49 my.ini
17 26.06.2023 16:46 <DIR> mysql
18 26.06.2023 17:09 5 mysql.pid
19 26.06.2023 17:09 3.560 mysql_error.log
20 26.06.2023 16:46 <DIR> performance_schema
21 26.06.2023 16:46 <DIR> phpmyadmin
22 26.06.2023 16:46 <DIR> test
23                               11 Datei(en), 35.681.381 Bytes
24                               6 Verzeichnis(se), 60.240.384.
000 Bytes frei

```

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

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

`c:\xampp\mysql\bin\mysql irgendwelche aufrufparameter`
2. Sie fügen den Pfad dem Windows-Suchpfad an: WINDOWS + PAUSE → SYSTEM → INFO → ERWEITERTE SYSTEMEINSTELLUNGEN → UMGEBUNGSVARIABLEN → unter Systemvariablen PATH markieren → BEARBEITEN → NEU → Pfadangabe `c:\xampp\mysql\bin` eingeben und ganz oft OK drücken. Nun lässt sich der Client von jeder Stelle aus aufrufen. Sie müssen aber die aktuelle

COMMAND-Box erst schließen und eine neue öffnen, damit die Änderungen übernommen werden.

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

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

```
c:\>mysqladmin -uroot -pPasswordAlt password PasswordNeu.
```

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



Hinweis: Wenn Sie das Panel mit Beenden verlassen, könnte die Fehlermeldung wie in [Bild 4.12](#) mit einer Folge weiterer Absturzmeldungen erscheinen. Beenden Sie das Panel über den Windows Task Manager. Markieren Sie anschließend im Datei Explorer die Datei `c:\xampp\xampp-control.ini` und gehen Sie über die rechte Maustaste zu den Eigenschaften. Wechseln Sie zur Registerkarte Sicherheit, selektieren Sie Jeder und klicken Sie auf Bearbeiten; erlauben Sie jetzt den Vollzugriff.

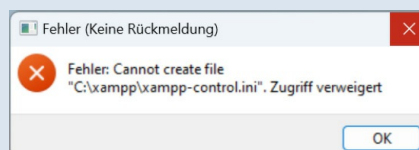


Bild 4.12 XAMPP-Absturz

4.3 Andere Installationen mit Docker

Einen SQL Server in einem Docker laufen zu lassen, ist eine wirklich ressourcenschonende und elegante Art, verschiedene SQL Server parallel auf einem Rechner bzw. unter einem Betriebssystem laufen zu lassen. Ich möchte hier kurz vorstellen, wie ich unter Debian⁶ im Docker die verschiedenen Server installiere und aufrufe.⁷ Als Docker Repository habe ich folgende Internetquelle eingetragen:

<https://hub.docker.com/>.

4.3.1 MySQL

Zunächst wird das Image aus [Doc21] heruntergeladen und installiert. Die aktuelle MySQL-Version für Linux im Docker ist 8.0.33.

```
1 adams@DebianSQL:~$ sudo docker pull mysql
2 Using default tag: latest
3 latest: Pulling from library/mysql
4 46ef68baacb7: Pull complete
5 [...]
6 Status: Downloaded newer image for mysql:latest
7 docker.io/library/mysql:latest
```

Damit wir später (MySQL-)Client-Programme mit dem Server verbinden können, müssen wir noch ein künstliches Netzwerk erstellen. Die dann laufenden Dockerinstanzen können über

dieses Netzwerk so kommunizieren, als wären sie miteinander verbundene Rechner.

```
1 adams@DebianSQL:~$ sudo docker network create mydockernet
```

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

```
1 adams@DebianSQL:~$ sudo docker run --name mysql_server01 -v /media/sf_src/:/home/buch --network mydockernet -e MYSQL_ROOT_PASSWORD=weinschlauchX10! -d mysql: latest
```

Die Parameter im Einzelnen:

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

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

- `mysql:latest`
Es wird die aktuellste Version des Servers im Docker-Image – hier 8.0.23 – gestartet.

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

```
1  adams@DebianSQL:~$ sudo docker run -it --n
ame mysql_client01 -v /media/sf_src/:/
      home/buch --network mydockernet --rm
mysql mysql -hmysql_server01 -uroot -
      pweinschlauchX10!
2  mysql: [Warning] Using a password on the c
ommand line interface can be insecure.
3  Welcome to the MySQL monitor. Commands end
with ; or \g.
4  Your MySQL connection id is 8
5  Server version: 8.0.33 MySQL Community Ser
ver - GPL
6  [...]
7  mysql> quit
8  Bye
```

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

Diese Parameter sind neu:

- `-it`
Eine Kombination aus `-i` und `-t`. Mit `-i` wird die

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

- `--rm`
Beim Start einer Dockerinstanz wird ein Dateisystem erstellt, auf welchem die Programme arbeiten können. Dieser Parameter bereinigt dieses Dateisystem nach Beendigung. Will man die Dateien beispielsweise zwecks Analyse bestehen lassen, lässt man `--rm` weg.
- Das erste `mysql`
Die Dockerinstanz wird auf Basis des Docker-Image `mysql` erstellt.
- Das zweite `mysql`
Der MySQL Client ist eine ausführbare Datei mit dem Namen `mysql`. Eine hier leider verwirrende Namensgleichheit.
- `-hmysql_server01`
Das ist kein Docker-Parameter, sondern ein Parameter des MySQL Clients. Er weist den Client an, sich mit dem Server namens `mysql_server01` zu verbinden.
- `-uroot`
Auch ein Client-Parameter: Der Client meldet sich mit dem Benutzernamen `root` an.
- `-pweinschlauchX10!`
Und noch ein Client-Parameter: das zum Benutzernamen passende Passwort.

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


```
1  adams@DebianSQL:~$ sudo docker exec -it mysql_server01 bash
2  bash-4.4# mysql -uroot -pweinschlauchX10!
3  mysql: [Warning] Using a password on the command line interface can be insecure.
4  Welcome to the MySQL monitor. Commands end with ; or \g.
5  Your MySQL connection id is 9
6  Server version: 8.0.33 MySQL Community Server - GPL
7  [...]
8  mysql> exit
9  Bye
10 bash-4.4# exit
11 exit
12 adams@DebianSQL:~$
```

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

```
1  adams@DebianSQL:~$ sudo docker stop mysql_server01
```

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

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



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

```
210 # secure-file-priv= NULL
211 secure-file-priv= /var/lib/mysql/uploads/
```

Das entsprechende Unterverzeichnis muss noch mit `mkdir` angelegt und mit `chown mysql:mysql` dem Server zur Bearbeitung zur Verfügung gestellt werden.

Bei der Installation unter Windows ist das entsprechende Verzeichnis schon eingerichtet worden und in `C:/ProgramData/MySQL/MySQL Server 8.0/my.ini` eingetragen:

```
210 # Secure File Priv.
211 secure-file-priv="C:/ProgramData/MySQL/MySQL Server 8.0/Uploads"
```

4.3.2 MariaDB

Das Vorgehen entspricht völlig dem von MySQL (siehe [Abschnitt 4.3.1](#)) und wird deshalb hier kürzer abgehandelt. Auch hier wird das Image aus [\[Doc21\]](#) heruntergeladen und installiert. Die dort verfügbare MariaDB-Version ist 11.0.2.

Zeile 2 ist nur notwendig, wenn Sie das Docker-Netzwerk nicht schon bei einer anderen Dockerinstanz – wie oben bei MySQL – eingerichtet haben.

```
1 adams@DebianSQL:~$ sudo docker pull mariadb
```

```

2  adams@DebianSQL:~$ sudo docker network create mydockernet
3  adams@DebianSQL:~$ sudo docker run --name mariadb_server01 -v /media/sf_src/:/home/buch --network mydockernet -e MYSQL_ROOT_PASSWORD=weinschlauchX10! -d mariadb:latest

```

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

```

1  adams@DebianSQL:~$ sudo docker run -it --name mariadb_client01 -v /media/sf_src/:/home/buch --network mydockernet --rm mariadb mariadb -hmariadb_server01 -uroot -pweinschlauchX10!
2  Welcome to the MariaDB monitor. Commands end with ; or \g.
3  Your MariaDB connection id is 4
4  Server version: 11.0.2-MariaDB-1:11.0.2+maria~ubu2204 mariadb.org binary distribution
5  [...]
6  MariaDB [(none)]> quit
7  Bye

```



Hinweis: Anders als bei der XAMPP-Installation in [Abschnitt 4.2](#) hat der Konsolen-Client hier nicht mehr den veralteten Namen `mysql`, sondern `mariadb`.

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

```
1 adams@DebianSQL:~$ sudo docker start maria
db_server01
2 mariadb_server01
3 adams@DebianSQL:~$ sudo docker stop mariad
b_server01
```

4.3.3 PostgreSQL

Wie bei der Einrichtung des MySQL-Containers⁸ (siehe [Abschnitt 4.3.1](#)) und des MariaDB-Containers (siehe [Abschnitt 4.3.2](#)), so gestaltet sich auch die Einrichtung des PostgreSQL-Containers denkbar einfach. Die Bedeutung der einzelnen Parameter lesen Sie bitte in den obigen Abschnitten nach. Unser Docker beinhaltet PostgreSQL in der Version 13.2. Zur Erinnerung: Zeile 10 ist nur notwendig, wenn Sie das Docker-Netzwerk noch nicht definiert haben.

```
1 adams@DebianSQL:~$ sudo docker pull postgr
es
2 Using default tag: latest
3 latest: Pulling from library/postgres
4 5b5fe70539cd: Pull complete
5 [...]
6 f06aa6280d55: Pull complete
7 Status: Downloaded newer image for postgre
s:latest
8 docker.io/library/postgres:latest
9
10 adams@DebianSQL:~$ sudo docker network cre
ate mydockernet
11 adams@debian:~$ sudo docker run --name pos
tgresql_server01 -v /media/sf_src:/home/buch
--network mydockernet -e POSTGRES_PASSWORD=wei
nschlauchX10! -d postgres: latest
12 adams@DebianSQL:~$ sudo docker run --name
```

```

postgresql_server01 -v /media/sf_src/:/home/bu
ch --network mydockernet -e POSTGRES_PASSWORD=
weinschlauchX10! -d postgres:latest
13 adams@DebianSQL:~$ sudo docker run -it --r
m --network mydockernet postgres psql -h postg
resql_server01 -U postgres
14 Password for user postgres:
15 psql (15.3 (Debian 15.3-1.pgdg120+1))
16 [...]
17 postgres=# quit
18 adams@DebianSQL:~$ sudo docker stop postgr
esql_server01

```

Alternativ zum Docker kann eine kostenfreie Version für Windows von <https://www.postgresql.org/download/windows/> heruntergeladen werden.

4.3.4 Microsoft SQL Server

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

Anders als bei den drei anderen Docker-Images verwende ich eine Microsoft-Internetquelle als Docker-Repository. Auch verwende ich nicht explizit den Befehl `docker pull`, sondern Docker erkennt, wenn ein Image angesprochen ist, welches er noch nicht kennt, und versucht es dann aus der angegebenen Quelle herunterzuladen. Dabei wurde hier und heute bei mir der SQL Server 2019 Version 15.0.4102.2 verwendet.

```

1 adams@DebianSQL:~$ sudo docker network cre
ate mydockernet
2 adams@DebianSQL:~$ sudo docker run --name
mssql_server01 --network mydockernet -v / medi

```

```

a/sf_src/~/home/buch -e 'ACCEPT_EULA=Y' -e 'SA
_PASSWORD=weinschlauchX10!' -p 1433:1433 -d mcr
.microsoft.com/mssql/server:latest
3 Unable to find image 'mcr.microsoft.com/ms
sql/server:latest' locally
4 latest: Pulling from mssql/server
5 bbf84ab0c13: Pull complete
6 [...]
7 Status: Downloaded newer image for mcr.mic
rosoft.com/mssql/server:latest
8
9 adams@DebianSQL:~$ sudo docker exec -u 0 -
it mssql_server01 bash
10 root@aa7ff192eac9:/# /opt/mssql-tools/bin/
sqlcmd -U sa -P weinschlauchX10!
11 1> SELECT 'Hello World!' AS greetings;
12 2> GO
13 greetings
14 -----
15 Hello World!
16
17 (1 rows affected)
18 1> quit
19 root@aa7ff192eac9:/# exit
20 exit
21 adams@DebianSQL:~$ sudo docker stop mssql_
server01

```

Der Parameter `-u 0` in Zeile 9 sollte nur dann verwendet werden, wenn man sich als root im Container anmelden möchte. Anders als bei den anderen SQL Server Containern, ist man innerhalb des MS SQL Server Containers kein root.

Zu allen Serversystemen existieren Unmengen an Wikis, Tutorials, YouTube-Videos und Foren. Zu fast jedem Aspekt wie *Installation*, *Sicherheit*, *Performance*, *Verlässlichkeit* usw. gibt es Spezialdokumentationen, die in der Regel aktueller und

ausführlicher sind, als ich es hier in einem Einführungsbuch über SQL sein kann und will.

¹ Stand 23.06.2023: `mysql-installer-web-community-8.0.33.0.msi`

² Eine gute Einführung finden Sie hier: [[MyS21d](#)].

³ Ich habe noch nie versucht, mehrere Instanzen auf einem Rechner unterzubringen, und kann daher nicht viel Erhellendes zu dieser Option beitragen.

⁴ $X = \{W|L\}$ und steht für {Windows|Linux}, Apache, MySQL, PHP und Perl.

⁵ Dabei muss ich erwähnen, dass in XAMPP der MariaDB-Server der Version 10.4.28 verwendet wird, aber auf <https://downloads.mariadb.org/> schon die Versionen 11.1.1 bzw. 11.2.0 *preview* verfügbar sind.

⁶ Debian-Version: 12 (Bookworm)

⁷ Eine Einführung in die Docker-Technologie finden Sie beispielsweise unter [[Arb17](#)].

⁸ Container ist ein anderes Wort für eine Dockerinstanz.

5 Datenbank und Tabellen anlegen

5.1 Die Programmiersprache SQL



Was ist SQL?

- Grundkurs
 - Geschichte und Struktur von SQL
- Vertiefendes
 - SQL:2023 vs. MySQL vs. MariaDB vs. PostgreSQL vs. T-SQL

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

1. Data Definition Language (DDL): Befehlssatz zum Anlegen, Ändern und Löschen von Datenbanken, Tabellen usw. und ihren Strukturen.

2. Data Manipulation Language (DML): Befehlssatz zum Einfügen, Ändern, Löschen und Auslesen von Daten aus den Tabellen.
3. Data Control Language (DCL) oder Data Administration Language (DAL): Befehlssatz zur Administration von Datenbanken.

Anders als bei imperativen Programmiersprachen wie C# oder Python wird kein Algorithmus implementiert. Vielmehr sagt man, was man haben möchte, und der Datenbankserver ermittelt das Ergebnis. Solche Sprachen nennt man *deklarativ* (siehe [[Wik23f](#)]).

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

1986 wurde der erste SQL-Standard vom ANSI³ verabschiedet, der 1987 von der ISO⁴ ratifiziert wurde (SQL1). 1992 wurde der Standard überarbeitet und als SQL-92 (oder auch SQL2) veröffentlicht. Es folgte die Version SQL:1999 (ISO/IEC 9075:1999, auch SQL3 genannt) mit ihrer letzten Revision SQL:2011 (ISO/IEC 9075:2011). Danach kam SQL:2016 (ISO/IEC 9075:2016), welcher 2019 um den Datentyp der mehrdimensionalen Arrays erweitert wurde (SQL/MDA:2019). Der aktuelle Standard ist SQL:2023 (ISO/IEC 9075:2023)⁵.

Abgrenzung zu SQL:2023⁶

Die hier vorgestellten Befehle sind alle mit dem MySQL Community Server, MariaDB, PostgreSQL und MS SQL Server getestet worden. Schwerpunkt der Darstellung ist MySQL. Die Befehlssätze von MySQL/MariaDB, PostgreSQL und Transact-SQL⁷ enthalten Befehle und Datentypen, die es im Standard

SQL:2023 nicht gibt. Ebenso gibt es in SQL:2023 welche, die in diesen Produkten nicht implementiert sind.

Der aktuelle Standard ist noch taufrisch: Am 1. Juni 2023 wurde SQL:2023 verabschiedet. Leider konnte ich keine Sprachspezifikation online finden, aber unter [\[Eis23b\]](#) finden Sie eine gute Übersicht über die neu eingeführten Features. Diese Features sind teilweise schon in den Servern verfügbar, aber bei weitem nicht vollständig und bei allen. Beispielhaft sei auf die Implementierungen in PostgreSQL verwiesen ([\[Eis23a\]](#)).

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

5.2 Anlegen der Datenbank



Jeder Hausbau fängt beim Fundament an.

- Grundkurs

- Der Aufruf des MySQL Clients
- Anzeigen aller Datenbanken des Servers mit `SHOW DATABASES`
- Anlegen einer Datenbank mit `CREATE DATABASE`

Bedingtes Anlegen der Datenbank mit `IF NOT EXISTS`

- Bedingtes Anlegen der Datenbank mit IF EXISTS
- Löschen einer Datenbank mit DROP DATABASE
- Vertiefendes
 - Unterschied DATABASE und SCHEMA
 - Zuweisen einer Zeichenkodierung für die Datenbank mit CHARACTER SET
 - Zuweisen einer Sortierung für die ganze Datenbank mit COLLATE



Die Quelltexte dieses Kapitels stehen in der Datei `listing01.sql` (siehe [Listing 28.1](#), [Listing 28.41](#) und [Listing 28.26](#)).

5.2.1 Wie rufe ich den MySQL Client auf?

Bevor wir eine Datenbank legen, möchte ich kurz die Verwendung des MySQL Clients vorstellen. Der Zugriff auf den MySQL Server kann auf verschiedenen Wegen erfolgen. MySQL selbst bringt zwei Werkzeuge mit: den MySQL Query Browser, welcher in der MySQL Workbench integriert ist, und den MySQL/MariaDB Client⁸.

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

```
1 adams@DebianSQL:~$ sudo docker exec -it my
```

```
sql_server01 bash
2  bash-4.4# mysql -uroot -p
3  Enter password:
4  Welcome to the MySQL monitor. Commands end
  with ; or \g.
5  Your MySQL connection id is 9
6  Server version: 8.0.33 MySQL Community Ser
ver - GPL
7  [...]
8  mysql>
```

In [Zeile 2](#) wird der Client aufgerufen – hier unter Linux im Docker-Container. Der Aufruf setzt voraus, dass die Datei `mysql` im Suchpfad liegt. Ist dies nicht der Fall, muss der Pfad – wie beispielsweise für eine Windows-XAMPP-Installation – mit angegeben werden⁹:

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

Die Aufrufparameter des Clients und was er noch so drauf hat, entnehmen Sie bitte dem Handbuch. Mit dem Parameter `-u` wird ein Benutzer angemeldet, hier `root`. Wird dieser Parameter nicht verwendet, wird der Benutzer `guest` angemeldet, der darf aber fast nichts. `-p` hat zur Folge, dass Sie nach dem Passwort gefragt werden. Die entsprechende Eingabezeile lasse ich aus Platzgründen zukünftig weg.

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

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

Die [Zeile 8](#) ist der eigentliche Eingabeprompt. Hier werden die

----- Eingabe prompt -----
Kommandos/Anweisungen eingegeben. Der Prompt `mysql>`¹⁰ zeigt an, dass er auf eine neue Anweisung wartet. Der Prompt `->` hingegen markiert, dass die Anweisung noch nicht abgeschlossen ist und weitere Eingaben zur laufenden Anweisung gemacht werden können.

Natürlich kann man den MySQL Client auch wieder verlassen:

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

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

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

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

5.2.2 Wie lege ich eine Datenbank an?



SQL:2023

```
CREATE SCHEMA datenbankname
[DEFAULT CHARACTER SET zeichensatz]
;
MySQL/MariaDB
CREATE {DATABASE|SCHEMA} [IF NOT EXISTS] d
atenbankname
[[DEFAULT] CHARACTER SET zeichensatz]
[[DEFAULT] COLLATE sortierung]
[DEFAULT ENCRYPTION {'Y' | 'N'}]
;
```

PostgreSQL

```
CREATE DATABASE datenbankname
[ENCODING zeichenkodierung]
[LC_COLLATE sortierung]
[LC_CTYPE ctype]
;
```

T-SQL

```
CREATE DATABASE datenbankname
[COLLATE sortierung]
;
```

In MySQL und MariaDB sind DATABASE und SCHEMA nur zwei Wörter für das Gleiche (siehe [Tabelle 2.1](#)). In SQL:2023 gibt es den Begriff *Datenbank* nicht, sondern nur das *Schema*.

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

inhaltliche Gruppen unterteilt und mit verschiedenen Zugriffsrechten versehen werden.

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

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

Anschließend folgt der Name der Datenbank, die in unserem Beispiel `oshop` heißen soll. Zum Zeichensatz und zur Sortierung kommen wir später¹¹.

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

Bitte beachten Sie, dass am Ende eines SQL-Befehls immer ein Semikolon steht!¹² Ohne dieses geht der MySQL Client davon aus, dass der Befehl noch nicht vollständig ist:

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

Sie sehen, wie der Befehl auf mehrere Zeilen verteilt wurde. Sie sollten sich dies angewöhnen, da der SQL-Quelltext dadurch erheblich leichter zu lesen und zu verstehen ist. Das

Gleiche gilt auch für die Einrückungen, die eine gewisse inhaltliche Gruppierung der Anweisung verdeutlichen.

Aber Achtung! In der letzten Zeile steht, dass es eine Warnung gegeben hat. Mit `SHOW WARNINGS`¹³ kannman sich dieWarnungen bezüglich des letzten Befehls anschauen:

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

DieWarnung bedeutet, dass es die Datenbank schon gibt und daher nicht angelegt wird.

5.2.3 Wie lösche ich eine Datenbank?



SQL:2023

`DROP SCHEMA`

`datenbankname`

`{CASCADE|RESTRICT}`

`;`

MySQL/MariaDB

`DROP {DATABASE|SCHEMA}`

`[IF EXISTS]`


```
datenbankname
;

PostgreSQL, T-SQL

DROP SCHEMA
[IF EXISTS]
datenbankname
;
```

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

In der SQL:2023-Version löscht die Angabe CASCADE alle in der Datenbank angesiedelten Objekte wie Tabellen, Ansichten etc. Die Angabe RESTRICT verweigert das Löschen der Datenbank, solange noch andere Objekte der Datenbank vorhanden sind.

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

Probieren wir das mal aus:

```
1  mysql> DROP DATABASE IF EXISTS oshop;
2  Query OK, 0 rows affected (0.29 sec)
3
4  mysql> DROP DATABASE oshop;
```

```
5 ERROR 1008 (HY000): Can't drop database 'o
shop'; database doesn't exist
```

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

5.2.4 Wie weise ich einen Zeichensatz zu?

Die Befehlsreferenz in [Abschnitt 5.2.2](#) gibt noch weitere Optionen an. Eine ist hier jetzt relevant: CHARACTER SET.

Was ist ein Zeichensatz? Auf dem Computer werden Buchstaben, Ziffern, Satz- und Sonderzeichen durch Zahlen kodiert. So ist beispielsweise der Buchstabe A im ASCII¹⁴ die Zahl 0x41 und a die Zahl 0x61. Da für die Kodierung des ASCII nur ein Byte (= 8 Bit) zur Verfügung steht, können nur 2⁸ = 256 verschiedene Zahlen zur Kodierung verwendet werden.

Die ersten 128 sind im Wesentlichen die Steuerzeichen (wie z.B. der Zeilenumbruch), das Leerzeichen, die lateinischen Buchstaben, die Ziffern 0–9, Satz- und einfache Sonderzeichen. Die restlichen 128 wurden mehr oder weniger willkürlich dazu verwendet, Umlaute oder andere sprachspezifische Sonderzeichen abzubilden.

Und hier fing das Unglück an. Fast jeder Computer- oder Betriebssystemhersteller hat da sein eigenes Süppchen gekocht. So ist beispielsweise das Zeichen Ü im Zeichensatz ISO/IEC 8859-1 mit der Zahl 0xDC kodiert und in Codepage 850 mit 0x9A.

Dieses Problem und die Beschränkung auf 256 Zeichen, was die Darstellung z.B. ostasiatischer Schriften unmöglich macht, haben dazu geführt, dass man eine neue, leicht erweiterbare Kodierung von Schriftzeichen baute. Unicode ward geboren!

Unicode selbst liegt in verschiedenen Formatierungen vor. Derzeit gerne verwendet werden UTF-8 und UTF-16. Mir ist kein Einsatz von UTF-32 bekannt.

Welche Zeichensätze von Ihrem Server unterstützt werden, können Sie leicht mit `SHOW CHARACTER SET` herausfinden. Bei mir waren es in MySQL 41 und in MariaDB 40!¹⁵ Möchten Sie die Ausgabe auf bestimmte Zeichensätze einschränken, geht das natürlich auch:

```
1  mysql> SHOW CHARACTER SET LIKE 'latin%';
2  +-----+-----+-----+-----+
-----+-----+
3  | Charset | Description | Default collatio
n | Maxlen |
4  +-----+-----+-----+-----+
-----+-----+
5  | latin1 | cp1252 West European | latin1_s
wedish_ci | 1 |
6  | latin2 | ISO 8859-2 Central European | l
atin2_general_ci | 1 |
7  | latin5 | ISO 8859-9 Turkish | latin5_tur
kish_ci | 1 |
8  | latin7 | ISO 8859-13 Baltic | latin7_gen
eral_ci | 1 |
9  +-----+-----+-----+-----+
-----+-----+
10 4 rows in set (0.00 sec)
11
12 mysql> SHOW CHARACTER SET LIKE 'utf%';
13 +-----+-----+-----+-----+
-----+-----+
14 | Charset | Description | Default collatio
n | Maxlen |
15 +-----+-----+-----+-----+
-----+-----+
16 | utf16 | UTF-16 Unicode | utf16_general_c
i | 4 |
```

```

17 | utf16le | UTF-16LE Unicode | utf16le_gen
   |         |                   | eral_ci | 4 |
18 | utf32 | UTF-32 Unicode | utf32_general_c
   |      |                   | i | 4 |
19 | utf8mb3 | UTF-8 Unicode | utf8_general_c
   |      |                   | i | 3 |
20 | utf8mb4 | UTF-8 Unicode | utf8mb4_0900_a
   |      |                   | i_ci | 4 |
21 +-----+-----+-----+-----+
   +-----+
22 5 rows in set (0.00 sec)

```

Für die meisten Anwendungen in Deutschland sind die Zeichensätze `latin1`, `cp850` und `utf8mb4` ausreichend. Welchen Zeichensatz Sie tatsächlich brauchen, ist genau zu untersuchen und hängt in der Regel von der Datenquelle ab.

Stammen Ihre Daten aus einer älteren Quelle und sind ggf. über ein `COMMAND`-Terminal eingegeben worden, ist `cp850` vermutlich richtig¹⁶. Neue Anwendungen sollten von Anfang an `utf8mb` oder `utf16` verwenden. Diese Kodierungen ist recht stabil bezüglich der Sonderzeichen und Umlaute und machen auch die Verwaltung von anderssprachlichen Zeichen einfach möglich.

Die Kodierung UTF-8 ist unter Unix bzw. Linux und in der Internetkommunikation weit verbreitet. Microsoft in seiner .NET-Umgebung, Tcl und Java verwenden UTF-16. Falls Sie also eine Anwendung bauen, die mit ADO/.NET arbeiten soll, ist die Verwendung von `utf16` sinnvoll.



Hinweis: Die Unterscheidung zwischen `utf8mb4` und `utf8` gibt es nur in MySQL und MariaDB. Sie sollten besser `utf8mb4` als Zeichenkodierung verwenden. Die UTF-8-Norm schreibt eigentlich eine maximale Kodierung in 4 Bytes vor. In obigem Listing in [Zeile 19](#)

steht aber, dass `utf8` nur 3 Bytes speichern kann. Diesem *Fehler* wurde mit der MySQL-Version 5.5.3 abgeholfen, indem man die Kodierung `utf8mb4` einführte.¹⁷

So sieht unser Anlegen einer Datenbank bisher aus:

```
1 DROP DATABASE IF EXISTS oshop;  
2 CREATE DATABASE oshop  
3 DEFAULT CHARACTER SET utf8mb4  
4 ;
```

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

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

Insgesamt stehen damit 5508 Collations mit 68 Varianten für German_zur Verfügung. Aber damit sind wir eigentlich schon beim nächsten Abschnitt angekommen.

5.2.5 Wie weise ich eine Sortierung zu?

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

Die verfügbaren Sortierungen lassen sich leicht mit SHOW COLLATION anzeigen. Bei meiner MySQL-Installation sind es 286 und bei MariaDB 506, was bei 41 bzw. 40 Zeichensätzen schon deutlich macht, dass es mehrere Sortierungen für einen Zeichensatz geben kann.

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

```

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

```

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

Für `utf8mb4` gibt es in MySQL 89 und in MariaDB 217 verschiedene Sortierungen.

- Für die *Wörterbuchsortierung* nach DIN 5007-1 die Collation `utf8mb4_unicode_ci` mit dem Nachteil, dass Umlaute tatsächlich nicht von ihren Substituten unterschieden werden. Konkret bedeutet das, eine Suche nach Müller liefert alle Müller und alle Muller – das mussman schon wollen.

Tabelle 5.1 Sortiernormen für die deutsche Sprache

Zeichen	DIN 5007-1 (Wörterbuch)	DIN 5007-2 (Telefonbuch)
Ä	A	AE
Ö	O	OE
Ü	U	UE
ß	s	SS

- Für die *Telefonbuchsortierung* nach DIN 5007-2 die Collations `utf8mb4_de_pb_0900_ai_ci`, `utf8mb4_de_pb_0900_as_cs` und `utf8mb4_german2_ci`, wobei pb für *phonebook* steht und 900 die entsprechende Codepage ist¹⁸:

```
1 mysql> SHOW COLLATION LIKE 'utf8mb4_de%';
```

```

2  +-----+-----+-----+
-+-----+-----+-----+
3  | Collation | Charset | Id | Default | Com
   | piled | Sortlen |
4  +-----+-----+-----+
-+-----+-----+-----+
5  | utf8mb4_de_pb_0900_ai_ci | utf8mb4 | 256
   | | Yes | 0 |
6  | utf8mb4_de_pb_0900_as_cs | utf8mb4 | 279
   | | Yes | 0 |
7  +-----+-----+-----+
-+-----+-----+-----+
8
9  mysql> SHOW COLLATION LIKE 'utf8mb4_german
%';
10 +-----+-----+-----+-----+
    +-----+-----+
11 | Collation | Charset | Id | Default | Com
   | piled | Sortlen |
12 +-----+-----+-----+-----+
    +-----+-----+
13 | utf8mb4_german2_ci | utf8mb4 | 244 | | Y
   | es | 8 |
14 +-----+-----+-----+-----+
    +-----+-----+

```



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

Aufgabe 5.2: Wofür steht die Codepage 900?

Das Anlegen der Datenbank kann jetzt vervollständigt werden:

```
1  DROP DATABASE IF EXISTS oshop;
```



```
2 CREATE DATABASE oshop
3 CHARACTER SET utf8mb4
4 COLLATE utf8mb4_german2_ci;
```

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

```
1 mysql> SHOW DATABASES;
2 +-----+
3 | Database |
4 +-----+
5 | information_schema |
6 | mysql |
7 | oshop |
8 | performance_schema |
9 | sys |
10 +-----+
11 5 rows in set (0.01 sec)
```



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

5.3 Anlegen der Tabellen



Die Tabelle als universeller Datencontainer

- Grundkurs

- Vermeiden von Tipparbeit mit USE
 - Anlegen einer Tabelle mit CREATE TABLE
 - Bedingtes Anlegen einer Tabelle mit IF EXISTS
 - Datentypen und Zusätze für Tabellenspalten
 - Festlegen des Primärschlüssels mit PRIMARY KEY
 - Anzeigen aller Tabellen einer Datenbank mit SHOW TABLES
 - Festlegen des Fremdschlüssels mit FOREIGN KEY
- Vertiefendes
- Festlegen der Constraints mit ON UPDATE und ON DELETE zur Wahrung der referenziellen Integrität
 - Entscheidung bzgl. ENUM und separater Tabelle
 - Performancebetrachtungen bzgl. DOUBLE und DECIMAL
 - Genauigkeitsbetrachtungen bzgl. DOUBLE und DECIMAL
 - Entscheidung bzgl. NOT NULL inkl. einer Performancebetrachtung
 - Die Drei-Schichten-Architektur und das SRP-Prinzip



Die Quelltexte dieses Kapitels stehen in der Datei `listing02.sql` (siehe [Listing 28.2](#), [Listing 28.42](#) und [Listing 28.27](#)).

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

5.3.1 Welche Datentypen gibt es?

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

In SQL gibt es auch Datentypen. Den Spalten wird beim Erstellen der Tabelle ein Datentyp zugewiesen. In [Abschnitt 26.1](#) sind auszugsweise die in MySQL möglichen Datentypen aufgelistet. Datentypen, die nicht SQL:2023-Standard sind, werden durch ein Sternchen (*) gekennzeichnet. Datentypen, die in MySQL nicht, aber in SQL:2023 vorkommen, werden hier nicht angegeben.

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



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

5.3.2 Wie lege ich eine Tabelle an?



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

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



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

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

1. **Spaltenname:** Der wird klein geschrieben, wird immer im Singular formuliert und ergibt sich aus dem ER-Modell.
2. **Datentyp:** Dieser legt fest, was für eine Art von Information in der Spalte verwaltet und wie diese kodiert wird. Mögliche Datentypen finden Sie in [Abschnitt 26.1](#).
3. **Zusätze:** Mit diesen kann man eine Spalte ausführlicher bestimmen. Eine Liste möglicher Zusätze finden Sie in [Abschnitt 26.1.8](#).

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

Versuchen wir nun, die Tabelle *adresse* (siehe [Bild 3.1](#) in [Abschnitt 3.1](#)) zu erstellen:

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

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

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



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

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

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

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



Hinweis: `AUTO_INCREMENT` ist kein SQL-Standard, sondern ein MySQL-/MariaDB-Zusatz. SQL:2023 verwendet den Zusatz `GENERATED art AS IDENTITY` (siehe [Abschnitt 26.1.8](#)).

In den [Zeilen 3](#) ff. werden diverse `VARCHAR(255)`-Spalten deklariert. Grundsätzlich gilt, dass bei einem `VARCHAR` nur so viel Speicherplatz verbraucht wird²¹, wie man Text in die

Spalte einfügt. Somit kann man ohne Speicherplatzverschwendung eine ausreichend großzügige obere Grenze angeben. Verwundern tut einen das schon, besonders bei Feldern wie der Hausnummer ([Zeile 4](#)). Aber aus der Perspektive des Platzverbrauchs ist es völlig egal, ob bei der Hausnummer VARCHAR(255) oder VARCHAR(10) steht, wenn Sie maximal zehnstellige Daten erwarten. Was aber, wenn aus irgendeinem Grund elf Stellen gebraucht werden? Bei VARCHAR(10) wird abgeschnitten, bei VARCHAR(255) nicht²².

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

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

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

```
1  mysql> SHOW TABLES;
2  +-----+
3  | Tables_in_oshop |
4  +-----+
```

```
5 | adresse |  
6 +-----+
```

Als zweite Tabelle wird die Tabelle kunde angelegt:

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



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

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

Nun sind die anderen schnell gemacht:

```
1 CREATE TABLE bank (  
2 bank_id CHAR(12),  
3 bankname VARCHAR(255),  
4 lkz CHAR(2),
```



```

5   deleted TINYINT UNSIGNED NOT NULL DEFAULT
6   0,
7   PRIMARY KEY(bank_id)
8 );
9 CREATE TABLE bankverbindung (
10  kunde_id INT UNSIGNED,
11  bankverbindung_nr INT UNSIGNED,
12  bank_id CHAR(12),
13  kontonummer CHAR(25),
14  iban CHAR(34),
15  deleted TINYINT UNSIGNED NOT NULL DEFAULT
16  0,
17  PRIMARY KEY(kunde_id, bankverbindung_nr)
18 );

```

In der [Zeile 2](#) wird ein Primärschlüssel deklariert, der keine AUTO_INCREMENT, ja noch nicht einmal ein INT ist. Die bank_id ist das, was man landläufig als Bankleitzahl bezeichnet. Der Inhalt dieses Feldes ist also vorgegeben. In bestimmten Ländern kann die Bankleitzahl auch Bindestriche, Leerzeichen oder sogar Buchstaben²³ enthalten. Somit muss ein Datentyp für Zeichen verwendet werden. Analoges gilt für kontonummer und iban.

Auch die bankverbindung_nr ist keine AUTO_INCREMENT, da die Nummerierung für jeden Kunden (jede Kundennummer) neu begonnen wird. Man nennt solche Konstruktionen *Nummernkreis*. In der [Zeile 16](#) wird ein zusammengesetzter Primärschlüssel deklariert (siehe [Definition 8](#) in [Abschnitt 2.1.3](#)).



Aufgabe 5.4: Bauen Sie die drei CREATE TABLE auf den SQL:2023-Standard um. Beachten Sie dabei auch die Datentypen.

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

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

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

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

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

In der [Zeile 4](#) wird die *Aufzählung* mit einem ENUM deklariert. Die Werte werden als Zeichenketten in einer Liste angegeben.

Wie bei anderen Datentypen kann man eine Vorbelegung mit DEFAULT festlegen.

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

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

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

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

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

Tabelle 5.2 Entscheidungshilfe Datentyp ENUM

Kriterium	ENUM	Tabelle

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

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



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

5.3.4 Wann ein DECIMAL und wann ein DOUBLE?

Sowohl DOUBLE [PRECISION] als auch DECIMAL sind dazu da, Zahlen mit Nachkommastellen zu verarbeiten (siehe [Tabelle 26.2](#) in [Abschnitt 26.1.1.2](#)).

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

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

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

Ich wollte mir diesen Effekt anhand einer einfachen Performancemessung anschaulich machen. Ich habe daher bei steigender Anzahl von Datensätzen einmal auf einem DOUBLE- und einmal auf einem DECIMAL-Feld eine Durchschnittsberechnung durchgeführt.

Das Ergebnis können Sie in [Bild 5.1](#) betrachten²⁶. Man erkennt, dass die Berechnung mit einem DOUBLE ab einer gewissen Anzahl zu verarbeitender Zahlen deutlich schneller ist als mit einem DECIMAL. Warum sollte man dann überhaupt DECIMAL verwenden?

Der wichtigste Vorteil von DECIMAL ist die Vermeidung von Rundungsfehlern. Obwohl es ein blödes Beispiel ist (siehe [Listing 27.2](#) in [Abschnitt 27.2](#)), die Summe von 1000 einzelnen 0.001-Werten überrascht mit:

1	+-----+-----+	
2	FLOAT DECIMAL	

```

3  +-----+-----+
4  | 1.00000000474974513 | 1.00000000000000000 |
5  +-----+-----+

```

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

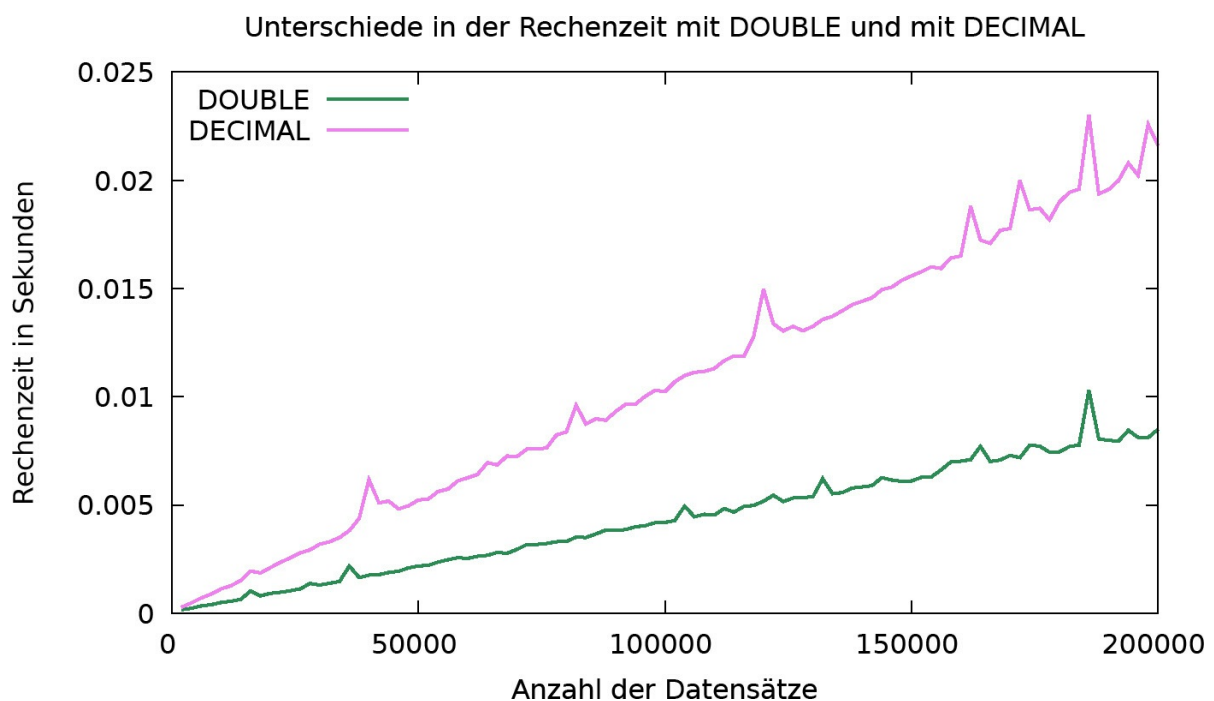


Bild 5.1 Unterschiedlicher Zeitverbrauch von Berechnungen bei DOUBLE und DECIMAL

Tabelle 5.3 Entscheidungshilfe DOUBLE oder DECIMAL

Kriterium	DOUBLE	DECIMAL
Breite des Wertebereichs	groß	klein
Schnelligkeit der Berechnung	wichtig	nicht so wichtig

Rundungsfehler	tolerierbar	nicht tolerierbar
kaufmännische Berechnung	trifft nicht zu	trifft zu
fiskalische Berechnung	trifft nicht zu	trifft zu

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



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

5.3.5 Wann verwende ich NOT NULL?

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

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

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

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

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

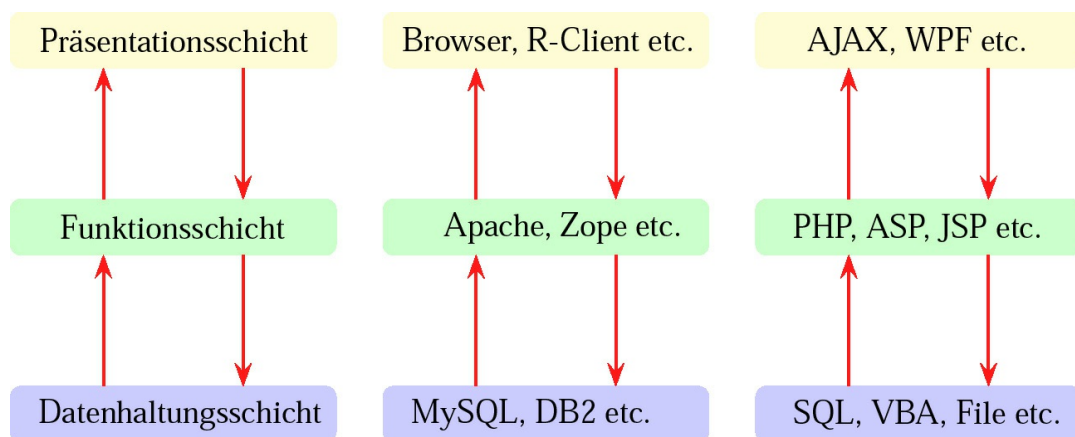


Bild 5.2 Drei-Schichten-Architektur

Jede Schicht sollte danach nur die Aufgaben bearbeiten, für die sie zuständig ist (*Single Responsibility Principle*). Da stellt sich nun die Frage, wer die Plausibilitätsprüfungen – und somit auch die Pflichtfeldprüfung – vornehmen muss. In trivialen Fällen sicherlich die Präsentationsschicht, ansonsten aber die Funktionsschicht. Aus der Perspektive der Datenhaltungsschicht ist die Eingabeplausibilisierung aber ein ALP: ein anderer Leute Problem.

Haben die oberen Schichten einen Datensatz geprüft und

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

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

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

Und jetzt kommt das große ABER. In [[SZT+09](#)] findet sich ein Hinweis darüber, dass NULL-Werte die Performance belasten. Das hat mir keine Ruhe gelassen, und ich habe mir dazu einen kleinen Versuch ausgedacht (Näheres siehe [Kapitel 27.3](#)). Das Ergebnis in [Bild 5.3](#) ist für mich überraschend gewesen.

Zwar lässt sich feststellen, dass die Suche nach bzw. der Vergleich mit leeren Inhalten wie dem Leer-String ' ' messbar schneller verläuft als die Suche nach bzw. der Vergleich mit NULL, aber sooo beeindruckend schneller auch wieder nicht^{[27](#)}.

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

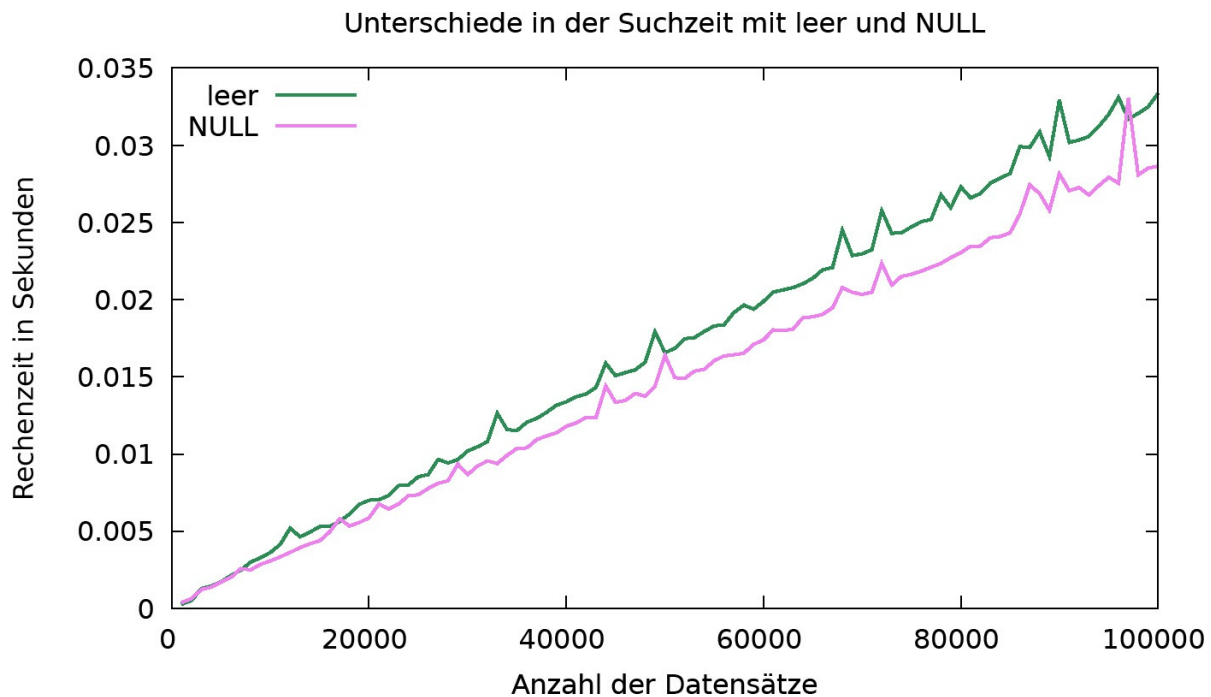


Bild 5.3 Performanceunterschied zwischen leer und NULL

Tabelle 5.4 Entscheidungshilfe NULL oder NOT NULL

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

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

Das Anlegen der Tabelle adresse hätte dann folgendes

Aussehen:

```
1 CREATE TABLE adresse (  
2     adresse_id INT UNSIGNED AUTO_INCREMENT,  
3     strasse VARCHAR(255) NOT NULL DEFAULT '',  
4     hnr VARCHAR(255) NOT NULL DEFAULT '',  
5     lkz CHAR(2) NOT NULL DEFAULT '',  
6     plz CHAR(9) NOT NULL DEFAULT '',  
7     ort VARCHAR(255) NOT NULL DEFAULT '',  
8     deleted TINYINT UNSIGNED NOT NULL DEFAULT  
9     0,  
10    PRIMARY KEY(adresse_id)  
11 );
```

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

5.3.6 Wie lege ich einen Fremdschlüssel fest?

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

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

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

```
1 CREATE TABLE kunde (  
2     kunde_id INT UNSIGNED AUTO_INCREMENT,  
3     nachname VARCHAR(255) NOT NULL DEFAULT ''
```

```

,
4  vorname VARCHAR(255) NOT NULL DEFAULT '',
5  rechnung_adresse_id INT UNSIGNED,
6  liefer_adresse_id INT UNSIGNED,
7  bezahlart INT UNSIGNED NOT NULL DEFAULT 0
,
8  art ENUM('unb', 'prv', 'gsch') NOT NULL D
EFAULT 'unb',
9  deleted TINYINT UNSIGNED NOT NULL DEFAULT
0,
10 PRIMARY KEY(kunde_id),
11 FOREIGN KEY(rechnung_adresse_id) REFEREN
CES adresse(adresse_id),
12 FOREIGN KEY(liefer_adresse_id) REFERENCE
S adresse(adresse_id)
13 );

```

In [Zeile 11](#) wird mit dem Schlüsselwort `FOREIGN KEY` ein Fremdschlüssel benannt. In Klammern steht der Spaltenname des Fremdschlüssels. Nach dem Schlüsselwort `REFERENCES` wird die Tabelle angegeben, auf die der Fremdschlüssel zeigt; in den runden Klammern steht der Name des Primärschlüssels, der die Werte für den Fremdschlüssel liefert. Bei diesem Beispiel werden sie als Constraints²⁸ der Tabellen deklariert.

Obwohl die [Zeilen 11ff](#) in SQL:2023 genauso funktionieren würden, hier die gleiche Tabelle nun in SQL:2023, aber durch Zusätze bei der Spaltendeklaration.

```

1  CREATE TABLE kunde (
2  kunde_id INT GENERATED BY DEFAULT AS IDENT
ITY,
3  [...]
4  rechnung_adresse_id INT REFERENCES adresse
(adresse_id),
5  liefer_adresse_id INT REFERENCES adresse(a

```

```
    adresse_id),  
    6    [...]  
    7    PRIMARY KEY(kunde_id)  
    8 );
```

Bei den Deklarationen der Spalten `rechnung_adresse_id` und `liefer_adresse_id` wird durch den Zusatz `REFERENCES` ([Zeile 4ff](#)) der Fremdschlüssel festgelegt. In [Zeile 2](#) wird der Primärschlüssel deklariert. Sie sehen hier ein Beispiel für die Verwendung von `GENERATED`.

In MySQL/MariaDB wird es jetzt kompliziert. Wir müssen uns nun mit einer ihrer Besonderheiten beschäftigen: den *Engines*. Der MySQL oder MariaDB Server ist *eigentlich* nur so etwas wie eine Arbeitsumgebung. Er stellt Funktionen zum Verbindungsaufbau oder der Administration zur Verfügung. Er überprüft empfangene SQL-Befehle auf ihre Richtigkeit und baut die Befehle in ein standardisiertes Format um. Das Ergebnis eines Befehls wird auch vom Server verwaltet an den Client gesendet. Aber das Dazwischen, das Ausführen der Befehle, das machen die Engines^{[29](#)}.

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

In [Tabelle 24.3](#) finden Sie einen Auszug der Engines, die Teil der aktuellen MySQL- oder MariaDB-Distribution sind. Für eine genauere Beschreibung der Engines empfehle ich dringend, die entsprechenden Handbuchkapitel zu lesen^{[30](#)}.

Darüber hinaus gibt es viele Engines von kommerziellen Anbietern oder für spezielle Anforderungen. So ist beispielsweise mit der Engine *Brighthouse* das Verwalten von Datawarehouse-Objekten^{[31](#)} besonders gut möglich.

Da wir beim CREATE TABLE in MySQL oder MariaDB keine Engine angegeben haben, wird die InnoDB verwendet. Lassen wir uns das durch MySQL bzw. MariaDB mithilfe von SHOW CREATE TABLE bestätigen (siehe [Zeile 15](#)):

```
1 Create Table: CREATE TABLE 'kunde' (  
2   'kunde_id' int unsigned NOT NULL AUTO_INCREMENT,  
3   'nachname' varchar(255) COLLATE utf8mb4_german2_ci NOT NULL DEFAULT '',  
4   'vorname' varchar(255) COLLATE utf8mb4_german2_ci NOT NULL DEFAULT '',  
5   'rechnung_adresse_id' int unsigned DEFAULT NULL,  
6   'liefer_adresse_id' int unsigned DEFAULT NULL,  
7   'bezahlart' int unsigned NOT NULL DEFAULT '0',  
8   'art' enum('unb','prv','gsch') COLLATE utf8mb4_german2_ci NOT NULL DEFAULT 'unb',  
9   'deleted' tinyint unsigned NOT NULL DEFAULT '0',  
10  PRIMARY KEY ('kunde_id'),  
11  KEY 'rechnung_adresse_id' ('rechnung_adresse_id'),  
12  KEY 'liefer_adresse_id' ('liefer_adresse_id'),  
13  CONSTRAINT 'kunde_ibfk_1' FOREIGN KEY ('rechnung_adresse_id') REFERENCES 'adresse' ('adresse_id'),  
14  CONSTRAINT 'kunde_ibfk_2' FOREIGN KEY ('liefer_adresse_id') REFERENCES 'adresse' ('adresse_id')  
15 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_german2_ci
```

Warum war dieser Vortrag über Engines nötig? Weil die

Unterstützung von Fremdschlüsseln sehr stark von der Engine abhängt. Bei MyISAM beispielsweise läuft es letztlich darauf hinaus, dass der Zusatz FOREIGN KEY zur Kenntnis genommen wird, aber keine praktische Konsequenz daraus folgt. Ganz anders bei der InnoDB.

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

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

In der Ausgabe des SHOW CREATE TABLE oben steht vor dem FOREIGN KEY in [Zeile 13](#) auf einmal das Schlüsselwort CONSTRAINT. Hier wird eine Randbedingung festgelegt, die von der InnoDB-Engine überwacht wird.

Die Überwachung bemerkt, ob sich der Wert oder das Vorhandensein des Primärschlüsselwerts, auf welchen der Fremdschlüssel verweist, verändert. Der Primärschlüsselwert kann durch DELETE^{[32](#)} verschwinden oder durch ein UPDATE^{[33](#)} verändert werden. Aber wie soll darauf reagiert werden? Ziel ist der Erhalt der referenziellen Integrität (siehe [Definition 22](#) in [Abschnitt 2.3](#)).

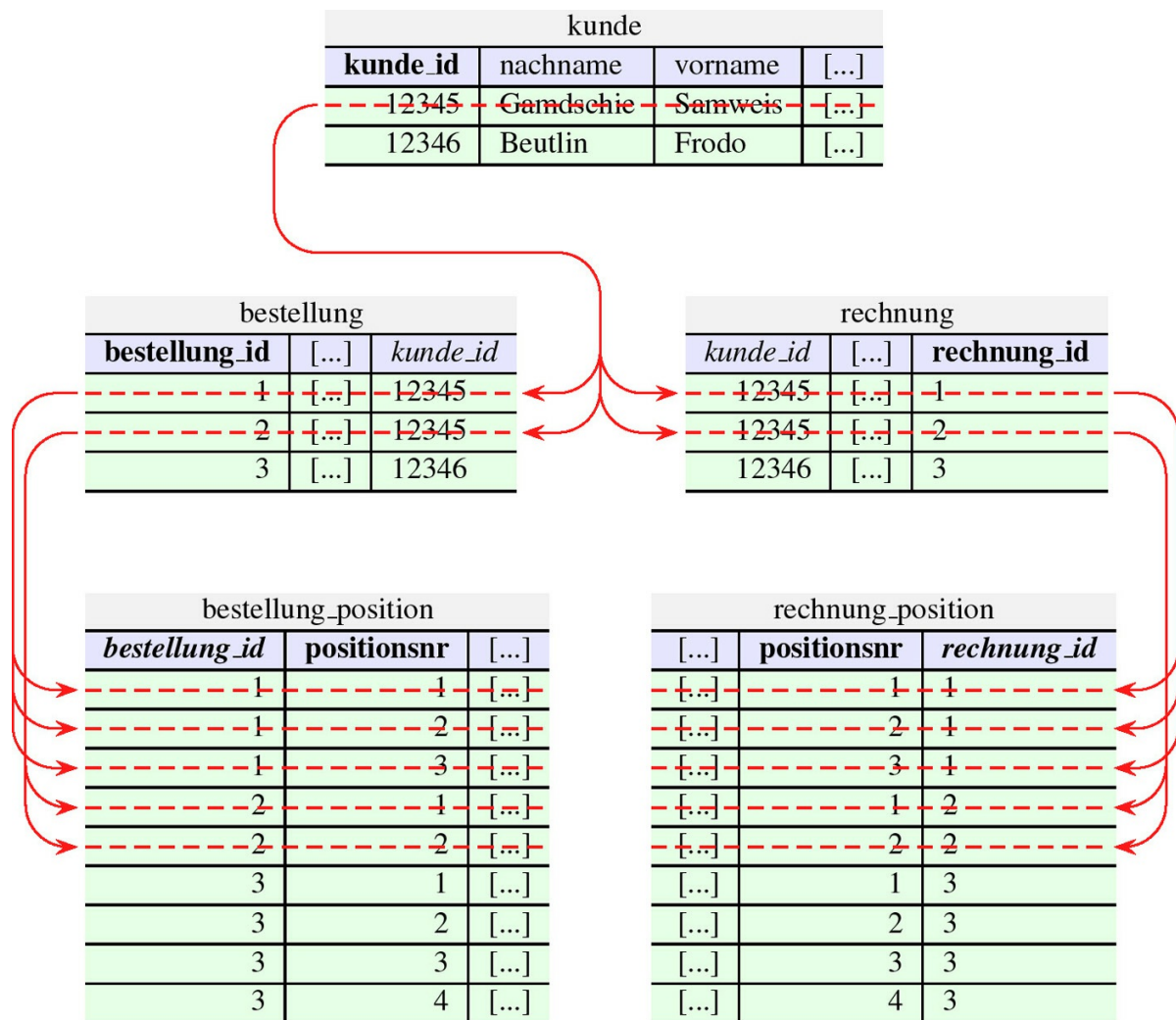


Bild 5.4 Löschweitergabe am Beispiel kunde

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



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

Beispiel: Betrachten wir das Löschen eines Kunden (siehe [Bild 5.4](#)). Sie löschen einen Kunden. Dann werden alle Bestellungen, die sich auf den Kunden beziehen, gelöscht. Es werden alle Buchungen im Rechnungswesen zu dem Kunden gelöscht. Es wäre so, als hätte es diesen Kunden niemals gegeben! Das widerspricht natürlich nicht nur den gesetzlichen Aufbewahrungspflichten eines Unternehmens, sondern kann auch nicht im Sinne des Anwenders sein. Ausweg ist die schon angesprochene Spalte `deleted` (siehe [Abschnitt 2.3.1](#)).



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

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

In [Abschnitt 5.3.2](#) haben wir eine erste Version von CREATE TABLE kennengelernt. Diese muss nun erweitert werden:



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

```

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

```

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

```

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

```

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

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

Anders bei der Lieferadresse ([Zeile 18](#)). Wenn die Lieferadresse gelöscht wird, wird von der Programmlogik die Rechnungsadresse verwendet. In Folge kann der Fremdschlüsselwert auf NULL gesetzt werden. Beachten Sie bitte dazu die [Zeile 6](#); es wird kein NOT NULL verwendet³⁴.

In T-SQL funktioniert dieser Quelltext nicht! Neben Gründen, die sich aus dem T-SQL-Dialekt ergeben – beispielsweise kein UNSIGNED, IDENTITY anstelle von AUTO_INCREMENT etc. –, akzeptiert er die beiden Fremdschlüssel-Constraints nicht in der Form: *Introducing FOREIGN KEY constraint 'FK_kunde__liefer_ad__0F624AF8' on table 'kunde' may cause cycles or multiple cascade paths. Specify ONDELETENOACTION or ONUPDATENOACTION, or modify other FOREIGN KEY constraints.*

Hintergrund ist folgendes Problem: Durch kaskadierendes Löschen oder Ändern können zyklische oder mehrfache sich widersprechende Änderungen auf einem Datensatz ausgelöst werden. So könnte einmal eine Änderung und einmal eine Löschung auf demselben Datensatz innerhalb der gleichen Transaktion ausgelöst werden. Viele Datenbankprodukte ignorieren das Problem oder stecken viel Analyseaufwand hinein. Der MS SQL Server begegnet dem dadurch, dass er schlichterweise zählt, wie viele sich widersprechende

Kaskaden auf einem Datensatz definiert werden. Bei mehr als einem wirft er die obige Fehlermeldung aus³⁵. Aufgrund dieser Einschränkung des Systems ist der T-SQL-Quelltext entsprechend abgeändert.



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



Aufgabe 5.7: Erstellen Sie die CREATE TABLE für SQL:2023.

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

Modus	Bedeutung
RESTRICT	Alle Lösch- und Änderungsoperationen auf den Primärschlüsselwert werden zurückgewiesen, wenn es einen dazugehörenden Fremdschlüsselwert gibt. Bei der InnoDB ist dies funktionsgleich mit NO ACTION.
CASCADE	Bei einer Löschoperation werden alle Zeilen in der Fremdschlüsseltabelle gelöscht, die auf den gelöschten Primärschlüsselwert verweisen. Bei einer Änderung des Primärschlüsselwerts werden die Fremdschlüsselwerte ebenfalls geändert.
SET NULL	Der Fremdschlüsselwert wird auf NULL gesetzt. Dies setzt natürlich voraus, dass der

	Fremdschlüssel nicht mit NOT NULL deklariert ist.
SET DEFAULT	Der Fremdschlüsselwert wird auf die serverseitig eingestellte Vorbelegung gesetzt. InnoDB akzeptiert, aber ignoriert diese Angabe.
NO ACTION	Alle Lösch- und Änderungsoperationen auf den Primärschlüsselwert werden zurückgewiesen, wenn es einen dazugehörenden Fremdschlüsselwert gibt. Bei der InnoDB ist dies funktionsgleich mit RESTRICT.

5.3.7 Wie kann ich Tabellen aus anderen herleiten?

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

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



MySQL/MariaDB

```
CREATE TABLE [datenbankname.] tabellennam
e_neu
[() LIKE [datenbankname.] tabellennamalt
()]
;
```

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

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

Der neue Tabellename muss nicht der gleiche wie der alte sein. Sie hätten der neuen Tabelle auch den Namen wurstbrot geben können. Bitte achten Sie dann darauf, dass der Name des Primärschlüssels nicht mehr passt (siehe [Zeile 15](#) unten). Entweder Sie benennen den Primärschlüssel um oder Sie nehmen das in Kauf.

```
1  mysql> CREATE TABLE wurstbrot LIKE adresse
;
2  Query OK, 0 rows affected (0.03 sec)
3
4  mysql> SHOW CREATE TABLE wurstbrot\G
5  ***** 1. row *****
*****
6          Table: wurstbrot
7  Create Table: CREATE TABLE 'wurstbrot' (
8      'adresse_id' int unsigned NOT NULL AUTO_INCREMENT,
9      'strasse' varchar(255) COLLATE utf8mb4_german2_ci NOT NULL DEFAULT '',
10     'hnr' varchar(255) COLLATE utf8mb4_german2_ci NOT NULL DEFAULT '',
11     'lkz' char(2) COLLATE utf8mb4_german2_ci NOT NULL DEFAULT '',
12     'plz' char(9) COLLATE utf8mb4_german2_ci NOT NULL DEFAULT '',
13     'ort' varchar(255) COLLATE utf8mb4_german2_ci NOT NULL DEFAULT '',
```

```
14      'deleted' tinyint unsigned NOT NULL DE  
FAULT '0',  
15      PRIMARY KEY ('adresse_id')  
16 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 CO  
LLATE=utf8mb4_german2_ci
```

Weitergehend und letztlich mächtiger ist das Konzept der Vererbung von Tabellen. In SQL:2023 kann man angeben, dass eine Tabelle die Spezifikation einer anderen Tabelle beinhaltet. SQL:2023 verwendet dazu das Schlüsselwort UNDER:



SQL:2023

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

Die Tabellenstruktur ist in SQL:2023 somit keine Kopie der *elterntabelle*, sondern eine Spezialisierung. Die Elterntabelle wird dabei um die eigenen Spalten und Constraints erweitert, also spezialisiert.

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



PostgreSQL

```
CREATE TABLE tabellenname  
(
```

```
tabellenspezifikation
) INHERITS (elterntabelle[, elterntabelle]*)
;
```

Die bei der Vererbung entstehenden Probleme wie Namensgleichheit in der Eltern- und Kindtabelle werden in den SQL:2023-Implementationen unterschiedlich gelöst. Bitte schauen Sie in das Handbuch.

5.3.8 Ich brauche mal eben kurz 'ne Tabelle!

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

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



SQL:2023

```
CREATE [GLOBAL|LOCAL] TEMPORARY TABLE ta
bellename
tabellenspezifikation
;
```

MySQL/MariaDB, PostgreSQL

```
CREATE TEMPORARY TABLE tabellename
```



```
tabellenspezifikation  
;  
T-SQL  
CREATE TABLE #tabellenname  
tabellenspezifikation  
;
```

Die dabei angelegte Tabelle ist an eine bestimmte Existenzbedingung geknüpft. In MySQL und MariaDB ist diese Existenzbedingung die Verbindung³⁶ oder *Session*. Wird die Verbindung geschlossen beziehungsweise die Session beendet, wird automatisch die temporäre Tabelle entfernt. Sie ist auch nur in dieser Verbindung sichtbar, sodass mehrere Verbindungen temporäre Tabellen gleichen Namens anlegen können.



Hinweis: Bitte beachten Sie bei T-SQL das #-Zeichen vor dem Tabellennamen. Durch dieses Zeichen, wird die Tabelle temporär.

In SQL:2023 können Sie ebenfalls eine temporäre Tabelle anlegen, aber dort können Sie die Sichtbarkeit der Tabelle genauer festlegen. Wird die Option LOCAL verwendet, ist das Verhalten so, wie gerade bei MySQL/MariaDB beschrieben. Bei GLOBAL können auch andere Verbindungen (Sessions) die temporäre Tabelle verwenden.



Aufgabe 5.8: Schauen Sie sich die Liste der Tabellen in oshop genau an. Was fällt hier auf?

```
mysql> CREATE TEMPORARY TABLE tmp_kunde
kunde;
Query OK, 0 rows affected (0.04 sec)
mysql> SHOW TABLES;
+-----+
| Tables_in_oshop |
+-----+
| adresse |
| artikel |
| artikel_nm_lieferant |
| artikel_nm_warengruppe |
| bank |
| bankverbindung |
| bestellung |
| bestellung_position |
| kunde |
| lieferant |
| rechnung |
| rechnung_position |
| warengruppe |
+-----+
```

¹ Aussprache: EsKjuEl; die Aussprache SiQwL ist falsch, da damit der SQL-Vorläufer SEQUEL gemeint ist.

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

³ Abkürzung für: American National Standards Institute; US-amerikanisches Institut zur Normierung, vergleichbar mit dem deutschen DIN.

⁴ Abkürzung für: International Organization for Standardization.

⁵ Siehe [[Wik23t](#)].

⁶ Als Quelle habe ich [[Whe19](#)] verwendet. Diese referenziert zwar nur SQL:2016, aber ich gehe naiverweise davon aus, dass SQL:2023 bzgl. unserer Inhalte abwärtskompatibel ist.

⁷ Transact-SQL (T-SQL) ist der Name des SQL-Dialektes des Microsoft SQL Servers.

⁸ MariaDB bringt ebenfalls einen bis auf Kleinigkeiten kompatiblen Client mit.

⁹ Lassen Sie sich nicht irritieren: Obwohl Sie die `mysql.exe` aufgerufen haben, wird bei XAMPP ein MariaDB Client gestartet. Im Docker-Container müssen Sie allerdings `mariadb` aufrufen, da der Client ab der Version 11 umbenannt wurde.

¹⁰ Der MariaDB Client meldet sich mit dem Prompt `MariaDB [(none)]>`. In den eckigen Klammern steht die gerade verwendete Datenbank. Da wir noch keine ausgewählt haben, eben `(none)`. Später steht dort beispielsweise `oshop`. Der MySQL Client kennt dieses Feature nicht und ich persönlich bin mir auch nicht sicher, ob ich es mag.

¹¹ Die Verschlüsselung überlasse ich dem interessierten Leser.

¹² Dies ist nicht nötig, wenn Sie den SQL-Befehl aus einer Anwendung heraus auf den Server ausführen wollen. Die MySQL-APIs von PHP, Python, C# etc. hängen das Semikolon immer an.

¹³ Die ganzen `SHOW`-Befehle sind nicht Teil des SQL:2023-Standards. Sie werden deshalb hier syntaktisch nicht näher erläutert.

¹⁴ Siehe [[Ass76](#)].

¹⁵ MySQL hat den Zeichensatz `gb18030`, *China National Standard GB18030*, noch zusätzlich.

¹⁶ Denken Sie lieber darüber nach, die Daten mit einem Tool wie `iconv` in etwas Moderneres zu konvertieren.

¹⁷ BTW: `utf8` ist nur ein Alias auf `utf8mb3`.

¹⁸ Die Spalte `Pad_attribute` habe ich aus Platzgründen weggelassen.

¹⁹ Selbst bei nicht typisierten Sprachen werden intern Datentypen vergeben.

²⁰ lat.: *proprietas* = Eigentümlichkeit

²¹ +2 Bytes zur Speicherung der Textlänge.

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

²³ Z.B. das 'X' für die Prüfziffer 10.

²⁴ PostgreSQL braucht dazu einen Datentyp: `CREATE TYPE kundenart AS ENUM('unb', 'prv', 'gsch');`.

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

²⁶ Mehr zum Versuch: siehe [Kapitel 27.1](#).

²⁷ Falls jemand einen signifikanten Fehler im Versuchsaufbau findet, lasse er es mich wissen. Für einen Versuchsaufbau, der den Effekt deutlicher ausfallen lässt, wäre ich sehr dankbar ;-)

²⁸ engl.: Randbedingung

²⁹ Die Abgrenzung ist nicht ganz so einfach, aber das soll uns hier und jetzt nicht stören.

³⁰ Übersicht finden Sie für MySQL hier [[MyS21c](#)] und für MariaDB hier [[Mar23c](#)].

³¹ Siehe [[Wik23e](#)].

³² Siehe [Abschnitt 9.3](#).

³³ Siehe [Abschnitt 9.2](#).

³⁴ Ich verletze hier bewusst die Drei-Schichten-Architektur (siehe [Bild 5.2](#) in [Abschnitt 5.3.5](#)), da ich die Möglichkeiten der Sprache aufzeigen möchte.

³⁵ Eine schöne Diskussion darüber finden Sie unter [[Sta18](#)].

³⁶ Erinnern Sie sich noch an die Verbindungsnummer, die im MySQL/MariaDB Client angezeigt wird?

6 Indizes anlegen

6.1 Index für Anfänger



Indizes vernünftig eingesetzt beschleunigen viele der Datenbankzugriffe enorm.

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



Die Quelltexte dieses Kapitels stehen in der Datei `listing01.sql` (siehe [Listing 28.3](#), [Listing 28.43](#) und [Listing 28.28](#)).

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

Zunächst möchte ich hier eine kleine Performancemessung¹ vorstellen, die den Effekt von Indizes verdeutlichen soll. Die Darstellung in [Bild 6.1](#) zeigt, dass die Such- und

Sortierdauer mit Index nahezu konstant² bleibt. Die Such- und Sortierdauer ohne Index steigt hingegen linear an. Aus der Grafik ergibt sich unmittelbar, dass ein Suchen und Sortieren mithilfe von Indizes erheblich schneller abläuft als ohne.

Interessant ist, dass es für die ersten 50 bis 75 Zeilen umgekehrt ist (siehe [Bild 6.2](#)). Hier scheint der Verwaltungsaufwand für Indizes den Nutzen noch zu überlagern. Entscheidend ist aber der erhebliche Performancegewinn mit einem Index.

Wie kommt dieser Effekt zustande? Der *normale* Index organisiert die Daten einer Tabelle in Form von B-Bäumen.³ Letztlich nutzt man aus, dass die Daten in einer Sortierung vorliegen und nur mit sehr wenigen Zugriffen garantiert, dass das zu suchende Element gefunden wird.

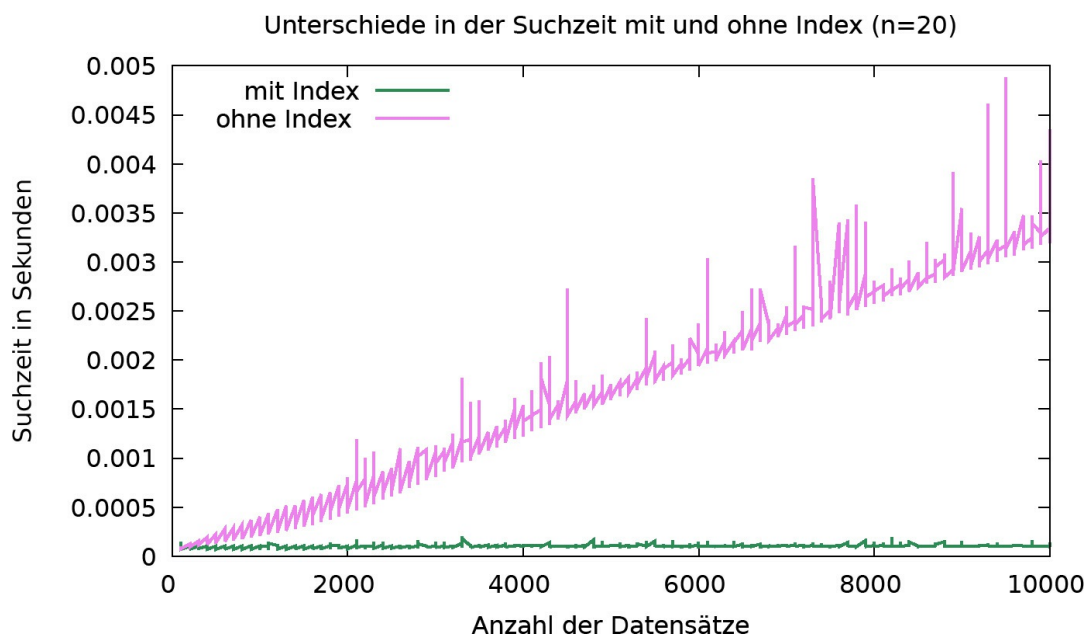


Bild 6.1 Zeitdauer von Suchoperationen mit und ohne Index

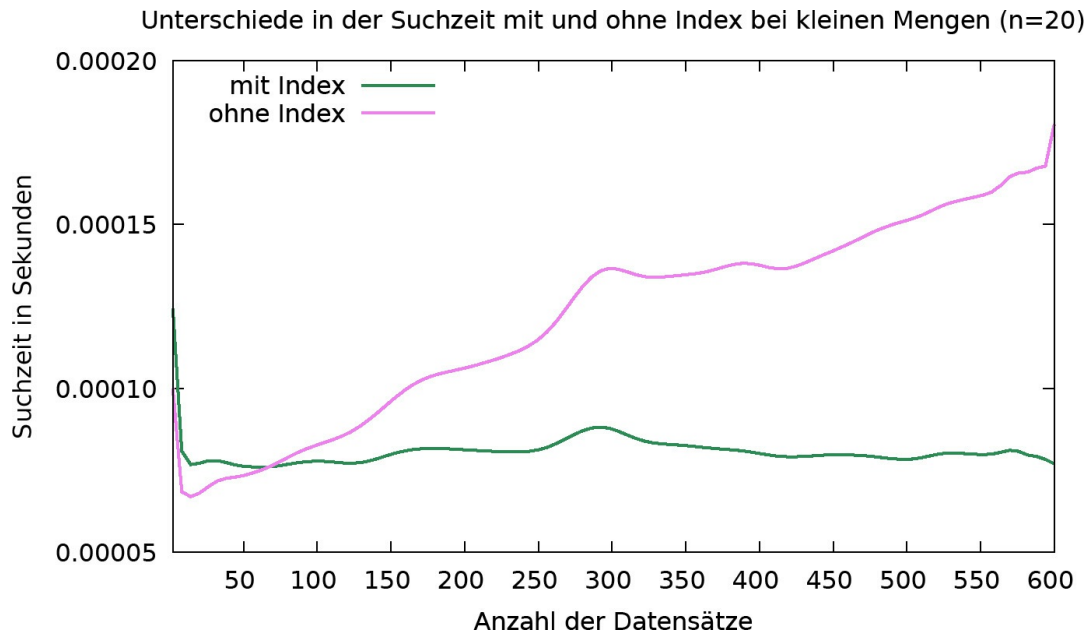


Bild 6.2 Zeitdauer von Suchoperationen mit und ohne Index bei kleinen Mengen (geglättet)

6.1.1 Wann wird ein Index automatisch erstellt?

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

Und weil dem so ist, wird automatisch mit der Deklaration zum Primär- und Fremdschlüssel ein Index angelegt.⁴ Hilfreich ist hier der Befehl `SHOW INDEX`:

```

1  mysql> SHOW INDEX FROM kunde\G
2  ***** 1. row *****
*****
3      Table: kunde
4      Non_unique: 0
5      Key_name: PRIMARY
6      Seq_in_index: 1
7      Column_name: kunde_id

```



```

8          Collation: A
9          Cardinality: 0
10         Sub_part: NULL
11         Packed: NULL
12         Null:
13         Index_type: BTREE
14         Comment:
15 Index_comment:
16         Visible: YES
17         Expression: NULL
18 ***** 2. row *****
19         Table: kunde
20         Non_unique: 1
21         Key_name: rechnung_adresse_id
22 Seq_in_index: 1
23         Column_name: rechnung_adresse_id
24         Collation: A
25         Cardinality: 0
26         Sub_part: NULL
27         Packed: NULL
28         Null: YES
29         Index_type: BTREE
30         Comment:
31 Index_comment:
32         Visible: YES
33         Expression: NULL
34 ***** 3. row *****
35         Table: kunde
36         Non_unique: 1
37         Key_name: liefer_adresse_id
38 [...]

```

Die Bedeutungen der Angaben sind:

- Table: Name der Tabelle, aus der der Index gebildet wird.
- Non_unique: Gibt an, ob die Werte des Index einmalig

sind. Dies ist bei allen Schlüsseln der Fall (siehe [Definition 8](#) in [Abschnitt 2.1.3](#)). Der Wert 0 steht dafür, dass keine Dubletten vorkommen dürfen, der Wert 1 eben doch. Deshalb wird bei PRIMARY KEY-Indizes hier immer eine 0 stehen.

- **Key_name:** Name des Index. Dieser kann frei vergeben werden, wenn man den Index per CREATE INDEX anlegt. Bei Primärschlüsseln heißt er automatisch PRIMARY, weshalb dieser Name nicht manuell vergeben werden kann.
- **Seq_in_index:** Hier wird angegeben, an welcher Stelle die Spalte Column_name im Index steht. Bei Indizes, die aus mehreren Spalten zusammengesetzt sind (z.B. bei bankverbindung), gibt es auch eine Spalte mit der Sequenznummer 2.
- **Column_name:** Ein Index wird entweder durch einen Spaltennamen oder einen Ausdruck (Expression) festgelegt. Falls er durch einen Spaltennamen festgelegt wird, steht dieser hier, ansonsten NULL.
- **Collation:** Da ein Index die Daten in einer Sortierung vorhalten soll, kann diese angegeben werden. Bei MySQL können hier die Werte A für *aufsteigend*, D für *absteigend* oder NULL für *nicht sortiert* auftreten.
- **Cardinality:** Je mehr eindeutige Werte in einem Index vorkommen, desto sinnvoller ist es, danach zu suchen. Der Wert, der hier angegeben wird, schätzt die Anzahl der eindeutigen Werte. Der Wert muss somit nicht exakt sein. Je höher der Wert im Verhältnis zur Gesamtanzahl der Datensätze ist, desto höher ist die Wahrscheinlichkeit, dass für Suchoperationen der Index verwendet wird.
- **Sub_part:** Wird ein Index manuell erstellt, kann man

angeben, dass er beispielsweise nur auf den ersten zehn Zeichen eines Nachnamens basieren soll. Ist dies nicht der Fall, steht hier NULL, ansonsten wird hier eine Zahl angegeben.

- **Packed:** Ein Schlüssel kann gepackt (komprimiert) werden. Steht hier NULL, ist er nicht komprimiert.
- **NULL:** Ist die Spalte nicht mit NOT NULL deklariert, steht hier YES, ansonsten nichts oder NO.
- **Index_type:** Es gibt verschiedene Typen von Indizes. Bei MySQL sind derzeit folgende möglich: BTREE, FULLTEXT, HASH und RTREE.
- **Comment:** Dieser Kommentar wird vom System gefüllt. Wird ein Index beispielsweise deaktiviert, steht hier disabled. Dies ist also nicht der Kommentar, der beim CREATE INDEX mitgegeben wird.
- **Index_comment:** Kommentar, der bei der Erstellung dem Index mitgegeben wird. Er kann maximal 1024 Zeichen lang sein.
- **Visible:** Legt fest, ob der Index dem Optimierer bekannt gemacht wird oder nicht.
- **Expression:** Ein Index wird entweder durch einen Spaltennamen (Column_name) oder einem Ausdruck festgelegt. Falls er durch einen Ausdruck festgelegt wird, steht dieser hier, ansonsten NULL.



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

6.1.2 Wie kann ich einen Index manuell erstellen?



MySQL/MariaDB, PostgreSQL, T-SQL

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



Hinweis: In SQL:2023 gibt es keinen Befehl zum Anlegen oder Löschen von Indizes. In vielen DBMSen, wie beispielsweise PostgreSQL und MS SQL Server, können Indizes aber mit der hier vorgestellten Syntax angelegt werden.

Zunächst stellt sich aber die Frage, wann man einen manuell erstellten Index braucht. Grundsätzlich dann, wenn man Such- und Sortieroperationen beschleunigen will⁵.

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

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

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

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

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

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

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

```

1  mysql> SHOW INDEX FROM kunde\G
2  ***** 1. row *****
*****
3  [...]
4  ***** 2. row *****
*****
5  [...]
6  ***** 3. row *****
*****
7  [...]
8  ***** 4. row *****
*****
9          Table: kunde
10         Non_unique: 1
11         Key_name: idx_kunde_nachname_vorname
12  Seq_in_index: 1
13         Column_name: nachname
14         Collation: A
15         Cardinality: 0
16         Sub_part: NULL
17         Packed: NULL
18         Null:
19  Index_type: BTREE
20  [...]
21  ***** 5. row *****
*****
22         Table: kunde
23         Non_unique: 1
24         Key_name: idx_kunde_nachname_vorname
25  Seq_in_index: 2
26         Column_name: vorname
27         Collation: A
28         Cardinality: 0
29         Sub_part: NULL
30         Packed: NULL
31         Null:
32  Index_type: BTREE
33  [...]

```

Der Teil 1. row bis 3. row ist uns schon aus obiger Betrachtung bekannt und hat sich nicht geändert. Es sind zwei weitere Einträge hinzugekommen, für jede Indexspalte eine. Sie können auch gut erkennen, dass die Angabe Seq_in_index in der [Zeile 12](#) für die erste Indexspalte 1 und in [Zeile 25](#) für die zweite Indexspalte 2 ist.

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

```
1 CREATE INDEX idx_bank_bankname
2     ON bank (bankname);
3
4 CREATE INDEX idx_bankverbindung_bankid_kon
tonummer
5     ON bankverbindung (bank_id, kontonumme
r);
6
7 CREATE INDEX idx_bankverbindung_iban
8     ON bankverbindung (iban);
```



Aufgabe 6.2: Erstellen Sie die verbleibenden Indizes zu den Tabellen laut der Auflistung in [Abschnitt 6.1.2](#).

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

6.2 Und jetzt etwas genauer



Es lohnt sich, Indizes genauer zu betrachten:

- Vertiefendes

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

Der Befehl CREATE INDEX kann ein bisschen mehr als das, was in der Version in [Abschnitt 6.1.2](#) angegeben wurde:



MySQL/MariaDB, PostgreSQL, T-SQL

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

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

6.2.1 Wie kann ich die Schlüsseleigenschaft erzwingen?

Nach [Definition 8](#) in [Abschnitt 2.1.3](#) müssen bei Schlüsseln Spaltenwerte eindeutig sein. Bei Primärschlüsseln wird dies durch PRIMARY KEY automatisch vom Server kontrolliert. Aber was ist mit anderen Schlüsselkandidaten?

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

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

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

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



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

6.2.2 Wie kann ich Dubletten verhindern?

Was ist eine Dublette? Schnell eine Definition dazu:



Definition 31: Dublette

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

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

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

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

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

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

Neben der Möglichkeit, durch eigene Programme die Daten auf Dubletten zu überprüfen, könnte man auch die Annahme auf Seiten der Tabelle verweigern. Man baut einen Index auf alle signifikanten Felder mit der Option `UNIQUE`. Da ein

Indexschlüssel in MySQL maximal 3072 Bytes lang sein darf, legen wir sinnvolle Spaltenlängen fest. Für unser Beispiel sähe dies wie folgt aus:

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



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

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

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

- *Einfügen:* Wenn neue Datensätze hinzukommen, muss für jeden Datensatz die neue Position innerhalb der Indizierung gefunden werden. Falls es dabei zu einem nicht ausbalancierten B-Baum kommt, muss dieser umgebaut werden.⁶
- *Löschen:* Wenn ein Datensatz gelöscht wird, muss dieser auch aus dem Index entfernt werden. Auch dabei kann es passieren, dass der B-Baum ins Ungleichgewicht kommt und umgebaut werden muss.

- *Ändern*: Falls die Änderung ein Feld betrifft, welches Teil des Index ist, kann auch dadurch ein Umbau notwendig sein (Mischung zwischen Einfügen und Löschen).
- Zusätzlich verbraucht ein Index auch noch Speicherplatz; ein glücklicherweise immer unwichtiger werdendes Argument.

Ist aber die Einfüge-Operation wirklich messbar langsamer? Da ist wohl mal wieder eine Performancemessung fällig: Es werden jeweils 19.000 Datensätze in eine Tabelle ohne Index und eine mit Index importiert. In [Bild 6.3](#) erkennt man deutlich, dass das Einfügen in Tabellen mit Indizes mehr Zeit verbraucht als ohne.⁷

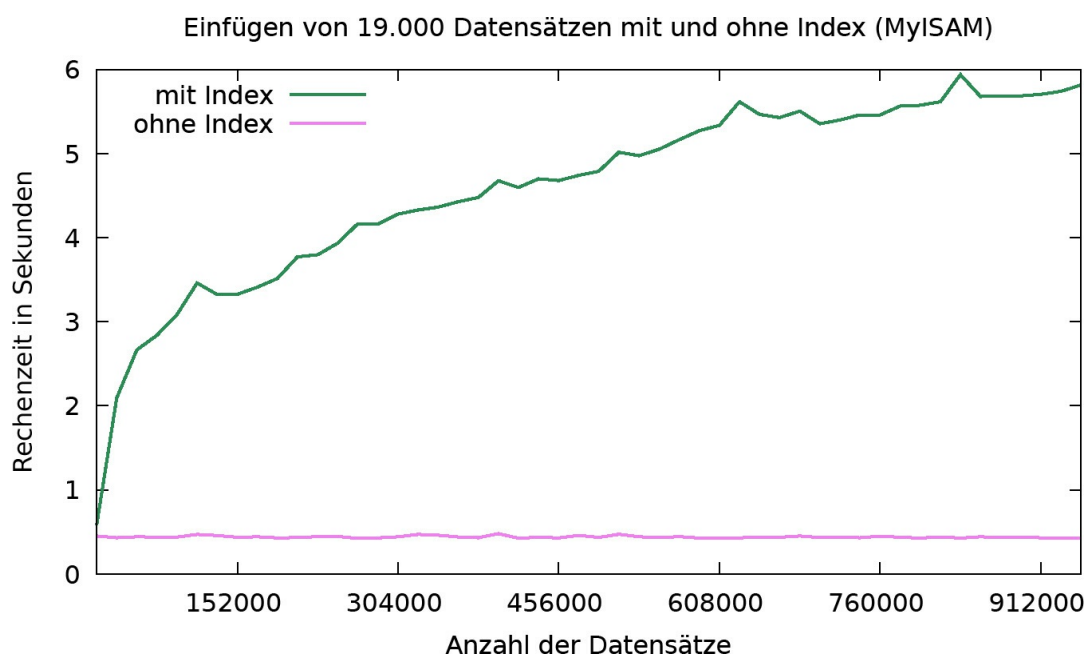


Bild 6.3 Zeitdauer Einfüge-Operationen mit und ohne Index

Sie müssen deshalb abwägen, ob und welche Indizes Sie verwenden wollen. Die [Tabelle 6.1](#) soll Ihnen bei der Entscheidung helfen.

Lässt sich das Einfügen. Ändern oder Löschen großer

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

Tabelle 6.1 Entscheidungshilfe Index oder nicht

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

6.2.3 Was bedeutet Indexselektivität?

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

Bei numerischen Datentypen (siehe die [Abschnitte 26.1.1.1](#))

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

Bei zeichenbasierten Datentypen (siehe [Tabelle 26.3](#) in [Abschnitt 26.1.2](#)) muss Zeichen für Zeichen verglichen werden. Besteht der zu suchende String aus vielen Zeichen und sind die indizierten Elemente ebenfalls lang, ergeben sich dadurch lange Suchzeiten.

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

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



Definition 32: Indexselektivität

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

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

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

Die Indexselektivität von 1 bedeutet, dass jede Zeile der Tabelle durch den Index eindeutig identifiziert werden kann. Unser Beispiel liefert aber auch eine Indexselektivität von 1, wenn man nur die ersten beiden Zeichen nimmt. Offensichtlich können diese beiden Zeichen viel schneller

verglichen und abgespeichert/geladen werden als bei voller Länge.

Kann ich also durch einen kleineren und damit schnelleren Index genau die gleiche Ergebnisqualität bekommen? Ich möchte das Ganze an einem echten Beispiel näher untersuchen. Die Datei `blz_20120305.csv` enthält die Liste aller deutscher Banken mit Namen und Bankleitzahl zum Zeitpunkt 03.05.2012⁸.

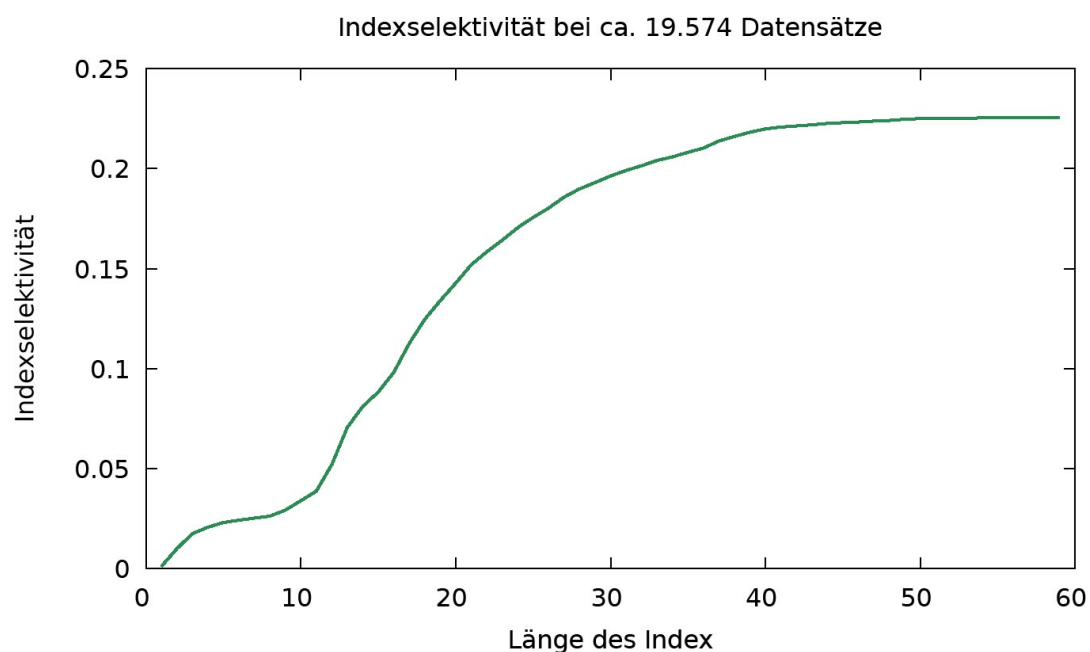


Bild 6.4 Indexselektivität am Beispiel Bankbezeichnung

Die maximale Länge einer Bankbezeichnung ist 59. In [Bild 6.4](#) können Sie sehen, wie sich die Indexselektivität bei diesem realen Beispiel gestaltet⁹. Leider erreichen wir die Indexselektivität von 1 bei Weitem nicht. Nach kurzem Nachdenken wird klar, dass dies auch nur bei Schlüsseln zu erwarten ist. Noch viel spannender ist aber, dass wir bis zu einer Indextlänge von ca. 30 einen steilen Anstieg haben und ab diesem Wert nur noch ein sehr flaches Anwachsen erkennen können.

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

```
1  mysql> CREATE INDEX idx_testdaten_bezeichn
ung
2      -> ON testdaten_bank (bezeichnung(30))
3
4  mysql> SHOW INDEX FROM testdaten\G
5  ***** 1. row *****
*****
6      Table: testdaten
7      Non_unique: 1
8      Key_name: idx_testdaten_bezeichnung
9      Seq_in_index: 1
10     Column_name: bezeichnung
11     Collation: NULL
12     Cardinality: 3336
13     Sub_part: 30
14     Packed: NULL
15     [...]
```

In der [Zeile 13](#) erkennen Sie die Einschränkung auf die ersten 30 Zeichen. MySQL schätzt, dass dies bei 19575 Zeilen zu 3336 Indexeinträgen führt ([Zeile 12](#))¹⁰.

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

6.2.4 Wie kann ich einen Index löschen?




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

PostgreSQL, T-SQL

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

Der Index – und damit seine Spezifikation – wird vollständig entfernt. Möchten Sie später den Index wieder verwenden, müssen Sie mit `CREATE INDEX` diesen wieder erstellen.



MySQL/MariaDB

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

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

```
1      ALTER TABLE bankverbindung DISABLE KEY
S;
```

Reaktivieren können Sie den Index mit `ENABLE KEYS`. Für die Tabelle `bankverbindung` sähe dies wie folgt aus:

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

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

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

Auch werden nur die Indizes ausgeschaltet, die nicht mit UNIQUE erstellt wurden. Mit anderen Worten: So richtig steht uns dieser Befehl nicht zur Verfügung und wir werden wohl im Bedarfsfall den Index löschen und wiederanlegen müssen. Schade :-)

¹ Der genaue Versuchsaufbau kann in [Kapitel 27.4](#) nachgelesen werden.

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

³ Ich möchte hier darauf verzichten, den B-Baum genauer zu erläutern. Sein Verständnis bringt nicht zwingend ein besseres Anwendungswissen bzgl. SQL. Trotzdem empfehle ich die Literatur, da B-Bäume ein schönes Stück Informatik sind. In [\[Wik23a\]](#) finden Sie eine übersichtliche Erklärung.

⁴ Natürlich werden Indizes auch bei anderen Gelegenheiten automatisch angelegt; beispielhaft sei hier nur der Spaltenzusatz UNIQUE (siehe [Tabelle 26.10](#)) erwähnt.

⁵ Aber nicht nur dann: Gruppierungen, DISTINCT, ...

⁶ Siehe [\[Wik23a\]](#).

⁷ Erstaunlicherweise ist die InnoDB erheblich schneller (siehe [Abschnitt 27.5](#)).

⁸ Siehe [[Bun16](#)].

⁹ Die genaue Messung können Sie in [Abschnitt 27.6](#) nachlesen.

¹⁰ Interessanterweise schätzt MariaDB hier 9733.

7 Werte in Tabellen einfügen

7.1 Daten importieren



Massenhaft neue Daten und wie man sie in Tabellen verstaut

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



Die Quelltexte dieses Kapitels stehen in der Datei `listing04.sql` (siehe [Listing 28.4](#), [Listing 28.44](#) und [Listing 28.29](#)). Sie brauchen auch die Dateien `artikel01.csv` und `artikel02.csv`.

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

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

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

7.1.1 Das CSV-Format

Die Artikelstammdaten für unseren Online-Shop werden nur selten manuell eingegeben, da diese in der Regel durch externe Anbieter zur Verfügung gestellt werden. Wenn Sie beispielsweise Kinderspielzeug anbieten wollen, bekommen Sie von den Herstellern Artikelstammdaten inklusive Fotos, z.B. per Online-Download. Diese Daten liegen in einem CSV-Format¹ vor und müssen in die Artikeltabelle importiert werden.



Definition 33: CSV-Format

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

- Ein Datensatz endet mit einem Steuerzeichen wie `\r\n`.
- Bei der letzten Datenzeile kann das Endezeichen entfallen.
- Die erste Zeile kann optional die Feldnamen enthalten.
- Es gibt ein Trennzeichen, welches die optionalen Feldnamen in der ersten Zeile und die Werte in den Datensätzen voneinander trennt.
- Pro Zeile muss immer die gleiche Anzahl von Trennzeichen verwendet werden.
- Werte können durch Hochkommas ' oder Gänsefüßchen " umschlossen werden.

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

Das Trennzeichen kann jedes beliebige Zeichen sein, sofern es sich eindeutig von den Daten und anderen Steuerzeichen

unterscheiden lässt. Neben dem Komma , sind das Semikolon ;, das kaufmännische Und &, der Lattenzaun #, der Slash /, das Leerzeichen und der Tabulator sehr beliebt.

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

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

Ich möchte nun die Artikelstammdaten importieren. Die Datei `artikel01.csv` sieht so aus, dass in der ersten Zeile die Spaltenüberschriften notiert sind, wobei das Trennsymbol ein Semikolon ist, und das Ende des Datensatzes durch `\n` markiert wird:

```
1  artikelnummer;artikelname;preis;waehrung
2  7856;Silberzwiebel;0.41;EUR
3  7863;Tulpenzwiebel;3.29;EUR
4  9010;Schaufel;15.00;USD
5  9015;Spaten;19.90;EUR
6  3001;Papier (100);2.30;EUR
7  3005;Tinte (gold);55.70;EUR
8  3006;Tinte (rot);6.20;EUR
9  3010;Feder;5.00;EUR
```

7.1.2 CSV-Daten laden mit `LOAD DATA INFILE`

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



MySQL/MariaDB

```
LOAD DATA [LOCAL] INFILE 'dateiname'  
  INTO TABLE tabellenname  
  [CHARACTER SET zeichensatz]  
  [FIELDS  
    [TERMINATED BY 'zeichenfolge']]  
  [LINES  
    [TERMINATED BY 'zeichenfolge']]  
  ;
```



Hinweis: In SQL:2023 gibt es keine Anweisung zum Importieren von externen Daten. In anderen Datenbankmanagementsystemen gibt es Entsprechungen: z.B. COPY in PostgreSQL oder BULK INSERT in T-SQL.

Zunächst muss angegeben werden, aus welcher CSV-Datei die Daten importiert werden sollen. Es reicht aber nicht aus, den Dateinamen ggf. mit seinem Pfad anzugeben. Die Datei `artikel01.csv` kann schließlich an zwei Orten liegen: auf dem Rechner, wo der MySQL oder MariaDB Server läuft, oder auf dem Client, über den der Befehl zum Import gerade versendet wird. Dies wird durch die Angabe oder das Weglassen von `LOCAL` entschieden. Wird `LOCAL` angegeben, werden die Daten zuerst vom Client zum Server transferiert und erst dann importiert. Lässt man `LOCAL` weg, sucht er die Datei in einem Serverpfad. In unserem Fall liegt die Datei auf dem Client-Rechner.

Die Zieltabelle ist `artikel`. und das Trennzeichen ist das

Semikolon. Die Datensätze werden nur durch \n getrennt, und die Zeichen sind in UTF-8 kodiert, sodass sich folgende erste Version ergibt:

```
1  mysql> LOAD DATA LOCAL INFILE 'artikel01.csv'  
2      -> CHARACTER SET utf8mb4  
3      -> INTO TABLE oshop.artikel  
4      -> FIELDS  
5      -> TERMINATED BY ';'   
6      -> LINES  
7      -> TERMINATED BY '\n'  
8      -> ;  
9  ERROR 2068 (HY000): LOAD DATA LOCAL INFILE  
   file request rejected due to  
       restrictions on access.
```

Alternativ könnte auch diese Meldung erscheinen:

```
1  ERROR 3948 (42000): Loading local data is  
   disabled; this must be enabled on both the cli  
   ent and server sides
```

Unser erster Versuch ging also glatt schief²: Die Fehlermeldungen können mehrere Gründe haben. Der wahrscheinlichste ist, dass es bei Ihrer Installation nicht gestattet ist, dass der Client lokale Dateien zwecks Import hochlädt.

Versuchen Sie es erneut, nachdem Sie den MySQL Client mit `--local-infile` gestartet und die globale Servervariable `local_infile` auf ON gesetzt haben:

```
1  root@cae6e2bec1b8:/#mysql --local-infile -u root -p  
2  mysql> SET GLOBAL local_infile = ON;  
3  Query OK, 0 rows affected (0.00 sec)
```

```

4
5  mysql> LOAD DATA LOCAL INFILE 'artikel01.csv'
6      -> INTO TABLE oshop.artikel -- Spare m
ir ein USE oshop
7      -> CHARACTER SET utf8mb4
8      -> FIELDS
9      -> TERMINATED BY ';'
10     -> LINES
11     -> TERMINATED BY '\n'
12     -> ;
13  Query OK, 9 rows affected, 12 warnings (0.
03 sec)
14  Records: 9 Deleted: 0 Skipped: 0 Warnings:
10

```

Die zwölf Warnungen machen uns auch nicht glücklich.
Schauen wir uns die mal genauer an:

```

1  mysql> SHOW WARNINGS;
2  +-----+-----+-----+-----+
-----[...]-+
3  | Level | Code | Message [...] |
4  +-----+-----+-----+-----+
-----[...]-+
5  | Warning | 1366 | Incorrect integer value
: 'artikelnummer' for column 'artikel_id' at r
ow 1 |
6  | Warning | 1366 | Incorrect decimal value
: 'preis' for column 'einzelpreis' at row 1 |
7  | Warning | 1265 | Data truncated for colu
mn 'waehrung' at row 1
|
8  | Warning | 1261 | Row 1 doesn't contain d
ata for all columns [...] |
9  | Warning | 1261 | Row 2 doesn't contain d
ata for all columns [...] |
10 | Warning | 1261 | Row 3 doesn't contain d

```

```

ata for all columns [...] |
11 | Warning | 1261 | Row 4 doesn't contain d
ata for all columns [...] |
12 | Warning | 1261 | Row 5 doesn't contain d
ata for all columns [...] |
13 | Warning | 1261 | Row 6 doesn't contain d
ata for all columns [...] |
14 | Warning | 1261 | Row 7 doesn't contain d
ata for all columns [...] |
15 | Warning | 1261 | Row 8 doesn't contain d
ata for all columns [...] |
16 | Warning | 1261 | Row 9 doesn't contain d
ata for all columns [...] |
17 +-----+-----+-----+-----+
-----[...]-+
18 12 rows in set (0.00 sec)

```



Aufgabe 7.1: Die drei Warnungen in den [Zeilen 5](#) ff. beziehen sich auf die erste Zeile (row 1). Was ist denn mit der ersten Zeile?

Aufgabe 7.2: Die Warnungen in den [Zeilen 8](#) ff. haben auch eine gemeinsame Ursache. Vergleichen Sie die Anzahl der Angaben pro Zeile in der csv-Datei und die Anzahl der Spalten der Tabelle. Was fällt auf?

Wir brauchen eine neue Version von `LOAD DATA INFILE`:



MySQL/MariaDB

```

LOAD DATA [LOW_PRIORITY|CONCURRENT] [LOCAL
] INFILE 'dateiname'
[REPLACE|IGNORE]
INTO TABLE tabellenname

```

```
[CHARACTER SET zeichensatz]  
[FIELDS  
[TERMINATED BY 'zeichenfolge']  
[[OPTIONALLY] ENCLOSED BY 'zeichen']  
[ESCAPED BY 'zeichen'  
]  
[LINES  
[STARTING BY 'zeichenfolge']  
[TERMINATED BY 'zeichenfolge']  
IGNORE anzahl LINES]  
[(spaltenname,...)]  
[SET spaltenname = ausdruck,...)]  
;  
;
```

Die erste Aufgabe können wir mit IGNORE lösen. Mit dieser Option legen wir fest, wie viele Zeilen (*anzahl*) zu Beginn der Datei übersprungen werden sollen. Die zweite Aufgabe entsteht dadurch, dass ich nicht angegeben habe, welche Felder ich in der CSV-Datei erwarte.

Auf zumnächsten Versuch:

```
1  mysql> DELETE FROM oshop.artikel;  
2  Query OK, 8 rows affected (0.08 sec)  
3  
4  mysql> LOAD DATA LOCAL INFILE 'artikel01.c  
sv'  
5      -> INTO TABLE oshop.artikel  
6      -> CHARACTER SET utf8mb4  
7      -> FIELDS  
8      -> TERMINATED BY ';'   
9      -> LINES  
10     -> TERMINATED BY '\n'  
11     -> IGNORE 1 LINES  
12     -> (artikel_id, bezeichnung, einzelpre  
is, waehrung)
```

```

13      -> SET deleted=0;
14 Query OK, 8 rows affected (0.03 sec)
15 Records: 8 Deleted: 0 Skipped: 0 Warnings:
0

```

Keine Warnungen, keine Fehler :-). Die [Zeile 11](#) löst die erste Aufgabe, indem sie anweist, die erste Zeile zu überspringen. Aber was macht [Zeile 1](#)? Im Vorgriff auf spätere Kapitel³ muss hier die Tabelle erst einmal wieder geleert werden. Dies erreiche ich durch DELETE.

Die zweite Aufgabe wird durch zwei Angaben gelöst. In der [Zeile 12](#) wird eine Spaltenliste angegeben. Diese Spaltenliste weist den Feldern der CSV-Datei die Spalten der Tabelle zu. Das erste Feld wird der Spalte `artikel_id` zugewiesen usw. Fehlt diese Spaltenliste, versucht er, alle Spalten der Tabelle in der CSV-Datei zu finden. In der [Zeile 13](#) wird der fehlenden Spalte `deleted` die 0 zugewiesen.



Aufgabe 7.3: Warum ist [Zeile 13](#) beim `LOAD DATA INFILE` eigentlich überflüssig? Tipp: `DESCRIBE artikel`.

Ein Blick in die Tabelle `artikel` bestätigt es: Die Werte wurden korrekt übernommen.

```

1  mysql> SELECT * FROM oshop.artikel;
2  +-----+-----+-----+-----+
--+-----+-----+
3  | artikel_id | bezeichnung | einzelpreis |
  waehrung | deleted |
4  +-----+-----+-----+-----+
--+-----+-----+
5  | 3001 | Papier (100) | 2.300000 | EUR | 0

```

6		3005		Tinte (gold)		55.700000		EUR	
0		7		3006		Tinte (rot)		6.200000	
		8		3010		Feder		5.000000	
		9		7856		Silberwiebel		0.410000	
		10		7863		Tulpenzwiebel		3.290000	
0		11		9010		Schaufel		15.000000	
		12		9015		Spaten		19.900000	
13		+-----+-----+-----+-----+							
		-+-----+-----+							

BULK INSERT in T-SQL



T-SQL

```
BULK INSERT '[tabellenname.schemaname.]tab
ellenname'
FROM dateiname
[WITH (
[FIRSTROW='zeilennummer']
[FORMATFILE='dateiname'])]
;
```

In T-SQL gibt es eine sehr mächtige Importschnittstelle, deren Möglichkeiten ich hier nur anreißen möchte.⁴ Dort können Sie über eine Formatdatei mehr oder weniger frei die Art des Imports steuern. Der Artikeldatenimport sähe so aus:

```
1 >1 BULK INSERT artikel
```

```

2  >2 FROM '/home/[...]/src/mssql/artikel01.c
sv'
3  >3 WITH
4  >4 (FIRSTROW=2, FORMATFILE='/home/[...]/sr
c/mssql/artikel.fmt', KEEPIDENTITY);
5  GO

```

Die Formatdatei hätte für diesen simplen Fall folgenden Aufbau:

```

1  11.0
2  4
3  1 SQLCHAR 0 12 ";" 1 artikel_id ""
4  2 SQLCHAR 0 510 ";" 2 bezeichnung German_P
honeB[...]
5  3 SQLCHAR 0 41 ";" 3 einzelpreis ""
6  4 SQLCHAR 0 6 "\n" 4 waehrung German_Phone
B[...]

```

7.1.3 Was ist, wenn ich geänderte Werte importieren will?

Nehmen wir an, wir bekämen eine neue Version unserer Artikelstammdaten zugeschickt. Würden wir jetzt die Daten wieder importieren, stellt sich die Frage, was mit Datensätzen passiert, die den gleichen Primärschlüsselwert haben?

Beispiel: Die Preise unserer Artikelstammdaten haben sich geändert, und es sind neue hinzugekommen:

```

1  artikelnummer;artikelname;preis;waehrung
2  7856;Silberzwiebel;0.51;EUR
3  7863;Tulpenzwiebel;3.39;EUR
4  9010;Schaufel;14.95;USD
5  9015;Spaten;19.90;EUR

```

```
6 3001;Papier (100);2.30;EUR
7 3005;Tinte (gold);55.70;EUR
8 3006;Tinte (rot);6.20;EUR
9 3007;Tinte (blau);4.13;EUR
10 3010;Feder;5.00;EUR
```

Was passiert jetzt mit den Artikeln, die den gleichen Primärschlüsselwert haben, aber sich im Preis unterscheiden (7856, 7863 und 9010)? Das entscheidet die Option REPLACE|IGNORE:

- IGNORE: Datensätze, die den gleichen Primärschlüsselwert haben, werden nicht importiert. Dies soll verhindern, dass man versehentlich schon vorhandene Datensätze überschreibt. Man erwartet eigentlich keine geänderten, sondern nur neue Datensätze.
- REPLACE: Datensätze, die den gleichen Primärschlüsselwert haben, werden komplett ersetzt, d.h., der alte Datensatz wird gelöscht und der neue eingefügt.

Probieren wir das Ganze mal aus, zunächst mit IGNORE:

```
1 mysql> LOAD DATA LOCAL INFILE 'artikel02.csv' IGNORE
2 [...]
3      -> (artikel_id, bezeichnung, einzelpreis, waehrung);
4 Query OK, 1 row affected (0.06 sec)
5 Records: 9 Deleted: 0 Skipped: 8 Warnings:
6 0
```

Beachten Sie bitte [Zeile 5](#): Von neun Datensätzen wurden acht übersprungen. Dies entspricht auch unseren Erwartungen. Alle Datensätze mit gleichem Primärschlüsselwert wurden wegen

IGNORE nicht importiert. Der neue Datensatz 3007 wurde hingegen anstandslos importiert:

```
1  mysql> SELECT * FROM artikel;
2  +-----+-----+-----+-----+
3  | artikel_id | bezeichnung | einzelpreis | waehrung | deleted |
4  +-----+-----+-----+-----+
5  | 3001 | Papier (100) | 2.300000 | EUR | 0
6  |
7  | 3005 | Tinte (gold) | 55.700000 | EUR |
8  |
9  | 3006 | Tinte (rot) | 6.200000 | EUR | 0
10 |
11 | 3007 | Tinte (blau) | 4.130000 | EUR | 0
12 |
13 | 3010 | Feder | 5.000000 | EUR | 0 |
14 | 7856 | Silberwiebel | 0.410000 | EUR | 0
15 |
16 | 7863 | Tulpenzwiebel | 3.290000 | EUR |
17 | 0 |
18 | 9010 | Schaufel | 15.000000 | USD | 0 |
19 | 9015 | Spaten | 19.900000 | EUR | 0 |
20 +-----+-----+-----+-----+
21 +-----+-----+-----+-----+
```

Und jetzt mit REPLACE:

```
1  mysql> DELETE FROM artikel;
2  Query OK, 8 rows affected (0.08 sec)
3
4  mysql> LOAD DATA LOCAL INFILE 'artikel01.csv'
5          -> INTO TABLE artikel
6  [...]
7  Query OK, 8 rows affected (0.08 sec)
```

```

8  Records: 8 Deleted: 0 Skipped: 0 Warnings:
9
10 mysql> LOAD DATA LOCAL INFILE 'artikel02.c
sv' REPLACE
11      -> INTO TABLE artikel
12  [...]
13  Query OK, 17 rows affected (0.07 sec)
14  Records: 9 Deleted: 8 Skipped: 0 Warnings:
15  0

```

Zuerst müssen wir wieder die Tabelle leeren ([Zeile 1](#)). Anschließend mit der ersten Version der Artikelstammdaten füllen ([Zeile 4](#)). In [Zeile 10](#) wird die zweite Version der Artikelstammdaten mit der Option REPLACE importiert.

Das Ergebnis: 17 Datensätze sind betroffen. Acht wurden gelöscht und einer neu eingefügt:

```

1  mysql> SELECT * FROM artikel;
2  +-----+-----+-----+-----+
3  | artikel_id | bezeichnung | einzelpreis | waehrung | deleted |
4  +-----+-----+-----+-----+
5  | 3001 | Papier (100) | 2.300000 | EUR | 0
6  |
7  | 3005 | Tinte (gold) | 55.700000 | EUR |
8  |
9  | 3006 | Tinte (rot) | 6.200000 | EUR | 0
10 |
11 | 3007 | Tinte (blau) | 4.130000 | EUR | 0
12 |
13 | 3010 | Feder | 5.000000 | EUR | 0 |
14 | 7856 | Silberwiebel | 0.510000 | EUR | 0
15 |
16 | 7863 | Tulpenzwiebel | 3.390000 | EUR |

```

0							
12		9010		Schaufel		14.950000	USD 0
13		9015		Spaten		19.900000	EUR 0
14	+	-----	+	-----	+	-----	+
	-	-----	+	-----	+	-----	+

7.2 Daten anlegen



Erlesene Daten, liebevoll und einzeln eingefügt

- Grundkurs
 - Manuelles Anlegen einzelner Zeilen mit INSERT INTO ... SET
 - Manuelles Anlegen mehrerer Zeilen mit INSERT INTO ... VALUES
- Vertiefendes
 - Binäre Daten einfügen
 - Einfügereihenfolgen wegen der referenziellen Integrität

Schauen wir uns noch einmal das ER-Modell zur Artikelverwaltung an. Jedem Artikel muss mindestens eine Warengruppe zugeordnet werden, eine Warengruppe kann beliebig viele Artikel enthalten. Wir wissen darüber hinaus, dass die $n:m$ -Verknüpfung eine Hilfstabelle impliziert (siehe [Definition 21](#) in [Abschnitt 2.2.4](#)). Wir wollen nun dem Artikel die Warengruppe zuordnen. Ich beziehe mich dabei auf die Daten in `artikel02.csv`.

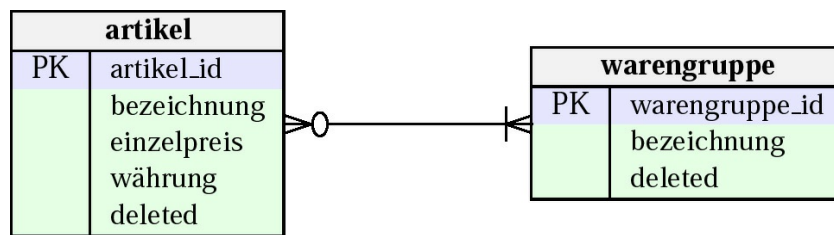


Bild 7.1 ER-Modell: Artikel zu Warengruppe

Die Artikel mit der führenden 3 sind sicherlich leicht als Bürobedarf zu kategorisieren. Die Artikel mit der führenden 7 gehören zu Pflanzen und Gartenbedarf, die mit der führenden 9 zu Werkzeug und Gartenbedarf.

Unser Vorgehen ist in zwei Phasen aufgeteilt:

1. Die Warengruppen werden in der Tabelle warengruppe angelegt.
2. In der Hilfstabelle werden die Verknüpfungen zwischen den Artikeln und diesen Warengruppen vorgenommen.

7.2.1 Wie lege ich mehrere Zeilen mit einem Befehl an?

Der Befehl zum Anlegen einer Zeile heißt `INSERT INTO`. Hier ein Syntax-Auszug:



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

```

INSERT INTO tabellenname [(spaltenliste)]
VALUES
(werteliste)
[, (werteliste)] *
;
  
```

Die *werteliste* muss genau so viele Werte enthalten, wie Spalten in der *spaltenliste* angegeben wurden. Ebenso müssen die Werte in der *werteliste* immer passend zur Reihenfolge in der *spaltenliste* sein.

Die *spaltenliste* ist zwar optional, wird sie aber nicht angegeben, müssen zu allen Spalten der Tabelle in der Reihenfolge des CREATE TABLE in der *werteliste* Werte angegeben werden.

Wenn nötig, können Sie diese Reihenfolge durch DESCRIBE *tabellenname* oder SHOW CREATE TABLE *tabellenname* ermitteln.

```
1  INSERT INTO warengruppe (bezeichnung, deleted)
2  VALUES
3      ('Bürobedarf', 0)
4      , ('Pflanzen', 0)
5      , ('Gartenbedarf', 0)
6      , ('Werkzeug', 0)
7  ;
```

Das Ergebnis:

```
1  mysql> SELECT * FROM warengruppe;
2  +-----+-----+-----+
3  | warengruppe_id | bezeichnung | deleted |
4  +-----+-----+-----+
5  | 1 | Bürobedarf | 0 |
6  | 2 | Pflanzen | 0 |
7  | 3 | Gartenbedarf | 0 |
8  | 4 | Werkzeug | 0 |
9  +-----+-----+-----+
+
```

Wir können mehrere Dinge hier erkennen:

- Zeichenketten werden in SQL immer von Hochkommata ' umschlossen, numerische Werte nicht.
- In der *spaltenliste* sind nur die Spalten angegeben, zu denen neue Werte bekannt sind.
- Jede *werteliste* enthält gleich viele Angaben und immer in der gleichen Reihenfolge.
- Obwohl keine Angaben über die Spalte *warengruppe_id* gemacht wurden, werden diese automatisch generiert.



Aufgabe 7.4: Warum werden die Werte von der Spalte *warengruppe_id* automatisch generiert?

Aufgabe 7.5: Dieser Quelltext würde ebenfalls das gleiche Ergebnis liefern. Warum?

```
INSERT INTO warengruppe (warengruppe_id,
    bezeichnung) VALUES
    (1, 'Bürobedarf')
    ,(2, 'Pflanzen')
    ,(3, 'Gartenbedarf')
    ,(4, 'Werkzeug')
    ;
```

Aufgabe 7.6: Wenn Sie beide Quelltexte hintereinander ausführen, erhalten Sie folgende Fehlermeldung:

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

Interpretieren Sie diese Fehlermeldung.

7.2.2 Wie kann ich eine einzelne Zeile anlegen?



MySQL/MariaDB

```
INSERT INTO tabellenname  
SET  
spaltenname=wert  
[, spaltenname=wert] *  
;
```

Wir wollen jetzt Verknüpfungen in die Hilfstabelle eintragen. Dazu möchte ich die Zeilen einzeln anlegen, um bei der Zuordnung die Übersicht nicht zu verlieren. Und wie geht das?

Zuerst die lahmte Antwort: Indem Sie in der schon bekannten Variante `INSERT INTO ... VALUES` nur eine *werteliste* angeben. Und jetzt die spannende Antwort: Alternativ können wir bei MySQL und MariaDB eine eigene Variante verwenden: `INSERT INTO ... SET`.

Bevor wir die neuen Daten einfügen, sollten wir zuerst die Daten einzeln notieren, um das Verhältnis zwischen Artikelstammdaten und Warengruppen besser zu verstehen.

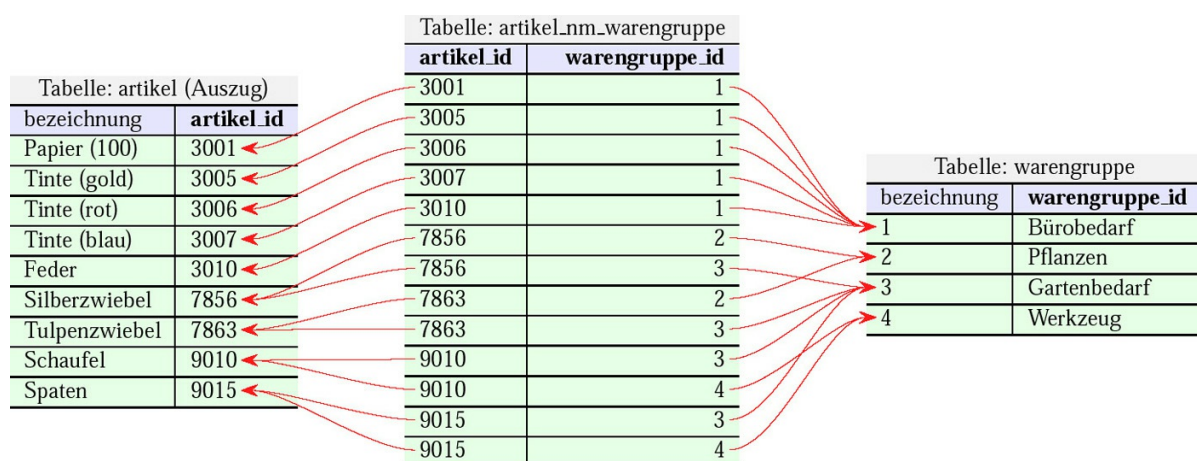


Bild 7.2 Hilfstabelle: Artikel zu Warengruppe

In [Bild 7.2](#) wird die Funktionsweise der Hilfstabelle deutlich. Jede Verknüpfung in der Hilfstabelle verweist auf einen Artikel und eine Warengruppe. Als Folge kann ein Artikel zu mehreren Warengruppen gehören (z.B. 7863) und eine Warengruppe mehrere Artikel beinhalten (z.B. 4).

Der nächste Schritt ist nun, für jede Verknüpfung ein eigenes INSERT INTO zu erstellen. Sie können beim Lesen des Befehls schneller erkennen, welcher Wert welcher Spalte zugeordnet wird:

```
1  INSERT INTO artikel_nm_warengruppe
2  SET
3      artikel_id=3001
4      ,warengruppe_id=1
5  ;
```



Aufgabe 7.7: Erstellen Sie die INSERT INTO für alle Zuordnungen, sodass jeder Artikel seine Warengruppe kennt und jede Warengruppe alle entsprechenden Artikel enthält.

7.2.3 Vorsicht Constraints!

Zuerst eine kleine Fehlermeldung:

```
1  mysql> SELECT bestellung_id, kunde_id, adr
2  esse_id, datum FROM bestellung;
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```



```

3 | bestellung_id | kunde_id | adresse_id |
datum |
4 +-----+-----+-----+-----+
-----+
5 | 1 | 1 | 1 | 2012-03-24 17:41:00 |
6 | 2 | 2 | 2 | 2012-03-23 16:11:00 |
7 +-----+-----+-----+-----+
-----+

```

```

8
9 mysql> INSERT INTO bestellung (datum) VALU
ES (NOW());
10 ERROR 1452 (23000): Cannot add or update a
   child row: a foreign key constraint fails ('o
shop'. 'bestellung', CONSTRAINT 'bestellung_ibf
k_2' FOREIGN KEY ('adresse_id') REFERENCES 'a
dresse' ('adresse_id') ON DELETE RESTRICT ON U
PDATE RESTRICT)

```

Die Fehlermeldung besagt übersetzt soviel wie: Ich konnte den Datensatz nicht anlegen, da der Constraint einen NULL-Wert verbietet. Also 2. Versuch:

```

1 mysql> INSERT INTO bestellung (adresse_id,
datum) VALUES (10, NOW());
2 ERROR 1452 (23000): Cannot add or update a
   child row: a foreign key constraint fails ('o
shop'. 'bestellung', CONSTRAINT 'bestellung_ibf
k_2' FOREIGN KEY ('adresse_id') REFERENCES 'a
dresse' ('adresse_id') ON DELETE RESTRICT ON U
PDATE RESTRICT)

```

Ging schon wieder schief, da es keine Adresse 10 gibt. Also zuerst eine Adresse 10 einfügen:

```

1 mysql> INSERT INTO adresse (adresse_id, st
rasse) VALUES (10, 'Oberer Weg');
2 Query OK, 1 row affected (0.09 sec)
3

```

```
4 mysql> INSERT INTO bestellung (adresse_id,  
datum) VALUES (10, NOW());  
5 ERROR 1452 (23000): Cannot add or update a  
child row: a foreign key constraint fails ('o  
shop'. 'bestellung', CONSTRAINT 'bestellung_ibf  
k_1' FOREIGN KEY ('kunde_id') REFERENCES 'kun  
de' ('kunde_id') ON DELETE RESTRICT ON UPDATE  
RESTRICT)
```

Schön, wenigstens eine neue Fehlerquelle. Aber leider die gleiche Art von Ursache. Sie sehen, dass bei Tabellen, die Constraints beachten, die Reihenfolge wichtig ist, in der die Daten importiert, angelegt oder kopiert werden.

Die fehlgeschlagenen Versuche bedeuten natürlich nicht, dass man keine neue Bestellung anlegen kann. Es muss aber darauf geachtet werden, dass der neue Datensatz zu allen geforderten Constraints eine gültige Angabe enthält.



Hinweis: Bei Constraints muss beim Anlegen neuer Zeilen in mehreren Tabellen auf die Reihenfolge geachtet werden, um die referenzielle Integrität zu wahren. Warum? Siehe [Abschnitt 2.3](#).

7.2.4 Einfügen von binären Daten über einen C#-Client

Mit dem Datentyp BLOB und dessen Verwandten steht uns die Möglichkeit offen, binäre Daten wie Bilder, Musik, PDF-Dateien etc. in Tabellen abzulegen. Stellt sich nur die Frage, wie man die Dinger da reinbekommt.



Aufgabe 7.8: Wir könnten für unsere Artikel eine Bildergalerie anlegen wollen.

1. Modellieren Sie die Tabelle `bild`. Sie soll mindestens die Bilddaten, einen Dateinamen, die Dateigröße und den Dateityp enthalten.
2. Erweitern Sie das ER-Modell derart, dass man zu jedem Artikel beliebig viele Bilder ablegen kann.
3. Erstellen Sie das passende `CREATE TABLE`.

Das Einfügen erfolgt in der Regel über einen Client. Ich verwende hier als Beispiel einen C#-Client; PHP-Beispiele finden Sie zu Zehntausenden im Internet.



Wir brauchen die aus dem Verzeichnis `src/InsertBlob` die Dateien: `spaten01.png`, `spaten02.png`, `spaten03.png` und aus dem Verzeichnis `src/InsertBlob/InsertBlob` die Datei `Hauptfenster.cs`.

Hier die Tabelle:

```
1  mysql> CREATE TABLE bild (  
2      -> bild_id INT UNSIGNED AUTO_INCREMENT  
PRIMARY KEY,  
3      -> bild BLOB,  
4      -> dateiname VARCHAR(255),  
5      -> dateityp VARCHAR(255),  
6      -> dateigroesse BIGINT UNSIGNED,  
7      -> artikel_id INT UNSIGNED REFERENCES  
artikel(artikel_id)  
8      -> );
```

Zuerst wird mithilfe eines `OpenFileDialog`s die Datei ausgewählt: `spaten01.png`. Nur wenn der `OpenFileDialog` mit OK beendet wurde, werden die nachfolgenden Anweisungen ausgeführt.

Listing 7.1 INSERT: Hauptfenster.cs, Teil 1

```
18     private void BtnnBild_Click(object sender, EventArgs e) {
19         OpenFileDialog dlg = new OpenFileDialog();
20         dlg.DefaultExt = "png";
21         dlg.Filter = "png-Dateien|*.png|jpg-Dateien|*.jpg|Alle Dateien|*.*";
22         if (dlg.ShowDialog() == System.Windows.Forms.DialogResult.OK) {
```

Jetzt werden einige Objekte deklariert und initialisiert, die wir später brauchen. Das Objekt `MySQLConn` repräsentiert die Verbindung zwischen dem SQL Server und der Anwendung. `MySQLCmd` enthält alle Informationen, die man für die Ausführung eines Befehls braucht. Dazu gehört der Befehl selbst (`strInsert`) und die Verbindung, über die der Befehl ausgeführt werden soll.

In `iDateigroesse` wird die Größe der Datei in Bytes gemessen abgespeichert. `binBilddaten` ist ein Array vom Datentyp `Byte`. In dieses Array werden die Bilddaten abgespeichert, die wir aus der Datei auslesen werden.

Listing 7.2 INSERT: Hauptfenster.cs, Teil 2

```
23     MySqlConnection MySQLConn = new MySqlConnection();
24     MySqlCommand MySQLCmd = new MySqlCommand();
25     String strInsert;
```

```

26         Int32 iDateigroesse;
27         Byte[] binBilddaten;
28         FileStream streamDatei;
29         String strName = Path.GetFullPath(dlg.FileName);
30         String strTyp = Path.GetExtension(dlg.FileName);
31
32         MySqlConnection.ConnectionString = "server=127.0.0.1";
33         MySqlConnection.ConnectionString += ";uid=root";
34         MySqlConnection.ConnectionString += ";pwd=";
35         MySqlConnection.ConnectionString += ";database=oshop;charset=utf8mb4";

```

Die Datei selbst wird über `streamDatei` angesprochen. Bleibt noch das Setzen der Verbindungsparameter. Dazu wird ein geläufiges Verfahren gewählt: der `ConnectionString`. Bitte achten Sie darauf, dass Sie ggf. andere Parameter einstellen müssen.

Listing 7.3 INSERT: Hauptfenster.cs, Teil 3

```

37         try {
38             streamDatei = new FileStream(strName, FileMode.Open, FileAccess.Read);
39             if (streamDatei.Length > Int32.MaxValue) {
40                 throw new Exception("Datei zu gross");
41             }
42             iDateigroesse = Convert.ToInt32(streamDatei.Length);

```

Die Datei wird lesend geöffnet. Ich verwende dabei die statische Klasse `Path`. Diese hilft dabei, Dateizugriffe

unabhängig vom verwendeten Betriebssystem zu programmieren. Besonders die unterschiedlichen Verzeichnisbegrenzer⁵ werden bequem verwaltet.

Problem: Die Größe der Datei Length ist vom Datentyp long. Ich kann später mit der Methode Read() aber nur Blöcke mit einer maximalen Größe von MaxInt32 einlesen. Ich verbiete größere Dateien, indem ich ggf. eine Ausnahme auslöse.



Hinweis: Wenn Sie große Dateien einfügen möchten, müssen diese blockweise eingelesen werden. Beachten Sie dabei, dass ggf. die Variable max_allowed_packet auf dem Server verändert werden muss.

Jetzt wird der Inhalt der Datei eingelesen und im Array binBilddaten abgelegt. Da wir die Datei nicht mehr brauchen, kann sie geschlossen werden.

Listing 7.4 INSERT: Hauptfenster.cs, Teil 4

```
44          binBilddaten = new byte[iDateigroesse];
45          streamDatei.Read(binBilddaten, 0,
46          iDateigroesse);
46          streamDatei.Close();
```

Der eigentliche Befehl wird nun gebastelt. Dabei wird ein ganz normaler INSERT verwendet. Anstelle der eigentlichen Daten werden aber Platzhalter verwendet. Die Nummer 9015 ist fest vorgegeben, da es sich um die Artikelnummer des Spatens handelt.

Listing 7.5 INSERT: Hauptfenster.cs, Teil 5

```
48          strInsert = "INSERT INTO ";
```

```

49         strInsert += "bild(bild, dateiname
, dateityp, dateigroesse, artikel_id)";
50         strInsert += " VALUES ";
51         strInsert += "(@bild, @dateiname,
@dateityp, @dateigroesse, 9015)";

```

Und los geht's: Die Verbindung zum Server wird geöffnet. Läuft da was schief, wird eine Ausnahme ausgelöst. Vielleicht ist der ConnectionString nicht richtig? Die gerade aufgebaute Verbindung wird dem MySqlCommand mitgeteilt, ebenso der SQL-Befehl. Mithilfe von Parameter. AddWithValue() werden die Platzhalter durch die konkreten Werte ersetzt, u.a. auch die binären Bilddaten⁶.

Da wir keine Ergebnisse vom Server zurückerwarten, wird mit ExecuteNonQuery() der SQL-Befehl zum Server gesendet und ausgeführt. Da wir die Verbindung zum Server nicht mehr brauchen, wird diese geschlossen.

Listing 7.6 INSERT: Hauptfenster.cs, Teil 6

```

53         MySqlConnection.Open();
54         MySqlCommand.Connection = MySqlConnection;
55         MySqlCommand.CommandText = strInsert;
56         MySqlCommand.Parameters.AddWithValue("
@dateiname", strName);
57         MySqlCommand.Parameters.AddWithValue("
@dateigroesse", iDateigroesse);
58         MySqlCommand.Parameters.AddWithValue("
@dateityp", strTyp);
59         MySqlCommand.Parameters.AddWithValue("
@bild", binBilddaten);
60
61         MySqlCommand.ExecuteNonQuery();

```

Zum Schluss noch eine Erfolgsmeldung und der catch() zur Fehlerbehandlung.

Listing 7.7 INSERT: Hauptfenster.cs, Teil 7

```
63         MessageBox.Show("Und wieder ein ne  
uer Spaten", "Spatenbild importiert!",  
64         MessageBoxButtons.OK, MessageBoxIcon.  
on.Asterisk);  
65  
66         MySqlConnection.Close();  
67     }  
68     catch (Exception ex) {  
69         MessageBox.Show(ex.Message, "Error  
",  
70         MessageBoxButtons.OK, MessageBoxIcon.  
on.Error);  
71     }  
72 }  
73 }  
74 }
```

7.2.5 Einfügen von binären Daten LOAD FILE

Eine andere Methode, BLOB-Daten in die Tabelle einzufügen, ist die Option `LOAD FILE`. Bei `INSERT`- oder `UPDATE`-Anweisungen können Dateiinhalte eingelesen und Spalten zugeordnet werden.

```
1  mysql> INSERT INTO bild  
2  -> SET  
3  -> bild = LOAD_FILE('/var/lib/mysql/upload  
s/spaten01.png'),  
4  -> dateiname = 'spaten01.png',  
5  -> dateityp = 'png',  
6  -> dateigroesse = 7168,  
7  -> artikel_id = 9015  
8  -> ;
```


In [Zeile 3](#) können Sie sehen, wie einfach die entsprechende Syntax ist.⁷ Innerhalb der Option wird der Dateiname mit vollständigem Datenpfad angegeben. Damit auf diesen zugegriffen werden kann, muss er allerdings in der Konfigurationsdatei des MySQL/MariaDB Servers (z.B. `/etc/mysql/my.cnf`) als Datenverzeichnis angemeldet werden. Diese Angabe ist zur Laufzeit des Servers nicht veränderbar:

```
210 # secure-file-priv= NULL
211 secure-file-priv= /var/lib/mysql/uploads/
```

Der Parameter `secure-file-priv` enthält die Verzeichnisse, aus denen der Server Dateien importieren oder in denen er Dateien exportieren darf.



Hinweis: Diese Möglichkeit ist in einer Produktivumgebung mit Vorsicht zu genießen! Besser ist es, diese Variable mit dem Wert `NULL` zu belegen, damit keine Dateien vom Server im- oder exportiert werden können.

Wie man diese Variable für einen SQL-Injection-Angriff ausnutzen kann, finden Sie gut in [\[Mik10\]](#) dargestellt.

7.3 Daten kopieren



Aus eins mach zwei: Daten aus Tabellen kopieren

- Vertiefendes

- Daten aus bestehenden Tabellen mit INSERT INTO ... SELECT übernehmen

Beim Anlegen temporärer Tabellen hatten wir die Situation (siehe [Abschnitt 5.3.8](#)), dass man für eine Marketingaktion nur Auszüge der Daten aus der Tabelle *kunde* benötigte.

Wir möchten alle Kunden, die zur Familie *Beutlin* gehören, separat auswerten. Dazu sind zwei Schritte notwendig: das Erstellen der temporären Tabelle und das Kopieren aller Kunden mit Nachnamen *Beutlin*.

Als Datenquelle für das Kopieren verwenden wir den Befehl SELECT, der mir Informationen aus Tabellen ermittelt und zur Verfügung stellt.



SQL:2023, MySQL/MariaDB, PostgreSQL

```
INSERT INTO tabellenname_ziel [(spaltenlis  
te)]
```

```
SELECT auswahl
```

```
;
```

T-SQL

```
SELECT auswahl_der_spalten
```

```
INTO tabellenname_ziel [(spaltenliste)]
```

```
FROM
```

```
SELECT auswahl_der_datenquelle
```

```
;
```

In unserem Fall ist die *auswahl* eine einfache Abfrage auf den Nachnamen:

```
1 mysql> CREATE TEMPORARY TABLE kunde_beutlin  
LIKE kunde;
```

```

2
3  mysql> INSERT INTO kunde_beutlin
4      -> SELECT *
5      -> FROM kunde
6      -> WHERE nachname='Beutlin'
7      -> ;
8  Query OK, 2 rows affected (0.04 sec)
9  Records: 2 Duplicates: 0 Warnings: 0
10
11  mysql> SELECT * FROM kunde_beutlin;
12  +-----+-----+-----+-----+
13  | kunde_id | nachname | vorname | rechnung
   _adresse_id | liefer_adresse_id | [...] |
14  +-----+-----+-----+-----+
15  | 2 | Beutlin | Frodo | 2 | NULL | [...] |
16  | 3 | Beutlin | Bilbo | 2 | 4 | [...] |
17  +-----+-----+-----+-----+
   +-----+-----+-----+-----+

```

Den Aufbau des Befehls SELECT werden wir noch später in [Teil IV](#) behandeln. Aber mit ein wenig Phantasie ist klar, wie hier der SELECT arbeitet. Das Sternchen hinter dem SELECT gibt an, dass alle Spalten der Tabelle ausgegeben werden sollen.

Da die Tabelle kunde und die temporäre Tabelle kunde_beutlin die gleiche Anzahl, Reihenfolge und Typisierung der Spalten haben, passt das SELECT * genau zu kunde_beutlin, und die Daten können sofort importiert werden.

Sind die Tabellen nicht exakte strukturelle Kopien, so muss der SELECT so aufgebaut werden, dass das Ergebnis in die Tabellenstruktur oder *spaltenliste* der Zieltabelle passt.

Beispiel: Hier werden nur die Vor- und Nachnamen aller Kunden in eine neue temporäre Tabelle kopiert.

```

1  mysql> CREATE TEMPORARY TABLE kunde_namen
LIKE kunde;
2  Query OK, 0 rows affected (0.04 sec)
3
4  mysql> INSERT INTO kunde_namen (vorname, n
achname)
5      -> SELECT vorname, nachname
6      -> FROM kunde
7      -> ;
8  Query OK, 5 rows affected (0.05 sec)
9  Records: 5 Duplicates: 0 Warnings: 0
10
11 mysql> SELECT * FROM kunde_namen;
12 +-----+-----+-----+-----+
-----[...]-+
13 | kunde_id | nachname | vorname | rechnung
_adresse_id | liefer_adresse_i[...] |
14 +-----+-----+-----+-----+
-----[...]-+
15 | 1 | Beutlin | Bilbo | NULL | NUL[...] |
16 | 2 | Beutlin | Frodo | NULL | NUL[...] |
17 | 3 | Earendil | Elrond | NULL | NUL[.
..] |
18 | 4 | Gamdschie | Samweis | NULL | NUL[...]
] |
19 | 5 | Telcontar | Elessar | NULL | NUL[...]
] |
20 +-----+-----+-----+-----+
-----[...]-+

```

¹ Siehe [\[Sha05\]](#).

² Falls nicht, lesen Sie trotzdem weiter ;-).

³ Siehe [Abschnitt 9.3](#).

⁴ Eine ausführliche Darstellung finden Sie unter [\[Cor22\]](#).

⁵ Linux bzw. Unixoiden: / und Windows: \

⁶ Um SQL-Injection-Angriffen vorzubeugen, sollten Sie sich mit `Prepare()` vertraut machen.

⁷ Unter Windows 11 lautet der Pfad: `C:\ProgramData\MySQL\MySQL Server 8.0\Uploads` (siehe Hinweis in [Abschnitt 4.3.1](#)).

TEIL III

Datenbank ändern

8 Datenbank und Tabellen umbauen

8.1 Eine Datenbank ändern



Erstaunlich, wie wenig man von einer Datenbank ändern kann.

- Grundkurs
 - ALTER SCHEMA, ALTER DATABASE
 - Ein neuer Zeichensatz
 - Eine neue Sortierung
- Vertiefendes: SHOW CREATE TABLE

Beim Anlegen der Datenbank gab es nur vier Angaben, die für eine Datenbank festgelegt werden konnten: Datenbankname, Zeichensatz, Sortierung und ob verschlüsselt werden soll. Der Datenbankname kann nicht geändert werden, aber Zeichensatz, Sortierung und der Verschlüsselungsschalter. Zusätzlich kann seit MySQL 8.0.22 eine Datenbank vor Veränderung geschützt werden, indem man die Eigenschaft `READ ONLY` setzt.

In SQL:2023 gibt es keinen Befehl zum Ändern einer Datenbank. Eine Änderung kann hier nur durch Anlegen einer neuen Datenbank mit den geänderten Parametern erfolgen. Anschließend werden die Datenbankobjekte einzeln in die neue Datenbank überführt.



MySQL/MariaDB

```
ALTER {DATABASE | SCHEMA} datenbankname
  [[DEFAULT] CHARACTER SET [=] zeichensatz]
  [[DEFAULT] COLLATE [=] sortierung]
  [[DEFAULT] ENCRYPTION [=] {'Y'|'N'}]
  [READ ONLY [=] {DEFAULT|0|1}]
;
```

Wie Sie die zur Verfügung stehenden Zeichensätze ermitteln, entnehmen Sie [Abschnitt 5.2.4](#) und die Sortierungen ab [Abschnitt 5.2.5](#).

Spannend sind jetzt folgende Fragen:

- Werden die Tabellenoptionen der schon vorhandenen Tabellen mit geändert oder werden nur neue Tabellen mit den geänderten Datenbankoptionen erstellt?
- Werden die Zeichensätze konvertiert oder nur anders interpretiert?
- Werden die Indizes automatisch neu gebildet oder muss man dies manuell tun?

Die erste Frage lässt sich leicht durch Ausprobieren beantworten:


```

1  mysql> SHOW CREATE DATABASE oshop\G
2  ***** 1. row *****
3          Database: oshop
4  Create Database: CREATE DATABASE 'oshop' /
*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE
utf8mb4_german2_ci */ /*!80016 DEFAULT ENCRYPT
ION='N' */
5
6  mysql> ALTER DATABASE oshop DEFAULT CHARAC
TER SET latin1;
7
8  mysql> SHOW CREATE DATABASE oshop\G
9  ***** 1. row *****
10         Database: oshop
11  Create Database: CREATE DATABASE 'oshop' /
*!40100 DEFAULT CHARACTER SET latin1 */ /*!800
16 DEFAULT ENCRYPTION='N' */

```



Hinweis: Mit `/*!401000 [...] */` wird ein bedingter Kommentar (Conditional Comment) programmiert. Hier: Bei einem Server der Version kleiner 4.1 ist dies ein Kommentar, ansonsten nicht. So können abwärtskompatible Befehle erstellt werden.

Mit `SHOW CREATE DATABASE` konnten wir uns davon überzeugen, dass die Datenbank einen neuen DEFAULT-Zeichensatz hat. Wie sieht es aber für die Tabellen aus? Hier hilft der gute alte Bekannte `SHOW CREATE TABLE`:

```

1  mysql> SHOW CREATE TABLE kunde\G
2  ***** 1. row *****
*****

```

```

3           Table: kunde
4 Create Table: CREATE TABLE 'kunde' (
5     'kunde_id' int unsigned NOT NULL AUTO_INCREMENT,
6     'nachname' varchar(255) COLLATE utf8mb4_german2_ci NOT NULL DEFAULT '',
7     'vorname' varchar(255) COLLATE utf8mb4_german2_ci NOT NULL DEFAULT '',
8     [...]
9 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_german2_ci

```

In der [Zeile 9](#) wird der Zeichensatz mit utf8mb4 angegeben. Die Tabellen behalten somit ihren Zeichensatz und werden nicht automatisch in den neuen konvertiert.

Bei neuen Tabellen sieht das anders aus:

```

1  mysql> CREATE TABLE bla (name VARCHAR(255)
2  );
3  mysql> SHOW CREATE TABLE bla\G
4  ***** 1. row *****
5           Table: bla
6 Create Table: CREATE TABLE 'bla' (
7     'name' varchar(255) DEFAULT NULL
8 ) ENGINE=InnoDB DEFAULT CHARSET=latin1

```

Der Zeichensatz hier ist der neu eingestellte der Datenbank: latin1. Wenn die alten Tabellen bei dem alten Zeichensatz bleiben, wird auch die letzte Frage beantwortet: Die Indizes brauchen nicht neu gebildet werden.



Aufgabe 8.1: Ermitteln Sie selber, ob eine geänderte Sortierung in den Tabellen übernommen wird.

8.2 Eine Datenbank löschen



Nothing lasts forever. Mit einem Handstreich alles löschen.

- Grundkurs: Löschen mit `DROP DATABASE`
- Vertiefendes: Was beim Löschen so alles passiert

Das Löschen einer Datenbank mit `DROP DATABASE` ist erfrischend einfach. Die Datenbank und alle ihre Elemente werden aus dem DBMS entfernt. Dazu gehören beispielsweise die Tabellen, Ansichten, Prozeduren und Trigger.



SQL:2023

```
DROP DATABASE datenbankname
```

```
;
```

MySQL/MariaDB

```
DROP {SCHEMA|DATABASE} [IF EXISTS] datenbankname
```

```
;
```

PostgreSQL, T-SQL

```
DROP DATABASE [IF EXISTS] datenbankname
```

```
;
```

Damit wir uns nichts kaputt machen, legen wir eine neue Datenbank an und überprüfen ihre Existenz.

```

1  mysql> CREATE DATABASE wurstbrot;
2
3  mysql> SHOW DATABASES;
4  +-----+
5  | Database |
6  +-----+
7  | information_schema |
8  | mysql |
9  | oshop |
10 | performance_schema |
11 | sys |
12 | wurstbrot |
13 +-----+
14 6 rows in set (0.00 sec)

```

Jetzt löschen wir sie wieder:

```

1  mysql> DROP DATABASE wurstbrot;
2  mysql> SHOW DATABASES;
3  +-----+
4  | Database |
5  +-----+
6  | information_schema |
7  | mysql |
8  | oshop |
9  | performance_schema |
10 | sys |
11 +-----+
12 5 rows in set (0.00 sec)

```

Nun hat MySQL aus dem Unterverzeichnis `wurstbrot` alle Dateien mit den Endungen `.BAK`, `.DAT`, `.HSH`, `.MRG`, `.MYD`, `.ISD`, `.MYI`, `.db` und `.frm` gelöscht. Ebenso wird dort die Datei `db.opt` gelöscht.

Abschließend wird das Verzeichnis selbst gelöscht. Sollten in dem Verzeichnis noch andere Dateien liegen, kann das

Verzeichnis nicht gelöscht werden. Es erscheint eine entsprechende Fehlermeldung:

```
1  mysql> DROP DATABASE oshop;  
2  ERROR 1010 (HY000): Error dropping database (can't rmdir './osshop/', [...])
```

Bei einigen DBMS können Sie die Datenbank nicht löschen, wenn diese gerade in einer Sitzung verwendet wird. Hier ein Beispiel in T-SQL:

```
1  1> CREATE DATABASE wurstbrot;  
2  2> GO  
3  1> USE wurstbrot;  
4  2> GO  
5  Changed database context to 'wurstbrot'.  
6  1> DROP DATABASE wurstbrot;  
7  2> GO  
8  Msg 3702, Level 16, State 3, Server aa7ff192eac9, Line 1  
9  Cannot drop database "wurstbrot" because it is currently in use.  
10 1> USE master;  
11 2> GO  
12 Changed database context to 'master'.  
13 1> DROP DATABASE wurstbrot;  
14 2> GO  
15 1>
```

Der Löschversuch in [Zeile 9](#) scheitert mit der entsprechenden Fehlermeldung. Erst der Wechsel zu einer anderen Datenbank (siehe [Zeile 12](#)) gibt die Datenbank wieder frei und sie kann daher auch gelöscht werden, falls nicht noch andere Sitzungen auf sie zugreifen.



Hinweis: Bevor Sie eine Datenbank löschen, sollten Sie unbedingt eine Sicherheitskopie der Datenbank gemacht haben! Nach dem DROP ist nämlich alles weg! Es erfolgt keine Rückmeldung, und es gibt auch kein *undo*.

8.3 Eine Tabelle ändern



Tabellen müssen sich anpassen lassen. Daher frei nach dem Motto des Zweiten Vatikanischen Konzils: *Tabula semper reformanda est*.

■ Grundkurs

- Tabelle mit `ALTER TABLE ... RENAME` umbenennen
- Spalte mit `ALTER TABLE ... ADD` hinzufügen
- Spaltenspezifikation mit `ALTER TABLE ... MODIFY` ändern
- Spalten mit `ALTER TABLE ... DROP` entfernen

■ Vertiefendes

- Umbenennen der Fremdschlüsselverweise
- Spalten einsortieren
- Was passiert bei Längenänderungen?
- Was passiert bei Zeichensatzänderungen?
- Von Zahl nach Zeichen konvertieren

- Wertebereich einer Zahl ändern
- Datum- und Uhrzeitwerte



Die Quelltexte dieses Kapitels stehen in der Datei `listing05.sql` (siehe [Listing 28.5](#), [Listing 28.45](#) und [Listing 28.30](#)).

Eine Tabelle kann in folgenden Eigenschaften nachträglich verändert werden:

- Name der Tabelle
- Spaltenspezifikation bzgl. des Namens, des Datentyps und der Zusätze
- Constraints
- Tabellenoptionen

Das ist so ziemlich alles, was eine Tabelle ausmacht. Je umfangreicher eine Tabellenänderung ist, desto eher sollten Sie darüber nachdenken, eine neue Tabelle anzulegen, die Datensätze einzeln zu lesen, umzubauen und in die neue hineinzuschreiben.

Der entscheidende Befehl zum Ändern einer Tabelle ist `ALTER TABLE`. Da dieser Befehl sehr umfangreich ist, wird er hier anhand von Fallbeispielen eingeführt. Dabei werden nicht alle Varianten des Befehls berücksichtigt.

8.3.1 Wie kann ich den Namen der Tabelle ändern?



MySQL/MariaDB, PostgreSQL

```
ALTER TABLE tabellenname_alt
    RENAME TO tabellenname_neu
;
```

T-SQL

```
EXEC sp_rename tabellenname_alt,
tabellenname_neu
;
```

Das Umbenennen ist schnell durchgeführt und überprüft.

```
1  mysql> SHOW TABLES;
2  +-----+
3  | Tables_in_oshop |
4  +-----+
5  | adresse |
6  | artikel |
7  [...]
8  | rechnung_position |
9  | warengruppe |
10 +-----+
11 mysql> ALTER TABLE adresse
12 -> RENAME TO wurstbrot;
13 mysql> SHOW TABLES;
14 +-----+
15 | Tables_in_oshop |
16 +-----+
17 | artikel |
18 | artikel_nm_lieferant |
19 [...]
20 | warengruppe |
21 | wurstbrot |
```



```
22  +-----+
23
24  mysql>
```

In [Zeile 21](#) können Sie sehen, dass die Tabelle adresse nun wurstbrot heißt. Stellt sich doch die Frage, was bei einem ALTER TABLE ... RENAME mit den Fremdschlüssel-Constraints passiert?

```
1  mysql> SHOW CREATE TABLE kunde\G
2  ***** 1. row *****
3          Table: kunde
4  Create Table: CREATE TABLE 'kunde' (
5  [...]
6  CONSTRAINT 'kunde_ibfk_1' FOREIGN KEY
('rechnung_adresse_id') REFERENCES 'wurstbrot
' ('adresse_id') ON DELETE RESTRICT ON UPDATE
CASCADE,
7  CONSTRAINT 'kunde_ibfk_2' FOREIGN KEY
('liefer_adresse_id') REFERENCES 'wurstbrot '
('adresse_id') ON DELETE SET NULL ON UPDATE CA
SCADE
8  ) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT C
HARSET=utf8mb4 COLLATE=utf8mb4_german2_ci
9
10 mysql> ALTER TABLE wurstbrot RENAME TO adr
esse;
```

Es haben diverse Fremdschlüssel auf eine Tabelle mit dem Namen adresse verwiesen. Eine Tabelle, die Fremdschlüssel auf adresse enthält, ist kunde. In [Zeile 6](#) ist zu sehen, dass die Verweise auf den neuen Namen wurstbrot aktualisiert wurden. Puh :-)



Hinweis: In SQL:2023 gibt es kein ALTER TABLE ... RENAME TO-Befehl. In den verschiedenen RDBMS-Implementierungen finden sich aber entsprechende Anweisungen wie beispielsweise in T-SQL die Systemprozedur sp_rename.

```
1      1> EXEC sp_rename 'adresse', 'wurs
tbrot';
2      2> GO
3      Caution: Changing any part of an o
bject name could break scripts and store
d procedures.
```

8.3.2 Wie kann ich eine Spalte hinzufügen?



SQL:2023, PostgreSQL, T-SQL

```
ALTER TABLE tabellenname
  ADD [COLUMN] spaltenspezifikation
;
```

MySQL/MariaDB

```
ALTER TABLE tabellenname
  ADD [COLUMN] spaltenspezifikation [{FIRST|AFTER}
spaltenname]
;
```

Die *spaltenspezifikation* in ALTER TABLE ... ADD entspricht der, die bei einem CREATE TABLE (siehe [Abschnitt 5.3](#)) möglich ist.

Wir wollen in der Tabelle kunde neben Nach- und Vornamen auch einen Firmennamen erfassen können.



Aufgabe 8.2: Wie lautet die Spaltenspezifikation für den Firmennamen? Begründen Sie Ihre Wahl.

Der aktuelle Stand der Tabelle kunde enthält keine Spalte zum Firmennamen. Dies könnten wir uns mit SHOW CREATE TABLE bestätigen lassen. Ein etwas weniger geschwätziger Befehl liefert uns auch alle notwendigen Informationen: DESCRIBE.

```
1  mysql> DESCRIBE kunde;
2  +-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+
3  | Field | Type | Null | Key | Default | Ex
[...] |
4  +-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+
5  | kunde_id | int unsigned | NO | PRI | NULL
L | au[...] |
6  | nachname | varchar(255) | NO | MUL | | [
...] |
7  | vorname | varchar(255) | NO | | | [...]
|
8  | rechnung_adresse_id | int unsigned | YES
| MUL | NULL | [...] |
9  | liefer_adresse_id | int unsigned | YES |
MUL | NULL | [...] |
10 | bezahlart | int unsigned | NO | | 0 | [.
..] |
11 | art | enum('unb','prv','gsch') | NO | |
unb | [...] |
12 | deleted | tinyint unsigned | NO | | 0 |
[...] |
13 +-----+-----+-----+-----+-----+-----+
```

```

-----+-----+-----+-----+---[...]--+
14  8 rows in set (0.00 sec)

```

Hinter dem Vornamen wird jetzt die Spalte firmenname eingefügt:

```

1  ALTER TABLE kunde
2  ADD firmenname VARCHAR(255) NOT NULL DEFAU
LT '' AFTER vorname
3  ;

```

In [Zeile 8](#) taucht unser Firmenname auf – genau da, wo wir ihn haben wollten: nach dem Vornamen. Wird weder FIRST noch AFTER verwendet, wird die neue Spalte hinter der letzten eingefügt.

```

1  mysql> DESCRIBE kunde;
2  +-----+-----+-----+-----+---[...]--+
-----+-----+-----+-----+---[...]--+
3  | Field | Type | Null | Key | Default | Ex
[...] |
4  +-----+-----+-----+-----+---[...]--+
-----+-----+-----+-----+---[...]--+
5  | kunde_id | int unsigned | NO | PRI | NUL
L | au[...] |
6  | nachname | varchar(255) | NO | MUL | | [
...] |
7  | vorname | varchar(255) | NO | | | [...]
|
8  | firmenname | varchar(255) | NO | | | [..
.] |
9  [...]
10 +-----+-----+-----+-----+---[...]--+
-----+-----+-----+-----+---[...]--+

```

Nachträglich eine neue NOT NULL-Spalte hinzuzufügen, kann problematisch sein, wenn die Tabelle nicht leer ist. Was soll

denn in die schon angelegten Zeilen hineingeschrieben werden, wenn jetzt eine neue Spalte daher kommt?

MySQL und MariaDB belegen die Spalten mit Standardwerten (siehe [Abschnitt 26.1.7](#)). Sinnvoller wäre es, sich selbst eigene Standardwerte zu überlegen und diese in die Spaltenspezifikation per DEFAULT einzubauen.



Aufgabe 8.3: Verändern Sie die Spaltenspezifikationen in den entsprechenden CREATE TABLE-Anweisungen so, dass den Spalten sinnvolle Standardwerte zugewiesen werden.

8.3.3 Wie kann ich die Spezifikation einer Spalte ändern?



SQL:2023

```
ALTER TABLE tabellenname
```

```
    ALTER COLUMN spaltenname SET DATA TYPE  
    datentyp
```

```
;
```

MySQL/MariaDB, PostgreSQL

```
ALTER TABLE tabellenname
```

```
    MODIFY [COLUMN] spaltenname spaltenspezifikation
```

```
;
```

T-SQL

```
ALTER TABLE COLUMN tabellenname spaltenname
spaltenspezifikation
;
```

In SQL:2023 kann nur der Datentyp auf diese Art verändert werden. Für Zusätze wie NOT NULL müssen jeweils eigene ALTER TABLE ... ALTER COLUMN-Varianten verwendet werden.

In MySQL oder MariaDB steht bei ALTER TABLE ... MODIFY hinter dem *spaltenname* die neue Spaltenspezifikation, die genauso wie im CREATE TABLE aufgebaut ist.

Wir haben bisher die Bezahlart über eine ganze Zahl kodiert. Wie bei der Kundenart wäre es aber aussagekräftiger, einen ENUM zu verwenden. Wir wollen die Bezahlarten per Rechnung, per Bankeinzug und per Nachname anbieten. Hier der aktuelle Zustand der Tabelle kunde:

```
1  mysql> DESCRIBE kunde\G
2  ***** 1. row *****
3  [...]
4  ***** 7. row *****
5  Field: bezahlart
6  Type: int unsigned
7  Null: NO
8  Key:
9  Default: 0
10 Extra:
11 ***** 8. row *****
12 [...]
```

Die Spalte bezahlart ist eine vorzeichenlose ganze Zahl mit

der Vorbelegung 0. Jetzt wird die Spalte angepasst:

```
1 ALTER TABLE kunde
2     MODIFY
3     bezahllart ENUM('rechnung', 'bankeinzug
4     ', 'nachname') DEFAULT 'rechnung';
```

Das Ergebnis ist selbsterklärend:

```
1 mysql> DESCRIBE kunde\G
2  ***** 1. row *****
3  [...]
4  ***** 7. row *****
5  Field: bezahllart
6  Type: enum('rechnung','bankeinzug',
7  'nachname')
8  Null: YES
9  Key:
10 Default: rechnung
11 Extra:
12 ***** 8. row *****
13 *****
14 [...]
```

So weit, so einfach. Die Änderung der Spaltenspezifikation ist in vielen Fällen ohne großes Nachdenken möglich. So wird die Änderung von einem VARCHAR(255) auf einen CHAR(50) keine Probleme bereiten, oder?



Hinweis: Werden durch Typ- oder Längenänderung mehr Bytes zur Speicherung der Daten benötigt, könnte es passieren, dass ein mit der Spalte verbundener Index seine maximale Länge überschreitet.

```

1  mysql> CREATE INDEX idx_strasse ON a
  dresse (strasse);
2
3  mysql> ALTER TABLE adresse MODIFY st
  rasse VARCHAR(2000);
4  ERROR 1071 (42000): Specified key wa
  s too long; max key length is 3072 bytes

```

8.3.4 Zeichenbasierte Spalten in der Länge verändern

Hat die neue Spaltenspezifikation eine geringere Länge als die alte, wurden in früheren Serverversionen die Daten, die nicht in die neue passen, abgeschnitten. In den aktuellen Versionen liefern MySQL und MariaDB eine Fehlermeldung (siehe [Zeile 2](#)). Als Beispiel begrenzen wir die Spalte `bezeichnung` in einer Kopie der Tabelle `artikel` auf zehn Zeichen:

```

1  mysql> ALTER TABLE bla MODIFY bezeichnung
  CHAR(10);
2  Query OK, 9 rows affected, 6 warnings (0.2
  4 sec)
3  ERROR 1406 (22001): Data too long for colu
  mn 'bezeichnung' at row 1

```

Es ist sinnvoll, die Längen der Werte in den Spalten vorab zu ermitteln:

```

1  mysql> SELECT MAX(CHAR_LENGTH(bezeichnung)
  ) FROM artikel;
2  +-----+
3  | MAX(CHAR_LENGTH(bezeichnung)) |
4  +-----+

```



```
5 | 13 |
6 +-----+
```

Dieser Befehl liefert den Wert 13. Um noch ein wenig Luft nach oben zu haben, wäre die Länge 20 sicherlich ausreichend.



Aufgabe 8.4: Überprüfen Sie auf einer Kopie von Artikel (verwenden Sie `CREATE TEMPORARY TABLE ... LIKE` und `INSERT INTO ... SELECT`), ob die Länge 20 ohne Warnungen übernommen wird.

Eine Verlängerung der maximalen Länge einer zeichenbasierenden Spalte ist ohne Datenverlust möglich.

8.3.5 Zeichensatz verändern

Das kann richtig kompliziert werden. Zunächst sollte man sich darüber im Klaren sein, dass es keinen Sinn hat, Zeichensätze wie Kyrillisch (cp1251) nach Arabisch (cp1265) zu konvertieren. Aber die Konvertierungen zwischen `latin1`, `utf8mb4` und `cp850` sind schon wahrscheinlich.

Das Konvertieren der Daten sollte meines Erachtens nicht in den Tabellen durch einen SQL-Befehl erfolgen. Ich empfehle, die Daten zu exportieren, sie mit einem Konvertierungsprogramm wie beispielsweise unter Linux mit `iconv` oder `recode` umzuwandeln und sie in die geänderte Tabelle zu importieren.

Der Vorteil dieses Vorgehens ist, dass Sie jeden einzelnen Schritt unter Kontrolle haben und die Ergebnisse überprüfen können. Lassen Sie den Datenbankserver dies tun, erhalten Sie

ggf. Warnungen, aber die Daten sind dann vielleicht schon verfälscht.

8.3.6 Zeichenbasierte Spalten in numerische Spalten verändern

Weil wir eine intensive Auswertung über die örtliche Verteilung unserer Kunden mithilfe der Postleitzahl vornehmen wollen, werden wir in einer Kopie der Tabelle `adresse` die Postleitzahl in ein numerisches Feld umwandeln. Vergleiche und Sortierungen lassen sich dann erheblich schneller durchführen.

```
1  mysql> SELECT * FROM wurstbrot;
2  +-----+-----+-----+-----+-----+-----+
3  | adresse_id | strasse | hnr | lkz | plz |  |
4  | ort | deleted |
5  +-----+-----+-----+-----+-----+-----+
6  | 1 | Beutelhaldenweg | 5 | AL | 67676 | H
7  obbingen | 0 |
8  | 2 | Beutelhaldenweg | 1 | AL | 67676 | H
9  obbingen | 0 |
10 | 3 | Auf der Feste | 1 | GO | 54786 | Min
11 as Tirith | 0 |
12 | 4 | Letztes Haus | 4 | ER | 87567 | Bruc
13 htal | 0 |
14 | 5 | Baradur | 1 | MO | 62519 | Lugburz |
15 0 |
16 | 10 | Hochstrasse | 4a | DE | 44879 | Boc
17 hum | 0 |
18 | 11 | Industriegebiet | 8 | DE | 44878 |
19 Bochum | 0 |
20 +-----+-----+-----+-----+-----+-----+
```

```

--+-----+-----+-----+
13  mysql> ALTER TABLE wurstbrot MODIFY plz IN
T;
14  Query OK, 7 rows affected (0.00 sec)
15  Records: 7 Duplicates: 0 Warnings: 0
16
17  mysql> SELECT * FROM wurstbrot;
18  +-----+-----+-----+-----+-----+-----+
--+-----+-----+-----+-----+-----+-----+
19  | adresse_id | strasse | hnr | lkz | plz |
   | ort | deleted |
20  +-----+-----+-----+-----+-----+-----+
--+-----+-----+-----+-----+-----+-----+
21  | 1 | Beutelhaldenweg | 5 | AL | 67676 | H
   | obbingen | 0 |
22  | 2 | Beutelhaldenweg | 1 | AL | 67676 | H
   | obbingen | 0 |
23  | 3 | Auf der Feste | 1 | GO | 54786 | Min
   | as Tirith | 0 |
24  | 4 | Letztes Haus | 4 | ER | 87567 | Bruc
   | htal | 0 |
25  | 5 | Baradur | 1 | MO | 62519 | Lugburz |
   | 0 |
26  | 10 | Hochstrasse | 4a | DE | 44879 | Boc
   | hum | 0 |
27  | 11 | Industriegebiet | 8 | DE | 44878 |
   | Bochum | 0 |
28  +-----+-----+-----+-----+-----+-----+
--+-----+-----+-----+-----+-----+-----+

```



Hinweis: Die Daten werden hier problemlos automatisch konvertiert, weil es keine führende 0 gibt.

Probleme gäbe es nur, wenn die ermittelte Zahl nicht in den Zieldatentyp passt:

```
1 mysql> ALTER TABLE wurstbrot MODIFY plz TINYINT;  
2 ERROR 1264 (22003): Out of range value for column 'plz' at row 1
```

Was aber, wenn die Werte nicht nur numerisch sind?
Konvertieren wir die Hausnummer in eine numerische Spalte:

```
1 mysql> ALTER TABLE wurstbrot MODIFY hnr INT;  
2 ERROR 1265 (01000): Data truncated for column 'hnr' at row 6
```

8.3.7 Numerische Spalten im Wertebereich verändern

Wird der Wertebereich bei ganzzahligen Datentypen erweitert (z.B. INT nach BIGINT), ergeben sich normalerweise keine Probleme.

Ein ALTER TABLE von negativen Werten nach UNSIGNED wird mit einer Fehlermeldung abgebrochen. Konvertierungen von DOUBLE nach FLOAT bzw. DECIMAL nach DOUBLE oder FLOAT werden unter Ausgabe einer Warnung gerundet. Dabei gehen Nachkommastellen verloren, aber nicht der Wert der Zahl als solcher.^{[1](#)}

8.3.8 Datum- oder Zeitspalten verändern

Intern werden Datums- und Zeitangaben als ganzzahlige Werte oder Strings abgespeichert. Insofern erwarte ich keine Konvertierungsprobleme. Aber *schau'n mer mal*, wie schon der

Kaiser sagte: Legen wir zuerst eine temporäre Tabelle an, fügen einen Testdatensatz ein und konvertieren die Spalte zu BIGINTs:

```
1  mysql> CREATE TEMPORARY TABLE wurstbrot (
2      -> a DATETIME,
3      -> b TIME,
4      -> c YEAR
5      -> );
6
7  mysql> INSERT INTO wurstbrot (a, b, c)
8      -> VALUES ('2012-03-24 14:57:00', '14:
57:00', '2012');
9
10 mysql> ALTER TABLE wurstbrot
11     -> MODIFY a BIGINT,
12     -> MODIFY b BIGINT,
13     -> MODIFY c BIGINT
14     -> ;
15
16 mysql> SELECT * FROM wurstbrot;
17 +-----+-----+-----+
18 | a | b | c |
19 +-----+-----+-----+
20 | 20120324145700 | 145700 | 2012 |
21 +-----+-----+-----+
```

Die Konvertierung scheint keine Probleme gemacht zu haben. Konsequenterweise liefert eine entsprechende *Rückkonvertierung* auch wieder die ursprünglichen Werte:

```
1  mysql> ALTER TABLE wurstbrot
2      -> MODIFY a DATETIME,
3      -> MODIFY b TIME,
4      -> MODIFY c YEAR
5      -> ;
6
7  mysql> SELECT * FROM wurstbrot;
```

```

8  +-----+-----+-----+
9  | a | b | c |
10 +-----+-----+-----+
11 | 2012-03-24 14:57:00 | 14:57:00 | 2012 |
12 +-----+-----+-----+

```

Der gleiche Effekt kann auch bei der Konvertierung in einen zeichenbasierenden Datentyp beobachtet werden:

```

1  mysql> ALTER TABLE wurstbrot
2      -> MODIFY a VARCHAR(255),
3      -> MODIFY b VARCHAR(255),
4      -> MODIFY c VARCHAR(255)
5      -> ;
6
7  mysql> SELECT * FROM wurstbrot;
8  +-----+-----+-----+
9  | a | b | c |
10 +-----+-----+-----+
11 | 2012-03-24 14:57:00 | 14:57:00 | 2012 |
12 +-----+-----+-----+
13
14 mysql> ALTER TABLE wurstbrot
15      -> MODIFY a DATETIME,
16      -> MODIFY b TIME,
17      -> MODIFY c YEAR
18      -> ;
19
20 mysql> SELECT * FROM wurstbrot;
21 +-----+-----+-----+
22 | a | b | c |
23 +-----+-----+-----+
24 | 2012-03-24 14:57:00 | 14:57:00 | 2012 |
25 +-----+-----+-----+

```

Die Angaben werden einfach als Zeichenkette dargestellt. Zwar sind jetzt keine Datums- und Zeitoperationen mehr direkt möglich, aber der Wert ist erhalten geblieben, und mit

Konvertierungsfunktionen wie DATE () lassen sich diese auch weiterverarbeiten.

8.3.9 Wie kann ich aus einer Tabelle Spalten entfernen?



SQL:2023

```
ALTER TABLE tabellenname DROP COLUMN spaltenname
[RESTRICT|CASCADE] ;
```

MySQL/MariaDB, PostgreSQL

```
ALTER TABLE tabellenname DROP [COLUMN]
spaltenname ;
```

T-SQL

```
ALTER TABLE tabellenname DROP COLUMN spaltenname
;
```

Mit ALTER TABLE ... DROP werden Spalten aus den Tabellen entfernt. Dabei gehen natürlich auch alle darin enthaltenen Werte verloren.

Aus einer Kopie von Adresse möchte ich diverse Spalten entfernen, da ich sie nicht brauche. Zuerst der aktuelle Zustand:

```
1  mysql> DESCRIBE wurstbrot;
2  +-----+-----+-----+-----+-----+
---+-----+-----+
3  | Field | Type | Null | Key | Default | Ex
tra |
4  +-----+-----+-----+-----+-----+---
```

```

----+-----+-----+
  5 | adresse_id | int unsigned | NO | | 0 | N
ULL |
  6 | strasse | varchar(255) | NO | | | NULL |
  7 | hnr | varchar(255) | NO | | | NULL |
  8 | lkz | char(2) | NO | | | NULL |
  9 | plz | char(9) | NO | | | NULL |
 10 | ort | varchar(255) | NO | | | NULL |
 11 | deleted | tinyint unsigned | NO | | 0 |
NULL |
 12 +-----+-----+-----+-----+-----+
----+-----+-----+

```

Und weg damit:

```

1 ALTER TABLE wurstbrot
2     DROP deleted, DROP strasse, DROP ort;

```

Nur noch mal nachschauen, ob die Spalten wirklich entfernt wurden:

```

1 mysql> DESCRIBE wurstbrot;
2 +-----+-----+-----+-----+-----+
-----+-----+
  3 | Field | Type | Null | Key | Default | Ex
tra |
  4 +-----+-----+-----+-----+-----+
-----+-----+
  5 | adresse_id | int unsigned | NO | | 0 | N
ULL |
  6 | hnr | varchar(255) | NO | | | NULL |
  7 | lkz | char(2) | NO | | | NULL |
  8 | plz | char(9) | NO | | | NULL |
  9 +-----+-----+-----+-----+-----+
-----+-----+

```

In SQL:2023 hat man die Möglichkeit, mithilfe der Optionen RESTRICT und CASCADE zu steuern, was mit Daten passieren

soll, welche mit den zu löschenden Spalten – beispielsweise über einen Constraint – verknüpft sind. Bei RESTRICT – der Default – wird das Löschen verweigert, und bei CASCADE werden alle verknüpften Daten ebenfalls gelöscht. MySQL und MariaDB verhalten sich wie die Option RESTRICT.



Hinweis: Sie erhalten keine Warnung oder Rückfrage. Die Spalten und ihre Inhalte sind . . . weg!

8.4 Eine Tabelle löschen



Requiescat in pace. Eine Tabelle weniger.

- Grundkurs: Tabelle mit DROP TABLE löschen
- Vertiefendes
 - Löschreihenfolge bei Constraints
 - CASCADE und RESTRICT

Wie das Löschen der Datenbank ist auch das von Tabellen mit DROP TABLE einfach.



SQL:2023

```
DROP TABLE tabellenname [RESTRICT|CASCADE] ;
```

MySQL/MariaDB, PostgreSQL, T-SQL

```
DROP TABLE [IF EXISTS] tabellenname  
[RESTRICT|CASCADE] ;
```



Hinweis: Die Tabellen werden ohne Rückfrage oder Warnung endgültig entfernt. Es gibt kein Undo, um die Tabellen wieder herzustellen.

8.4.1 Einfach löschen

Schauen wir uns an, was wir an Tabellen haben:

```
1  mysql> SHOW TABLES;
2  +-----+
3  | Tables_in_oshop |
4  +-----+
5  | adresse |
6  | artikel |
7  | artikel_nm_lieferant |
8  | artikel_nm_warengruppe |
9  | bank |
10 | bankverbindung |
11 | bestellung |
12 | bestellung_position |
13 | bild |
14 | kunde |
15 | lieferant |
16 | rechnung |
17 | rechnung_position |
18 | warengruppe |
19 +-----+
20 14 rows in set (0.00 sec)
```

Jetzt löschen wir nicht nur eine, sondern gleichmehrere Tabellen.

```

1  mysql> DROP TABLE
2      > rechnung_position, rechnung, bestell
ung_position,bestellung
3      > ;
4
5  mysql> SHOW TABLES;
6  +-----+
7  | Tables_in_oshop |
8  +-----+
9  | adresse |
10 | artikel |
11 | artikel_nm_lieferant |
12 | artikel_nm_warengruppe |
13 | bank |
14 | bankverbindung |
15 | bild |
16 | kunde |
17 | lieferant |
18 | warengruppe |
19 +-----+
20 10 rows in set (0.00 sec)

```

Was passiert eigentlich mit den Indizes der entsprechenden Tabellen? Diese werden mit gelöscht. Durch das Löschen einer Tabelle werden alle damit verbundenen Indizes, Constraints etc. gelöscht.

8.4.2 Was bedeuten CASCADE und RESTRICT?



Aufgabe 8.5: Was für eine Fehlermeldung erwarte ich, wenn die Tabellen in folgender Reihenfolge zu löschen versucht werden: rechnung, rechnung_position?

Besteht ein Constraint zwischen zwei Tabellen – z.B. eine Primär-Fremdschlüsselbeziehung –, wird dieser verletzt, wenn man beispielsweise die Primärschlüsseltabelle löscht.

Wird die Option RESTRICT angegeben, scheitert der Löschversuch. Wird die Option CASCADE angegeben, werden alle verbundenen Datenobjekte wie Constraints oder Ansichten auch gelöscht. Schauen wir uns das mal in PostgreSQL² an:

```
1  oshop=# \dt
2  List of relations
3  Schema | Name | Type | Owner
4  -----+-----+-----+-----
5  public | adresse | table | postgres
6  public | artikel | table | postgres
7  public | artikel_nm_lieferant | table | po
stgres
8  public | artikel_nm_warengruppe | table |
postgres
9  public | bank | table | postgres
10 public | bankverbindung | table | postgres
11 public | bestellung | table | postgres
12 public | bestellung_position | table | pos
tgres
13 public | bild | table | postgres
14 public | kunde | table | postgres
15 public | lieferant | table | postgres
16 public | rechnung | table | postgres
17 public | rechnung_position | table | postg
res
18 public | warengruppe | table | postgres
19 (14 rows)
20
21 oshop=# DROP TABLE kunde RESTRICT;
22 ERROR: cannot drop table kunde because oth
er objects depend on it
```

```
23  DETAIL: constraint bankverbindung_kunde_id
    _fkey on table bankverbindung depends on table
    kunde
24  constraint bestellung_kunde_id_fkey on tab
    le bestellung depends on table kunde
25  constraint rechnung_kunde_id_fkey on table
    rechnung depends on table kunde
26  HINT: Use DROP ... CASCADE to drop the dep
    endent objects too.
27
28  oshop=# DROP TABLE kunde CASCADE;
29  NOTICE: drop cascades to 3 other objects
30  DETAIL: drop cascades to constraint bankve
    rbindung_kunde_id_fkey on table bankverbindung
31  drop cascades to constraint bestellung_kun
    de_id_fkey on table bestellung
32  drop cascades to constraint rechnung_kunde
    _id_fkey on table rechnung
```

¹ Natürlich lassen sich perverse Beispiele von DECIMAL-Zahlen erstellen, die bei einer Konvertierung nach FLOAT oder INT den Zahlenwert verlieren.

² In MySQL und MariaDB wird die Option CASCADE ignoriert und führt somit auch zu Fehlermeldungen: ERROR 3730 (HY000): Cannot drop table 'kunde' referenced by a foreign key constraint 'rechnung_ibfk_1' on table 'rechnung'. bei MySQL und ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails in MariaDB.

9 Werte in Tabellen verändern

9.1 WHERE-Klausel



We Want You! Mit WHERE nur die Zeilen auswählen, die wir wollen.

- Grundkurs
 - Bedingung und Wahrheitswerte
 - Verknüpfung von Teilbedingungen
- Vertiefendes: Groß- und Kleinschreibung



Die Quelltexte dieses Kapitels stehen in der Datei `listing06.sql` (siehe [Listing 28.6](#), [Listing 28.46](#) und [Listing 28.31](#)).

Wir werden in den nachfolgenden Kapiteln Befehle kennenlernen, die Tabelleninhalte auswerten, verändern und löschen. Diese Befehle werden mal auf alle Zeilen der Tabelle angewendet, mal auf eine Teilmenge.

Damit SQL unterscheiden kann, auf welche Teilmenge der

Damit SQL eingeschränkten Zugriff auf Datensätze ermöglicht, wird der Befehl eingeschränkt, müssen wir für die Zeilen Bedingungen formulieren können.



Definition 34: Bedingung

Eine *Bedingung* (engl: condition) ist eine Frage, die der Computer mit TRUE oder FALSE beantworten muss.

In SQL kann ein boolescher¹ Ausdruck auch noch einen dritten Wert annehmen: UNKNOWN. Dieser tritt in der Regel auf, wenn wir mit NULL-Werten vergleichen.

Für die Befehle UPDATE (siehe [Abschnitt 9.2](#)), DELETE (siehe [Abschnitt 9.3](#)) und SELECT (siehe [Teil IV](#)) werden die Bedingungen durch das Schlüsselwort WHERE eingeleitet.



SQL:2023

MySQL/MariaDB, PostgreSQL, T-SQL

```
{UPDATE | DELETE | SELECT} befehlsinhalt  
WHERE  
  bedingung  
;
```



Definition 35: WHERE-Klausel

Die *WHERE-Klausel* schränkt die voranstehende Operation auf die Zeilen ein, für welche die Bedingung TRUE ist.

9.1.1 Wie formuliere ich eine einfache Bedingung?

Es gibt zwei *sehr* einfache Bedingungen².

- 1 `SELECT * FROM artikel WHERE TRUE;`
- 2 `SELECT * FROM artikel WHERE FALSE;`



Aufgabe 9.1: Der Befehl `SELECT` mir liefert den Inhalt einer Tabelle. Welche Ausgabe erwarte ich bei [Zeile 1](#) und welche bei [2](#)?

Verwenden Sie in Ihrer Begründung die [Definition 35](#).

So triviale Bedingungen kommen normalerweise nicht vor. Vielmehr wollen wir Bedingungen bauen, die die inhaltlichen Eigenschaften einer Zeile untersucht: Ist `delete` auch mit 0 gefüllt? Ist der Kunde volljährig? Kommen die Kunden aus einem bestimmten Postleitzahlenbereich? Etc.

Dazu muss man Spalteninhalte miteinander oder gegen Konstanten oder Ausdrücke vergleichen. Eine Liste der in SQL möglichen Vergleichsoperatoren finden Sie in [Abschnitt 26.3.1](#).



Aufgabe 9.2: Führen Sie folgende Befehle aus und interpretieren Sie die Ergebnisse:

```
SELECT * FROM artikel WHERE artikel_id = 3010;
SELECT * FROM artikel WHERE artikel_id != 3010;
SELECT * FROM artikel WHERE einzelpreis < 5.0;
```



```
SELECT * FROM artikel WHERE einzelpreis <=
5.0;
SELECT * FROM artikel WHERE einzelpreis >
5.0;
SELECT * FROM artikel WHERE einzelpreis >=
5.0;
SELECT * FROM artikel WHERE einzelpreis BE
TWEEN 1.00 AND 15.00;
SELECT * FROM artikel WHERE einzelpreis NO
T BETWEEN 1.00 AND 15.00;
SELECT * FROM artikel WHERE artikel_id IN
(3001, 7856, 9015);
SELECT * FROM artikel WHERE artikel_id NOT
IN (3001, 7856, 9015);
SELECT * FROM artikel WHERE bezeichnung LI
KE 'Tinte%';
SELECT * FROM artikel WHERE bezeichnung =
'Tinte%';
SELECT * FROM kunde WHERE liefer_adresse_i
d = NULL;
SELECT * FROM kunde WHERE liefer_adresse_i
d IS NULL;
SELECT * FROM kunde WHERE liefer_adresse_i
d IS NOT NULL;
```

Aufgabe 9.3: Ermitteln Sie die passende WHERE-Klausel zum SELECT für folgende Anforderungen:

- a) Geben Sie den Kunden mit dem Namen 'Beutlin' aus.
- b) Geben Sie alle Kunden, die nicht 'Beutlin' heißen, aus.
- c) Geben Sie alle Kunden mit einer Kundennummer kleiner als 4 aus.
- d) Geben Sie alle Kunden mit einer Kundennummer größer oder gleich 4 aus.

- e) Geben Sie alle Kunden mit einer Kundennummer von 2 bis 4 aus.
- f) Geben Sie alle Kunden mit einer Kundennummer kleiner 2 oder größer 4 aus.
- g) Geben Sie alle Kunden, deren Nachname ein 'n' enthalten, aus.

9.1.2 Wird zwischen Groß- und Kleinschreibung unterschieden?

Ein Versuch sagt mehr als 1000Worte³:

```
1  mysql> CREATE TEMPORARY TABLE wurstbrot (
2      -> name VARCHAR(255)
3      -> );
4
5  mysql> INSERT INTO wurstbrot VALUES
6      -> ('Jaqueline')
7      -> ,('Kevin')
8      -> ,('kevin')
9      -> ,('jaqueline');
10
11 mysql> SELECT * FROM wurstbrot WHERE name
12 = 'kevin';
13 +-----+
14 | name |
15 +-----+
16 | Kevin |
17 | kevin |
18 +-----+
```

Offensichtlich wird in MySQL und MariaDB standardmäßig nicht zwischen Groß- und Kleinschreibung unterschieden:⁴.

Möchte man zwischen Groß- und Kleinschreibung unterscheiden, was beispielsweise bei Passwortinhalten⁵ notwendig wäre, so muss der Zusatz BINARY verwendet werden.⁶

```
1  mysql> ALTER TABLE wurstbrot
2      -> MODIFY name VARCHAR(255) BINARY
3      -> ;
4  Query OK, 4 rows affected, 1 warning (0.00
4 sec)
5  Records: 4 Duplicates: 0 Warnings: 1
6
7  mysql> SELECT * FROM wurstbrot WHERE name
= 'kevin';
```

```
8  +-----+
9  | name |
10 +-----+
11 | kevin |
12 +-----+
```

T-SQL steuert dies durch die Angabe _CS bzw. _CI in der Sortierung (siehe [Zeile 22](#)):

```
1  > DROP TABLE IF EXISTS #wurstbrot;
2  2> GO
3  1> CREATE TABLE #wurstbrot (
4  2> name NVARCHAR(15)
5  3> );
6  4> GO
7  1> INSERT INTO #wurstbrot VALUES
8  2> ('Jaqueline'),
9  3> ('Kevin'),
10 4> ('kevin'),
11 5> ('jaqueline')
12 6> ;
13 7> SELECT * FROM #wurstbrot WHERE name = '
kevin';
14 8> GO
15 (4 rows affected)
16 name
```

```

17  -----
18  Kevin
19  kevin
20
21  1> ALTER TABLE #wurstbrot
22  2> ALTER COLUMN name NVARCHAR(15) COLLATE
SQL_Latin1_General_CP1_CS_AS
23  3> ;
24  4> SELECT * FROM #wurstbrot WHERE name = '
kevin';
25  5> GO
26  name
27  -----
28  kevin

```



Hinweis: In SQL:2023 gibt es keine Möglichkeit, zwischen Groß- und Kleinschreibung zu unterscheiden.

9.1.3 Wie formuliere ich eine zusammengesetzte Bedingung?

Oft sind die fachlichen Anforderungen, die für die Operation zutreffen müssen, aus mehreren Bedingungen zusammengesetzt: Bochumer Kunden mit mehr als 10 Bestellungen im letzten Jahr, nicht gelöschte Bestellpositionen mit einer Menge von 0 usw.

Teilbedingungen müssen daher zu Gesamtbedingungen zusammengesetzt werden. Eine Liste der verfügbaren Verknüpfungsoperatoren finden Sie in [Abschnitt 26.3.2](#).



Hinweis: PostgreSQL und T-SQL kennen keinen logischen XOR-Operator. Dieser kann aber anhand folgender Gleichung simuliert werden: $AB = (A \vee B) \wedge \neg(A \wedge B)$



Aufgabe 9.4: Führen Sie folgende Befehle aus und interpretieren Sie die Ergebnisse:

```
SELECT * FROM artikel
WHERE
    waehrung = 'EUR' AND einzelpreis > 10.0;
```

```
SELECT * FROM artikel
WHERE
    waehrung = 'USD' OR einzelpreis <= 10.0;
```

```
SELECT * FROM artikel
WHERE
    waehrung != 'EUR';
```

```
SELECT * FROM kunde
WHERE
    (rechnung_adresse_id IS NULL) XOR (liefer_adresse_id IS NOT NULL);
```

Um die Ausführungsreihenfolge festzulegen und die Übersichtlichkeit zu erhöhen, werden Klammern verwendet. Verwenden Sie auch dann die Klammern, wenn die Ausführungsreihenfolge klar ist.



Aufgabe 9.5: Führen Sie folgende Befehle aus und interpretieren Sie die Ergebnisse:

```
SELECT * FROM artikel
WHERE
    artikel_id > 3006 AND einzelpreis > 10.0
    OR waehrung = 'EUR';
```

```
SELECT * FROM artikel
WHERE
    (artikel_id > 3006 AND einzelpreis > 10.0) OR (waehrung = 'EUR');
```

```
SELECT * FROM artikel
WHERE
    (artikel_id > 3006) AND (einzelpreis > 10.0 OR waehrung = 'EUR');
```

Aufgabe 9.6: Ermitteln Sie die passende WHERE-Klausel zum SELECT für folgende Anforderungen:

- a) Geben Sie alle Kunden, die 'Beutlin' heißen oder eine Lieferadresse haben, aus.
- b) Geben Sie alle Kunden mit dem Nachnamen 'Beutlin' und dem Vornamen 'Frodo' aus.
- c) Geben Sie alle Kunden, die entweder eine Lieferadresse haben oder Geschäftskunden sind, aus.

Der Aufbau der WHERE-Klausel kann beliebig kompliziert werden. Wir haben jetzt nur die einfachen Grundgerüste kennengelernt. So wird der Inhalt der Werteliste bei dem Operator IN oft durch eigene SELECT⁷ ermittelt. Auch der obere und untere Wert beim Operator BETWEEN wird meist

dynamisch aus Tabelleninhalten berechnet. Freuen wir uns also auf viel kompliziertere WHERE-Klauseln ;-).

9.2 Tabelleninhalte verändern



Alles ändert sich, nichts bleibt, wie es ist.

- Grundkurs
 - Stammdaten und Bewegungsdaten
 - Einfache Wertzuweisung mit UPDATE
 - Werte berechnen
- Vertiefendes
 - LOW_PRIORITY
 - IGNORE



Die Quelltexte dieses Kapitels stehen in der Datei `listing06.sql` (siehe [Listing 28.6](#), [Listing 28.46](#) und [Listing 28.31](#)).

Wir haben Tabelleninhalte mit `INSERT INTO` oder `LOAD DATA INFILE` erstellt. Viele Tabelleninhalte werden selten bis nie verändert, andere häufiger.



Definition 36: Stammdaten/Bewegungsdaten

Tabellen, deren Inhalte eine geringe Änderungswahrscheinlichkeit aufweisen und in den

Geschäftsprozessen eine nicht triviale Bedeutung haben, heißen *Stammdaten*. Tabellen, deren Inhalte eine hohe Änderungswahrscheinlichkeit besitzen und in den Geschäftsprozessen eine nicht triviale Bedeutung haben, heißen *Bewegungsdaten*.

In unserem Beispiel oshop würde ich die Tabellen wie folgt kategorisieren:

Tabelle 9.1 Stamm- und Bewegungsdaten im oshop

Stammdaten	Bewegungsdaten
adresse	bestellung
artikel	bestellung_position
artikel_nm_warengruppen	lagerbestand
artikel_nm_lieferant	rechnung
bank	rechnung_position
bankverbindung	
kunde	
bild	
lieferant	
warengruppe	

Wir werden in den Bewegungsdaten mit dem Befehl UPDATE Änderungen vornehmen:



SQL:2023, PostgreSQL, T-SQL

```
UPDATE tabellenname
SET
  spaltenname=ausdruck
  [, spaltenname=ausdruck] *
  [WHERE bedingung]
;
```

MySQL/MariaDB

```
UPDATE [LOW_PRIORITY] [IGNORE] tabellenname
SET
  spaltenname=ausdruck
  [, spaltenname=ausdruck] *
  [WHERE bedingung]
;
```

9.2.1 Szenario 1: Einfache Wertzuweisung

Der Kunde ändert die Mengenangabe in einer Bestellung. In der Bestellung 1 sollen zwei anstelle von einem Spaten aufgegeben werden.

Die Daten in der Tabelle `bestellung_position` sind folgende:

```
1  mysql> SELECT *
2      -> FROM bestellung_position
3      -> ;
4  +-----+-----+-----+
```

```

-+-----+-----+
5  | bestellung_id | position_nr | artikel_id
  | menge | deleted |
6  +-----+-----+-----+-----+
-+-----+-----+
7  [...]
8  | 1 | 3 | 9015 | 1.000000 | 0 |
9  [...]
10 +-----+-----+-----+-----+
-+-----+-----+

```

In der [Zeile 8](#) befindet sich der zu ändernde Datensatz. Um den Datensatz eindeutig zu identifizieren, eignet sich am besten der Primärschlüssel.



Aufgabe 9.7: Wie bekommt man heraus, welche Spalten den Primärschlüssel einer Tabelle bilden?

```

1  mysql> UPDATE bestellung_position
2      -> SET menge = 2
3      -> WHERE bestellung_id = 1 AND position
n_nr = 3
4      -> ;
5
6  mysql> SELECT *
7      -> FROM bestellung_position
8      -> ;
9  +-----+-----+-----+-----+
-+-----+-----+
10 | bestellung_id | position_nr | artikel_id
  | menge | deleted |
11 +-----+-----+-----+-----+
-+-----+-----+
12 [...]
13 | 1 | 3 | 9015 | 2.000000 | 0 |

```

```

14 [...]
15 +-----+-----+-----+
--+-----+-----+

```

In der [Zeile 13](#) kann der geänderte Wert überprüft werden.

9.2.2 Szenario 2: Berechnete Werte

Die Preise sollen pauschal um 1% erhöht werden.

```

1  mysql> SELECT *
2      -> FROM artikel
3      -> ;
4  +-----+-----+-----+
--+-----+-----+
5  | artikel_id | bezeichnung | einzelpreis |
   | waehrung   | deleted    |
6  +-----+-----+-----+
--+-----+-----+
7  | 3001 | Papier (100) | 2.300000 | EUR | 0
   |
8  | 3005 | Tinte (gold) | 55.700000 | EUR |
0 |
9  | 3006 | Tinte (rot)  | 6.200000 | EUR | 0
   |
10 | 3007 | Tinte (blau) | 4.130000 | EUR | 0
    |
11 | 3010 | Feder        | 5.000000 | EUR | 0 |
12 | 7856 | Silberwiebel | 0.510000 | EUR | 0
    |
13 | 7863 | Tulpenzwiebel | 3.390000 | EUR |
0 |
14 | 9010 | Schaufel     | 14.950000 | USD | 0 |
15 | 9015 | Spaten       | 19.900000 | EUR | 0 |
16 +-----+-----+-----+
--+-----+-----+

```

Jetzt werden die Preise aktualisiert. Da alle Preise aktualisiert werden sollen, kann auf eine WHERE-Klausel verzichtet werden.

```
1  mysql-> UPDATE artikel
2          -> SET einzelpreis = einzelpreis + e
    einzelpreis / 100.0
3          -> ;
4  Query OK, 9 rows affected (0.036 sec)
5  Rows matched: 9 Changed: 9 Warnings: 0
```

Wir haben hier erstmalig die *rechnerischen* Fähigkeiten von SQL kennengelernt. Eine Liste der mathematischen Operationen finden Sie in [Abschnitt 26.2.1](#) und von mathematischen Funktionen in [Abschnitt 26.2.2](#).

```
1  mysql> SELECT *
2          -> FROM artikel
3          -> ;
4  +-----+-----+-----+-----+
+-----+-----+
5  | artikel_id | bezeichnung | einzelpreis |
    waehrung | deleted |
6  +-----+-----+-----+-----+
+-----+-----+
7  | 3001 | Papier (100) | 2.323000 | EUR | 0
   |
8  | 3005 | Tinte (gold) | 56.257000 | EUR |
0 |
9  | 3006 | Tinte (rot) | 6.262000 | EUR | 0
   |
10 | 3007 | Tinte (blau) | 4.171300 | EUR | 0
    |
11 | 3010 | Feder | 5.050000 | EUR | 0 |
12 | 7856 | Silberwiebel | 0.515100 | EUR | 0
    |
13 | 7863 | Tulpenzwiebel | 3.423900 | EUR |
0 |
14 | 9010 | Schaufel | 15.099500 | USD | 0 |
```

```

15 | 9015 | Spaten | 20.099000 | EUR | 0 |
16 +-----+-----+-----+-----+
  +-----+-----+

```

Wir haben es nun mit sehr unschönen Preisen zu tun. Besser wäre ein Runden auf die zweite Stelle hinter dem Komma.

```

1  mysql> UPDATE artikel
2      -> SET einzelpreis = ROUND(einzelpreis
, 2)
3      -> ;
4  Query OK, 9 rows affected (0.031 sec)
5  Rows matched: 9 Changed: 9 Warnings: 0

```



Aufgabe 9.8: Schauen Sie sich die Preise an, und ermitteln Sie die Rundungsregeln.

Wir hätten diese beiden Schritte auch in einem erledigen können:

```

1  UPDATE artikel
2  SET einzelpreis = ROUND(einzelpreis + einz
elpreis / 100.0, 2)
3  ;

```

9.2.3 Szenario 3: Gebastelte Zeichenketten

Für den Briefverkehr soll eine Anrede verwendet werden. In der Tabelle kunde müssen wir diese Spalte zuerst einfügen.

```

1  ALTER TABLE kunde ADD anrede VARCHAR(255)
AFTER nachname;

```

Jetzt können wir aus dem Nachnamen eine Anrede generieren. Da wir keine Unterscheidungsmöglichkeit für Männer und Frauen haben, muss diese Anrede neutral gehalten sein. Dabei sollen alle Datensätze ausgespart werden, für die kein Nachname erfasst ist.

```
1  UPDATE kunde
2  SET anrede = CONCAT('Sehr geehrte/r Frau/Herr ', nachname)
3  WHERE nachname <> ''
4  ;
```



Aufgabe 9.9: Betrachten Sie das Ergebnis, und überlegen Sie sich die Funktionsweise von `CONCAT()`.

9.2.4 Was bedeuten `LOW_PRIORITY` und `IGNORE`?

Ein Update kann recht lange dauern. Entweder weil es auf einer großen Datenmenge arbeitet, oder weil es schwer zu berechnen ist. In beiden Fällen würden konkurrierende Zugriffe auf die Tabelle ausgebremst oder gar blockiert werden. Mit der Option `LOW_PRIORITY` wird der `UPDATE` so lange verzögert, bis keine Verbindung mehr auf der Tabelle arbeitet.

Eine Aktualisierung der Daten bricht ab, sobald ein Fehler auftritt. Man könnte beispielsweise allen Zeilen den gleichen Primärschlüssel zuweisen wollen oder in einem Fremdschlüssel einen Wert eintragen, der einem Constraint widerspricht. Will man, dass das Aktualisieren aber

weitergehen soll, gibt man die Option IGNORE an. Dubletten in Schlüsselspalten werden dabei nicht angelegt.

9.3 Tabelleninhalte löschen



Die verbotene Frucht. Fluch und Segen des Löschens von Daten.

- Grundkurs
 - Löschen mit DELETE
 - Tabellen mit einem Schlag leeren: TRUNCATE
- Vertiefendes
 - Constraints
 - AUTO_INCREMENT
 - LOW_PRIORITY, QUICK, IGNORE



Die Quelltexte dieses Kapitels stehen in der Datei `listing06.sql` (siehe [Listing 28.6](#), [Listing 28.46](#) und [Listing 28.31](#)).

Wir wollen eine Bestellposition löschen. Schauen wir uns vorher den Bestand an Bestellpositionen an:

```
1  mysql> SELECT *
2      -> FROM bestellung_position
3      -> ;
4  +-----+-----+-----+-----+
+-----+-----+-----+-----+
```

5		bestellung_id		position_nr		artikel_id	
		menge		deleted			
6	+	-----	+	-----	+	-----	-----
7		1		1		7856	30.000000 0
8		1		2		7863	50.000000 0
9		1		3		9015	1.000000 0
10		2		1		7856	10.000000 0
11		2		2		9010	5.000000 0
12	+	-----	+	-----	+	-----	-----
	+	-----	+	-----	+	-----	-----

Der Befehl DELETE ist einfach. Das Komplizierteste an ihm ist die WHERE-Klausel.



SQL:2023, PostgreSQL, T-SQL

```
DELETE FROM tabellenname
```

```
    [WHERE bedingung]
```

```
;
```

MySQL/MariaDB

```
DELETE FROM [LOW_PRIORITY] [QUICK] [IGNORE]
tabellenname
```

```
    [WHERE bedingung]
```

```
;
```

Löschen wir die Positionen der Bestellung 1:

```
1  mysql> DELETE FROM bestellung_position
2      -> WHERE bestellung_id = 1
3      -> ;
4
```



```

5  mysql> SELECT *
6      -> FROM bestellung_position
7      -> ;
8
9  +-----+-----+-----+
+-----+-----+
10 | bestellung_id | position_nr | artikel_id
   | menge | deleted |
11 +-----+-----+-----+
+-----+-----+
12 | 2 | 1 | 7856 | 10.000000 | 0 |
13 | 2 | 2 | 9010 | 5.000000 | 0 |
14 +-----+-----+-----+
+-----+-----+

```

Bitte beachten Sie, dass drei Zeilen aus der Tabelle gelöscht werden.



Hinweis: Die Daten werden ohne Rückfrage gelöscht! Nach der Drei-Schichten-Architektur (siehe [Bild 5.2](#) in [Abschnitt 5.3.5](#)) wird das Löschen durch andere Schichten der Anwendung plausibilisiert.

9.3.1 Und was passiert bei Constraints?

Können Sie sich noch an die Löschweitergabe erinnern (siehe [Kapitel 5.3.6](#))? Wären unsere Tabellen mit Löschweitergabe erstellt worden, würde Folgendes passieren:

```

1  mysql> DELETE FROM kunde
2      -> WHERE kunde_id = 1
3      -> ;
4  Query OK, 1 row affected (0.02 sec)

```

```

5
6  mysql> SELECT *
7      -> FROM bestellung
8      -> ;
9  +-----+-----+-----+-----+
-----+---[...]---+
10 | bestellung_id | kunde_id | adresse_id |
   | datum | [...] |
11 +-----+-----+-----+-----+
-----+---[...]---+
12 | 2 | 2 | 2 | 2012-03-23 16:11:00 | [...]
   |
13 +-----+-----+-----+-----+
-----+---[...]---+
14
15 mysql> SELECT *
16 -> FROM bestellung_position
17 -> ;
18 +-----+-----+-----+-----+
-+-----+-----+
19 | bestellung_id | position_nr | artikel_id
   | menge | deleted |
20 +-----+-----+-----+-----+
-+-----+-----+
21 | 1 | 1 | 7856 | 30.000000 | 0 |
22 | 1 | 2 | 7863 | 50.000000 | 0 |
23 | 1 | 3 | 9015 | 1.000000 | 0 |
24 +-----+-----+-----+-----+
-+-----+-----+

```

Bitte beachten Sie, dass in [Zeile 4](#) nur ein Hinweis auf eine gelöschte Zeile erscheint. Ebenso erscheint keine Warnung. Ein anschließender Blick in die Tabelle `bestellung` zeigt mir, dass alle Bestellungen des Kunden 1 ebenfalls gelöscht wurden ([Zeilen 8 ff](#)). Und nicht nur das: In der Tabelle `bestellung_position` sind alle Positionen der Bestellung gelöscht worden.

Jetzt sind unsere Tabellen aber nicht mit ON DELETE CASCADE, sondern mit ON DELETE RESTRICT erstellt worden. Der Löschversuch wird daher zurückgewiesen:

```
1  mysql> DELETE FROM kunde
2      -> WHERE kunde_id = 1
3      -> ;
4  ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails
('oshop'. 'bestellung', CONSTRAINT 'bestellung_ibfk_1' FOREIGN KEY ('kunde_id') REFERENCES
'kunde' ('kunde_id') ON DELETE RESTRICT ON UPDATE RESTRICT)
```

9.3.2 Was passiert mit dem AUTO_INCREMENT?

Ein immer wieder vorgetragenes Missverständnis: Der AUTO_INCREMENT nimmt als neuen Wert $id_{neu} = \text{Max}(id_{alt}) + 1$ an. Man könnte sogar annehmen, dass Lücken in der Zahlenfolge aufgefüllt werden.

Schauen wir uns das an einem Beispiel an:

```
1  mysql> SELECT kunde_id, nachname
2      -> FROM kunde
3      -> ;
4  +-----+-----+
5  | kunde_id | nachname |
6  +-----+-----+
7  | 3 | Beutlin |
8  | 2 | Beutlin |
9  | 5 | Earendilionn |
10 | 1 | Gamdschie |
11 | 4 | Telcontar |
12 +-----+-----+
13
```

```

14 mysql> DELETE FROM kunde
15 -> WHERE kunde_id IN (3,5)
16 -> ;
17
18 mysql> INSERT INTO
19 -> kunde (nachname, vorname)
20 -> VALUES ('Eichenschild', 'Thorin')
21 -> ;
22
23 mysql> SELECT kunde_id, nachname
24 -> FROM kunde
25 -> ;
26 +-----+-----+
27 | kunde_id | nachname |
28 +-----+-----+
29 | 1 | Gamdschie |
30 | 2 | Beutlin |
31 | 4 | Telcontar |
32 | 6 | Eichenschild |
33 +-----+-----+

```

Es werden somit keine Lücken gefüllt oder die obige *Max()*-Formel verwendet. Die Tabellen müssen sich irgendwo den zuletzt erzeugten oder nächsten AUTO_INCREMENT-Wert merken.

Und tatsächlich: Mit SHOW CREATE TABLE kriegt man heraus, welchen Wert der nächste AUTO_INCREMENT liefern wird (siehe [Zeile 7](#)).

```

1 mysql> SHOW CREATE TABLE kunde\G
2 ***** 1. row *****
3
4 Table: kunde
5 Create Table: CREATE TABLE 'kunde' (
6   'kunde_id' int unsigned NOT NULL AUTO_INCREMENT,
7   [...]

```

```
7 ) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT C  
HARSET=utf8mb4 COLLATE=utf8mb4_german2_ci
```



Hinweis: Dies bedeutet auch, dass ein Löschen des gesamten Inhalts nicht dazu führt, dass wieder mit 1 angefangen wird.

9.3.3 Was bedeuten LOW_PRIORITY, QUICK und IGNORE?

Ein Löschvorgang kann recht lange dauern. Entweder weil er auf einer großen Datenmenge arbeitet oder weil er eine schwer auszuwertende WHERE-Klausel hat. In beiden Fällen würden konkurrierende Zugriffe auf die Tabelle ausgebremst oder gar blockiert werden. Mit der Option LOW_PRIORITY wird der Löschvorgang solange verzögert, bis keine Verbindung mehr auf der Tabelle arbeitet.

Das Löschen von Datensätzen kann die Reorganisation von Indizes auslösen, was recht langsam ist. Bei MyISAM-Tabellen kann die Option QUICK das Löschen so organisieren, dass es schneller ist.

Ein Löschen der Daten bricht ab, sobald ein Fehler auftritt. So könnte durch eine Löschoperation ein Constraint verletzt werden. Will man, dass das Löschen mit der nächsten Zeile fortgesetzt werden soll, gibt man die Option IGNORE an.

9.3.4 Wie kann ich eine Tabelle komplett leeren?

Dazu gibt es zwei Möglichkeiten. Die erste ist ein DELETE ohne WHERE-Klausel. Da die WHERE-Klausel eine Einschränkung darstellt, bedeutet das Fehlen der WHERE-Klausel, dass das Löschen ohne Einschränkung erfolgt⁸:

```
1 DELETE FROM bestellung_position;
```

Damit wird die Tabelle aber nicht wieder in den *Startzustand* versetzt. Hierfür gibt es eine Alternative:



SQL:2023, T-SQL

```
TRUNCATE TABLE tabellenname ;
```

MySQL/MariaDB, PostgreSQL

```
TRUNCATE [TABLE] tabellenname ;
```

```
1 TRUNCATE bestellung_position;
```

Außer in der InnoDB-Engine wird die Tabelle komplett gelöscht und wieder neu erzeugt. Dies ist in der Regel sehr viel schneller, als jede Zeile einzeln zu löschen, wie es die InnoDB unter Umständen tut. Bei jeder Engine in MySQL und MariaDB wird der Zähler vom AUTO_INCREMENT wieder auf 1 gesetzt.

Wie bei DELETE werden – sofern die Engine es unterstützt – bei einem TRUNCATE die Constraints überprüft.



Hinweis: Bei einem TRUNCATE werden die Tabelleninhalte ohne Rückfrage endgültig gelöscht.

¹ Der Datentyp `BOOL` ist eigentlich nur ein anderer Ausdruck für ein `TINYINT` mit der Domäne 0 (`=FALSE`), 1 (`=TRUE`) und `NULL` (`=UNKNOWN`).

² In T-SQL gibt es keine booleschen Datentypen; man behilft sich mit dem Datentyp `BIT`. Eine gute Einführung finden Sie unter [\[Cal22\]](#).

³ Dabei ist ein Quelltext auch nur ein HaufenWorte. Darüber sollte man mal nachdenken ...

⁴ In PostgreSQL allerdings schon. Dort hätte man schreiben müssen:
`SELECT * FROM wurstbrot WHERE LOWER(name) = 'kevin';`.

⁵ Die natürlich NIE NIE NIE klartextlich abgespeichert werden.

⁶ Allerdings erscheint dabei ein Warnung: *'BINARY as attribute of a type' is deprecated and will be removed in a future release. Please use a CHARACTER SET clause with _bin collation instead.* (1287)

⁷ Siehe Unterabfragen in [Kapitel 13](#).

⁸ Aha!

TEIL IV

Datenbank auswerten

10 Einfache Auswertungen



Wie, SQL ist kein WRITE ONLY-Baustein? Wie wir die Daten wieder auslesen können.

- Grundkurs

- Konstanten
- Mathematische Operatoren
- Auswahl von Spalten
- Verwendung der WHERE-Klausel
- Sortieren mit ORDER BY
- Sortierreihenfolge mit ASC und DESC
- Groß- und Kleinschreibung bei der Sortierung
- Sortieren von Datum- und Uhrzeitwerten
- Mehrfachausgabe mit DISTINCT unterbinden
- Teilergebnisse mit LIMIT

- Vertiefendes

- Operatorenpriorität
- Zufallszahlen
- Variablen

- Collations
- Einfluss von Indizes auf Sortierung
- EXPLAIN
- Einfluss von Indizes auf DISTINCT
- Daten exportieren mit SELECT ... INTO OUTFILE
- Binäre Daten auslesen

Der Befehl SELECT ist das Herz von SQL. Alle anderen Befehle haben natürlich ihre Berechtigung, aber SELECT stellt mir die Daten aufbereitet zur Verfügung. Dabei kann der SELECT sowohl Säbel als auch Florett sein. Von kleinen trivialen Auswertungen bis zu mehrseitigen Analysen ist mit diesem Wunderkind alles möglich. Für mich als Programmierer ist es schon sehr erstaunlich, dass 90% der Programmierarbeit mit Varianten eines Befehls erfolgen – und eben genau das macht die Eleganz und den Reiz der SQL-Programmierung aus.



Die Quelltexte dieses Kapitels stehen in der Datei `listing07. sql` (siehe [Listing 28.7](#), [Listing 28.47](#) und [Listing 28.32](#)).

Die Dateien `perfIndexOrderBy01. sql` und `blz_20120305.csv` werden verwendet.

10.1 Ausdrücke



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

SELECT *ausdruck*

;

10.1.1 Konstanten

Der einfachste Ausdruck ist eine Konstante. Dies kann eine numerische oder eine zeichenbasierte Konstante sein.



Definition 37: Konstante

Eine *Konstante* ist ein Ausdruck, der zur Compilezeit festgelegt wird und zur Laufzeit seinen Wert nicht verändern kann.

Was heißen jetzt schon wieder zur Laufzeit und Compilezeit? Der Quelltext – hier SQL-Quelltext – wird durch einen Compiler zu einem ausführbaren Code¹ umgewandelt. Zur Compilezeit bedeutet also, dass der Wert während dieser Umwandlung festgelegt wird. Laufzeit ist der Zeitraum, wo das Programm – der SQL-Befehl – verarbeitet wird. Bei einer Konstanten kann also während der Ausführung des Befehls keine Veränderung vorgenommen werden.

Beispiel: Wir wollen in einem SQL-Skript auf der Konsole angeben, bei welchem Bearbeitungsschritt wir gerade sind.

```
1  mysql> SELECT 5, 'Beginn der Auswertung';
2  +---+-----+
3  | 5 | Beginn der Auswertung |
```

```

4  +---+-----+
5  | 5 | Beginn der Auswertung |
6  +---+-----+

```

Sowohl die 5 als auch die Zeichenkette Beginn der Auswertung sind Konstanten, da sie während der Laufzeit nicht verändert werden. Konstanten sind schwer zu erklären und super einfach zu verwenden.

10.1.2 Wie kann ich Berechnungen vornehmen?

Man kann mit SQL auch rechnen. Eine Übersicht der mathematischen Operatoren finden Sie in [Abschnitt 26.2.1](#).

```

1  mysql> SELECT -5, 9 + 4, 9 - 4, 9 * 4, 9 /
2  +---+-----+-----+-----+-----+-----+
-----+-----+-----+
3  | -5 | 9 + 4 | 9 - 4 | 9 * 4 | 9 / 4 | 9 D
IV 4 | 9 % 4 | 9 MOD 4 |
4  +---+-----+-----+-----+-----+-----+
-----+-----+-----+
5  | -5 | 13 | 5 | 36 | 2.2500 | 2 | 1 | 1 |
6  +---+-----+-----+-----+-----+-----+
-----+-----+-----+

```

Das unäre Minus setzt das Vorzeichen der nachfolgenden Zahl. Dieser Operator hat die höchste Ausführungspriorität, er wird deshalb vor allen anderen ausgeführt. Natürlich dürfen anstelle von Konstanten als Operanden auch Spaltennamen oder Variablen verwendet werden.²



Definition 38: Operatorenpriorität

Unter der *Operatorenpriorität* oder *Operatorenrangfolge* versteht man die Festlegung darüber, welche Operation in einer Anweisung vor anderen ausgeführt wird. Haben zwei Operatoren die gleiche *Operatorenpriorität*, so werden sie von links nach rechts ausgeführt.

Die Reihenfolge der Auswertung kann durch Klammern festgelegt werden. Da die Operatorenpriorität nicht von jedem auswendig gewusst wird und der besseren Lesbarkeit wegen, empfehle ich, immer zu klammern. Hier ein Beispiel:

```
1  mysql> SELECT 3 * 4 DIV 3, (3 * 4) DIV 3,  
2  3 * (4 DIV 3);  
3  +-----+-----+-----+-----+  
4  | 3 * 4 DIV 3 | (3 * 4) DIV 3 | 3 * (4 DIV  
5  3) |  
6  +-----+-----+-----+-----+  
7  | 4 | 4 | 3 |  
8  +-----+-----+-----+-----+  
9  | NULL | NULL | NULL |
```

Neben den einfachen mathematischen Operatoren gibt es eine Vielzahl von mathematischen Funktionen. In [Abschnitt 26.2.2](#) finden Sie die Funktionen und ihre Bedeutung.³ Bei der Verwendung von mathematischen Operatoren oder Funktionen ist aber auf den Wertebereich zu achten.

```
1  mysql> SELECT 9 / 0, 9 * NULL, SQRT(-5);  
2  +-----+-----+-----+  
3  | 9 / 0 | 9 * NULL | SQRT(-5) |  
4  +-----+-----+-----+  
5  | NULL | NULL | NULL |
```

In MySQL oder MariaDB liefert ein Funktionsparameter, der nicht im Definitionsbereich liegt, das Ergebnis NULL. Bei anderen Systemen wie beispielsweise PostgreSQL und T-SQL wird die Verarbeitung mit einer Fehlermeldung abgebrochen.

10.1.3 Wie ermittle ich Zufallszahlen?

Die Funktion zur Ermittlung von Zufallszahlen ist `RAND()` mit einem Wertebereich von `[0.0, 1.0[`. Dieser Funktion kann ein sogenannter *seed* mitgegeben werden. Die Berechnung von Zufallszahlen ist nicht wirklich dem Zufall überlassen.

Vielmehr müssen Sie sich das Ganze als einen riesigen Kreis vorstellen. Auf dem Kreis liegen die *Zufallszahlen*, und jede hat einen festen Vorgänger und einen festen Nachfolger. Rufen Sie `RAND()` oft genug auf, so würde sich der Kreis schließen und sich die Zahlenfolge wiederholen.

Die Qualität eines Zufallszahlengenerators ergibt sich aus der Größe dieses Kreises (ab wann sich die Zahlenfolge wiederholt), der Häufigkeitsverteilung der Zahlen auf dem Kreis (möglichst gleich verteilt) und der Vorhersagbarkeit der nächsten Zahl. Wer sich näher damit beschäftigen möchte, dem sei das epochale Werk von Donald E. Knuth [[Knu81](#)] empfohlen.

Der *seed* legt indirekt den Startpunkt auf diesem Kreis fest. Verwenden Sie den gleichen *seed*, erhalten Sie auch das gleiche Ergebnis.

```
1  mysql> SELECT RAND(1);
2  +-----+
3  | RAND(1) |
4  +-----+
```

```

5  | 0.40540353712197724 |
6  +-----+
7
8  mysql> SELECT RAND(1);
9  +-----+
10 | RAND(1) |
11 +-----+
12 | 0.40540353712197724 |
13 +-----+
14
15 mysql> SELECT RAND(1) FROM artikel;
16 +-----+
17 | RAND(1) |
18 +-----+
19 | 0.40540353712197724 |
20 | 0.8716141803857071 |
21 | 0.1418603212962489 |
22 | 0.09445909605776807 |
23 | 0.04671454713373868 |
24 | 0.9501954782290342 |
25 | 0.6108337804776 |
26 | 0.2035824984345422 |
27 | 0.18541118147355615 |
28 +-----+

```

Der Verweis auf die Artikeltabelle ist nur notwendig, damit mehrere Zufallszahlen in einer Anweisung generiert werden.



Aufgabe 10.1: Würde Sie den Befehl noch mal ausführen, bekämen Sie dann die gleichen oder andere Zufallszahlen?

Zufallszahlen im Intervall $[a, b[$, für $a \leq b$, gehen so: $\text{FLOOR}(a + \text{RAND}() * (b - a))$.

10.1.4 Wie stecke ich das Berechnungsergebnis in eine Variable?

In MySQL oder MariaDB werden Variablen, die außerhalb von gekapselten Anweisungen liegen⁴, mit einem Klammeraffen @ gekennzeichnet. Mit SELECT ... INTO können Sie Ergebnisse in eine Variable schreiben. Die Variablen lassen sich in späteren Berechnungen und Bedingungen weiterverwenden:

```
1  mysql> SELECT POWER(2,8)
2      -> INTO @x
3      -> ;
4
5  mysql> SELECT @x - 1;
6  +-----+
7  | @x - 1 |
8  +-----+
9  | 255   |
10 +-----+
```

Aber:

```
1  mysql> SELECT einzelpreis
2      -> INTO @y
3      -> FROM artikel
4      -> ;
5  ERROR 1172 (42000): Result consisted of more than one row
```



Aufgabe 10.2: Interpretieren Sie die Fehlermeldung. Überlegen Sie sich, was SELECT liefert und worin Sie das Ergebnis abspeichern wollen.

Man kann Variablen auch direkt mit einem Wert versehen:

```
1  mysql> SET @artnr=3010;
2
3  mysql> SELECT *
4      -> FROM artikel
5      -> WHERE artikel_id=@artnr
6      -> ;
7  +-----+-----+-----+-----+
-----+-----+
8  | artikel_id | bezeichnung | einzelpreis |
   | waehrung  | deleted    |
9  +-----+-----+-----+-----+
-----+-----+
10 | 3010 | Feder | 5.050000 | EUR | 0 |
11 +-----+-----+-----+-----+
-----+-----+
```

Variablen sind immer dann sehr hilfreich, wenn Informationen von einem SQL-Befehl an den anderen weitergereicht werden müssen. So können Sie in einem SELECT eine bestimmte Bestellung ermitteln und in einem zweiten die Positionen zu genau dieser Bestellung und in einem dritten Befehl vielleicht die Lieferadresse der Bestellung. Bei den letzten beiden SELECT-Anweisungen würde man in der WHERE-Klausel die bestellung_id auf Gleichheit mit dem Variablenwert testen.

10.2 Zeilen- und Spaltenwahl



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

SELECT

{*|spaltenliste|ausdruck}

```
FROM tabellenname
[WHERE bedingung]
;
```

Um den Inhalt einer Tabelle – z.B. `artikel` – komplett auszugeben, haben wir in den vorangegangenen Kapiteln eine einfache Version des `SELECT` verwendet:

```
1 SELECT * FROM artikel;
```

Der `*` steht hier als Platzhalter für alle Spalten einer Tabelle. Folgender Befehl hätte das gleiche Ergebnis:

```
1 SELECT
2     artikel_id, bezeichnung, einzelpreis,
waehrung, deleted
3     FROM artikel
4     ;
```

Diese *spaltenliste* kann sehr flexibel zusammengesetzt werden:

- Die Reihenfolge der Spalten kann beliebig sein.
OK: `SELECT deleted, bezeichnung, artikel_id, waehrung FROM artikel`
- Die Spalten können mehrfach vorkommen.
OK: `SELECT deleted, artikel_id, deleted FROM artikel`
- Es muss aber mindestens eine Spalte angegeben werden.
NOT OK: `SELECT FROM artikel`
- Es können Konstanten oder Ausdrücke vorkommen.
OK: `SELECT RAND(), TRUE, bezeichnung, artikel_id + 5000 FROM artikel`

Eine weitere schöne Sache ist, dass man den Spaltennamen oder Ausdrücken neue *Namen* – einen Alias – zuweisen kann, wobei das Schlüsselwort AS optional ist:

```
1  mysql> SELECT
2      -> artikel_id + 10000 AS Unsinn, delet
ed AS Löschkennzeichen
3      -> FROM artikel
4      -> ;
5  +-----+-----+
6  | Unsinn | Löschkennzeichen |
7  +-----+-----+
8  | 13010 | deleted |
9  [...]
10 | 17863 | deleted |
11 +-----+-----+
```

In [Zeile 6](#) werden nicht mehr die ursprünglichen Spaltennamen oder Ausdrücke als Überschriften verwendet. Der neue Name sollte selbsterklärend oder eine sinnvolle Abkürzung sein. Diese Namensvergabe wird uns noch häufiger begegnen.

Zur Auswahl der Zeilen wird wie beim UPDATE und DELETE die WHERE-Klausel verwendet. Dabei entsteht leicht die Annahme, dass die Spalten, die in der WHERE-Klausel stehen, auch in der *spaltenliste* vorkommen müssen.⁵ Hier ein Gegenbeispiel:

```
1  mysql> SELECT
2      -> artikel_id, bezeichnung
3      -> FROM artikel
4      -> WHERE einzelpreis BETWEEN 10 AND 10
0;
5  +-----+-----+
6  | artikel_id | bezeichnung |
7  +-----+-----+
8  | 3005 | Tinte (gold) |
```

```

9   | 9010 | Schaufel |
10  | 9015 | Spaten  |
11  +-----+-----+

```

Weitere Informationen zur Zeilenauswahl finden Sie in der Besprechung der WHERE-Klausel in [Abschnitt 9.1](#).

10.3 Sortierung



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

```

SELECT
    {*|spaltenliste|ausdruck}
FROM tabellenname
[WHERE bedingung]
[ORDER BY spaltenname [ASC|DESC] [,spaltenname
[ASC|DESC]]*]
;

```

Die Ausgabe von Daten in einer fachlich oder ergonomisch sinnvollen Reihenfolge ist eine oft gestellte Forderung an eine Auswertung. Beispiele: Kunden nach Umsatz absteigend, Artikelpreise nach Einzelpreis aufsteigend, Namenslisten nach Nach- und Vorname etc. Die entsprechende Option heißt ORDER BY.

Geben wir die Artikel nach Einzelpreis aufsteigend aus:

```

1  mysql> SELECT
2      -> artikel_id, bezeichnung, einzelprei
s

```

```

3      -> FROM artikel
4      -> ORDER BY einzelpreis ASC;
5  +-----+-----+-----+
-+
6  | artikel_id | bezeichnung | einzelpreis |
7  +-----+-----+-----+
-+
8  | 7856 | Silberzwiebel | 0.520000 |
9  | 3001 | Papier (100) | 2.320000 |
10 | 7863 | Tulpenzwiebel | 3.420000 |
11 | 3007 | Tinte (blau) | 4.170000 |
12 | 3010 | Feder | 5.050000 |
13 | 3006 | Tinte (rot) | 6.260000 |
14 | 9010 | Schaufel | 15.100000 |
15 | 9015 | Spaten | 20.100000 |
16 | 3005 | Tinte (gold) | 56.260000 |
17 +-----+-----+-----+
-+

```

Das Schlüsselwort ASC sortiert die Daten aufsteigend, DESC absteigend. Wird keine der beiden Angaben gemacht, werden die Daten aufsteigend sortiert.



Aufgabe 10.3: Führen Sie den Befehl einmal ohne ASC und anschließend mit einem DESC aus. Vergleichen Sie die Ergebnisse mit dem obigen.

Die Sortierung kann auch nach mehreren Spalten erfolgen:

```

1  mysql> SELECT
2      -> nachname, vorname
3      -> FROM kunde
4      -> ORDER BY nachname, vorname
5      -> ;
6  +-----+

```

```

7  | nachname | vorname |
8  +-----+-----+
9  | Beutlin  | Bilbo  |
10 | Beutlin  | Frodo  |
11 | Earendil | Elrond |
12 | Eichensch | Thorin |
13 | Gamdschie | Samweis |
14 | Telcontar | Elessar |
15 +-----+-----+

```

Bitte beachten Sie, dass die Liste in erster Priorität nach dem Nachnamen sortiert ist. Nur dort, wo im Nachnamen der gleiche Wert steht, wird nach Vorname sortiert. Deshalb steht der Bilbo vor dem Frodo. Die Sortierreihenfolge ist aber für beide Spalten unabhängig einstellbar:

```

1  mysql> SELECT
2      -> nachname, vorname
3      -> FROM kunde
4      -> ORDER BY nachname DESC, vorname ASC
5      -> ;
6  +-----+-----+
7  | nachname | vorname |
8  +-----+-----+
9  | Telcontar | Elessar |
10 | Gamdschie | Samweis |
11 | Eichensch | Thorin |
12 | Earendil | Elrond |
13 | Beutlin  | Bilbo  |
14 | Beutlin  | Frodo  |
15 +-----+-----+

```

Die Liste wird jetzt nach dem Nachnamen absteigend sortiert, aber die Vornamen wiederum aufsteigend⁶.



Hinweis: Ein gerne gemachter Fehler bei folgender Aufgabenstellung: *Geben Sie die Namensliste für Nach-*

und Vorname absteigend sortiert aus. Häufig wird folgende Lösung angeboten:

```
1  SELECT
2      nachname, vorname
3      FROM kunde
4      ORDER BY nachname, vorname DESC;
```

Diese Lösung ist aber falsch, da sich das DESC hier nur auf die letzte Spalte auswirkt und der Nachname aufsteigend sortiert wird. Denken Sie daran, dass ASC immer dann verwendet wird, wenn man für diese Spalte keine Angaben macht.

Würden Sie für die Spaltennamen Aliase verwenden, können Sie diese in die ORDER BY-Klausel einsetzen.

```
1  SELECT nachname n, vorname v FROM kunde OR
   DER BY n DESC, v DESC;
```

Die ORDER BY-Klausel kann anstelle des Spaltennamens oder eines Alias auch eine Zahl enthalten. Die Zahl ergibt sich aus einer 1 basierenden Indizierung der *spaltenliste*:

```
1  -- 1 2
2  SELECT nachname, vorname FROM kunde ORDER
   BY 1 DESC, 2 ASC;
```

Und so haben wir auch das Kommentarzeichen -- für einzeilige Kommentare eingeführt ([Zeile 1](#)). Bitte beachten Sie, dass ein Leerzeichen nach den zwei Strichen stehen muss.

10.3.1 Was muss ich bei der Sortierung von

Texten beachten?

Zunächst stellt sich die Frage, wann eine Zeichenkette größer als eine andere ist. Denn jede Sortierung basiert auf einer Festlegung darüber, wie eine Reihenfolge ermittelt werden kann. In der Informatik verwendet man dazu den Begriff der *Ordnung*:



Definition 39: Ordnung

Eine Menge M ist *wohlgeordnet*, wenn sie folgende Eigenschaften erfüllt:

- Trichotomie
Für die Elemente $a, b \in M$ gilt immer genau eine der folgenden Aussagen: $a = b$, $a < b$ oder $a > b$
- Transitivität
Für die Elemente $a, b, c \in M$ gilt, wenn $a \leq b$ und $b \leq c$, dann muss $a \leq c$ sein.
- Minimumexistenz
Es gibt ein $m \in M$, für welches gilt, dass für alle $x \in M$ $m \leq x$ ist.

Wann immer eine Menge wohlgeordnet ist, kann man die Menge sortieren. Machen wir uns die [Definition 39](#) anhand der natürlichen Zahlen klar⁷.

- Trichotomie: Für zwei Zahlen muss gelten, dass sie entweder gleich oder unterschiedlich sind. Wenn sie unterschiedlich sind, muss eine der beiden die kleinere sein.
- Transitivität: Wenn $5 < 10$ und $10 < 12$ ist, dann ist auch $5 < 12$.

- Minimumexistenz: Wir behaupten, dass 0 die gesuchte Zahl ist.

Wie ist das aber jetzt mit den Texten? Oft bekomme ich die Antwort, dass man nach der Länge des Textes sortieren kann. Und ja, das wäre möglich:

- Trichotomie: Die Längen zweier Texte sind entweder gleich oder unterschiedlich. Wenn sie unterschiedlich sind, muss ein Text der kürzere sein.
- Transitivität: Wenn ein Text kürzer als ein zweiter ist und der zweite kürzer als ein dritter Text ist, dann muss der erste auch kürzer als der dritte sein.
- Minimumexistenz: Wir behaupten, dass dies der leere Text mit einer Länge 0 ist.

Aber wer braucht das? Textsortierungen werden in der Regel als alphabetische Sortierungen betrachtet. Und jetzt fängt es an, kompliziert zu werden. Was ist ein Alphabet und wer legt die Reihenfolge fest? Es hat mich beispielsweise immer erstaunt, dass wir in Deutschland als Kinder ein Alphabet lernen, in dem die Umlaute und das ß nicht vorkommen. Schließlich sind diese Buchstaben ja keine Unfälle, sondern kommen sogar häufiger vor als ein X.

Eine textliche Sortierung verlangt somit, dass es eine Festschreibung über die Reihenfolge der gültigen Zeichen gibt. Historisch wurde die ASCII-Zeichenfolge mit ihrem Nummerncode verwendet. Was zu folgendem unglücklichen Ergebnis führt:

```
1  mysql> SET NAMES latin1;
2
3  mysql> CREATE TEMPORARY TABLE wurstbrot (
4      -> name VARCHAR(255) CHARACTER SET 'la
tin1' COLLATE 'latin1_bin'
```

```

5      -> );
6
7  mysql> INSERT INTO wurstbrot
8      -> VALUES
9      -> ('winfried')
10     -> ,('achim')
11     -> ,('olga')
12     -> ,('zechine')
13     -> ,('ägidius');
14
15  mysql> SELECT * FROM wurstbrot ORDER BY na
me;
16  +-----+
17  | name |
18  +-----+
19  | achim |
20  | olga |
21  | winfried |
22  | zechine |
23  | ägidius |
24  +-----+

```

In [Zeile 1](#) wird durch SET NAMES sichergestellt, dass die Konsoleneingabe in der hier gewünschten Kodierung zum Server geschickt wird. [Zeile 4](#) erstellt eine Spalte mit dem CHARACTER SET latin1 und einer dazu passenden Sortierung. Übrigens sehen Sie hier ein Beispiel dafür, dass Sie innerhalb einer Tabelle für Textspalten unterschiedliche Zeichensätze verwenden können (zum Thema Zeichensätze siehe [Abschnitt 5.2.4](#)).

In [Zeile 13](#) wird ein Name mit einem Umlaut eingefügt, der nach einer Sortierung in [Zeile 23](#) ganz an das Ende gestellt wird. Warum? Der ASCII hat in den ersten 128 Zeichen nur die Buchstaben für das amerikanische Alphabet. Länderspezifische Sonderzeichen haben demnach einen Code

> 127. Wird jetzt dieser Code als Basis der Sortierung verwendet, landen die Umlaute immer ganz am Ende.

Es war also nötig, eine von der Zeichenkodierung unabhängige Spezifikation für die Sortierung zu entwickeln. Das Ergebnis dieser Bemühungen sind die Collations (siehe [Kapitel 5.2.5](#)). Im Prinzip wird dabei jedem Zeichen oder einer Zeichenkombination eine eindeutige Nummer zugewiesen. Anhand dieser Nummer kann dann die Sortierung erfolgen:

```
1  mysql> SET NAMES utf8mb4;
2
3  mysql> CREATE TEMPORARY TABLE wurstbrot (
4      -> name VARCHAR(255) CHARACTER SET 'ut
utf8mb4' COLLATE 'utf8mb4_german2_ci'
5      -> );
6
7  mysql> INSERT INTO wurstbrot
8      -> VALUES
9      -> ('winfried')
10     -> ,('achim')
11     -> ,('olga')
12     -> ,('zechine')
13     -> ,('ägidius');
14
15  mysql> SELECT * FROM wurstbrot ORDER BY na
me;
16  +-----+
17  | name |
18  +-----+
19  | achim |
20  | ägidius |
21  | olga |
22  | winfried |
23  | zechine |
24  +-----+
```

Wenn Sie eine Sortierung auf zeichenbasierten Spalten machen, sollten Sie unbedingt auf den eingestellten Zeichensatz und die entsprechende Sortierung achten. So ist es beispielsweise unter Windows nicht möglich, Unicode-Zeichen über den MySQL Client einzugeben, da der Windows-COMMAND kein Unicode verarbeiten kann. Die Verarbeitung kann dann zu unerwünschten Resultaten führen.

Verwenden Sie dann lieber den Query-Browser der Workbench oder noch besser: Schreiben Sie in einer Sprache Ihrer Wahl (C#, PHP, Python, ...) einen eigenen Query-Browser. Sie glauben gar nicht, wie lehrreich das ist!

10.3.2 Wird zwischen Groß- und Kleinschreibung unterschieden?

Ändern wir das obige Beispiel ab, um diese Frage zu beantworten. Zuerst die Testdaten:

```
1  mysql> CREATE TEMPORARY TABLE wurstbrot (name VARCHAR(255));
2
3  mysql> INSERT INTO wurstbrot
4      -> VALUES
5      -> ('winfried')
6      -> ,('achim')
7      -> ,('olga')
8      -> ,('Olga')
9      -> ,('zechine');
```

Jetzt einmal aufsteigend und einmal absteigend sortiert.

```
1  mysql> SELECT * FROM wurstbrot ORDER BY name;
2  +-----+
```

```

3  | name |
4  +-----+
5  | achim |
6  | Olga |
7  | olga |
8  | winfried |
9  | zechine |
10 +-----+
11
12 mysql> SELECT * FROM wurstbrot ORDER BY name DESC;
13 +-----+
14 | name |
15 +-----+
16 | zechine |
17 | winfried |
18 | Olga |
19 | olga |
20 | achim |
21 +-----+

```

In beiden Sortierreihenfolgen erscheint die *große Olga* vor der *kleinen*. Würde zwischen Groß- und Kleinschreibung unterschieden, müsste die *große Olga* entweder als erste Zeile oder als letzte Zeile oder mal vor und mal nach der *kleinen Olga* erscheinen, aber nicht in jedem Fall gleich.

Möchte Sie zwischen Groß- und Kleinschreibung einen Unterschied machen, müssen Sie die zeichenbasierte Spalte mit BINARY als Zusatz deklarieren (siehe [Zeile 1](#)).

```

1  mysql> CREATE TEMPORARY TABLE wurstbrot (name VARCHAR(255) BINARY);
2
3  mysql> INSERT INTO wurstbrot
4      -> VALUES
5      -> ('winfried')
6      -> , ('achim')

```

```

7      -> ,('olga')
8      -> ,('Olga')
9      -> ,('zechine');
10
11 mysql> SELECT * FROM wurstbrot ORDER BY na
me;
12 +-----+
13 | name |
14 +-----+
15 | Olga |
16 | achim |
17 | olga |
18 | winfried |
19 | zechine |
20 +-----+
21
22 mysql> SELECT * FROM wurstbrot ORDER BY na
me DESC;
23 +-----+
24 | name |
25 +-----+
26 | zechine |
27 | winfried |
28 | olga |
29 | achim |
30 | Olga |
31 +-----+

```

Wie erwartet, wird die *große Olga* jetzt anders einsortiert.



Hinweis: In SQL:2023 gibt es keine Möglichkeit, zwischen Groß- und Kleinschreibung zu unterscheiden.

10.3.3 Wie werden Datums- und Uhrzeitwerte

sortiert?

Datums- und Uhrzeitwertewerden nach ihrem Alter sortiert. Das älteste Datum, die älteste Uhrzeit kommt als Erstes, das jüngste Datum bzw. die jüngste Uhrzeit als Letztes.

```
1  mysql> SELECT
2      -> bestellung_id, datum
3      -> FROM bestellung
4      -> ORDER BY datum
5      -> ;
6  +-----+-----+
7  | bestellung_id | datum |
8  +-----+-----+
9  | 2 | 2012-03-23 16:11:00 |
10 | 1 | 2012-03-24 17:41:00 |
11 +-----+-----+
```



Hinweis: Der aktuelle Zeitpunkt bleibt dabei völlig unberücksichtigt. So kann ein in der Zukunft liegendes Datum das älteste sein, wenn alle anderen noch weiter in der Zukunft liegen.

Die Uhrzeit wird mit 00:00:00Uhr beginnend sortiert. Die späteste Uhrzeit kommt als Letztes:

```
1  mysql> SELECT
2      -> bestellung_id, TIME(datum) Uhrzeit
3      -> FROM bestellung
4      -> ORDER BY uhrzeit
5      -> ;
6  +-----+-----+
7  | bestellung_id | Uhrzeit |
8  +-----+-----+
```

9		2		16:11:00	
10		1		17:41:00	
11	+-----+-----+				



Aufgabe 10.4: Ermitteln Sie die Bestellung, die als Letztes – bezogen auf Datum und Uhrzeit – aufgegeben wurde.

10.3.4 Wie kann ich das Sortieren beschleunigen?

Grundsätzlich gibt es zwei Methoden, wie die Sortierung vorgenommen wird:

- **Index:** Die zu sortierenden Felder sind in der gewünschten Reihenfolge in einem Index. SQL braucht dann nur die Vorsortierung des Index auszugeben, was sehr schnell geht.
- **Filesort⁸:** Die zu sortierenden Felder werden mithilfe integrierter Sortier Routinen wie gefordert geordnet.

Damit wir überhaupt einen Unterschied feststellen können, bauen wir eine Tabelle auf, die tausend Datensätze hat. Laden Sie das Experiment in [Abschnitt 27.7](#) und führen Sie Folgendes aus (kann ein paar Minuten dauern):

```
1 mysql> CALL datenerzeugen(150);
```

Um herauszufinden, welche der beiden Methoden verwendet wird, können Sie in MySQL den Befehl EXPLAIN verwenden.

Dieser sehr hilfreiche Befehl beschreibt den Ausführungsplan eines Befehls.

```
1  mysql> EXPLAIN
2  -> SELECT * FROM name_mit ORDER BY nachnam
e, vorname\G
3  **** 1. row ****
****
4          id: 1
5      select_type: SIMPLE
6          table: name_mit
7      partitions: NULL
8          type: index
9  possible_keys: NULL
10         key: idx_namemit_nnvn
11         key_len: 2046
12         ref: NULL
13         rows: 1500
14     filtered: 100.00
15     Extra: Using index
16
17 mysql> EXPLAIN
18 -> SELECT * FROM name_ohne ORDER BY nachna
me, vorname\G
19 **** 1. row ****
****
20         id: 1
21     select_type: SIMPLE
22         table: name_ohne
23     partitions: NULL
24         type: ALL
25 possible_keys: NULL
26         key: NULL
27     key_len: NULL
28         ref: NULL
29         rows: 1500
30     filtered: 100.00
31     Extra: Using filesort
```


Es lohnt sich, beim Erzeugen der Indizes (siehe [Abschnitt 6.1](#)) sehr genau den Bedarf zu planen und beim Formulieren der ORDER BY-Klausel gut aufzupassen, denn leider verliert man sehr schnell die Möglichkeit, den Index zu verwenden.

```
1  mysql> EXPLAIN
2      -> SELECT * FROM name_mit ORDER BY vor
name\G
3  ***** 1. row *****
4      id: 1
5      select_type: SIMPLE
6      table: name_mit
7      partitions: NULL
8      type: index
9      possible_keys: NULL
10     key: idx_namemit_nnv
11     key_len: 2046
12     ref: NULL
13     rows: 1500
14     filtered: 100.00
15     Extra: Using index; Using filesort
16
17  mysql> EXPLAIN
18      -> SELECT * FROM name_mit ORDER BY nac
hname DESC, vorname DESC\G
19  ***** 1. row *****
20     id: 1
21     select_type: SIMPLE
22     table: name_mit
23     partitions: NULL
24     type: index
25     possible_keys: NULL
26     key: idx_namemit_nnv
27     key_len: 2046
28     ref: NULL
29     rows: 1500
```

```
30         filtered: 100.00
31         Extra: Backward index scan; Using index
x
```

Das erste Beispiel kann den Index zwar noch verwenden, muss danach aber nochmalig eine Sortierung vornehmen ([Zeile 15](#)). Auch das zweite Beispiel benutzt den Index, durch die in MySQL 8.0 neu geschaffene Möglichkeit des *backward index scan*.

In einem Experiment konnte ich keine Performanceunterschiede zum *forward index scan* feststellen.^{[11](#)}

```
1  mysql> EXPLAIN
2      -> SELECT * FROM name_mit ORDER BY id,
        nachname \G
3  ***** 1. row *****
*****
4          id: 1
5      select_type: SIMPLE
6          table: name_mit
7  [...]
8  Extra: Using index; Using filesort
```

Zuerst wird ein Index verwendet ([Zeile 8](#)), weil die *id* als Primärschlüssel automatisch einen Index hat. Anschließend wird ein Filesort benötigt, obwohl auf *nachname* ein Index liegt. Man kann also nicht zwei Indizes kombinieren^{[12](#)}.

```
1  mysql> EXPLAIN
2      -> SELECT * FROM name_mit ORDER BY vor
        name, nachname\G
3  ***** 1. row *****
*****
4          id: 1
5      select_type: SIMPLE
6          table: name_mit
```

```
7 [...]
8      Extra: Using index; Using filesort
```

Beim letzten Beispiel wird der Vorname mit dem Filesort sortiert. Innerhalb einer Gruppe gleicher Vornamen kann der Index auf Nachname benutzt werden.

Was auch interessant ist, dass bei kleinen Datenmengen trotz Index der Filesort verwendet wird. Vermutlich kann dieser bei kleinen Datenmengen sehr schnell im RAM durchgeführt werden.

Als Beispiel sei hier die Tabelle kunde mit nur sechs Datensätzen genannt:

```
1  mysql> EXPLAIN
2      -> SELECT * FROM kunde ORDER BY nachna
me, vorname\G
3  ***** 1. row *****
*****
4      id: 1
5      select_type: SIMPLE
6      table: kunde
7      partitions: NULL
8      type: ALL
9      possible_keys: NULL
10     key: NULL
11     key_len: NULL
12     ref: NULL
13     rows: 6
14     filtered: 100.00
15     Extra: Using filesort
```

10.4 Mehrfachausgaben unterbinden



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

```
SELECT [DISTINCT]
    {*}|spaltenliste|ausdruck}
FROM tabellenname
[WHERE bedingung]
[ORDER BY spaltenname [ASC|DESC] [,spaltenname
    [ASC|DESC]]*]
;
```

Wenn wir wissen wollen, in welchen Städten wir Adressen haben, so können wir jetzt schon eine entsprechende Auswertung programmieren:

```
1  mysql> SELECT
2      -> ort
3      -> FROM adresse
4      -> ORDER BY ort;
5  +-----+
6  | ort |
7  +-----+
8  | Bochum |
9  | Bochum |
10 | Bruchtal |
11 | Hobbingen |
12 | Hobbingen |
13 | Lugburz |
14 | Minas Tirith |
15 +-----+
```

Das ORDER BY ist dringend nötig, da wir sonst nicht erkennen würden, welche Orte mehrfach vorkommen. Bei unserem

Datenbestand ist das noch sehr übersichtlich. Stellen wir uns einen Adressbestand von mehreren Hundert Adressen vor, so wäre der obige SELECT sicherlich nicht verwendbar. Und warum nicht? Weil er mich mit vielen überflüssigen Informationen zuschüttet.

Hier hilft eine Option des SELECT, der DISTINCT:

```
1  mysql> SELECT
2      -> DISTINCT ort
3      -> FROM adresse
4      -> ORDER BY ort;
5  +-----+
6  | ort |
7  +-----+
8  | Bochum |
9  | Bruchtal |
10 | Hobbingen |
11 | Lugburz |
12 | Minas Tirith |
13 +-----+
```

Nun werden die Orte nur einmal angezeigt.

Der DISTINCT vergleicht *nicht* die komplette Zeile, sondern nur die Inhalte, der in *spaltenliste* angegebenen Spalten oder Ausdrücke, wie dieses Beispiel zeigt:

```
1  mysql> SELECT
2      -> DISTINCT nachname
3      -> FROM kunde
4      -> ORDER BY nachname;
5  +-----+
6  | nachname |
7  +-----+
8  | Beutlin |
9  | Earendilionn |
10 | Eichenschild |
```

```
11 | Gamdschie |
12 | Telcontar |
13 +-----+
```



Aufgabe 10.5: Ein DISTINCT, der Schlüsselspaltenwerte unverändert auswählt, ist zwar schnell, aber sinnlos.

Begründen Sie dies mithilfe der [Definition 8](#) in [Abschnitt 2.1.3](#).

10.4.1 Fallstudie: Datenimport von Bankdaten

Unsere Tabelle bank ist bisher leer. Die Datei `blz_20120305.csv`¹³ enthält die Liste aller deutscher Banken mit Namen und Bankleitzahl zum Zeitpunkt 03.05.2012. Ziel ist es, diese Daten für den Aufbau unserer bank-Tabelle zu verwenden.

Ein Blick in die Datei zeigt uns, welche Informationen uns hier angeboten werden. Für uns sind folgende Informationen interessant: Bankleitzahl und Bezeichnung. Die anderen Informationen sind sicherlich nicht unwichtig, werden hier aber nicht weiter berücksichtigt. Wir wollen diese Daten importieren, auswerten, vorbereiten und in die Tabelle bank kopieren.

Schritt 1: Importieren

Wir legen dazu eine temporäre Tabelle an, die die gewünschten Informationen aufnehmen kann:


```
1 CREATE TEMPORARY TABLE bank_import (  
2 bankleitzahl CHAR(8),  
3 bezeichnung VARCHAR(255)  
4 );
```

Nun werden die Daten mithilfe von LOAD DATA INFILE importiert (siehe [Abschnitt 7.1](#)):

```
1 SET GLOBAL local_infile = ON;  
2 LOAD DATA LOCAL INFILE 'blz_20120305.csv'  
3 INTO TABLE bank_import  
4 CHARACTER SET utf8mb4  
5 FIELDS  
6 TERMINATED BY ';'   
7 LINES  
8 TERMINATED BY '\n'  
9 IGNORE 1 LINES  
10 (bankleitzahl, @dummy, bezeichnung);  
11 SET GLOBAL local_infile = OFF;
```

Die @dummy-Variable wird dazu verwendet, die nicht benötigten Informationen beim Einlesen zu ignorieren. Die Warnungen weisen darauf hin, dass nach der Spalte bezeichnung Daten einfach abgeschnitten werden. Das ist aber in Ordnung, denn schließlich brauchen wir diese nicht.

Schritt 2: Auswerten

Aus dem Import wissen wir, dass in der Tabelle 19574 Zeilen sind. Wie viele davon sind unterschiedliche Banken? Unterscheiden die sich in der Bankleitzahl oder kommt diese mehrfach vor?

```
1 SELECT DISTINCT bankleitzahl FROM bank_imp  
ort;
```

Nun sind es nur noch 4120. Haben die alle die gleiche Bezeichnung?

```
1  SELECT DISTINCT bankleitzahl, bezeichnung
   FROM bank_import;
```

Leider nein, es sind jetzt wieder 6105 verschiedene Einträge. Somit kommen für eine Bankleitzahl ggf. mehrere Bezeichnungen in Frage. So kann die Bankleitzahl nicht als Schlüssel verwendet werden. Muss denn die Bankleitzahl der Schlüssel sein? Können wir nicht die Tabelle bank und bankverbindung so umbauen, dass mit einer laufenden Nummer als Primärschlüssel gearbeitet wird?

Klar, geht das. Also ran an den nächsten Schritt:

Schritt 3: Vorbereiten

Unter der Annahme, dass beide Tabellen noch leer sind, müssen wir die Tabelle bankverbindung so präparieren, dass die Constraints keine Schwierigkeiten mehr machen. Alle Fremdschlüssel und Indizes werden deshalb gelöscht.

```
1  ALTER TABLE bankverbindung
2  DROP FOREIGN KEY bankverbindung_ibfk_1,
3  DROP FOREIGN KEY bankverbindung_ibfk_2,
4  DROP INDEX idx_bankverbindung_bankid_konto
   nummer,
5  DROP PRIMARY KEY
6  ;
```

Nun wird die Tabelle bank umgebaut. Da das Löschen eines Primärschlüssels und das Anlegen eines neuen Primärschlüssels nicht mit *einem* ALTER TABLE möglich ist, müssen hier *zwei* programmiert werden. Zusätzlich wird jetzt

eine neue Spalte mit dem Namen blz hinzugefügt. Der Optik wegen wird diese direkt nach dem Primärschlüssel einsortiert.

```
1 ALTER TABLE bank
2 DROP PRIMARY KEY
3 ;
4
5 ALTER TABLE bank
6 MODIFY bank_id INT UNSIGNED AUTO_INCREMENT
PRIMARY KEY,
7 ADD blz CHAR(12) NOT NULL DEFAULT '' AFTER
bank_id
8 ;
```

Und zum Schluss werden die Indizes und Schlüssel in der Tabelle bankverbindung wieder hergestellt:

```
1 ALTER TABLE bankverbindung
2 MODIFY bank_id INT UNSIGNED,
3 ADD PRIMARY KEY(kunde_id, bankverbindung_n
r),
4 ADD INDEX idx_bankverbindung_bankid_konton
ummer (bank_id, kontonummer),
5 ADD FOREIGN KEY (kunde_id)
6 REFERENCES kunde(kunde_id)
7 ON UPDATE RESTRICT
8 ON DELETE RESTRICT,
9 ADD FOREIGN KEY (bank_id)
10 REFERENCES bank(bank_id)
11 ON UPDATE RESTRICT
12 ON DELETE RESTRICT
13 ;
```

Schritt 4: Kopieren

Und auf zum letzten Schritt:

```
1  INSERT INTO bank (blz, bankname)
2  SELECT DISTINCT bankleitzahl, bezeichnung
FROM bank_import
3  ;
4
5  CREATE INDEX idx_bank_blzbankname
6  ON bank (blz, bankname)
7  ;
```

Zum einen haben wir den `DISTINCT` für Untersuchungen verwendet. Auch wenn wir hier nur an der Anzahl interessiert waren¹⁴, lieferte er uns doch eine entscheidende Information. Ohne den Test auf Dubletten hätte wir 19574 anstelle von 6105 Zeilen in die Tabelle `bank` importiert. Na, wenn sich das nicht gelohnt hat!

Zumanderen wurde der `DISTINCT` aktiv dafür verwendet, beim Import nur die einmaligen Datensätze zu verwenden.

10.4.2 Was muss ich beim `DISTINCT` bzgl. der Performance beachten?

Im Prinzip das Gleiche wie beim `ORDER BY` (siehe [Abschnitt 10.3](#)). Das Ermitteln von Dubletten kann bei sortierten Listen erheblich schneller ablaufen als bei nicht sortierten. Liegt ein Index auf den `DISTINCT`-Spalten, kann der Zugriff direkt über den Index abgearbeitet werden.

Wie oben bereits angesprochen, ist es aber sinnlos, auf Indizes mit der Eigenschaft `UNIQUE` einen `DISTINCT` abzusetzen.

10.5 Ergebnismenge ausschneiden



MySQL/MariaDB

```
SELECT [DISTINCT]
    {*}|spaltenliste|ausdruck
FROM tabellenname
[WHERE bedingung]
[ORDER BY spaltenname [ASC|DESC] [,spaltenname
[ASC|DESC]]*]
[LIMIT [offset,] anzahl]
;
```

Unter der Überschrift *Befehle, die Sie nicht im MySQL Client ausführen sollten*:

```
1  SELECT * FROM bank;
```

Wie sinnlos, über 6000 Datensätze an sich vorbeiziehen zu sehen. In MySQL und MariaDB gibt es eine einfache Möglichkeit, die Ausgabemenge zu beschränken: LIMIT.

10.5.1 Wie kann ich die ersten *n* Datensätze ausschneiden?

Wir wollen uns die ersten drei Zeilen der Tabelle bank anschauen:

```
1  mysql> SELECT
2      -> blz, bankname
3      -> FROM bank
4      -> LIMIT 3;
```

```

5
6  +-----+-----+
-----+
7  | blz | bankname |
8  +-----+-----+
-----+
9  | 10000000 | Bundesbank |
10 | 10010010 | Postbank |
11 | 10010111 | SEB |
12 +-----+-----+
-----+

```

Durch die Angabe *anzahl* hinter dem Schlüsselwort LIMIT werden nur die ersten drei Datensätze ausgegeben. Dabei werden ggf. vorhandene ORDER BY-Klauseln berücksichtigt. Gerade in Zusammenarbeit mit dem ORDER BY lassen sich häufige vorkommende Auswertungen erstellen. Wann immer Sie das erste oder letzte Element einer Liste brauchen, wird mit LIMIT 1 gearbeitet, z.B. der Kunde mit dem höchsten bzw. geringsten Umsatz oder der teuerste bzw. billigste Artikel oder die Bank mit der höchsten Bankleitzahl:

```

1  mysql> SELECT
2      -> blz, bankname
3      -> FROM bank
4      -> ORDER BY blz DESC
5      -> LIMIT 1
6      -> ;
7
8  +-----+-----+
9  | blz | bankname |
10 +-----+-----+
11 | 87096214 | Volksbank Chemnitz (Gf P2) |
12 +-----+-----+

```

Beschränken Sie die Ausgabemenge nicht nur auf eine Zeile, sondern auch auf eine Spalte, lässt sich das Ergebnis in eine

Variable schreiben, um es beispielsweise weiter verarbeiten zu können:

```
1  mysql> SELECT
2      -> blz
3      -> FROM bank
4      -> ORDER BY blz DESC
5      -> LIMIT 1
6      -> INTO @blzmin
7      -> ;
8
9  mysql> SELECT
10     -> bankname
11     -> FROM bank
12     -> WHERE blz = @blzmin
13     -> ;
14
15  +-----+
16  | bankname |
17  +-----+
18  | Volksbank Chemnitz |
19  | Volksbank Chemnitz (Gf P2) |
20  +-----+
```

10.5.2 Wie kann ich Teilmengen mittendrin ausschneiden?

Besonders bei Online-Anwendungen werden größere Datenmengen seitenweise z.B. in 20er Paketen ausgegeben. Dazu muss aus der Ergebnismenge 20 Zeilen ab einem bestimmten *offset* ausgeschnitten werden. Wir wandeln das Beispiel so ab, dass immer nur zwei Zeilen ausgeschnitten werden:

```
1  mysql> SELECT
```

```

2      -> blz, bankname
3      -> FROM bank
4      -> LIMIT 1, 2
5      -> ;
6
7  +-----+-----+
8  | blz | bankname |
9  +-----+-----+
10 | 10010010 | Postbank |
11 | 10010111 | SEB |
12 +-----+-----+

```



Aufgabe 10.6: Was ist hier falsch? Bitte vergleichen Sie das Ergebnis mit der Ausgabe bei LIMIT 3.

Der *offset* ist ein 0 basierender Zähler. So ergeben folgende Befehle das gleiche Ergebnis:

```

1  mysql> SELECT
2      -> blz, bankname
3      -> FROM bank
4      -> LIMIT 1;
5  +-----+-----+
6  | blz | bankname |
7  +-----+-----+
8  | 10000000 | Bundesbank |
9  +-----+-----+
10
11 mysql> SELECT
12      -> blz, bankname
13      -> FROM bank
14      -> LIMIT 0, 1;
15 +-----+-----+
16 | blz | bankname |
17 +-----+-----+

```



```

18 | 100000000 | Bundesbank |
19 +-----+-----+

```



Aufgabe 10.7: Geben Sie die Bankleitzahl und den Banknamen der Zeilen 1000 und 1005 aus. Vergleichen Sie die Ergebnisse:

```

1  +-----+-----+
-----+
2  | blz | bankname |
3  +-----+-----+
-----+
4  | 28062560 | Volksbank Lohne-Mühlen
   (Gf P2) |
5  | 28062740 | Volksbank Bookholzberg-
   Lemwerder |
6  | 28062740 | Volksbank Bookholzberg-
   Lemwerder (Gf P2) |
7  | 28062913 | Volksbank Bösel |
8  | 28062913 | Volksbank Bösel (Gf P2)
   |
9  +-----+-----+
-----+

```



Hinweis: Leider ergibt der Quelltext unten Fehlermeldungen, da im MySQL oder MariaDB Client LIMIT nur Konstanten akzeptiert werden. Aber was wird hier versucht?

```

1  SET @offset=0;
2  SET @anzahl=2;

```

```
3  SELECT blz, bankname FROM bank ORDER BY bl
z, bankname LIMIT '@offset', '@anzahl';
4  SELECT @offset + @anzahl INTO @offset;
5  SELECT blz, bankname FROM bank ORDER BY bl
z, bankname LIMIT @offset, @anzahl;
6  SELECT @offset + @anzahl INTO @offset;
```



Hinweis: LIMIT ist kein Standard-SQL, sondern kommt so z.B. in MySQL, MariaDB und PostgreSQL vor. Wobei PostgreSQL eine etwas andere Syntax hat: LIMIT 2 OFFSET 0 anstelle von beispielsweise LIMIT 0,2. Andere Systeme haben aber andere Lösungen dafür: siehe [[Wik23s](#)].

10.6 Ergebnisse exportieren

10.6.1 Wie lege ich eine Exportdatei auf dem Server an?



MySQL/MariaDB

```
SELECT [DISTINCT] {*|spaltenliste|ausdruck}
FROM tabellenname
[WHERE bedingung]
[ORDER BY spaltenname [ASC|DESC] [,spaltenname
[ASC|DESC]]*]
[LIMIT [offset,] anzahl]
```

```
[INTO OUTFILE 'dateiname' exportoptionen]
;
```

exportoptionen: siehe Optionen zum LOAD DATA INFILE in [Abschnitt 7.1.2](#)

Wir haben in [Abschnitt 7.1](#) gesehen, wie man Daten per LOAD DATA INFILE in eine Tabelle importieren kann. Es gibt auch eine entsprechende Variante des SELECT, der Daten in CSV-Format exportiert.

Wenn wir beispielsweise Bankdaten wieder in ein neutrales Datenaustauschformat bringen möchten, würde man eine CSV-Datei INTO OUTFILE wie folgt erzeugen:

```
1  SET GLOBAL local_infile = ON;
2  SELECT
3  blz Bankleitzahl, bankname Bankname
4  INTO OUTFILE '/var/lib/mysql/interchange/b
ank.csv'
5  FIELDS
6  TERMINATED BY ';'
7  ENCLOSED BY '"'
8  LINES
9  TERMINATED BY '\r\n'
10 FROM
11   bank
12   ORDER BY
13   Bankleitzahl, Bankname;
14 SET GLOBAL local_infile = OFF;
```

Es wird die Datei bank.csv auf dem Server mit folgendem Inhalt erstellt:

```
1  "100000000";"Bundesbank"
2  "10010010";"Postbank"
```

```
3  "10010111";"SEB"  
4  "10010222";"The Royal Bank of Scotland, Ni  
ederlassung Deutschland"  
5  [...]
```



Hinweis: Bitte beachten Sie den Sicherheitshinweis zu SQL-Injection in [Abschnitt 7.2.5](#).

10.6.2 Wie lege ich eine Exportdatei auf dem Client an?

Grundsätzlich ist das keine so gute Idee, die CSV-Datei auf dem Server anzulegen. Man kann den MySQL Client aber gut dafür verwenden, eine lokale Datei anzulegen:

```
1  mysql -D oshop -e "SELECT blz, bankname FR  
OM bank;" > bank.txt
```

Nun ist eine lokale Datei erstellt worden. Diese ist nicht so frei konfigurierbar erstellt wie bei einem INTO outfile, aber mit gängigen Importassistenten oder Programmen wie awk lässt sich eine neutrale CSV-Datei erstellen.

Wenn Sie den SELECT – oder noch weitere SQL-Befehle – in eine Datei packen, so lässt sich das Ganze noch weiter vereinfachen¹⁵:

```
1  mysql -D oshop < befehle.sql > bank.txt
```

10.6.3 Wie lese ich mithilfe eines C#-Client

binäre Daten aus?

In [Abschnitt 7.2.4](#) haben wir Bilder in die Tabelle `bild` eingefügt. Diese Daten wollen wir nun wieder auslesen und in Bilddateien auf dem Client ablegen. Auch hier möchte ich dazu einen C#-Client verwenden.

Zunächst werden ein paar benötigte Objekte gebastelt. Mit `dlg` wird das Verzeichnis ausgewählt, in welches die Bilder abgespeichert werden sollen. Die `MySQLCommand`- und `MySQLConnection`-Objekte sollten noch aus dem Obigen klar sein. Da wir nun Daten empfangen wollen, brauchen wir einen Container, der diese Daten aufnimmt: der `MySQL-DataReader`. Die restlichen Objekte sollten ebenfalls in ihrer Bedeutung bekannt sein.

Listing 10.1 SELECT: Hauptfenster.cs, Teil 1

```
18     private void btnnBild_Click(object sender, EventArgs e) {
19         FolderBrowserDialog dlg = new Folder
    BrowserDialog();
20         MySqlConnection MySqlConnection = new MySq
    l.Data.MySqlClient.MySqlConnection();
21         MySqlCommand MySqlCommand = new MySql.Da
    ta.MySqlClient.MySqlCommand();
22         MySqlDataReader MySqlData;
23
24         string strSelect;
25         Int32 iDateigroesse;
26         byte[] binBilddaten;
27         FileStream streamDatei;
28         string strName;
```

Nun werden die Verbindungsparameter angegeben. Bitte beachten Sie, dass Sie ggf. andere Werte eintragen müssen. Besonders die Angabe `server` muss überprüft werden.

Listing 10.2 SELECT: Hauptfenster.cs, Teil 2

```
30      MySqlConnection.ConnectionString = "server
=127.0.0.1";
31      MySqlConnection.ConnectionString += ";uid=
root";
32      MySqlConnection.ConnectionString += ";pwd=
";
33      MySqlConnection.ConnectionString += ";data
base=oshop;charset=utf8mb4";
```

Mit diesem SELECT werden alle relevanten Bilddaten ausgewählt.

Listing 10.3 SELECT: Hauptfenster.cs, Teil 3

```
35      strSelect = "SELECT bild, dateiname,
dateigroesse FROM bild";
```

Nur wenn ein Verzeichnis ausgewählt wurde, soll das Auslesen begonnen werden.

Listing 10.4 SELECT: Hauptfenster.cs, Teil 4

```
37      if (dlg.ShowDialog() != DialogResult
.OK) {
38          return;
39      }
```

Jetzt wird versucht, eine Verbindung zum Server herzustellen und den SELECT auszuführen. Sie könnten hier noch eine Warnung einbauen, falls keine Daten gefunden wurden.

Listing 10.5 SELECT: Hauptfenster.cs, Teil 5

```
41      try {
42          MySqlConnection.Open();
43
44          MySqlCommand.Connection = MySqlConnection;
```

```

45         MySqlCommand.CommandText = strSelect;
46         MySqlData = MySqlCommand.ExecuteReader()
;

```

Solange Daten vom MySqlDataReader-Objekt gefunden werden, läuft die while-Schleife. Durch Methoden des MySqlDataReader-Objekts wie GetOrdinal() und GetString() werden die Ergebnisse der Auswahl mit SELECT in passende C#-Variablen kopiert. Dabei wird bei jedem Schleifendurchlauf eine entsprechende Datei im ausgewählten Verzeichnis angelegt, und die Binärdaten werden mit Write() dort hineingeschrieben.

Listing 10.6 SELECT: Hauptfenster.cs, Teil 6

```

52         while (MySqlData.Read()) {
53             iDateigroesse = MySqlData.GetInt32(MySqlData.GetOrdinal("dateigroesse"));
54             binBilddaten = new byte[iDateigroesse];
55
56             MySqlData.GetBytes(MySqlData.GetOrdinal("bild"), 0,
57                 binBilddaten, 0, iDateigroesse);
58             strName = dlg.SelectedPath + Path.DirectorySeparatorChar
59                 + MySqlData.GetString("dateiname");
60             streamDatei = new FileStream(Path.GetFullPath(strName),
61                 FileMode.OpenOrCreate,
62                 FileAccess.Write);
63             streamDatei.Write(binBilddaten, 0, iDateigroesse);
64             streamDatei.Close();
65             lbDateien.Items.Insert(0, Path.GetFullPath(strName));
66         }

```

Zum Schluss noch alle offenen Verbindungen schließen und Ausnahmen auffangen:

Listing 10.7 SELECT: Hauptfenster.cs, Teil 7

```
68      MySqlData.Close();
69      MySqlConnection.Close();
70      }
71      catch (Exception ex) {
72          MessageBox.Show(ex.Message, "Error",
73                          MessageBoxButtons.OK, MessageBoxIcon
                          .Error);
```

¹ Wobei SQL-Quelltext anders als beispielsweise ein C-Programm nicht in eine EXE-Datei umgewandelt wird. Aber die SQL-Befehle werden in ein Format gebracht, welches von einer Laufzeitumgebung bzw. einem Interpreter ausgeführt werden kann.

² In PostgreSQL und T-SQL ergibt $9/4$ das gleiche Ergebnis wie $\text{DIV}(9,4)$. Sind die Werte der Division Integer, wird die ganzzahlige Division genommen. Ein $9.0/4.0$ bringt das gewünschte Ergebnis.

³ Ergänzende Beispiele finden Sie unter [\[MyS18b\]](#).

⁴ Siehe [Teil V](#).

⁵ Vermutlich entsteht der Fehler dadurch, dass man bei der imperativen Programmierung immer die Datenobjekte angeben muss, mit denen man gerade arbeiten will.

⁶ Ich habe hier das Schlüsselwort ASC nur des besseren Verständnisses wegen angegeben.

⁷ Ich weiß selber, dass das hier kein mathematischer Beweis ist :-).

⁸ Die Daten werden nicht zwingend in Dateien sortiert. Bis zu einer bestimmten Datenmenge kann das auch im RAM geschehen.

⁹ Der genaue Versuchsaufbau kann in [Abschnitt 27.7](#) nachgelesen werden.

¹⁰ Siehe [[Wik23h](#)].

¹¹ Sie finden das Experiment in `perfIndexOrderBy02.sql`.

¹² Natürlich ist Ihnen aufgefallen, dass es völlig sinnlos ist, auch noch nach dem Nachnamen zu sortieren. Wie, es ist Ihnen nicht aufgefallen? Warum ist es denn sinnlos?

¹³ Siehe [[Bun16](#)].

¹⁴ Was wir mit `COUNT(*)` etwas bequemer hätten haben können (siehe [Abschnitt 26.2.3](#)).

¹⁵ Das klappt übrigens auch in der COMMAND-Box unter Windows.

11 Tabellen verbinden



Drum prüfe, wer sich ewig bindet, ob FK auch zu PK findet: Daten für Auswertungen verknüpfen.

- Grundkurs
 - Kartesisches Produkt, CROSS JOIN
 - INNER JOIN mit zwei und mehr Tabellen
 - LEFT OUTER JOIN 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 müssen Sie das Handwerk dahinter verstehen, denn jeder DB-Designer muss wissen, wie die Daten später wieder zusammengebastelt werden müssen. Ohne den Aufwand zu kennen, lässt sich kein seriöses ER-Modell erstellen.¹



Die Quelltexte dieses Kapitels stehen in der Datei `listing08.sql` (siehe [Listing 28.8](#), [Listing 28.48](#) und [Listing 28.33](#)).

11.1 Heiße Liebe: Primär-Fremdschlüsselpaare



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

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

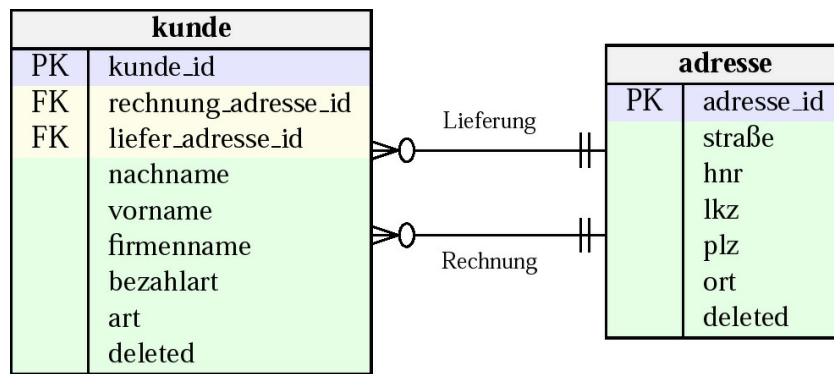


Bild 11.1 ER-Modell: kunde und adresse

Die bisher durchgeführten Auswertungen fanden immer auf *einer* Tabelle statt. Wenn wir uns die ER-Modelle für den Online-Shop anschauen, erkennen wir, dass inhaltlich zusammenhängende Informationen oft auf mehrere Tabellen verteilt sind.²

So stehen beispielsweise die Adressdaten eines Kunden in einer anderen Tabelle als sein Name. Für ein Rechnungsschreiben werden beide Informationen wieder miteinander zu verknüpfen sein. Eine Variante des SELECT erlaubt die Verwendung mehrerer Tabellen in einem SELECT. Wollen wir mal schauen, was dabei herauskommt:

```

1  mysql> SELECT
2      -> kunde_id, nachname, vorname, rechnu
ng_adresse_id, adresse_id
3      -> FROM
4      -> kunde, adresse
5      -> ;
6  +-----+-----+-----+-----+-----+
+-----+
7  | kunde_id | nachname | vorname | rechnung
_adresse_id | adresse_id |
8  +-----+-----+-----+-----+-----+
+-----+
9  | 1 | Gamdschie | Samweis | 1 | 1 |
  
```

10		2		Beutlin		Frodo		2		1	
11		3		Beutlin		Bilbo		2		1	
12		4		Telcontar		Elessar		3		1	
13		5		Earendilionn		Elrond		4		1	
14		6		Eichenschild		Thorin		NULL		1	
15		1		Gamdschie		Samweis		1		2	
16		2		Beutlin		Frodo		2		2	
17		3		Beutlin		Bilbo		2		2	
18		4		Telcontar		Elessar		3		2	
19		5		Earendilionn		Elrond		4		2	
20		6		Eichenschild		Thorin		NULL		2	
21		1		Gamdschie		Samweis		1		3	
22		2		Beutlin		Frodo		2		3	
23		3		Beutlin		Bilbo		2		3	
24		4		Telcontar		Elessar		3		3	
25		5		Earendilionn		Elrond		4		3	
26		6		Eichenschild		Thorin		NULL		3	
27		1		Gamdschie		Samweis		1		4	
28		2		Beutlin		Frodo		2		4	
29		3		Beutlin		Bilbo		2		4	
30		4		Telcontar		Elessar		3		4	
31		5		Earendilionn		Elrond		4		4	
32		6		Eichenschild		Thorin		NULL		4	
33		1		Gamdschie		Samweis		1		5	
34		2		Beutlin		Frodo		2		5	
35		3		Beutlin		Bilbo		2		5	
36		4		Telcontar		Elessar		3		5	
37		5		Earendilionn		Elrond		4		5	
38		6		Eichenschild		Thorin		NULL		5	
39		1		Gamdschie		Samweis		1		10	
40		2		Beutlin		Frodo		2		10	
41		3		Beutlin		Bilbo		2		10	
42		4		Telcontar		Elessar		3		10	
43		5		Earendilionn		Elrond		4		10	
44		6		Eichenschild		Thorin		NULL		10	
45		1		Gamdschie		Samweis		1		11	
46		2		Beutlin		Frodo		2		11	
47		3		Beutlin		Bilbo		2		11	

```

48 | 4 | Telcontar | Elessar | 3 | 11 |
49 | 5 | Earendillion | Elrond | 4 | 11 |
50 | 6 | Eichenschild | Thorin | NULL | 11 |
51 +-----+-----+-----+-----+
-----+-----+
52 42 rows in set (0.00 sec)

```

In [Zeile 4](#) werden zwei Tabellen hinter dem FROM angegeben. Das ist neu; bisher stand hier immer nur *die eine* Tabelle.

Im Ergebnis werden mir 42 Zeilen einer *neuen* Tabelle angezeigt. Aber wie sind diese Zeilen gebildet worden? Es fällt auf, dass die kunde_id sich nach dem Schema 1,2, 3, 4, 5,6 wiederholt. Ebenso interessant ist, dass die adresse_id mit jeweils sechs gleichen Werten auftaucht.

Betrachten Sie nur die Werte von kunde_id und adresse_id, so erkennen Sie Datenpaare, die wie folgt aufgebaut sind: Jede Adresse ist mit allen Kunden kombiniert worden. Adresse 1 mit Kunde 1 bis 6, Adresse 2 mit Kunde 1 bis 6 usw. Da es sechs Kunden und sieben Adressen gibt, erhalten wir 42 Kombinationen, das *Kartesische Produkt*.



Definition 40: Kartesisches Produkt

Seien A und B zwei endliche Mengen, $a \in A$ und $b \in B$, dann ist die Menge aller unterschiedlichen Paare (a,b) das *Kartesische Produkt* K der Mengen A und B .

Diese Definition lässt $A = B$ zu. Wenn n die Anzahl der Elemente in A ist und m die Anzahl der Elemente in B , dann hat K $n \times m$ viele Elemente.



Definition 41: CROSS JOIN

Das Kartesische Produkt zweier Mengen wird auch *CROSS JOIN* oder *Kreuzprodukt* genannt.

Toll, ein Kartesisches Produkt³, ja und?



Aufgabe 11.1: Betrachten Sie genau die Zahlenpaare `rechnung_adresse_id` und `adresse_id`. Formulieren Sie eine `WHERE`-Klausel, die genau die Zeilen übrig lässt, die zu einem Kunden die richtige Adressnummer angeben.

In den [Zeilen 9, 16, 17, 24](#) und [31](#) stehen jeweils die gleichen Werte. Ausformuliert bedeutet dies: Im Fremdschlüssel `rechnung_adresse_id` steht der gleiche Wert wie im Primärschlüssel `adresse_id`. Das sind genau die Elemente des Kartesischen Produkts, die eine gültige Verknüpfung zwischen den beiden Tabellen darstellen. Die `WHERE`-Klausel sollte somit nur diese Zeilen übrig lassen:

```
1  mysql> SELECT
2      -> kunde_id, nachname, vorname, rechnu
rechnung_adresse_id, adresse_id
3      -> FROM
4      -> kunde, adresse
5      -> WHERE
6      -> rechnung_adresse_id = adresse_id
7      -> ;
8  +-----+-----+-----+-----+
+-----+
9  | kunde_id | nachname | vorname | rechnung
_adresse_id | adresse_id |
10 +-----+-----+-----+-----+
```

```

-----+-----+
11  | 1 | Gamdschie | Samweis | 1 | 1 |
12  | 2 | Beutlin   | Frodo   | 2 | 2 |
13  | 3 | Beutlin   | Bilbo   | 2 | 2 |
14  | 4 | Telcontar | Elessar | 3 | 3 |
15  | 5 | Earendil  | Elrond  | 4 | 4 |
16  +-----+-----+-----+-----+
-----+-----+
17  5 rows in set (0.00 sec)

```



Aufgabe 11.2: Was ist mit den Adressen mit den Primärschlüsselwerten 5, 10 und 11 passiert? Wie können Sie das inhaltlich interpretieren? Und was ist mit dem Kunden *Thorin Eichenschild*?

Die Reduzierung des Kartesischen Produkts auf die Zeilen mit passenden Schlüsselpaaren ist schon ein `INNER JOIN`. Was uns nun noch fehlt, ist eine *coole SELECT-Erweiterung* dazu.

11.2 INNER JOIN zwischen zwei Tabellen



Definition 42: INNER JOIN

Der *INNER JOIN* zweier Tabellen ist die Teilmenge des kartesischen Produkts, für welche gilt, dass die Fremdschlüsselwerte zu den Primärschlüsselwerten passen.

Die Formulierung über das Kartesische Produkt mit der passenden `WHERE`-Klausel ist für die Programmierung ein

wenig sperrig. Stellen Sie sich vor, dass die Ergebnismenge weitere Bedingungen erfüllen muss oder keine echten Tabellen verwendet werden, sondern Unterabfragen⁴. Deshalb gibt es eine eigene Syntax für den INNER JOIN:



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

```
SELECT [DISTINCT]
{*|spaltenliste|ausdruck}
FROM
tabellennamefk INNER JOIN tabellennamepk
ON tabellennamefk.fk = tabellennamepk.pk
[WHERE bedingung]
[ORDER BY spaltenname [ASC|DESC] [,spalten
name [ASC|DESC]]*]
[LIMIT [offset,] anzahl]
[INTO OUTFILE 'dateiname' exportoptionen]
;
```

Der obige Befehl sähe umgebaut so aus:

```
1  SELECT
2      kunde_id, nachname, vorname, rechnung_ad
   resse_id, adresse_id
3      FROM
4      kunde INNER JOIN adresse ON rechnung_adr
   esse_id = adresse_id
5  ;
```



Aufgabe 11.3: Bauen Sie den 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
| Bruchtal |
+-----+-----+-----+
-+-----+-----+

```

11.2.1 Bauanleitung für einen INNER JOIN

Die Programmierung eines INNER JOIN fällt meinen Schülerinnen und Schülern oft schwer. Deshalb will ich hier ein wenig ausführlicher beschreiben, wie Sie einen INNER JOIN zusammenbauen können. Die passende Aufgabe dazu: 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:

bankverbindung
kunde_id
bankverbindung_id
bank_id

bank
bank_id

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:

bankverbindung
kunde_id
bankverbindung_id
bank_id

bank
bank_id

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

bankverbindung
kunde_id
bankverbindung_id
bank_id

bank
bank_id

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:

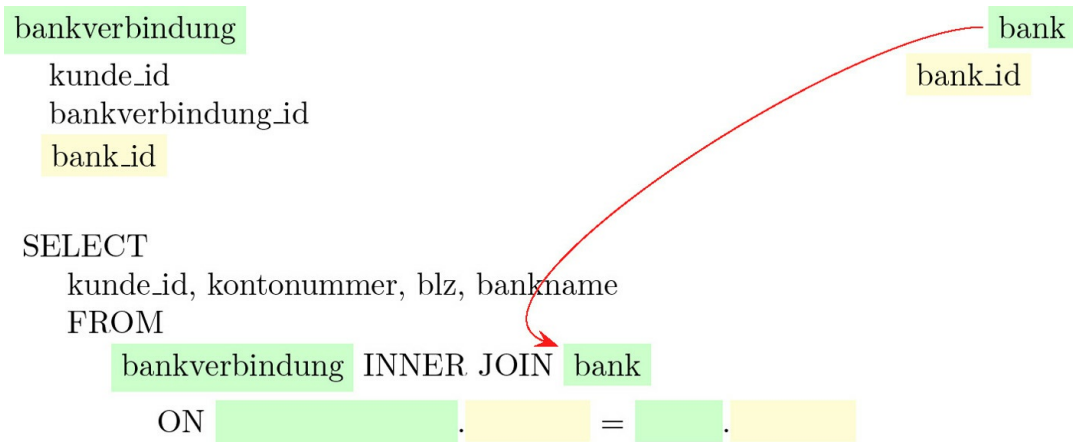
bankverbindung
kunde_id
bankverbindung_id
bank_id

bank
bank_id

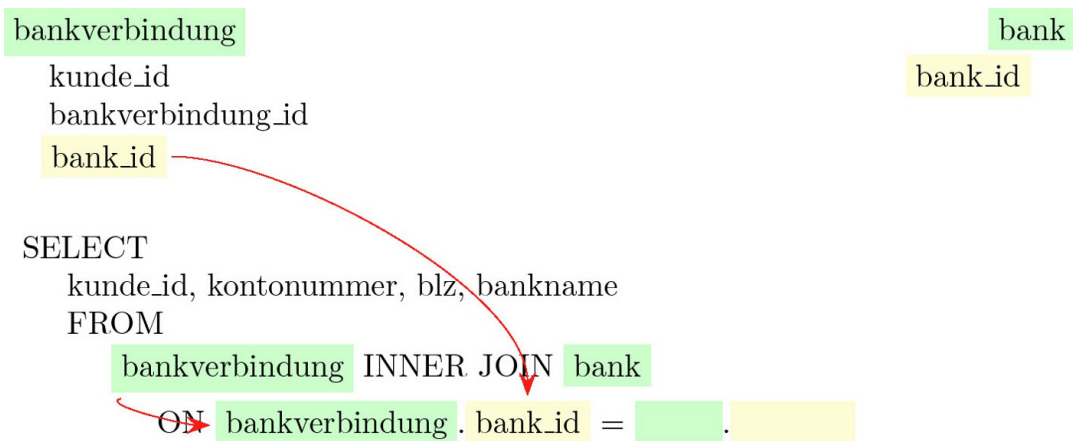
```
SELECT
  kunde_id, kontonummer, blz, bankname
FROM
  bankverbindung INNER JOIN
  ON . = .
```



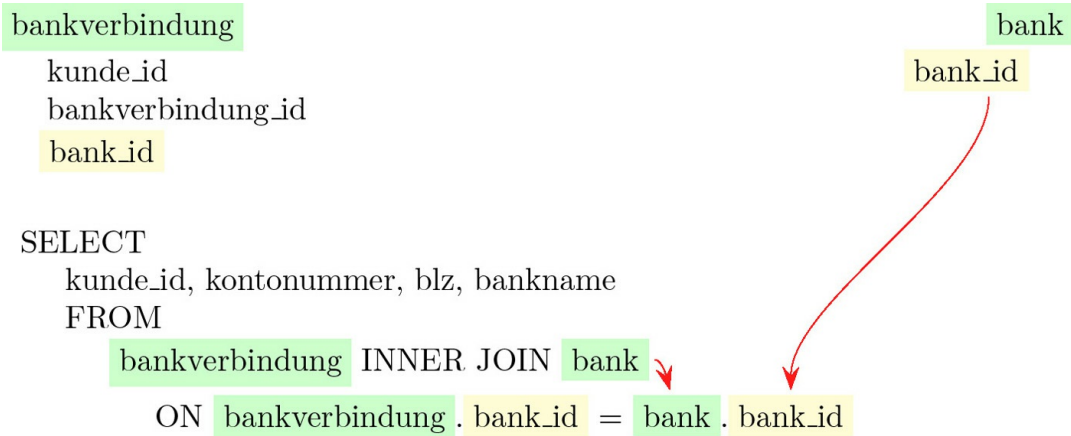
7. **Primärschlüsseltabelle eintragen:** Fügen Sie rechts vom INNER JOIN den Tabellennamen für die Tabelle mit dem markierten Primärschlüssel ein:



8. **Fremdschlüssel eintragen:** Fügen Sie zwischen dem ON und dem Gleichheitszeichen den Namen der Fremdschlüsseltabelle, einen Punkt und den Namen des Fremdschlüssels ein:



9. **Primärschlüssel eintragen:** Fügen Sie nach dem = den Namen der Primärschlüsseltabelle, einen Punkt und den Namen des Primärschlüssels ein:



10. Fertig!

```

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

Ein zweites Beispiel: Zu einem Kunden werden die Kontonummern ausgegeben.

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, kontonu
mmer;
```

	nachname	vorname	kontonummer
	Beutlin	Bilbo	3333333331
	Beutlin	Frodo	2222222221
	Earendillionn	Elrond	5555555551
	Gamdschie	Samweis	1111111111
	Gamdschie	Samweis	1111111112
	Telcontar	Elessar	4444444441

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 Lieferanten den Namen, Firmennamen und die Adresse aus.

Aufgabe 11.7: 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 11.8: Geben Sie zu jeder Bestellung die Kundennummer, das Bestelldatum und die Positionen aus. Die Sortierung soll nach Kundennummer, Bestellnummer und Position erfolgen.

Aufgabe 11.9: Geben Sie zu jeder Position den Artikelnamen aus. Es soll nach Artikelname sortiert werden.

11.2.2 Abkürzende Schreibweisen



SQL:2023, 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] [, spalten
name [ASC|DESC]]*]
[LIMIT [offset,] anzahl]
[INTO OUTFILE 'dateiname' exportoptionen]
;
```

Falls der Name des Fremdschlüssels genau der gleiche ist wie der des Primärschlüssels, können Sie die ON-Klausel durch eine kürzere Variante mit USING ersetzen:

```

1  SELECT
2      kunde_id, kontonummer, blz, bankname
3      FROM
4      bankverbindung INNER JOIN bank USING(b
ank_id)
5  ;

```

Bei den Fremdschlüsseln, die anders als die Primärschlüssel heißen, wie beispielsweise `rechnung_adresse_id`, ist eine solche Abkürzung nicht möglich.



Aufgabe 11.10: 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 heißen und die Verknüpfung definieren, spricht man von einem NATURAL JOIN.



Definition 43: NATURAL JOIN

Werden zwei Tabellen über die Spalten verknüpft, die den gleichen Namen haben, spricht man von einem *NATURAL JOIN*.

```

1  SELECT
2      kunde_id, kontonummer, blz, bankname
3      FROM
4      bankverbindung NATURAL JOIN bank
5  ;

```



Hinweis: Bitte beachten Sie, dass bei unseren Tabellen in einem NATURAL JOIN auch die Spalte deleted in die Verknüpfung mit einfließt.

11.2.3 Als Datenquelle für temporäre Tabellen

Wir haben oben die Bankverbindung zu einem Kunden ermittelt. Annahme: Wir wollen jetzt viele Auswertungen der Kunden inklusive der Bankverbindung machen, dann müssen jedes Mal im entsprechenden SELECT die Informationen mit INNER JOIN verknüpft werden.

Obwohl INNER JOIN-Anweisungen durch passend gewählte Indizes sehr beschleunigt werden, entsteht trotzdem eine Rechnerlast auf dem Server, die jedes Mal das gleiche – oder fast das gleiche – Ergebnis liefert.⁵

Nun können wir plausibel annehmen, dass sich die Kundenstammdaten (siehe [Definition 36](#)) für einen gewissen Auswertungszeitraum nicht ändern. Daher könnten wir statt dessen das Ergebnis des INNER JOIN in eine (temporäre) Tabelle ablegen und mit dieser weiterarbeiten.

Als Beispiel wollen wir die Kundendaten mit Rechnungsanschrift und allen Bestellungen ausgeben und danach die Kundendaten mit Rechnungsanschrift mit allen Bankverbindungen.

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

```
1  SELECT
2  k.kunde_id, k.nachname, k.vorname, a.stras
se, a.hnr, a.plz, a.ort
```

```

3 FROM
4     kunde k INNER JOIN adresse a ON k.rech
nung_adresse_id = a.adresse_id
5 ;

```

Sie bemerken bitte, dass hier abkürzende Alias ([Zeile 4](#)) verwendet werden. Das ist gerade bei Verknüpfungen sehr beliebt, weil 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 ON k.r
echnung_adresse_id = a.adresse_id;
6
7  Query OK, 5 rows affected (0.09 sec)
8  Records: 5 Duplicates: 0 Warnings: 0
9
10 mysql> SELECT kunde_id, nachname, ort FROM
    tmp_kadresse;
11  +-----+-----+-----+
12  | kunde_id | nachname | ort |
13  +-----+-----+-----+

```

```

14 | 1 | Gamdschie | Hobbingen |
15 | 2 | Beutlin | Hobbingen |
16 | 3 | Beutlin | Hobbingen |
17 | 4 | Telcontar | Minas Tirith |
18 | 5 | Earendilionn | Bruchtal |
19 +-----+-----+-----+-----+

```

Jetzt zur ersten Auswertung: Alle Kunden mit Adressdaten und ihren Bestellungen:

```

1  mysql> SELECT
2      -> t.kunde_id, t.nachname, t.ort, b.be
stellung_id, DATE(b.datum)
3      -> FROM
4      -> bestellung b INNER JOIN tmp_kadress
e t USING(kunde_id);
5
6  +-----+-----+-----+-----+
-----+-----+
7  | kunde_id | nachname | ort | bestellung_i
d | DATE(b.datum) |
8  +-----+-----+-----+-----+
-----+-----+
9  | 1 | Gamdschie | Hobbingen | 1 | 2012-03-
24 |
10 | 2 | Beutlin | Hobbingen | 2 | 2012-03-23
   |
11 +-----+-----+-----+-----+
-----+-----+

```



Aufgabe 11.11: 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 b
a USING(bank_id);
7
8  mysql> CREATE TEMPORARY TABLE tmp_kbankein
zug
9      -> SELECT
10     -> ka.*, kb.bankverbindung_nr, kb.kont
onummer,
11     -> kb.iban, kb.blz, kb.bankname
12     -> FROM
13     -> tmp_kadresse ka INNER JOIN tmp_kban
k kb USING(kunde_id);
14
15  mysql> SELECT kunde_id, nachname, ort, ban
kname 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 B	
				ank					
25	+	-----	+	-----	+	-----	+	-----	+
		-----	+						

Das Zusammenspiel der tatsächlichen und temporären Tabellen sei hier in einer Baumstruktur dargestellt. Die Blätter⁶ sind die Ursprungstabellen. Zwei davon werden in jeweils eine temporäre Tabelle mit INNER JOIN zusammengefasst. Die beiden neu erzeugten temporären Tabellen werden wiederum in eine temporäre Tabelle vereinigt.

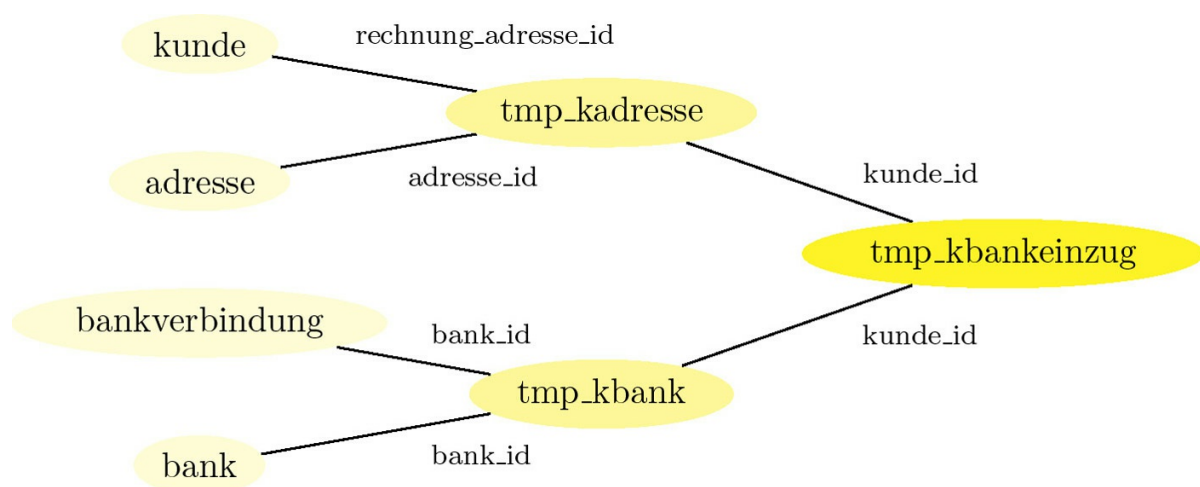


Bild 11.2 Temporäre Tabellen als Datenquelle

Jede der Tabellen kann nun unabhängig voneinander verwendet werden, da die Datensätze in den temporären Tabellen keine Verweise auf die ursprünglichen Datensätze sind, sondern Kopien. Diese *Unabhängigkeit* ist sehr wichtig, wenn Sie mit vielen konkurrierenden Zugriffen auf die Tabellen kunde etc. rechnen.

Bei der Betrachtung von Transaktionen (siehe [Kapitel 19](#)) werden wir noch sehen, dass Tabellen für Operationen gesperrt werden. Durch die temporären Tabellen kann aber auf der Tabelle künde gearbeitet werden, während parallel dazu 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?: JOIN über Nichtschlüsselspalten

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) unsigned | NO | | 0 | NULL
6  | nachname | varchar(255) | NO | | | NULL
   |
```



```

7  | vorname | varchar(255) | NO | | | NULL |
8  | strasse | varchar(255) | NO | | | NULL |
9  | hnr     | varchar(255) | NO | | | NULL |
10 | plz     | char(9)      | NO | | | NULL |
11 | ort     | varchar(255) | NO | | | NULL |
12 +-----+-----+-----+-----+-----+
-----+-----+
13
14 mysql> DESCRIBE tmp_kbank;
15 +-----+-----+-----+-----+-----+
+-----+-----+-----+
16 | Field | Type | Null | Key | Default | Extra |
17 +-----+-----+-----+-----+-----+
+-----+-----+-----+
18 | kunde_id | int unsigned | NO | | NULL |
NULL |
19 | bankverbindung_nr | int unsigned | NO |
| NULL | NULL |
20 | kontonummer | char(25) | NO | | | NULL |
21 | iban | char(34) | NO | | | NULL |
22 | blz | char(12) | NO | | | NULL |
23 | bankname | varchar(255) | NO | | | NULL
|
24 +-----+-----+-----+-----+-----+
+-----+-----+-----+

```

Und tatsächlich: Für einen INNER JOIN ist es nicht nötig, Primär-Fremdschlüsselpaare zu bilden. Dem Befehl ist es völlig egal, was bei einem INNER JOIN in der ON- oder USING-Klausel steht. Es wird nur die Verträglichkeit der Datentypen untersucht.

Natürlich erfolgt die überwiegende Anzahl der Verknüpfungen auf Primär-Fremdschlüsselpaaren. Aber hier haben wir ein Beispiel dafür, dass auch die Verknüpfung über Nicht-schlüsselspalten⁷ 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 ON k.r
   echnung_adresse_id <= a.adresse_id;
5  +-----+-----+-----+-----+---
   ---+-----+-----+
6  | kunde_id | vorname | strasse | hnr | plz
   | ort |
7  +-----+-----+-----+-----+---
   ---+-----+-----+
8  | 1 | Samweis | Beutelhaldenweg | 5 | 6767
6 | Hobbingen |
9  | 1 | Samweis | Beutelhaldenweg | 1 | 6767
6 | Hobbingen |
10 | 2 | Frodo | Beutelhaldenweg | 1 | 67676
   | Hobbingen |
11 | 3 | Bilbo | Beutelhaldenweg | 1 | 67676
   | Hobbingen |
12 | 1 | Samweis | Auf der Feste | 1 | 54786
   | Minas Tirith |
13 | 2 | Frodo | Auf der Feste | 1 | 54786 |
   Minas Tirith |
14 | 3 | Bilbo | Auf der Feste | 1 | 54786 |
   Minas Tirith |
15 | 4 | Elessar | Auf der Feste | 1 | 54786
   | Minas Tirith |
16 | 1 | Samweis | Letztes Haus | 4 | 87567 |
   Bruchtal |
17 | 2 | Frodo | Letztes Haus | 4 | 87567 | B
   ruchtal |
18 | 3 | Bilbo | Letztes Haus | 4 | 87567 | B
   ruchtal |
19 [...]
```

```

20 | 2 | Frodo | Baradur | 1 | 62519 | Lugbur
z |
21 | 3 | Bilbo | Baradur | 1 | 62519 | Lugbur
z |
22 | 4 | Elessar | Baradur | 1 | 62519 | Lugb
urz |
23 | 5 | Elrond | Baradur | 1 | 62519 | Lugbu
rz |
24 | 1 | Samweis | Hochstrasse | 4a | 44879 |
Bochum |
25 | 2 | Frodo | Hochstrasse | 4a | 44879 | B
ochum |
26 | 3 | Bilbo | Hochstrasse | 4a | 44879 | B
ochum |
27 | 4 | Elessar | Hochstrasse | 4a | 44879 |
Bochum |
28 | 5 | Elrond | Hochstrasse | 4a | 44879 |
Bochum |
29 | 1 | Samweis | Industriegebiet | 8 | 4487
8 | Bochum |
30 | 2 | Frodo | Industriegebiet | 8 | 44878
| Bochum |
31 | 3 | Bilbo | Industriegebiet | 8 | 44878
| Bochum |
32 | 4 | Elessar | Industriegebiet | 8 | 4487
8 | Bochum |
33 | 5 | Elrond | Industriegebiet | 8 | 44878
| Bochum |
34 +-----+-----+-----+-----+
---+-----+-----+
35 28 rows in set (0.00 sec)

```

In [Zeile 4](#) 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](#) 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 in [Abschnitt 11.2.3](#) 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](#)). Wir haben eine $n:m$ -Verknüpfung zwischen den Tabellen `artikel` und `warengruppe`.

Unser erster Versuch besteht darin, wie oben beschrieben vorzugehen, indem wir zwei $1:n$ -Verknüpfungen erstellen:

1. **Ermitteln der beteiligten Tabellen:**
`artikel_nm_warengruppe` und `artikel`.
2. **Ermitteln der Primär- und Fremdschlüssel:** In der Tabelle `artikel_nm_warengruppe` gibt es keine *echten* Primärschlüssel, aber die beiden Fremdschlüssel `warengruppe_id` und `artikel_id`. In der Tabelle `artikel` gibt es nur den Primärschlüssel `artikel_id`.
3. **Fremdschlüssel festlegen:** Laut Namenskonvention muss `artikel_id` in der Tabelle `artikel_nm_warengruppe` der gesuchte Fremdschlüssel sein.
4. **Primärschlüssel festlegen:** Wir markieren `artikel_id` in `artikel`.
5. **Schablone benutzen:** Jetzt schreiben Sie auf das Blatt die Schablone.
6. **Fremdschlüsseltabelle eintragen:** Fügen Sie links vom INNER JOIN den Tabellennamen für die Tabelle mit dem markierten Fremdschlüssel ein.
7. **Primärschlüsseltabelle eintragen:** Fügen Sie an die passende Stelle den Tabellennamen für die Tabelle mit dem markierten Primärschlüssel ein.

8. **Fremdschlüssel eintragen:** Fügen Sie an die passende Stelle den Spaltennamen des markierten Fremdschlüssels ein.
9. **Primärschlüssel eintragen:** Fügen Sie an die passende Stelle den Spaltennamen des markierten Primärschlüssels ein.
10. **Fertig!**

```

1  mysql> SELECT
2      -> a.bezeichnung, nm.warengruppe_id
3      -> FROM
4      -> artikel_nm_warengruppe nm INNER JOI
N artikel a USING(artikel_id);
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 INNER JOI
N warengruppe w USING(warengruppe_id);
5  +-----+-----+
6  | bezeichnung | artikel_id |
7  +-----+-----+
8  | Bürobedarf | 3001 |
9  | Bürobedarf | 3005 |
10 | Bürobedarf | 3006 |
11 | Bürobedarf | 3007 |
12 | Bürobedarf | 3010 |
13 | Gartenbedarf | 7856 |
14 | Gartenbedarf | 7863 |
15 | Gartenbedarf | 9010 |
16 | Gartenbedarf | 9015 |
17 | Pflanzen | 7856 |
18 | Pflanzen | 7863 |
19 | Werkzeug | 9010 |
20 | Werkzeug | 9015 |
21 +-----+-----+

```

Und jetzt kommt's: Wir können sehen, dass die [Zeile 4](#) selbst als Ergebnis eine neue Tabelle erzeugt, denn das Ergebnis besteht aus Zeilen und Spalten (siehe [Definition 7](#) in [Abschnitt 2.1.2](#)). Und tatsächlich können wir diese Zeile anstelle einer einzelnen Tabelle in den ersten Befehl oben einsetzen:

```

1  SELECT
2  w.bezeichnung, a.bezeichnung
3  FROM
4  artikel_nm_warengruppe nm INNER JOIN waren
gruppe w USING(warengruppe_id)
5              INNER JOIN artikel a USING(art
ikel_id)
6  ;

```

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

```

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:2023, PostgreSQL, T-SQL

```

SELECT [DISTINCT]
{*|spaltenliste|ausdruck}
FROM
tablename [INNER JOIN tablename
ON tablename.spaltenname = tablename.spalten
name)]*
[WHERE bedingung]
[ORDER BY spaltenname [ASC|DESC] [,spalt
enname [ASC|DESC]]*]
[. . .]

```



```

;
MySQL/MariaDB
SELECT [DISTINCT]
{*|spaltenliste|ausdruck}
FROM
tablename [INNER JOIN tablename
{ON tablename.spaltenname = tablename.spalte
nname|USING(spaltenname)}]*
[WHERE bedingung]
[ORDER BY spaltenname [ASC|DESC] [,spalt
enname [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 11.12: Warum konnte beim letzten SELECT kein NATURAL JOIN verwendet werden?

Aufgabe 11.13: Erweitern Sie diesen SELECT um die Positionen, in denen der Artikel vorkommt.

Aufgabe 11.14: Erweitern Sie das Ergebnis der letzten Aufgabe um die Daten der Bestellung.

Aufgabe 11.15: Erweitern Sie das Ergebnis der letzten Aufgabe um die Daten des Kunden.

Aufgabe 11.16: 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,
3      -> b.datum
4      -> FROM
5      -> bestellung b INNER JOIN kunde k USI
6      -> NG(kunde_id)
7      -> ORDER BY
8      -> k.nachname, k.vorname
9      -> ;
10
11  +-----+-----+-----+-----+
12  | kunde_id | nachname | vorname | datum |
13  +-----+-----+-----+-----+
14  | 2 | Beutlin | Frodo | 2012-03-23 16:11:00 |
15  | 1 | Gamdschie | Samweis | 2012-03-24 17:41:00 |
16  +-----+-----+-----+-----+
```



Aufgabe 11.17: Es werden zwar alle Bestellungen ausgegeben, aber nicht alle Kunden. Warum?

Möchte Sie aber alle Kunden sehen, auch wenn diese keine Bestellungen aufgegeben haben, können Sie keinen INNER

JOIN verwenden; dazu wird ein OUTER JOIN, hier ein RIGHT OUTER JOIN, notwendig sein.

```
1  mysql> SELECT
2      -> k.kunde_id, k.nachname, k.vorname,
b.datum
3      -> FROM
4      -> bestellung b RIGHT OUTER JOIN kunde
k USING(kunde_id)
5      -> ORDER BY
6      -> k.nachname, k.vorname
7      -> ;
8  +-----+-----+-----+-----+
-----+
9  | kunde_id | nachname | vorname | datum |
10 +-----+-----+-----+-----+
-----+
11 | 3 | Beutlin | Bilbo | NULL |
12 | 2 | Beutlin | Frodo | 2012-03-23 16:11:0
0 |
13 | 5 | Earendilionn | Elrond | NULL |
14 | 6 | Eichenschild | Thorin | NULL |
15 | 1 | Gamdschie | Samweis | 2012-03-24 17:
41:00 |
16 | 4 | Telcontar | Elessar | NULL |
17 +-----+-----+-----+-----+
-----+
```

Zuerst fällt auf, dass nicht zwei, sondern sechs Zeilen ausgegeben werden, denn jetzt werden auch die Kunden angezeigt, für die keine Bestellungen vorliegen ([Zeilen 11](#), [13](#), [14](#) und [16](#)). Da SQL nicht weiß, was es in den entsprechenden Spalten an Werten eintragen soll, wird hier NULL verwendet.



Definition 45: OUTER JOIN

Der *OUTER JOIN* zweier Tabellen ist der *INNER JOIN* dieser beiden Tabellen, der um folgende Zeilen erweitert wird: Zeilen der rechten (*RIGHT OUTER JOIN*) oder linken (*LEFT OUTER JOIN*) Tabelle, für welche keine passenden Paarung gefunden wurde.

Wird der *INNER JOIN* um Zeilen aus der linken und der rechten Tabelle erweitert, spricht man von einem *FULL OUTER JOIN*.

Keine Sorge, die Sache ist komplizierter zu erklären als zu benutzen. Betrachten wir noch einmal obiges Beispiel. Der *INNER JOIN* liefert uns nur die Zeilen, für welche ein passendes Primär-Fremdschlüsselpaar gefunden wird. Der *RIGHT JOIN* in [Zeile 4](#) erweitert jetzt das Ergebnis des *INNER JOIN*. Um welche Zeilen? Laut [Definition 45](#) um die Zeilen der rechts vom *JOIN* stehenden Tabelle, für die keine passenden Primär-Fremdschlüsselpaare gefunden werden.

Und tatsächlich, es sind dies die Kunden ohne Bestellung; für diese Primärschlüsselwerte kann kein passender Fremdschlüsselwert in *bestellung* gefunden werden.

Und das mit dem *LEFT* und *RIGHT* ist einfach nur banal. Bei *LEFT* wird um die Zeilen der Tabelle, die links vom Wort *JOIN* steht, erweitert. Bei *RIGHT* um die Tabelle, die rechts vom Wort *JOIN* steht.

Vertauschen Sie die beiden Tabellennamen und machen aus *RIGHT* ein *LEFT*, kommt genau das gleiche Ergebnis heraus:

```
1  SELECT
2      k.kunde_id, k.nachname, k.vorname, b.dat
um
3      FROM
4      kunde k LEFT OUTER JOIN bestellung b USI
```

```

NG(kunde_id)
5      ORDER BY
6      k.nachname, k.vorname
7      ;

```

Ein `LEFT OUTER JOIN` oder `RIGHT OUTER JOIN` wird immer dann verwendet, wenn nicht nur die Zeilen einer Tabelle interessant sind, für die es eine Paarung gibt. Nehmen Sie beispielsweise eine Liste von Vertretern und die von den Vertretern abgeschlossenen Verträge. Wollten Sie nun die Anzahl der abgeschlossenen Verträge pro Vertreter wissen, würde ein `INNER JOIN` alle Vertreter unterdrücken, die noch keinen Vertrag abgeschlossen haben. In einer Übersichtsauswertung wäre dies sicherlich fehlerhaft.

Hurra, wir haben eine neue Variante des `SELECT`:



SQL:2023, 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 MariaDB⁸ kennen keinen `FULL OUTER JOIN`. Bei beiden ist das Schlüsselwort `OUTER` optional, und es gilt: Wird

nur JOIN angegeben, wird ein INNER JOIN verwendet.



MySQL/MariaDB

```
SELECT [DISTINCT]
{*|spaltenliste|ausdruck}
FROM
tabname [[RIGHT [OUTER]|LEFT [OUTER]|INNER
] JOIN tabname
{ON tabname.spaltenname = tabname.spaltenname|USING(spaltenname)}]*
[WHERE bedingung]
[ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
[. . .]
;
```

Ein weiteres Beispiel: Wir möchten die Lieferanten⁹ und ihre Artikel wissen.

```
1  mysql> SELECT
2      -> l.lieferant_id, l.firmenname, a.art
   ikel_id, a.bezeichnung
3      -> FROM
4      -> artikel_nm_lieferant nm INNER JOIN
   artikel a USING(artikel_id)
5      -> INNER JOIN lieferant l USING(liefer
   ant_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 | Silb
erwiebel |
17 | 1 | Gartenbedarf AllesGrün | 7863 | Tulp
enzwiebel |
18 | 1 | Gartenbedarf AllesGrün | 9010 | Scha
ufel |
19 | 1 | Gartenbedarf AllesGrün | 9015 | Spat
en |
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.art
ikel_id, a.bezeichnung
3      -> FROM
4      -> artikel_nm_lieferant nm INNER JOIN
artikel a USING(artikel_id)
5      -> RIGHT JOIN lieferant l USING(liefer
ant_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 | Silb
erwiebel |
17  | 1 | Gartenbedarf AllesGrün | 7863 | Tulp
enzwiebel |
18  | 1 | Gartenbedarf AllesGrün | 9010 | Scha
ufel |
19  | 1 | Gartenbedarf AllesGrün | 9015 | Spat
en |
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 keineWare liefern:

```

1  mysql> SELECT l.firmenname
2          -> FROM
3          -> artikel_nm_lieferant nm INNER JOIN
artikel a USING(artikel_id)
4          -> RIGHT JOIN lieferant l USING(liefer
ant_id)

```



```

5      -> WHERE artikel_id IS NULL
6      -> ;
7      +-----+
8      | firmenname |
9      +-----+
10     | Office International |
11     +-----+

```

Durch das IS NULL in [Zeile 5](#) werden alle Zeilen aus der Ergebnismenge entfernt, die eine Artikelnummer haben. Übrig bleiben die, für welche keine Artikelnummer gefunden wird.

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

```

1  mysql> SELECT k.kunde_id, k.nachname, k.vo
rname
2      -> FROM
3      -> bestellung b RIGHT JOIN kunde k USI
NG(kunde_id)
4      -> WHERE b.bestellung_id IS NULL;
5      +-----+-----+-----+
6      | kunde_id | nachname | vorname |
7      +-----+-----+-----+
8      | 3 | Beutlin | Bilbo |
9      | 5 | Earendil | Elrond |
10     | 6 | Eichensch | Thorin |
11     | 4 | Telcontar | Elessar |
12     +-----+-----+-----+

```



Aufgabe 11.18: Welche Kunden haben keine eigene Lieferadresse?

>**Aufgabe 11.19:** Welcher Artikel ist noch nie bestellt worden?



Hinweis: Ist der OUTER JOIN Teil einer Kette, müssen Sie darauf achten, dass in der Kette das Ergebnis des OUTER JOINS nicht wieder durch einen INNER JOIN verloren geht.



Aufgabe 11.20: Warum verschwindet hier die Firma Office International?

```
SELECT DISTINCT l.firmenname
FROM
  artikel_nm_lieferant nm INNER JOIN artike
l a USING(artikel_id)
                                RIGHT JOIN liefer
ant l USING(lieferant_id)
                                INNER JOIN bestel
lung_position USING(artikel_id)
;
```

Aufgabe 11.21: Warum verschwinden hier wieder alle Kunden ohne Bestellung?

```
SELECT DISTINCT k.kunde_id, k.nachname, k.
vorname
FROM
  bestellung b RIGHT JOIN kunde k USING(
kunde_id)
                                INNER JOIN bestellung_pos
ition USING(bestellung_id)
;
```

Aufgabe 11.22: Unter der Annahme, dass Sie keine Constraints verwenden: Wie kann man mit einem OUTER JOIN verletzte referenzielle Integritäten (siehe [Definition 22](#) in [Abschnitt 2.3](#)) 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, bestel-
lung_id, datum
3 oshop=# FROM
4 oshop=# bestellung FULL OUTER JOIN kunde U
5 USING(kunde_id);
6 kunde_id | nachname | vorname | bestellung
_id | datum
7 -----+-----+-----+-----
8          1 | Gamdschie | Samweis | 1 | 2012-0
3-24 17:41:00
9          2 | Beutlin | Frodo | 2 | 2012-03-23
16:11:00
10         5 | Earendilionn | Elrond | |
11         6 | Eichenschild | Thorin | |
12         4 | Telcontar | Elessar | |
13         3 | Beutlin | Bilbo | |
```

Mangels fachlich sinnvollem Beispiel ist dies hier das gleiche Ergebnis wie bei einem RIGHT OUTER JOIN.

11.5 Narzissmus pur: Self Join

Wir wollen den Kunden die Möglichkeit bieten, in einem Forum Beiträge zu formulieren. Mit unseren bisherigen Tabellen ist das nicht möglich. Ein kurzes Brainstorming zeigt uns, dass wir

eine Tabelle mit Anmeldedaten und eine Tabelle mit Beiträgen brauchen (siehe ER-Modell in [Bild 11.3](#)).

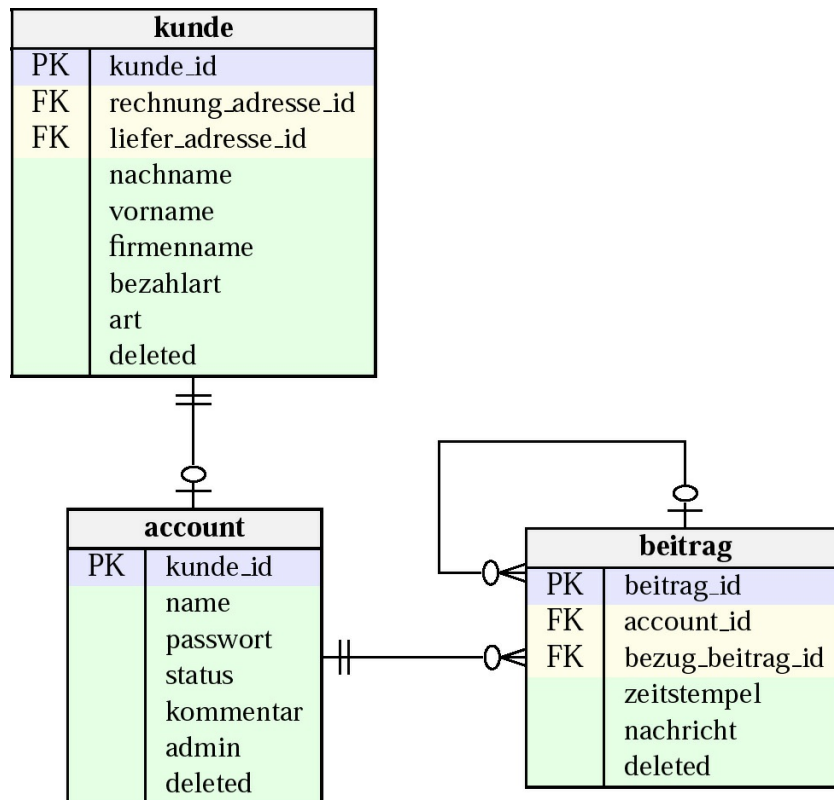


Bild 11.3 ER-Modell: Kundenforum

Zunächst die einfachen Dinge: Jeder Kunde kann einen Account haben, muss er aber nicht. Umgekehrt muss aber ein Account auf einen Kunden zeigen. Wir haben also eine 1:1-Verknüpfung.

Jeder Account kann mehrere Beiträge erfassen, muss er aber nicht. Ein Beitrag hingegen muss auf genau einen Account verweisen: eine 1:n-Verknüpfung.

In Foren ist es üblich, dass man auf Beiträge antworten kann. Dazu muss ein Beitrag wissen, auf welchen anderen Beitrag er sich bezieht. Deshalb gibt es in der Tabelle `beitrag` den Fremdschlüssel `bezug_beitrag_id`. In diesen Fremdschlüssel

trage ich den Wert von `beitrag_id` ein, den die ursprüngliche Nachricht hat.

Aus Sicht der Tabelle ist der Fremdschlüssel `bezug_beitrag_id` ein Fremdschlüssel, der auf eigene Datensätze verweist. Eine solche Verknüpfung nennt man *Self Join*.



Definition 46: Self Join

Enthält eine Tabelle einen Fremdschlüssel mit Primärschlüsselwerten der eigenen Tabelle, so nennt man diese Art der Verknüpfung *Self Join*.

Bitte beachten Sie, dass ein Self Join auch ein `INNER JOIN`, `OUTER JOIN` oder `CROSS JOIN` sein kann!

In [Bild 11.3](#) können Sie sehen, wie man einen Self Join sofort erkennt. Er verweist eben auf sich selbst.



Aufgabe 11.23: Erstellen Sie zu den beiden Tabellen `account` und `beitrag` passende `CREATE TABLE`- und ggf. `CREATE INDEX`-Befehle. Füllen Sie die beiden Tabellen mit passenden Inhalten. Beachten Sie dabei folgende Hinweise:

- `account`: Der Primärschlüssel `kunde_id` hat keinen eigenen Zähler.
- `account`: Der Inhalt der Spalte `name` muss ein Schlüssel sein.
- `account`: Der Status kann zwei Werte haben: `aktiv` und `gesperrt`.

- account: Der Kommentar muss einen langen Text aufnehmen können.
- account: Die Spalte admin ist vom Typ BOOL.
- beitrags: Die Spalte account_id ist der Fremdschlüssel auf die Spalte kunde_id in der Tabelle account.
- beitrags: 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.
- beitrags: Der Nachrichtentext muss Platz für längere Texte haben.
- beitrags: Auf eine Thread-Verwaltung wird hier verzichtet.

Nun stehen uns Testdaten¹⁰ zur Verfügung, deren Nachrichtentexte ich aus Platzmangel auf 20 Stellen in der Ausgabe begrenze:

```

1  mysql> SELECT
2      -> kunde_id, name, status, admin
3      -> FROM account;
4  +-----+-----+-----+-----+
5  | kunde_id | name  | status | admin |
6  +-----+-----+-----+-----+
7  | 1 | admin | aktiv | 1 |
8  | 2 | frodo | aktiv | 0 |
9  | 3 | bilbo | aktiv | 0 |
10 | 5 | elle  | aktiv | 0 |
11 +-----+-----+-----+-----+
12

```

```

13  mysql> SELECT
14      -> beitrags_id, account_id, bezug_beitrags_id, LEFT(nachricht, 20)
15      -> FROM beitrags;
16  +-----+-----+-----+
17  | beitrags_id | account_id | bezug_beitrags_id | LEFT(nachricht, 20) |
18  +-----+-----+-----+
19  | 1 | 1 | 1 | |
20  | 2 | 2 | 1 | Der Lieferservice is |
21  | 3 | 3 | 2 | Das finde ich auch. |
22  | 4 | 5 | 2 | Aber ein wenig langs |
23  | 5 | 2 | 4 | Finde ich nicht. |
24  | 6 | 5 | 1 | Angebot könnte besse |
25  +-----+-----+-----+

```

Jetzt kommt der Self Join: Wir wollen die Antworten auf Nachricht 2 wissen:

```

1  mysql> SELECT nachricht
2      -> FROM
3      -> beitrags INNER JOIN beitrags ON bezug_
4      -> WHERE
5      -> beitrags_id = 2;
6  ERROR 1066 (42000): Not unique table/alias
: 'beitrags'

```

Die Fehlermeldung in [Zeile 6](#) 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 3](#) deutlich. Wenn er die beiden Spalteninhalte vergleichen soll, ist unklar, ob mit beitrags_id

jetzt die Spalte der linken oder rechten Tabelle `beitrag` gemeint ist.

Spätestens jetzt ist die Vergabe eines Alias kein *nice to have* mehr, sondern ein *must be*. Indem man der Tabelle jeweils einen eindeutigen Alias – links `ant` für `beitrag` in der Rolle einer Antwort und rechts `orig` für `beitrag` in der Rolle der Originalnachricht – gibt und diesen bei der Angabe der Spalten auch verwendet, sind alle Unklarheiten beseitigt.

```
1  mysql> SELECT ant.nachricht 'Antwort'
2          -> FROM
3          -> beitrags ant INNER JOIN beitrags orig
4          -> ON ant.bezugsbeitrag_id = orig.beitrag_id
rag_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.

Exkurs: Common Table Expression (CTE)

Leider kann unsere bisherige Auswertung der Forenbeiträge immer nur direkte Antworten auf einen Beitrag ausgeben; Antworten auf Antworten in dynamischer Tiefe gehen so nicht, müsste ich doch für jede Antworttiefe einen neuen `INNER JOIN` anhängen.

Dieses Problem kann mit Hilfe einer Common Table Expression (CTE) gelöst werden.¹¹



Definition 47: Common Table Expression

Eine *Common Table Expression (CTE)* ist eine virtuelle Tabelle, die vor der Befehlsausführung definiert wird, im nachfolgenden Befehl Verwendung findet und danach sofort verworfen wird. Der SELECT-Ausdruck, der die virtuelle Tabelle bestimmt, wird durch das Schlüsselwort **WITH** eingeleitet.

In den ersten vier Zeilen wird eine virtuelle Tabelle mit Hilfe des **SELECT** in den runden Klammern aufgebaut. Wir hatten so etwas Ähnliches schon mal bei temporären Tabellen (siehe [Abschnitt 5.3.8](#)). Unsere virtuelle Tabelle bekommt hier den Namen `ant_auf` als Abkürzung für *Antwort auf*. Anders als bei temporären Tabellen, ist diese nicht für die Dauer der Sitzung verfügbar, sondern nur für einen **SELECT**.

Der zweite **SELECT** in [Zeile 5](#) kann nun auf die Tabelle `ant_auf` hinter dem **FROM** als Datenquelle zugreifen, so als wäre sie eine echte Tabelle.

```
1  mysql> WITH ant_auf AS
2      -> (
3      -> SELECT * FROM beitrags WHERE beitrags_id = 2
4      -> )
5      -> SELECT beitrags_id, beitrags_id, nachricht FROM ant_auf;
6  +-----+-----+-----+-----+
7  | beitrags_id | beitrags_id | nachricht |
8  +-----+-----+-----+-----+
```

```

8  +-----+-----+-----+-----+
-----+
9  | 3 | 2 | Das finde ich auch. |
10 | 4 | 2 | Aber ein wenig langsam. |
11 +-----+-----+-----+-----+
-----+

```

Neue Syntax, aber gleiches Ergebnis? Erstmal ja, aber wir wollten ja mehr, nämlich dynamisch alle Antworten auf einen Beitrag in beliebiger Tiefe. Dazu kann die Variante `WITH RECURSIVE` der CTE verwendet werden.

Der nicht rekursive Teil der virtuellen Tabelle ist ein ganz normaler `SELECT` auf die Antworten zum Beitrag 2. Im rekursiven Teil wird auf der rechten Seite des `INNER JOIN` der Name der virtuellen Tabelle `ant_auf` als Datenquelle angegeben. Nochmal: Innerhalb der Definition der Datenquelle `ant_auf` wird `ant_auf` als Datenquelle verwendet – also rekursiv aufgerufen. Dieser rekursive Aufruf erfolgt so lange eine passende Primär-Fremdschlüsselpaarung in der `ON`-Klausel gefunden wird.

Der `UNION ALL` verbindet beide `SELECT`-Ergebnisse zu einer neuen Tabelle (mehr in [Abschnitt 14.1](#)).

Ihnen fällt sicherlich auf, dass nun auch der Beitrag 5 als indirekte Antwort auf Beitrag 2 ausgegeben wird.

```

1  mysql> WITH RECURSIVE ant_auf AS
2      -> (
3      -> -- nicht rekursiver Teil
4      -> SELECT * FROM beitrags WHERE beitrags.bezug_beitrags_id = 2
5      -> UNION ALL
6      -> -- rekursiver Teil
7      -> SELECT ant.*
8      -> FROM beitrags ant INNER JOIN ant_auf
orig ON ant.bezug_beitrags_id = orig.beitrags_id

```

```

id
9      -> )
10     -> SELECT beitrags_id, bezug_betrags_id
, nachricht FROM ant_auf;
11  +-----+-----+-----+
-----+
12  | beitrags_id | bezug_betrags_id | nachricht |
13  +-----+-----+-----+
-----+
14  | 3 | 2 | Das finde ich auch. |
15  | 4 | 2 | Aber ein wenig langsam. |
16  | 5 | 4 | Finde ich nicht. |
17  +-----+-----+-----+
-----+

```

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 in [Abschnitt 11.2.4](#). Beide verwendeten die Spalte kunde_id, aber diese war in den temporären Tabellen nicht als Primär- oder Fremdschlüssel deklariert. Eine Verknüpfung über diese beiden Spalten kann somit sehr teuer werden.

Grundsätzlich ist aber der Zusammenbau von Daten aus verschiedenen Tabellen teurer als das Auslesen der Daten aus einer Tabelle. Mit temporären Tabellen – wie oben

beschrieben – können Sie übliche Verbindungen vorbauen, damit diese nicht jedes Mal neu erstellt werden müssen. Eine weitere Möglichkeit sind redundante Daten.

Redundante Daten sind nach [Definition 13](#) 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 ,
6   ADD l_strasse VARCHAR(255),
7   ADD l_ort VARCHAR(255),
8   ADD l_aktuell BOOL NOT NULL DEFAULT TRUE
9 ;
```

Straße und Ort sollen Zusammenbauten der Spalten strasse mit hnr und lkz mit plz mit ort sein.¹² In periodischen Abständen werden die Daten aus der Adresstabelle in die Kundentabelle kopiert.

```
1 mysql> UPDATE kunde INNER JOIN adresse ON
2   rechnung_adresse_id = adresse_id
3   -> SET
4   -> r_strasse = CONCAT(strasse, ' ', hn
5   r),
6   -> r_ort = CONCAT(lkz, '-', plz, ' ',
7   ort),
8   -> r_aktuell = TRUE;
```

```

6
7  mysql> SELECT
8      -> kunde_id, r_strasse, r_ort, r_aktue
ll
9      -> FROM kunde;
10
11  +-----+-----+-----+-----+
-----+-----+
12  | kunde_id | r_strasse | r_ort | r_aktuell
   |
13  +-----+-----+-----+-----+
-----+-----+
14  | 1 | Beutelhaldenweg 5 | AL-67676 Hobbing
en | 1 |
15  | 2 | Beutelhaldenweg 1 | AL-67676 Hobbing
en | 1 |
16  | 3 | Beutelhaldenweg 1 | AL-67676 Hobbing
en | 1 |
17  | 4 | Auf der Feste 1 | GO-54786 Minas Tir
ith | 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 11.24: Überlegen Sie mal, lässt sich beim `UPDATE` eine geschickte `WHERE`-Klausel einbauen?

Jetzt wird jedes Mal, wenn sich eine Adresse innerhalb der Periode ändert, die Spalte `r_aktuell` oder `l_aktuell` auf `FALSE` gesetzt. Dies könnten Sie beispielsweise mit einem Trigger¹³ erreichen. Mithilfe eines Events¹⁴ wird jetzt in periodischen Abständen nachgeschaut, ob sich die Adressdaten geändert haben. Falls ja, werden die redundanten Daten neu aufgebaut. Falls Sie die Änderung sofort in den redundanten Daten ändern wollen, können Sie dies ebenfalls im Trigger machen.



Aufgabe 11.25: Erweitern Sie das `UPDATE` so, dass gleichzeitig auch die Lieferadresse gesetzt wird. Vorsicht beim zweiten `JOIN` und der `WHERE`-Klausel!

¹ So, wie wir es in den ersten Kapiteln getan haben ;-).

² Dies ist gerade der entscheidende Unterschied zu OO-Datenbanken oder NoSQL-Ansätzen.

³ Wird von den Azubis gerne *Orgienjoin* genannt.

⁴ Siehe [Kapitel 13](#).

⁵ Der Query Cache ist seit der MySQL Version 8.0 abgeschaltet (siehe [\[MyS18c\]](#)).

⁶ die Knoten ganz links

⁷ Immerhin sind es Fremdschlüssel.

⁸ Über ein diesbezüglich sehr verwirrendes Verhalten von MySQL und MariaDB habe ich einen Bugreport ([\[Ada23\]](#)) für MariaDB angelegt. Für MySQL habe ich darauf verzichtet, da Oracle bei der Registrierung zu viele Angaben von mir wissen wollten.

⁹ In der Datei `listing08.sql` werden diese angelegt.

¹⁰ Die entsprechenden Befehle stehen in `listing08.sql`.

¹¹ Eine tiefere Betrachtung von CTE muss hier leider aus Platzgründen entfallen. Eine gute Einführung finden Sie unter [[MyS19b](#)].

¹² Eine von mir willkürlich gefällte Designentscheidung, um eine bessere Übersicht zu haben.

¹³ Siehe [Kapitel 22](#).

¹⁴ Siehe [Kapitel 23](#).

12 Differenzierte Auswertungen



Auswertungen mithilfe von Aggregatfunktionen erstellen.

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



Die Quelltexte dieses Kapitels stehen in der Datei `listing09.sql` (siehe [Listing 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 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  | 12.5777777778 |  
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)
2  ) FROM kunde;
3  +-----+
4  | COUNT(DISTINCT(rechnung_adresse_id)) |
5  +-----+
6  | 4 |
7  +-----+

```

Was ist unser teuerster und unser billigster Einzelpreis?

Die Tabelle artikel enthält in der Spalte einzelpreis unsere Preise.

```

1  SELECT MAX(einzelpreis), MIN(einzelpreis)
2  FROM artikel;
3  +-----+-----+
4  | MAX(einzelpreis) | MIN(einzelpreis) |
5  +-----+-----+
6  | 56.260000 | 0.520000 |
7  +-----+-----+

```

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 a
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 'Positions
wert'
3      -> FROM
4      -> bestellung_position bp INNER JOIN a
rtikel 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 artike
l 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.

- a) Ergänzen Sie das ER-Modell in [Bild 3.2](#) um die Lagerverwaltung in `listing09.sql`.
- b) Ermitteln Sie die durchschnittliche Bestellmenge.
- c) Ermitteln Sie die Anzahl der Artikel mit der Währung USD.
- d) Ermitteln Sie die Anzahl der Privatkunden.
- e) Ermitteln Sie die Anzahl der Kunden, die noch keine Bestellung aufgegeben haben.
- f) Was ist unsere kleinste und was unsere größte Bestellmenge?
- g) Ermitteln Sie, wie viele *ml* Tinte noch auf Lager sind.
- h) 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] [,spalten
name [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      -> ;
```

	Bestellnummer	Anzahl der Positionen
1	1	3
2	2	2

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 a
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
3      -> .menge) 'Anzahl bestellter Artikel'
4      -> FROM
5      -> bestellung_position bp INNER JOIN a
6      -> rtikel a USING(artikel_id)
7      -> GROUP BY
8      -> artikel_id
9      -> ;
10
11 +-----+-----+
12 | Artikelname | Anzahl bestellter Artikel |
13 |-----+-----+
14 | Silberzwiebel | 40.000000 |
15 | Tulpenzwiebel | 50.000000 |
16 | Schaufel | 5.000000 |
17 | Spaten | 1.000000 |
18 +-----+-----+
```

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
3      -> .menge) 'Anzahl bestellter Artikel'
4      -> FROM
5      -> bestellung_position bp RIGHT JOIN a
```

```

rtikel 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 a
rtikel a USING(artikel_id)
9      -> GROUP BY
10     -> artikel_id
11     -> ;
12  +-----+-----+-----+-----+

```

```

--+
13  | Artikelname | Anzahl bestellter Artikel
   |
14  +-----+-----+-----+-----+-----+
--+
15  | Papier (100) | 0 |
16  | Tinte (gold) | 0 |
17  | Tinte (rot)  | 0 |
18  | Tinte (blau) | 0 |
19  | Feder        | 0 |
20  | Silberzwiebel | 40.000000 |
21  | Tulpenzwiebel | 50.000000 |
22  | Schaufel      | 5.000000  |
23  | Spaten        | 1.000000  |
24  +-----+-----+-----+-----+
--+

```

Mit dem CASE 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 T
HEN 0
5  oshop=# ELSE SUM(bp.menge)
6  oshop=# END AS Anzahl_bestellter_Art
ikel
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 a
rtikel a USING(artikel_id)
9      -> GROUP BY
10     -> artikel_id WITH ROLLUP
11     -> ;

```

artikel_id	Anzahl bestellter Artikel
3001	0
3005	0
3006	0
3007	0
3010	0
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] [,spalten
name [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 a
rtikel 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 f
unction

```

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



```

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

12.4 Noch Fragen?

12.4.1 Kann ich nach Ausdrücken gruppieren?

In der syntaktischen Beschreibung von GROUP BY 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'
t', COUNT(*) 'Anzahl'
3      -> FROM
4      -> bank
5      -> GROUP BY
6      -> 'Clearinggebiet'
7      -> ORDER BY 'Anzahl' DESC
8      -> ;
9  +-----+-----+
```

10		Clearinggebiet		Anzahl	
11	+-----+				
12		7		1249	
13		5		1193	
14		6		1100	
15		2		989	
16		4		556	
17		3		527	
18		8		287	
19		1		205	
20	+-----+				

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) 'M
onat', 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) 'Clearinggebie
t', COUNT(*) 'Anzahl'
4      -> FROM
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.

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

13 Auswertungen mit Unterabfragen



Und wenn Du denkst, es geht nicht mehr, dann kommt ein SUBSELECT daher.

- Vertiefendes
 - Warum eine Unterabfrage?
 - Korrelierende und nicht korrelierende Unterabfragen
 - Skalar-, Listen- und Tabellenunterabfragen
 - IN(), ALL() und ANY()
 - EXISTS
 - Unterschied zu CTE



Die Quelltexte dieses Kapitels stehen in der Datei `listing10.sql` (siehe [Listing 28.10](#), [Listing 28.50](#) und [Listing 28.35](#)).

Sie benötigen ebenfalls die Datei `kunden01.csv`.

Unterabfragen gehören definitiv nicht mehr zu einem Grundkurs. Sie sind das Tor zu richtig schweren Auswertungen, da es jetzt möglich ist, in *einer* Abfrage *beliebig viele* Zwischenabfragen durchzuführen und weiterzuverwerten.

13.1 Das Problem und die Lösung

Betrachten wir folgende Aufgabenstellung: Wir wollen den Umsatz pro Kunde wissen.

Schritt 1: Die Umsätze müssen sich aus den Bestellungen ergeben. Also ermitteln wir zuerst den Umsatz pro Bestellung. Dazu wird ein `INNER JOIN` zu der Tabelle `artikel` gebraucht, da dort der Preis abgelegt ist.

```
1  mysql> SELECT
2      -> bp.bestellung_id, SUM(bp.menge * a.
3      -> FROM
4      -> bestellung_position bp INNER JOIN a
5      -> GROUP BY
6      -> bp.bestellung_id
7      -> ;
8  +-----+-----+
9  | bestellung_id | Bestellwert |
10 +-----+-----+
11 | 5 | 488.550000000000 |
12 | 4 | 26.460000000000 |
13 | 1 | 206.700000000000 |
14 | 2 | 80.700000000000 |
15 | 3 | 39.400000000000 |
16 +-----+-----+
```

Schritt 2: Den Bestellungen den Kunden zuordnen. Dazu

Schritt 2: Den Bestellungen den Kunden zuordnen. Dazu müssen Verknüpfungen zwischen den Tabellen bestellung und kunde geknüpft werden:

```
1  mysql> SELECT
2      -> k.nachname, k.vorname, SUM(bp.menge
3      * a.einzelpreis) Bestellwert
4      -> FROM
5      -> bestellung_position bp INNER JOIN a
rtikel a USING(artikel_id)
6      -> INNER JOIN bestellung b USING(beste
llung_id)
7      -> GROUP BY
8      -> bp.bestellung_id
9      -> ;
```

nachname	vorname	Bestellwert
Gamdschie	Samweis	488.55000000000000
Gamdschie	Samweis	26.46000000000000
Gamdschie	Samweis	206.70000000000000
Beutlin	Frodo	80.70000000000000
Gamdschie	Samweis	39.40000000000000

Schritt 3: Wir müssen wiederum eine Gruppierung vornehmen, und zwar pro Kunde. Und für jeden Kunden die Aggregatfunktion SUM(Bestellwert) ausführen. Aber wie soll das hier funktionieren?

Eine Möglichkeit wären die schon weiter oben in [Abschnitt 5.3.8](#) vorgestellten temporären Tabellen.

```
1  mysql> CREATE TEMPORARY TABLE tmp_bestellw
ert
2      -> SELECT
3      -> k.nachname, k.vorname, SUM(bp.menge
```

```

* a.einzelpreis) bestellwert
4      -> FROM
5      -> bestellung_position bp INNER JOIN a
rtikel a USING(artikel_id)
6      -> INNER JOIN bestellung b USING(beste
llung_id)
7      -> INNER JOIN kunde k USING(kunde_id)
8      -> GROUP BY
9      -> bp.bestellung_id;
10
11 mysql> SELECT * FROM tmp_bestellwert;
12 +-----+-----+-----+
13 | nachname | vorname | bestellwert |
14 +-----+-----+-----+
15 | Gamdschie | Samweis | 488.550000000000 |
16 | Gamdschie | Samweis | 26.460000000000 |
17 | Gamdschie | Samweis | 206.700000000000 |
18 | Beutlin | Frodo | 80.700000000000 |
19 | Gamdschie | Samweis | 39.400000000000 |
20 +-----+-----+-----+
21
22 mysql> SELECT
23      -> nachname, vorname, SUM(bestellwert)
Umsatz
24      -> FROM tmp_bestellwert
25      -> GROUP BY
26      -> nachname, vorname;
27 +-----+-----+-----+
28 | nachname | vorname | Umsatz |
29 +-----+-----+-----+
30 | Gamdschie | Samweis | 761.110000000000 |
31 | Beutlin | Frodo | 80.700000000000 |
32 +-----+-----+-----+

```

Natürlich hat diese Lösung einen gewissen Charme. Besonders der Aspekt, dass man die Bestellwertdaten ggf. mehrfach auswerten kann. Aber es geht auch anders. Man kann die Abfrage von [Zeile 2](#) bis [9](#) in [Zeile 24](#) als Unterabfrage einbauen.

```

1  mysql> SELECT
2      -> bw.nachname, bw.vorname, SUM(bw.bes
tellwert) Umsatz
3      -> FROM
4      -> (
5      -> SELECT
6      -> k.nachname, k.vorname, SUM(bp.menge
* a.einzelpreis) bestellwert
7      -> FROM
8      -> bestellung_position bp INNER JOIN a
rtikel a USING(artikel_id)
9      -> INNER JOIN bestellung b USING(beste
llung_id)
10     -> INNER JOIN kunde k USING(kunde_id)
11     -> GROUP BY
12     -> bp.bestellung_id
13     -> ) AS bw
14     -> GROUP BY nachname, vorname;
15  +-----+-----+-----+
16  | nachname | vorname | Umsatz |
17  +-----+-----+-----+
18  | Gamdschie | Samweis | 761.1100000000000 |
19  | Beutlin | Frodo | 80.7000000000000 |
20  +-----+-----+-----+

```

Die [Zeilen 4](#) bis [13](#) stellen eine Unterabfrage dar. In der Regel werden diese durch Klammerung besonders kenntlich gemacht; meist ist die Klammerung sogar aus syntaktischen Gründen zwingend. In [Zeile 13](#) haben wir dem Ergebnis noch einen Namen verpasst: bw. Dies erlaubt es uns, gezielt auf die Spalten der *neuen* Tabelle zuzugreifen, und verhindert Fehlermeldungen über mehrdeutige Spalten- oder Tabellennamen.



Definition 49: Unterabfrage

Kommt innerhalb eines SELECT, UPDATE, INSERT oder DELETE ein weiterer SELECT oder JOIN vor, so nennt man diesen eine *Unterabfrage* oder auch *SUBSELECT*.

Unterabfragen können wiederum Unterabfragen enthalten.

Die Anweisung, die die Unterabfrage enthält, wird *Hauptanweisung* oder *Oberanweisung* genannt.

Warum nennt man die *Oberanweisung* nicht *Oberabfrage*? Weil es auch ein INSERT, UPDATE oder DELETE sein kann, der sich einer Unterabfrage bedient. Ein Beispiel für eine solche Unterabfrage haben wir schon in [Abschnitt 11.6](#) kennengelernt. Hier wurde eine Unterabfrage innerhalb eines UPDATES verwendet.

Wir wollen uns nun mit verschiedenen Typen von Unterabfragen beschäftigen. Diese unterscheiden sich hinsichtlich der Komplexität und der Verflechtung mit der Oberanweisung.

13.2 Nicht korrelierende Unterabfrage



Definition 50: Korrelierende Unterabfrage

Kann eine Unterabfrage für sich alleine ausgeführt werden, so handelt es sich um eine *nicht korrelierende Unterabfrage*. Benötigt die Unterabfrage Ausdrücke oder Spalten der Oberanweisung, so handelt es sich um eine *korrelierende Unterabfrage*.

13.2.1 Skalarunterabfrage

13.2.1.1 Beispiel 1: Banken mit höchster BLZ

Fangen wir mit einem einfachen Beispiel an. Aus dem Import der Bankdaten wissen wir, dass es zu jeder Bankleitzahl mehrere Banknamen geben kann. Wir wollen die Banknamen der Bank wissen, die die höchste Bankleitzahl hat.

Schritt 1 – Unterabfrage ermitteln: Die Unterabfrage soll mir die höchste Bankleitzahl liefern.

```
1  mysql> SELECT MAX(blz) FROM bank;
2  +-----+
3  | MAX(blz) |
4  +-----+
5  | 87096214 |
6  +-----+
```

Schritt 2 – Bau der Oberanweisung ohne Unterabfrage: Die Oberanweisung will mir abhängig von einer WHERE-Klausel Banknamen ausgeben.

```
1  mysql> SELECT bankname FROM bank WHERE blz
= '37010050';
2  +-----+
3  | bankname |
4  +-----+
5  | Postbank |
6  +-----+
```

Schritt 3 – Einbau der Unterabfrage in die Oberanweisung: Jetzt wird im Vergleich die Konstante in [Zeile 1](#) geklammert und durch die Unterabfrage ersetzt.

```

1  mysql> SELECT bankname
2      -> FROM bank
3      -> WHERE blz =
4      -> (
5      -> SELECT MAX(blz)
6      -> FROM bank
7      -> )
8      -> ;
9  +-----+
10 | bankname |
11 +-----+
12 | Volksbank Chemnitz |
13 | Volksbank Chemnitz (Gf P2) |
14 +-----+

```



Aufgabe 13.1: Führen Sie mal einen EXPLAIN aus und interpretieren Sie die Ausgabe.

13.2.1.2 Beispiel 2: Überdurchschnittlich teure Artikel

Wir wollen alle Artikel ermitteln, die einen überdurchschnittlichen Einzelpreis haben.

Schritt 1 – Unterabfrage ermitteln: Wir brauchen den durchschnittlichen Einzelpreis, um damit vergleichen zu können. Diesen können wir einfach ermitteln:

```

1  mysql> SELECT
2      -> AVG(einzelpreis) AS durchschnittspr
eis
3      -> FROM artikel;
4  +-----+

```



```

5  | durchschnittspreis |
6  +-----+
7  | 12.57777777778 |
8  +-----+

```

Unsere Unterabfrage liefert eine sehr einfache Tabelle zurück, nämlich eine mit einer Zeile und einer Spalte.



Definition 51: Skalarunterabfrage

Liefert die Unterabfrage eine Tabelle mit nur einer Zeile und nur einer Spalte – also einen einzigen Wert –, so nennt man diese *Skalarunterabfrage*.

Schritt 2 – Bau der Oberanweisung ohne Unterabfrage: Wie muss denn die Oberanweisung aussehen? Letztlich ein einfaches SELECT mit einer WHERE-Klausel. Als Platzhalter für die Unterabfrage verwende ich hier eine willkürliche Zahl (siehe [Zeile 4](#)).

```

1  SELECT *
2    FROM artikel
3    WHERE
4    einzelpreis > 30;

```

Schritt 3 – Einbau der Unterabfrage in die Oberanweisung: In der [Zeile 4](#) steht eine 30 als Platzhalter für die Unterabfrage. Zuerst wird um die 30 ein Klammerpaar gesetzt und dann der SELECT aus Schritt 1 dort hineinkopiert. Etwas verschönert sieht das Ergebnis so aus:

```

1  mysql> SELECT *
2          -> FROM artikel
3          -> WHERE
4          -> einzelpreis > (

```

```

5      -> SELECT
6      -> AVG(einzelpreis) AS durchschnittspr
eis
7      -> FROM artikel
8      -> )
9      -> ;
10  +-----+-----+-----+-----+
+-----+-----+
11  | artikel_id | bezeichnung | einzelpreis |
   | waehrung | deleted |
12  +-----+-----+-----+-----+
+-----+-----+
13  | 3005 | Tinte (gold) | 56.260000 | EUR |
   | 0 |
14  | 9010 | Schaufel | 15.100000 | USD | 0 |
15  | 9015 | Spaten | 20.100000 | EUR | 0 |
16  +-----+-----+-----+-----+
+-----+-----+

```

Der Vorteil gegenüber temporären Tabellen liegt auf der Hand. Wird durch ein INSERT, DELETE oder UPDATE der Durchschnittswert der Preise verändert, so muss die temporäre Tabelle komplett neu gebildet werden. Bei Unterabfragen wird erst zum Zeitpunkt der Ausführung die Unterabfrage ausgeführt und enthält damit automatisch¹ den aktuellen Wert.

13.2.1.3 Beispiel 3: Überdurchschnittlich wertvolle Bestellungen

Auch Skalarunterabfragen können kompliziert werden: wenn wir beispielsweise wissen wollen, welche Bestellungen einen überdurchschnittlichen Auftragswert haben.

Schritt 1 – Unterabfrage ermitteln: Zuerst müssen wir den

.....

Auftragswert ermitteln. Das haben wir schon auf [Kapitel 13.1](#) gemacht.

```
1  SELECT
2  bp.bestellung_id, SUM(bp.menge * a.einzelpre
3  reis) bestellwert
4  FROM
5  bestellung_position bp INNER JOIN artikel
6  a USING(artikel_id)
7  GROUP BY
8  bp.bestellung_id
9  ;
```

Davon müssen wir jetzt den Durchschnitt ermitteln:

```
1  mysql> SELECT AVG(bw.bestellwert)
2      -> FROM (
3      -> SELECT
4      -> bp.bestellung_id, SUM(bp.menge * a.
5  einzelpreis) bestellwert
6      -> FROM
7      -> bestellung_position bp INNER JOIN a
8  rtikel a USING(artikel_id)
9      -> GROUP BY
10     -> bp.bestellung_id
11     -> ) AS bw
12     -> ;
13
14  +-----+
15  | AVG(bw.bestellwert) |
16  +-----+
17  | 168.362000000000000 |
18  +-----+
```

Jetzt haben wir eine Abfrage, die uns den durchschnittlichen Bestellwert angibt. Falls Sie den Alias bw in [Zeile 9](#) vergessen haben sollten, bekommen Sie folgende schöne Fehlermeldung:

```
1 ERROR 1248 (42000): Every derived table must have its own alias
```

Warum schön? Weil sie noch mal verdeutlicht, dass das Ergebnis eines SELECT immer eine Tabelle ist.

Schritt 2 – Bau der Oberanweisung ohne Unterabfrage: Ich baue zuerst eine Version mit einem Platzhalter:

```
1 SELECT
2     bp1.bestellung_id, SUM(bp1.menge * a1.ei
nzelpreis) bestellwert1
3     FROM
4     bestellung_position bp1 INNER JOIN artik
el a1 USING(artikel_id)
5     GROUP BY
6     bp1.bestellung_id
7     HAVING
8     bestellwert1 > 100;
```

Da wir das Ergebnis eines GROUP BY einschränken wollen, muss in [Zeile 7](#) das HAVING verwendet werden. Dort steht auch der Platzhalter 100, der gleich umgebaut wird. Die Nummerierung mit a1 usw. habe ich nur für später eingebaut (siehe unten).

Schritt 3 – Einbau der Unterabfrage in die Oberanweisung: Anstelle der Konstante 100 wird jetzt in Klammern die Unterabfrage eingesetzt.

```
1 mysql> SELECT
2     -> bp1.bestellung_id, SUM(bp1.menge *
a1.einzelpreis) bwert1
3     -> FROM
4     -> bestellung_position bp1 INNER JOIN
artikel a1 USING(artikel_id)
5     -> GROUP BY
6     -> bp1.bestellung_id
```

```

7      -> HAVING
8      -> bwert1 >
9      -> (
10     -> SELECT AVG(bw.bwert2)
11     -> FROM
12     -> (
13     -> SELECT
14     -> bp2.bestellung_id, SUM(bp2.menge *
a2.einzelpreis) bwert2
15     -> FROM
16     -> bestellung_position bp2 INNER JOIN
artikel a2
17     -> USING(artikel_id)
18     -> GROUP BY
19     -> bp2.bestellung_id
20     -> ) AS bw
21     -> )
22     -> ;
23 +-----+-----+
24 | bestellung_id | bwert1 |
25 +-----+-----+
26 | 5 | 488.5500000000000 |
27 | 1 | 206.7000000000000 |
28 +-----+-----+

```

13.2.2 Listenunterabfrage

13.2.2.1 Beispiel 1: IN()



Definition 52: Listenunterabfragen

Liefert die Unterabfrage eine Tabelle mit mehreren Zeilen und nur einer Spalte, so bezeichne ich dies als *Listenunterabfrage*.

Es gibt nicht nur Skalarunterabfragen. Sie könnten auch *mehrere* Zeilen in *einer* Spalte zurückliefern. Das Ergebnis kann dann natürlich nicht einfach mit dem Gleichheitsoperator in Bedingungen verwendet werden. Vielmehr brauchen wir Vergleichsoperatoren, die mit Mengen arbeiten. Ich habe dabei die Operatoren IN, ALL und ANY im Blick.²

Wir wollen alle Rechnungen³, für die es auch eine Bestellung gibt.



Aufgabe 13.2: Erweitern Sie das ER-Model in [Bild 3.2](#) um das Rechnungswesen.

Schritt 1 – Unterabfrage ermitteln: Die Unterabfrage muss mir die Primärschlüsselwerte der Bestellungen ermitteln:

```
1  mysql> SELECT bestellung_id FROM bestellun
g;
2  +-----+
3  | bestellung_id |
4  +-----+
5  | 1 |
6  | 3 |
7  | 4 |
8  | 5 |
9  | 2 |
10 | 6 |
11 +-----+
```

Schritt 2 – Bau der Oberanweisung ohne Unterabfrage: Die Oberanweisung soll die Primärschlüsselwerte von rechnung ausgeben, die in einer IN-Liste enthalten sind.

```
1  SELECT rechnung_id
```

```

2  FROM rechnung
3  WHERE bestellung_id IN (1, 2, 3)
4  ;

```

Schritt 3 – Einbau der Unterabfrage in die Oberanweisung:

Die Liste im IN() ist oben eine Liste von Konstanten. Diese soll nun durch die Unterabfrage ersetzt werden:

```

1  mysql> SELECT rechnung_id
2      -> FROM rechnung
3      -> WHERE bestellung_id IN
4      -> (
5      -> SELECT bestellung_id FROM bestellun
g
6      -> )
7      -> ;
8  +-----+
9  | rechnung_id |
10 +-----+
11 | 1 |
12 | 2 |
13 | 3 |
14 | 4 |
15 | 5 |
16 | 6 |
17 +-----+

```



Aufgabe 13.3: Bauen Sie das Beispiel so um, dass Sie gerade die Rechnungen ermitteln, zu denen es keine Bestellungen gibt.

13.2.2.2 Beispiel 2: ALL()

Wir wollen wissen, ob der Gartenbedarf unsere teuersten Artikel enthält. Anders: Welche Artikel sind teurer als die des Gartenbedarfs? Das `IN()` hilft uns hier nicht weiter. Was wir brauchen, ist ein Prädikat, welches uns mitteilt, ob ein Wert kleiner *oder* gleich *oder* größer als alle Elemente einer Liste ist. Das macht der `ALL()`.

Dem `ALL()` geht immer ein Vergleichsoperator (siehe [Abschnitt 26.3.1](#)) voran: `menge <= ALL([...])`; dieser kann mit `NOT` verbunden werden.

Schritt 1 – Unterabfrage ermitteln: Die Unterabfrage ermittelt uns alle Preise der Gartenbedarfsartikel:

```
1  mysql> SELECT a.einzelpreis
2      -> FROM
3      -> artikel_nm_warengruppe INNER JOIN a
artikel a USING(artikel_id)
4      -> INNER JOIN warengruppe w USING(ware
ngruppe_id)
5      -> WHERE
6      -> w.bezeichnung = 'Gartenbedarf';
7  +-----+
8  | einzelpreis |
9  +-----+
10 | 0.520000 |
11 | 3.420000 |
12 | 15.100000 |
13 | 20.100000 |
14 +-----+
```

Schritt 2 – Bau der Oberanweisung ohne Unterabfrage: Die Oberanweisung muss alle Artikelpreise ermitteln:

```
1  SELECT
2  a.bezeichnung, a.einzelpreis
3  FROM
4  artikel a
```



```

5      WHERE
6      a.einzelpreis > ALL(SELECT 10)
7      ;

```

Leider kann ich in [Zeile 6](#) nicht so etwas schreiben wie (100, 200, 300), da hier explizit eine Unterabfrage erwartet wird.

Schritt 3 – Einbau der Unterabfrage in die Oberanweisung:

Die Unterabfrage wird jetzt zwischen das Klammerpaar des ALL() eingefügt.

```

1  mysql> SELECT
2      -> a.bezeichnung, a.einzelpreis
3      -> FROM
4      -> artikel a
5      -> WHERE
6      -> a.einzelpreis > ALL
7      -> (
8      -> SELECT a.einzelpreis
9      -> FROM
10     -> artikel_nm_warengruppe INNER JOIN a
11     rtikel a USING(artikel_id)
12     -> INNER JOIN warengruppe w USING(ware
13     ngruppe_id)
14     -> WHERE
15     -> w.bezeichnung = 'Gartenbedarf'
16     -> )
17     -> ;
18
19  +-----+-----+
20  | bezeichnung | einzelpreis |
21  +-----+-----+
22  | Tinte (gold) | 56.260000 |
23  +-----+-----+

```

13.2.2.3 Beispiel 3: ALL()

Die Beutlins haben einige Rechnungen. Es wäre doch interessant zu wissen, ob es irgendeinen Kunden gibt, der mehr Umsatz gemacht hat als die Beutlins.

Schritt 1 – Unterabfrage ermitteln: Die Unterabfrage muss uns die Summe aller Rechnungssummen der Beutlins liefern. Ermitteln wir zuerst die Rechnungssummen:

```
1  mysql> SELECT
2      -> rechnung_id, kunde_id, SUM(einzelpr
3      -> eis*menge) AS umsatz
4      -> FROM
5      -> rechnung_position INNER JOIN artike
6      -> l USING(artikel_id)
7      -> INNER JOIN rechnung USING(rechnung_
8      -> id)
9      -> GROUP BY
10     -> rechnung_id
11     -> ;
```

	rechnung_id	kunde_id	umsatz
	5	1	488.550000000000
	7	3	67.860000000000
	10	5	87.350000000000
	11	5	1125.200000000000
	4	1	26.460000000000
	8	3	47.990000000000
	1	1	206.700000000000
	2	2	80.700000000000
	3	1	39.400000000000

Die Aufnahme der kunde_id ([Zeile 2](#)) wäre hier noch nicht nötig. Da ich die Kundennummer aber später noch brauche,

muss ich zur Rechnung auch die Kundennummer ermitteln.
Dazu erfolgt ein INNER JOIN auf die Tabelle rechnung ([Zeile 5](#)).

Als Nächstes müssen diese Rechnungssummen pro Kunde zu einer Umsatzsumme addiert werden.

```
1  mysql> SELECT SUM(umsatz) AS umsatzsumme
2      -> FROM
3      -> (
4      -> SELECT
5      -> rechnung_id, kunde_id, SUM(einzelpr
eis*menge) AS umsatz
6      -> FROM
7      -> rechnung_position INNER JOIN artike
l USING(artikel_id)
8      -> INNER JOIN rechnung USING(rechnung_
id)
9      -> GROUP BY
10     -> rechnung_id
11     -> ) AS ksum
12     -> GROUP BY
13     -> kunde_id
14     -> ;
15  +-----+
16  | umsatzsumme |
17  +-----+
18  | 761.110000000000 |
19  | 115.850000000000 |
20  | 1212.550000000000 |
21  | 80.700000000000 |
22  +-----+
```

Als Letztes erfolgt die Einschränkung auf die Beutlins ([Zeile 10](#)).

```
1  mysql> SELECT SUM(umsatz) AS umsatzsumme
2      -> FROM
```

```

3      -> (
4      -> SELECT
5      -> rechnung_id, kunde_id, SUM(einzelpr
eis*menge) AS umsatz
6      -> FROM
7      -> rechnung_position INNER JOIN artike
l USING(artikel_id)
8      -> INNER JOIN rechnung USING(rechnung_
id)
9      -> INNER JOIN kunde USING(kunde_id)
10     -> WHERE nachname = 'beutlin'
11     -> GROUP BY
12     -> rechnung_id
13     -> ) AS ksum
14     -> GROUP BY
15     -> kunde_id
16     -> ;
17 +-----+
18 | umsatzsumme |
19 +-----+
20 | 115.850000000000 |
21 | 80.700000000000 |
22 +-----+

```

Für eine Unterabfrage nicht schlecht, was?

Schritt 2 – Bau der Oberanweisung ohne Unterabfrage: Die Oberanweisung muss mir eine Liste von Kundennummern zurückliefern, deren Umsatzsummen alle größer sind als der Inhalt von der ALL()-Liste. Die Oberanweisung sieht in weiten Teilen wie die Unterabfrage aus:

```

1  SELECT kunde_id, SUM(umsatz) AS umsatzsumm
e
2  FROM
3  (
4  SELECT
5  rechnung_id, kunde_id, SUM(einzelpreis

```

```

*menge) AS umsatz
6      FROM
7      rechnung_position INNER JOIN artikel U
SING(artikel_id)
8      INNER JOIN rechnung USING(rechnung_id)
9      GROUP BY
10     rechnung_id
11   ) AS ksum
12   GROUP BY
13   kunde_id
14   HAVING umsatzsumme > ALL (SELECT 100)
15   ;

```

Schritt 3 – Einbau der Unterabfrage in die Oberanweisung:

Um die Sache noch schöner zu machen, habe ich den Kundennamen noch *drangejoint*.

```

1  mysql> SELECT
2      -> kunde_id, kunde.nachname, kunde.vor
name, SUM(umsatz) AS umsatzsumme
3      -> FROM
4      -> (
5      -> SELECT
6      -> rechnung_id, kunde_id, SUM(einzelpr
eis*menge) AS umsatz
7      -> FROM
8      -> rechnung_position INNER JOIN artike
l USING(artikel_id)
9      -> INNER JOIN rechnung USING(rechnung_
id)
10     -> GROUP BY rechnung_id
11     -> ) AS ksum
12     -> INNER JOIN kunde USING(kunde_id)
13     -> GROUP BY kunde_id
14     -> HAVING umsatzsumme > ALL
15     -> (
16     -> SELECT SUM(umsatz) AS umsatzsumme
17     -> FROM

```

```

18      -> (
19      -> SELECT
20      -> rechnung_id, kunde_id, SUM(einzelpr
eis*menge) AS umsatz
21      -> FROM
22      -> rechnung_position INNER JOIN artike
l USING(artikel_id)
23      -> INNER JOIN rechnung USING(rechnung_
id)
24      -> INNER JOIN kunde USING(kunde_id)
25      -> WHERE nachname = 'beutlin'
26      -> GROUP BY rechnung_id
27      -> ) AS ksum
28      -> GROUP BY kunde_id
29      -> )
30      -> ;
31
32  +-----+-----+-----+-----
-----+
33  | kunde_id | nachname | vorname | umsatzsu
mme |
34  +-----+-----+-----+-----
-----+
35  | 5 | Earendilonn | Elrond | 1212.5500000
00000 |
36  | 1 | Gamdschie | Samweis | 761.1100000000
00 |
37  +-----+-----+-----+-----
-----+

```

Schön, gell?

13.2.2.4 Beispiel 4: ANY()

Wir wollen alle Artikel wissen, die teurer sind als irgendein Artikel der Warengruppe Bürobedarf. Im Gegensatz zu ALL()

liefert das ANY() schon dann ein TRUE, wenn mindestens einer der Vergleiche mit den Listenelementen stimmt.

Schritt 1 – Unterabfrage ermitteln: Die Unterabfrage muss mir alle Artikelpreise der Warengruppe Bürobedarf liefern.

```
1  SELECT a.einzelpreis
2  FROM
3  artikel_nm_warengruppe INNER JOIN artikel
a USING(artikel_id)
4  INNER JOIN warengruppe w USING
(warengruppe_id)
5  WHERE
6  w.bezeichnung = 'Bürobedarf'
7  ;
```

	einzelpreis
	2.320000
	56.260000
	6.260000
	4.170000
	5.050000

Schritt 2 – Bau der Oberanweisung ohne Unterabfrage: Die Oberanweisung muss alle Artikel mit ihren Preisen ausgeben:

```
1  SELECT
2  a.bezeichnung, a.einzelpreis
3  FROM
4  artikel a
5  WHERE
6  a.einzelpreis > ANY(SELECT 10)
7  ;
```

Schritt 3 – Einbau der Unterabfrage in die Oberanweisung: Anstelle des SELECT 10 wird jetzt die Unterabfrage eingebaut.

```

1  SELECT
2    a.bezeichnung, a.einzelpreis
3    FROM
4    artikel a
5    WHERE
6    a.einzelpreis > ANY
7    (
8      SELECT a.einzelpreis
9      FROM
10     artikel_nm_warengruppe INNER JOIN arti
11     kel a USING(artikel_id)
12     INNER JOIN warengruppe w USING(warengr
13     uppe_id)
14     WHERE
15     w.bezeichnung = 'Bürobedarf'
16   )
17 ;
18 +-----+-----+
19 | bezeichnung | einzelpreis |
20 +-----+-----+
21 | Tinte (gold) | 56.260000 |
22 | Tinte (rot) | 6.260000 |
23 | Tinte (blau) | 4.170000 |
24 | Feder | 5.050000 |
25 | Tulpenzwiebel | 3.420000 |
26 | Schaufel | 15.100000 |
27 | Spaten | 20.100000 |
28 +-----+-----+

```

Die Ausgabe ergibt sich daraus, dass es ausreicht, dass der Artikel größer als einer der Preise in der Warengruppe *Bürobedarf* ist.⁴ So ist die Tulpenzwiebel beispielsweise teurer als Papier und taucht deshalb hier auf.

Ärgerlich ist, dass hier die Artikel der Warengruppe *Bürobedarf* auch auftauchen. Hier hilft nur eine zweite Unterabfrage in den [Zeilen 6–13](#):


```

1  SELECT
2      a.bezeichnung, a.einzelpreis
3  FROM
4      artikel a
5  WHERE
6      a.artikel_id NOT IN
7      (
8          SELECT
9              artikel_id
10         FROM
11             artikel_nm_warengruppe nm INNER JOIN w
arengruppe w USING(warengruppe_id)
12         WHERE w.bezeichnung = 'Bürobedarf'
13     )
14  AND
15      a.einzelpreis > ANY
16      (
17          SELECT a.einzelpreis
18         FROM
19             artikel_nm_warengruppe nm INNER JOIN a
rtikel a USING(artikel_id)
20         INNER JOIN warengruppe w USING(warengr
uppe_id)
21         WHERE
22             w.bezeichnung = 'Bürobedarf'
23     )
24  ;
25  +-----+-----+
26  | bezeichnung | einzelpreis |
27  +-----+-----+
28  | Tulpenzwiebel | 3.420000 |
29  | Schaufel | 15.100000 |
30  | Spaten | 20.100000 |
31  +-----+-----+

```

13.2.2.5 Also, der Unterschied zwischen IN(),

ALL() und ANY() ist mir nicht ganz klar!

- **IN()** Diese Abfrage prüft, ob ein Wert in der Liste enthalten ist. Falls er enthalten ist, liefert sie TRUE, sonst FALSE.

Beispiele:

```
[...] 5 IN (1, 2, 3, 4, 5) [...] 7 IN (1, 2, 3, 4, 5)
⇒ TRUE ⇒ FALSE
```

Ein besonderer Fall entsteht, wenn NULL ins Spiel kommt:

```
[...] 5 IN (1, NULL, 3, 4, 5)
⇒ TRUE [...] 7 IN (1, NULL, 3, 4, 5) ⇒ NULL
[...] NULL IN (1, 2, 3, 4, 5)
⇒ NULL
```

- **ALL()** Diese Abfrage prüft, ob ein Wert bezüglich eines Vergleichsoperators für alle Werte der Liste TRUE ergibt. Nur dann wird der Gesamtausdruck auch TRUE. Beispiele⁵:

```
[...] 1 <= ALL (1, 2, 3, 4, 5) [...] 3 <= ALL (1, 2, 3, 4, 5)
⇒ TRUE ⇒ FALSE
```

Taucht in der Liste ein NULL auf, wird immer das Ergebnis auf NULL gesetzt, wenn nicht schon einer der Vergleiche ein FALSE liefert:

```
[...] 1 <> ALL (1, 2, 3, 4, NULL) [...] 1 <> ALL (NULL, 2, 3, 4, 5)
⇒ FALSE ⇒ NULL
```

- **ANY()** Diese Abfrage prüft, ob ein Wert bezüglich eines Vergleichsoperators für mindestens einen Wert der Liste TRUE ergibt. Der Gesamtausdruck wird dann auch TRUE.

```
[...] 3 <= ANY (1, 2, 3, 4, 5) [...] 7 <= ANY (
1, 2, 3, 4, 5)
⇒ TRUE ⇒ FALSE
```

Auch hier ist der Fall NULL zu betrachten. Ergibt der Vergleich mit irgendeinem der Listenelemente TRUE, reicht das aus. Wird kein passender Vergleich gefunden und es kommt ein NULL vor, ist das Ergebnis auch NULL:

```
[...] 3 >= ANY (1, 2, 3, 4, NULL) [...] 1 >= ANY
(NULL, 2, 3, 4, 5)
⇒ TRUE ⇒ NULL
```

13.2.2.6 Gibt es einen Unterschied zwischen NOT IN() und <> ALL()?

Nein, siehe [\[MyS21b\]](#).

13.2.3 Tabellenunterabfrage

Unterabfragen können nicht nur auf *einer* Spalte arbeiten wie bei der Skalar- und der Listenunterabfrage, sondern auch mit Tabellen, die mehrere Spalten enthalten.



Definition 53: Tabellenunterabfragen

Liefert die Unterabfrage eine Tabelle mit mehreren Spalten, so bezeichne ich diese als *Tabellenunterabfrage*.

Es fällt mir sehr schwer, ein einigermaßen sinnvolles Beispiel zu konstruieren. Also lasse ich es und zeige den Effekt anhand einer konstruierten Situation. Wieder wollen wir wissen, welcher Kunde – aber diesmal mit Namen – im Jahr 2011 und 2012 Rechnungen hatte.

```
1  mysql> SELECT DISTINCT k.nachname, k.vorna
me
2      -> FROM rechnung r INNER JOIN kunde k
USING(kunde_id)
3      -> WHERE
4      -> YEAR(r.datum) = '2011' AND (k.nachn
ame, k.vorname) IN
5      -> (
6      -> SELECT
7      -> k.nachname, k.vorname
8      -> FROM
9      -> rechnung r INNER JOIN kunde k USING
(kunde_id)
10     -> WHERE
11     -> YEAR(r.datum) = '2012'
12     -> )
13     -> ;
14  +-----+-----+
15  | nachname | vorname |
16  +-----+-----+
17  | Gamdschie | Samweis |
18  +-----+-----+
```

In [Zeile 4](#) geschieht etwas ganz Neues. Bisher haben wir immer nur skalare Werte miteinander verglichen. Hier werden zwei Listen miteinander verglichen. Auf der linken und der rechten Seite vom IN stehen Listen mit je zwei Spalten.



Hinweis: Beim Vergleich mit Tabellenunterabfragen müssen Sie darauf achten, dass die Anzahl der Spalten

auf der rechten und linken Seite der Vergleichsoperation gleich ist.

13.3 Korrelierende Unterabfrage

[Definition 50](#) in [Abschnitt 13.2](#) führte Unterabfragen ein, die unabhängig von der Oberanweisung sind. Es gibt auch welche, die sich auf die Oberabfrage beziehen und deshalb nicht unabhängig von der Oberanweisung ausgeführt werden können.

13.3.1 Beispiel 1: Rechnungen mit vielen Positionen

Ich möchte alle Rechnungen mit mindestens drei Positionen.

```
1  mysql> SELECT rechnung_id
2      -> FROM rechnung r
3      -> WHERE
4      -> (
5      -> SELECT
6      -> COUNT(position_nr)
7      -> FROM rechnung_position rp
8      -> WHERE rp.rechnung_id = r.rechnung_id
9      -> ) >= 3
10     -> ;
11  +-----+
12  | rechnung_id |
13  +-----+
14  | 1 |
```

```

15  | 5 |
16  +-----+

```

In der WHERE-Klausel der Unterabfrage wird in [Zeile 8](#) mit der Rechnungsnummer der Oberanweisung ([Zeile 1](#)) verglichen⁶.

Da die Unterabfrage jetzt von einem Spalteninhalt der Oberanweisung abhängt, wird sie nicht einmal ausgeführt, sondern für jeden Spalteninhalt der Oberanweisung neu. In unserem Fall wird für jede Rechnungsnummer – also 11-mal – die Anzahl der Rechnungspositionen berechnet.



Hinweis: Weil die korrelierende Unterabfrage mehrfach ausgeführt wird, ist sie ein Performancerisiko.

13.3.2 Beispiel 2: EXISTS

Ich möchte alle Rechnungen, zu denen es eine Bestellung gibt. Hier kann der EXISTS-Operator helfen, da er überprüft, ob das Ergebnis eines SELECT überhaupt eine Zeile zurückliefert.

```

1  mysql> SELECT rechnung_id
2          -> FROM rechnung r
3          -> WHERE
4          -> EXISTS
5          -> (
6          -> SELECT bestellung_id
7          -> FROM bestellung b
8          -> WHERE b.bestellung_id = r.bestellun
rechnung_id
9          -> )
10         -> ;
11  +-----+

```

12		rechnung_id	
13	+-----+		
14		1	
15		2	
16		3	
17		4	
18		5	
19		6	
20	+-----+		

13.4 Common Table Expression versus Unterabfrage

Lehnen wir uns mal kurz zurück, dann fällt Ihnen sicherlich ein, dass wir nicht nur temporäre Tabellen (siehe [Abschnitt 5.3.8](#)) als Datenquelle anstelle von Unterabfragen verwenden können, sondern dass es ja auch noch die Common Table Expressions gab (siehe [Abschnitt 11.5](#)).

Schauen wir uns nochmal das Beispiel aus [Abschnitt 13.2.2.3](#) an. Es sollten alle Kunden ermittelt werden, die mehr Umsatz gemacht haben als die beiden Beutlins.

```

1  SELECT kunde_id, kunde.nachname, kunde.vor
   name, SUM(umsatz) AS umsatzsumme
2  FROM
3    (
4      SELECT
5        rechnung_id, kunde_id, SUM(einzelpreis
*menge) AS umsatz
6      FROM
7        rechnung_position INNER JOIN artikel U
   SING(artikel_id)
8      INNER JOIN rechnung USING(rechnung_id)

```

```

9      GROUP BY
10     rechnung_id
11   ) AS ksum
12   INNER JOIN kunde USING(kunde_id)
13   GROUP BY
14   kunde_id
15   HAVING umsatzsumme > ALL
16   (
17     SELECT SUM(umsatz) AS umsatzsumme
18     FROM
19     (
20     SELECT
21     rechnung_id, kunde_id, SUM(einzelpreis
22 *menge) AS umsatz
23     FROM
24     rechnung_position INNER JOIN artikel U
25     SING(artikel_id)
26     INNER JOIN rechnung USING(rechnung_id)
27     INNER JOIN kunde USING(kunde_id)
28     WHERE nachname = 'beutlin'
29     GROUP BY
30     rechnung_id
31   ) AS ksum
32   GROUP BY
33   kunde_id
34   )
35 ;

```

Aus den Zeilen 20–28 lässt sich ohne die WHERE-Klausel eine virtuelle Tabelle bauen, die mir alle Rechnungssummen liefert:

```

1  WITH rsum AS
2  (
3    SELECT
4    rechnung_id, kunde_id, nachname, vorname
5    ,
6    SUM(einzelpreis*menge) AS umsatz_r
7    FROM

```



```

7      rechnung_position INNER JOIN artikel USI
NG(artikel_id)
8      INNER JOIN rechnung USING(rechnung_id)
9      INNER JOIN kunde USING(kunde_id)
10 GROUP BY rechnung_id
11 )

```

Die virtuelle Tabelle rsum wiederum verwende ich zur Berechnung der Kundensummen:

```

1 ksum AS
2 (
3     SELECT
4     kunde_id, nachname, vorname, SUM(umsatz_
r) AS umsatz_k
5     FROM rsum
6     GROUP BY kunde_id
7 )

```

Und daraus fische ich mir die Umsätze der Beutlins:

```

1 bsum AS
2 (
3     SELECT umsatz_k
4     FROM ksum
5     WHERE nachname='Beutlin'
6 )

```

Nun noch mit Kommata getrennt zusammengebaut und et voilà:

```

1 MySQL> WITH
2     -> rsum AS (
3     -> SELECT
4     -> rechnung_id, kunde_id, nachname, vo
rname,
5     -> SUM(einzelpreis*menge) AS umsatz_r
6     -> FROM

```

```

7      -> rechnung_position INNER JOIN artike
l USING(artikel_id)
8      -> INNER JOIN rechnung USING(rechnung_
id)
9      -> INNER JOIN kunde USING(kunde_id)
10     -> GROUP BY rechnung_id
11     -> ),
12     -> ksum AS (
13     -> SELECT
14     -> kunde_id, nachname, vorname, SUM(um
satz_r) AS umsatz_k
15     -> FROM rsum
16     -> GROUP BY kunde_id
17     -> ),
18     -> bsum AS (
19     -> SELECT umsatz_k
20     -> FROM ksum
21     -> WHERE nachname='Beutlin'
22     -> )
23     -> SELECT kunde_id, nachname, vorname,
umsatz_k
24     -> FROM ksum
25     -> WHERE umsatz_k > ALL (SELECT * FROM
bsum)
26     -> ;
27  +-----+-----+-----+-----+
-----+
28  | kunde_id | nachname | vorname | umsatz_k
|
29  +-----+-----+-----+-----+
-----+
30  | 1 | Gamdschie | Samweis | 761.1100000000
00 |
31  | 5 | Earendilionn | Elrond | 1212.5500000
00000 |
32  +-----+-----+-----+-----+
-----+

```

Es fallen sofort ein paar Vorteile auf, und ich möchte hier zwei aus meiner Sicht wichtige hervorheben: Erstmal ist der Quelltext deutlich einfacher zu lesen und zu verstehen. Da keine in sich verschachtelten SELECT verwendet werden, muss man nur ein CTE nach dem anderen verstehen. Zum Zweiten können CTE in anderen CTE per Namensreferenz einfach wiederverwendet werden.

Weitere vertiefende Vergleiche finden Sie hier: [\[MyS19a\]](#).

13.5 Fallstudie Datenimport

Ein anderer Online-Shop hat pleite gemacht. Sie haben im Rahmen einer Kundenübernahme die Kunden- und Adressdaten des Shops als CSV-Datei⁷ erhalten.

```
nachname;vorname;strasse;hnr;lkz;plz;ort;ktmr;  
blz;  
Brandybock;Gormadoc;Brandyschloss;1;BL;57990;B  
ockenburg;9999999991;28063253  
Starrkopf;Malva;Brandyschloss;1;BL;57990;Bock  
enburg;9999999992;28063253  
[...]
```

Schritt 1 –Daten importieren: Um die Daten auswerten und umbauen zu können, erstelle ich eine temporäre Tabelle, die die Daten erst mal unverändert importieren kann.

```
1 CREATE TEMPORARY TABLE tmp_import (  
2     nachname VARCHAR(255),  
3     vorname VARCHAR(255),  
4     strasse VARCHAR(255),  
5     hnr VARCHAR(255),  
6     lkz VARCHAR(255),  
7     plz VARCHAR(255),
```

```

8      ort VARCHAR(255),
9      ktnr VARCHAR(255),
10     blz VARCHAR(255)
11 );
12
13 SET GLOBAL local_infile = ON;
14 LOAD DATA LOCAL INFILE 'kunden01.csv'
15     INTO TABLE tmp_import
16     FIELDS
17     TERMINATED BY ';'
18     LINES
19     TERMINATED BY '\n'
20     IGNORE 1 LINES
21     (nachname, vorname, strasse, hnr, lkz, p
22     lz, ort, ktnr, blz)
23     ;
24 SET GLOBAL local_infile = OFF;

```



Aufgabe 13.4: Überprüfen Sie, dass keine Warnung vorkommt, und schauen Sie nach, ob die Daten genauso in der Tabelle angekommen sind, wie Sie es sich vorgestellt haben.

Schritt 2 – Kunden ermitteln: Wir wollen jetzt alle neuen Kunden anhand ihres Namens identifizieren. Diese neuen Kunden sollen gleich in die Kundentabelle eingefügt werden.

```

1  INSERT INTO kunde (nachname, vorname)
2      SELECT DISTINCT nachname, vorname
3      FROM tmp_import tmp
4      WHERE
5          (tmp.vorname, tmp.nachname) NOT IN
6          (SELECT vorname, nachname FROM kunde)
7      ;

```

In der [Zeile 2](#) werden alle unterschiedlichen Namen ermittelt. Das DISTINCT soll verhindern, dass mehrfach vorkommende Namen mehrfach übernommen werden. In unserem Fall eine unnötige Vorsichtsmaßnahmen.

Durch das NOT IN in der [Zeile 5](#) wird ermittelt, ob der Name nicht schon in Tabelle kunde vorhanden ist.



Aufgabe 13.5: Welcher Kunde ist aussortiert worden? Erstellen Sie dazu ein passendes SELECT.

Aufgabe 13.6: Handelt es sich in [Zeile 5](#) um eine skalare Unterabfrage, eine Listenunterabfrage oder Tabellenunterabfrage?



Aufgabe 13.7: Handelt es sich in [Zeile 6](#) um eine korrelierende oder nicht korrelierende Unterabfrage?

Schritt 3 – Adressen ermitteln: Wie bei den Kunden muss hier überprüft werden, ob die Adresse schon vorhanden ist. Schließlich will ich keine sinnlosen Dubletten⁸.

```
1  INSERT INTO adresse (strasse, hnr, lkz, plz, ort)
2      SELECT DISTINCT strasse, hnr, lkz, plz, ort
3      FROM tmp_import tmp
4      WHERE
5          (tmp.strasse, tmp.hnr, tmp.lkz, tmp.plz, tmp.ort) NOT IN
6          (SELECT strasse, hnr, lkz, plz, ort FROM adresse)
7      ;
```

Schritt 4 – Einbau der Fremdschlüssel in die Importtabelle:

Wir haben jetzt die neuen Kunden- und Adressdaten eingefügt. Es fehlen aber die Verknüpfungen. Dazu könnte man jetzt verschiedene Wege gehen, aber ich möchte hier die Unterabfragen verwenden, die hier sehr effektiv sind.

Zunächst muss die Importtabelle um die Fremdschlüssel erweitert werden:

```
1 ALTER TABLE tmp_import ADD kunde_id INT UN  
SIGNED,  
2 ADD adresse_id INT UNSIGNED,  
3 ADD bank_id CHAR(12)  
4 ;
```

Sind die Spalten angelegt worden, wird jetzt für jeden Datensatz der Importtabelle der Fremdschlüssel ermittelt und eingetragen.

```
1 UPDATE tmp_import t SET  
2 t.kunde_id =  
3 (SELECT kunde_id  
4 FROM kunde k  
5 WHERE t.vorname = k.vorname AND t.nach  
name = k.nachname  
6 ),  
7 t.adresse_id =  
8 (SELECT adresse_id  
9 FROM adresse a  
10 WHERE  
11 t.strasse = a.strasse  
12 AND t.hnr = a.hnr  
13 AND t.lkz = a.lkz  
14 AND t.plz = a.plz  
15 AND t.ort = a.ort  
16 ),  
17 t.bank_id =
```

```

18      (SELECT bank_id FROM bank WHERE t.blz =
bank.blz LIMIT 1)
19  ;

```

In der [Zeile 2](#) wird für jede Zeile die ermittelte Kundennummer eingetragen, in [Zeile 7](#) die ermittelte Adressnummer und in [Zeile 17](#) die ermittelte bank_id.



Aufgabe 13.8: Handelt es sich in den [Zeilen 2, 7 und 17](#) um korrelierende oder nicht korrelierende Abfragen?

Schritt 5 – Bankverbindung ermitteln: Jetzt können wir die Bankverbindungen in die entsprechende Tabelle eintragen.

```

1  INSERT INTO
2      bankverbindung (kunde_id, bankverbindung
_nr, bank_id, kontonummer, iban)
3      SELECT DISTINCT kunde_id, 1, bank_id, kt
nr, CONCAT(blz, ktnr)
4      FROM tmp_import tmp
5      WHERE
6          (tmp.kunde_id, 1, tmp.bank_id, tmp.ktn
r) NOT IN
7          (SELECT kunde_id, 1, bank_id, kontonum
mer FROM bankverbindung)
8  ;

```

Schritt 6 – Einbau des Fremdschlüssels auf die

Bankverbindung: Die Bankverbindung ist jetzt auch bekannt. Da der erste Teile des Primärschlüssels die Kundennummer ist und wir annehmen können, dass jeder der neuen Kunden nur eine Bankverbindung hat, brauchen wir nur eine 1 in die Spalte bankverbindung_nr eintragen.

```
1 ALTER TABLE tmp_import
2 ADD bankverbindung_nr INT UNSIGNED NOT NUL
L DEFAULT 1
3 ;
```

Schritt 7 – Abschluss: Wir haben die neuen Kunden angelegt, wir haben die neuen Adressen angelegt, wir haben die neuen Bankverbindungen angelegt. Die Bankverbindungen waren schnell über die Kundennummer und die Bankverbindungsnummer mit dem Kunden verknüpft (Schritt 6). Bleibt noch, die Kunden mit der Adresse zu verknüpfen. Es stehen zwei Möglichkeit offen: die Rechnungs- oder Lieferadresse. Wir entscheiden uns für die Rechnungsadresse:

```
1 UPDATE kunde k SET
2     k.rechnung_adresse_id =
3     (SELECT adresse_id FROM tmp_import tmp W
HERE tmp.kunde_id = k.kunde_id)
4     WHERE k.rechnung_adresse_id IS NULL
5 ;
```



Aufgabe 13.9: Ist [Zeile 2](#) in Schritt 7 eine korrelierende oder nicht korrelierende Unterabfrage?

Zum Abschluss überprüfen wir, ob der Import geklappt hat, was natürlich nur wegen der wenigen Daten möglich ist (schließlich ist der Shop ja pleite gegangen).

```
1 mysql> SELECT kunde_id, nachname, vorname,
2     strasse, ort
3     -> FROM
4     -> kunde LEFT JOIN adresse ON kunde.re
chnung_adresse_id = adresse_id
```



```

4      -> ;
5  +-----+-----+-----+-----+
-----+-----+
6  | kunde_id | nachname | vorname | strasse
   | ort      |
7  +-----+-----+-----+-----+
-----+-----+
8  | 1 | Gamdschie | Samweis | Beutelhaldenwe
   | g | Hobbingen |
9  | 2 | Beutlin   | Frodo  | Beutelhaldenweg |
   | Hobbingen   |
10 | 3 | Beutlin   | Bilbo  | Beutelhaldenweg |
   | Hobbingen   |
11 | 4 | Telcontar | Elessar | Auf der Feste
   | Minas Tirith |
12 | 5 | Earendilionn | Elrond | Letztes Haus
   | Bruchtal     |
13 | 6 | Eichenschild | Thorin | NULL | NULL
   |
14 | 7 | Brandybock | Gormadoc | Brandyschlos
   | s | Bockenburg |
15 | 8 | Starrkopf  | Malva  | Brandyschloss |
   | Bockenburg   |
16 | 9 | Brandybock | Madoc  | Brandyschloss |
   | Bockenburg   |
17 | 10 | Goldwert  | Hanna  | Brandyschloss |
   | Bockenburg   |
18 | 11 | Brandybock | Marmamadoc | Brandysch
   | loss | Bockenburg |
19 | 12 | Bolger   | Adaldrida | Brandyschloss
   | Bockenburg   |
20 | 13 | Brandybock | Gorbado | Brandyschlo
   | ss | Bockenburg |
21 | 14 | Tuk      | Mirabella | Brandyschloss | B
   | ockenburg    |
22 | 15 | Brandybock | Rorimac | Brandyschlos
   | s | Bockenburg |
23 | 16 | Guld     | Menegilda | Brandyschloss |

```

```

Bockenburg |
24 | 17 | Brandybock | Saradoc | Brandyschlos
s | Bockenburg |
25 | 18 | Tuk | Esmeralda | Brandyschloss | B
ockenburg |
26 | 19 | Brandybock | Meriadoc | Brandyschlo
ss | Bockenburg |
27 | 20 | Brandybock | Merimac | Bocklandstra
ße | Krickloch |
28 | 21 | Lochner | Rufus | Maggots Weg | Mag
gots Hof |
29 | 22 | Tuk | Peregrin | Stockstraße | Buck
elstadt |
30 +-----+-----+-----+---
-----+-----+

```

13.6 Wie ticken Unterabfragen intern?

Schauen wir uns den (gekürzten) EXPLAIN zu Schritt 3 in Beispiel 3 aus [Abschnitt 13.2.1.3](#) an:

```

1  mysql> EXPLAIN /wie oben/\G
2  ***** 1. row ***
3      id: 1
4      select_type: PRIMARY
5      table: a1
6      type: ALL
7  possible_keys: PRIMARY
8      key: NULL
9      key_len: NULL
10     ref: NULL
11     rows: 2
12     filtered: 100.00
13     Extra: Using temporary
14  ***** 2. row ***

```

```

15         id: 1
16     select_type: PRIMARY
17         table: bp1
18         type: ref
19 possible_keys: PRIMARY,artikel_id
20         key: artikel_id
21     key_len: 4
22         ref: oshop.a1.artikel_id
23         rows: 3
24         filtered: 100.00
25         Extra: NULL
26 ***** 3. row ***
27         id: 2
28     select_type: SUBQUERY
29         table: <derived3>
30         type: ALL
31 possible_keys: NULL
32         key: NULL
33     key_len: NULL
34     ref: NULL
35     rows: 6
36     filtered: 100.00
37     Extra: NULL
38 ***** 4. row ***
39         id: 3
40     select_type: DERIVED
41         table: a2
42         type: ALL
43 possible_keys: PRIMARY
44         key: NULL
45     key_len: NULL
46     ref: NULL
47     rows: 2
48     filtered: 100.00
49     Extra: Using temporary
50 ***** 5. row ***
51         id: 3
52     select_type: DERIVED

```

```
53         table: bp2
54         type: ref
55 possible_keys: PRIMARY,artikel_id
56         key: artikel_id
57     key_len: 4
58     ref: oshop.a2.artikel_id
59     rows: 3
60     filtered: 100.00
61     Extra: NULL
```

Betrachten wir zunächst 1. row und 2. row. Beide haben die id: 1 (siehe [Zeilen 3](#) und [15](#)), was bedeutet, dass beide Ausgaben zum gleichen SELECT gehören. Dies wird ebenfalls durch die Angabe PRIMARY in den [Zeilen 4](#) und [16](#) bestätigt. Die Angaben in 1. row und 2. row beschreiben somit die Art und Weise, wie der INNER JOIN zwischen den beiden Tabellen bestellung_position und artikel organisiert wird. Verwendet wird die Spalte artikel_id ([Zeilen 19](#) und [22](#)) und die damit verbundenen Indizes (Primärschlüssel: [Zeile 7](#), Fremdschlüssel [19](#)).

Jetzt 3. row. Mit der id: 2 wird mir angezeigt, dass es ein neuer SELECT sein muss. Unter type ([Zeile 28](#)) wird mit SUBQUERY mitgeteilt, dass dieser SELECT eine Unterabfrage außerhalb einer FROM-Klausel ist. Der Tabellename <derived3> enthält den Hinweis auf die id 3. Die nächsten beiden Zeilen haben die id: 3. Wir schließen daraus, dass 3. row eine Unterabfrage auf die Tabelle, die durch den SELECT mit der id: 3 entsteht, ist.

Zum Schluss schauen wir uns 4. row und 5. row an. Beide gehören zum gleichen SELECT, da sie die gleiche id: 3 haben. Durch den select_type DERIVED ([Zeile 40](#)) wird mir mitgeteilt, dass MySQL die Unterabfrage vorab ausführt und das Ergebnis in eine – unsichtbare – temporäre Tabelle abspeichert. Die restlichen Angaben entsprechen denen von

1. row und 2. row, da hier der gleiche INNER JOIN ausgeführt wird.

13.7 Aufgaben



Aufgabe 13.10: Geben Sie den Kunden mit der kleinsten Postleitzahl aus.

Aufgabe 13.11: Geben Sie die jüngste Bestellung aus.

Aufgabe 13.12: Geben Sie den Lieferanten mit den meisten Artikel an.

Aufgabe 13.13: Ermitteln Sie die Bankleitzahlen mit den meisten Banken.

Aufgabe 13.14: Geben Sie das Clearinggebiet (1. Stelle der Bankleitzahl) aus, welches die wenigsten Banken hat.

Aufgabe 13.15: Ermitteln Sie alle Bestellungen, in denen Artikel vorkommen, die ihre Mindestmenge auf dem Lager unterschritten haben.

Aufgabe 13.16: Geben Sie alle Kundennamen aus, die eine Rechnung im Mahn- oder Inkassoverfahren haben.

Aufgabe 13.17: Geben Sie alle Artikel aus, die in der Warengruppe Gartenbedarf und in der Warengruppe Pflanzen sind.

Aufgabe 13.18: Gibt es Kunden, die nur in 2011 Rechnungen hatten?

Aufgabe 13.19: Geben Sie alle Rechnungen aus, die keine Artikel der Warengruppe Gartenbedarf haben.

Aufgabe 13.20: Ermitteln Sie alle Rechnungen, zu denen es keine Bestellungen gibt.

-
- ¹ Hier sind Nebenläufigkeiten bei Transaktionen zu berücksichtigen. Mehr in [Kapitel 19](#).
 - ² In [Abschnitt 26.3.1](#) finden Sie eine entsprechende Übersicht.
 - ³ In `listing10.sql` wird der Datenbestand entsprechend erweitert.
 - ⁴ Was für ein Satz :- (Aber ich bin sicher, Sie verstehen mich :-)
 - ⁵ Die Beispiele sind eine Verkürzung von beispielsweise `SELECT 3 <= ALL (SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4 UNION SELECT 5)`.
 - ⁶ Genau diese Verknüpfung zwischen der Oberanweisung und der Unterabfrage ist die *Korrelation*, die dieser Abfrage den Namen gegeben hat.
 - ⁷ Die Daten sind in der Datei `kunden01.csv`.
 - ⁸ Obwohl man das hier diskutieren kann. Zwei Personen können die gleiche Adresse haben, und trotzdem würde man diese nicht als *eine* Adresse im System abspeichern.

14 Mengenoperationen



Haben Sie die Mengenlehre in der Grundschule auch so gehasst? Ich hoffe, das hat sich gelegt.

- Grundkurs
 - Auswertungen vereinigen mit UNION
 - Schnittmenge mit INTERSECT
 - Differenzmenge mit EXCEPT
- Vertiefendes
 - ALL und DISTINCT
 - Simulation von INTERSECT
 - Simulation von EXCEPT



Die Quelltexte dieses Kapitels stehen in der Datei `listing11.sql` (siehe [Listing 28.11](#), [Listing 28.51](#) und [Listing 28.36](#)).

14.1 Die Vereinigung mit UNION

Wir möchten alle verwendeten Adressen ausgeben. Unser erster Ansatz dürfte wie folgt aussehen:

```
1  mysql> SELECT
2      -> strasse, hnr, lkz, plz, ort
3      -> FROM
4      -> kunde INNER JOIN adresse ON rechnun
g_adresse_id = adresse_id
5      -> ;
6  +-----+-----+-----+-----+-----+
7  | strasse | hnr | lkz | plz | ort |
8  +-----+-----+-----+-----+-----+
9  | Beutelhaldenweg | 5 | AL | 67676 | Hobbi
nngen |
10 | Beutelhaldenweg | 1 | AL | 67676 | Hobbi
nngen |
11 | Beutelhaldenweg | 1 | AL | 67676 | Hobbi
nngen |
12 | Auf der Feste | 1 | GO | 54786 | Minas T
irith |
13 | Letztes Haus | 4 | ER | 87567 | Bruchtal
|
14 | Brandyschloss | 1 | BL | 57990 | Bockenb
urg |
15 | Brandyschloss | 1 | BL | 57990 | Bockenb
urg |
16 | Brandyschloss | 1 | BL | 57990 | Bockenb
urg |
17 | Brandyschloss | 1 | BL | 57990 | Bockenb
urg |
18 | Brandyschloss | 1 | BL | 57990 | Bockenb
urg |
19 | Brandyschloss | 1 | BL | 57990 | Bockenb
urg |
20 | Brandyschloss | 1 | BL | 57990 | Bockenb
urg |
```



```

21 | Brandyschloss | 1a | BL | 57990 | Bocken
    | burg |
22 | Brandyschloss | 1a | BL | 57990 | Bocken
    | burg |
23 | Brandyschloss | 1a | BL | 57990 | Bocken
    | burg |
24 | Brandyschloss | 1a | BL | 57990 | Bocken
    | burg |
25 | Brandyschloss | 1a | BL | 57990 | Bocken
    | burg |
26 | Brandyschloss | 1a | BL | 57990 | Bocken
    | burg |
27 | Bocklandstraße | 4 | BL | 57996 | Krickl
    | och |
28 | Maggots Weg | 2 | BL | 57980 | Maggots H
    | of |
29 | Stockstraße | 1 | AL | 45689 | Buckelsta
    | dt |
30 +-----+-----+-----+-----+-----+
    -----+
31 21 rows in set (0.00 sec)

```

In der Ausgabe fehlen aber die Adressen der Lieferanten. Ein zweiter INNER JOIN liefert mir zwar diese Adressen,

```

1  mysql> SELECT
2      -> strasse, hnr, lkz, plz, ort
3      -> FROM
4      -> lieferant INNER JOIN adresse USING(
    | adresse_id)
5      -> ;
6  +-----+-----+-----+-----+-----+
    -----+
7  | strasse | hnr | lkz | plz | ort |
8  +-----+-----+-----+-----+-----+
    -----+
9  | Hochstrasse | 4a | DE | 44879 | Bochum |
10 | Industriegebiet | 8 | DE | 44878 | Bochu

```

```

m |
11 | Industriegebiet | 8 | DE | 44878 | Bochu
m |
12 +-----+-----+-----+-----+-----+
-----+

```

aber ich möchte *alle* in *einer* Ausgabe haben. Hier hilft ein UNION:



SQL:2023, T-SQL

MySQL/MariaDB, PostgreSQL

SELECT *auswahl*

SELECT *auswahl*

[UNION [ALL]

[UNION [ALL|DISTINCT]

SELECT *auswahl*]*

SELECT *auswahl*]*

;

;

Bei SQL:2023 kann die Option DISTINCT zwar nicht angegeben werden, ist aber – wie bei MySQL und MariaDB – der Default. Mit dieser neuen Möglichkeit versuchen wir es noch einmal:

```

1  mysql> SELECT
2      -> strasse, hnr, lkz, plz, ort
3      -> FROM
4      -> kunde INNER JOIN adresse ON rechnun
g_adresse_id = adresse_id
5      -> UNION
6      -> SELECT
7      -> strasse, hnr, lkz, plz, ort
8      -> FROM
9      -> lieferant INNER JOIN adresse USING(
adresse_id)

```

```

10      -> ;
11  +-----+-----+-----+-----+-----+
-----+
12  | strasse | hnr | lkz | plz | ort |
13  +-----+-----+-----+-----+-----+
-----+
14  | Beutelhaldenweg | 5 | AL | 67676 | Hobbi
ngen |
15  | Beutelhaldenweg | 1 | AL | 67676 | Hobbi
ngen |
16  | Auf der Feste | 1 | GO | 54786 | Minas T
irith |
17  | Letztes Haus | 4 | ER | 87567 | Bruchtal
|
18  | Brandyschloss | 1 | BL | 57990 | Bockenb
urg |
19  | Brandyschloss | 1a | BL | 57990 | Bocken
burg |
20  | Bocklandstraße | 4 | BL | 57996 | Krickl
och |
21  | Maggots Weg | 2 | BL | 57980 | Maggots H
of |
22  | Stockstraße | 1 | AL | 45689 | Buckelsta
dt |
23  | Hochstrasse | 4a | DE | 44879 | Bochum |
24  | Industriegebiet | 8 | DE | 44878 | Bochu
m |
25  +-----+-----+-----+-----+-----+
-----+
26  11 rows in set (0.00 sec)

```



Aufgabe 14.1: Mit der Option ALL erhalte ich nachfolgende Ausgabe. Welche Bedeutung haben die Optionen DISTINCT und ALL also?

```

+-----+-----+-----+-----+-----+
-----+

```

[illegible]

```

burg |
| Bocklandstraße | 4 | BL | 57996 | Krickl
och |
| Maggots Weg | 2 | BL | 57980 | Maggots H
of |
| Stockstraße | 1 | AL | 45689 | Buckelsta
dt |
| Hochstrasse | 4a | DE | 44879 | Bochum |
| Industriegebiet | 8 | DE | 44878 | Bochu
m |
| Industriegebiet | 8 | DE | 44878 | Bochu
m |
+-----+-----+-----+-----+-----+
-----+
24 rows in set (0.00 sec)

```

Es gibt folgende Szenarien, die der UNION hauptsächlich abarbeitet:

1. Zwischen zwei Tabellen bestehen zwei Verknüpfungen. Denken Sie beispielsweise an die Tabellen `verein` und `spielpaarung`. In der Tabelle `spielpaarung` würde mit zwei Fremdschlüsseln auf die Tabelle `verein` verwiesen werden: `heimmannschaft` und `gastmannschaft`. Möchte ich jetzt wissen, wie viele Tore eine Mannschaft geschossen hat, wird ein Teil des UNION über die `heimmannschaft` und der zweite über die `gastmannschaft` sein.
2. Eine Tabelle wird über den gleichen Primärschlüssel mit zwei verschiedenen Tabellen verknüpft. Das ist das Szenario, was wir oben beschrieben haben. Die Tabelle `adresse` wird einmal mit `lieferant` und einmal mit `kunde` verknüpft.
3. Tabellen werden aufgeteilt, aber es werden Auswertungen

über alle Daten benötigt. Eine Analyse der Datenzugriffe könnte ergeben, dass bestimmte Datensätze über einen Zeitraum von einem Monat oft und danach nur noch selten verwendet werden. Es ist sinnvoll, diese Tabelle – besonders, wenn sie groß und performancerelevant ist – in mehrere Tabellen aufzuteilen. In der einen stehen die Daten mit häufigem Zugriff und in der anderen die mit seltenem. Für Auswertungen müssen diese Tabellen wieder vereinigt werden. Eine temporäre Tabelle oder ein UNION wäre dann eine gute Wahl.

4. Hinzufügen von *dummy*-Zeilen z.B. für Listboxen auf der Oberfläche. Oft werden die Inhalte einer Listbox direkt aus einer Tabelle ermittelt. Denken Sie beispielsweise an unsere Kundenart oder die Bankleitzahl. Ein `SELECT DISTINCT` mit `ORDER BY`, und schon hat man alle verwendbaren Alternativen. Die Vorbelegung der Listbox soll aber ein Text wie `Bitte auswählen!` sein. Diesen Text müsste man mit einem UNION dem eigentlichen `SELECT` anfügen.



Hinweis: Sie müssen darauf achten, dass beide `SELECT` des `UNION` die gleichen Spalten in der gleichen Reihenfolge zurückliefern, sonst droht diese Fehlermeldung:

```
ERROR 1222 (21000):  
  The used SELECT statements have a different number of columns
```

14.2 Die Schnittmenge

Manchmal brauchen Sie die Zeilen, die auch in einer anderen Tabelle vorkommen. Man nennt dies die Schnittmenge zweier Tabellen. MySQL kennt im Gegensatz zu MariaDB¹ keinen entsprechenden Operator, aber man kann das Verhalten mit einer Unterabfrage emulieren.

14.2.1 Mit INTERSECT

 SQL:2023	PostgreSQL/MariaDB	T-SQL
SELECT <i>auswahl</i>	SELECT <i>auswahl</i>	SELECT <i>auswahl</i>
[INTERSECT [ALL]	[INTERSECT [ALL DISTINCT]	[INTERSECT
SELECT <i>auswahl</i>]*	SELECT <i>auswahl</i>]*	SELECT <i>auswahl</i>]*
;	;	;

Ich möchte Folgendes wissen: Welche Kunden haben sowohl 2011 als auch 2012 eine Rechnung erhalten (Beispiel in PostgreSQL ausgeführt)?

```
1 oshop=# SELECT kunde_id FROM rechnung
2 oshop=# WHERE EXTRACT(ISOYEAR FROM datum)
= '2011'
3 oshop=# INTERSECT
4 oshop=# SELECT kunde_id FROM rechnung
5 oshop=# WHERE EXTRACT(ISOYEAR FROM datum)
= '2012'
```

```

6  oshop-# ;
7
8  kunde_id
9  -----
10         1

```

Wie bei UNION werden Dubletten hier unterdrückt. Mit INTERSECT ALL würden ggf. vorhandene Dubletten ausgegeben werden. Die Anforderungen an die einzelnen SELECT sind wie bei einem UNION. Alle SELECT müssen die gleiche Anzahl von Spalten mit den gleichen Reihenfolgen von Datentypen haben.

14.2.2 Mit Unterabfragen

In MySQL muss dies mittels einer Unterabfrage emuliert werden.

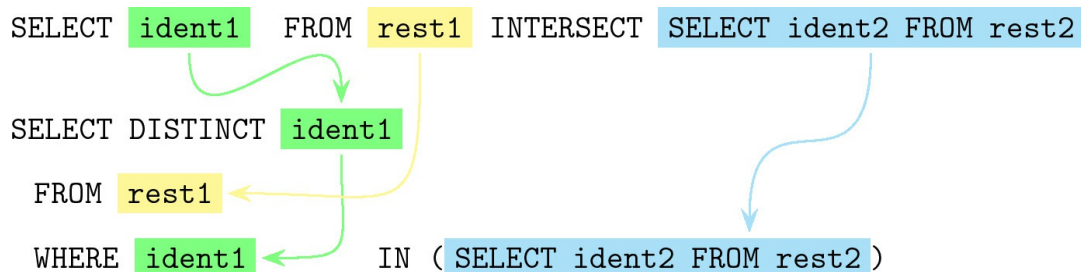


Bild 14.1 Umbau von INTERSECT in MySQL und MariaDB

Schritt 1 – Unterabfrage ermitteln: Wir brauchen alle Kundennummern – und zwar nur einmal –, die im Jahr 2011 eine Rechnung hatten:

```

1  SELECT DISTINCT kunde_id
2  FROM rechnung
3  WHERE YEAR(datum) = '2011'
4  ;

```


Schritt 2 – Bau der Oberanweisung ohne Unterabfrage: Die sieht fast gleich aus. Nur dass hier mit AND das *Enthaltesein* in einer Liste verknüpft wird. Die Werte 1, 2,3 sind nur willkürlich, um den Befehl ausprobieren zu können.

Ein weiterer Punkt ist das DISTINCT. Es verhindert die Mehrfachausgabe von Kundennummern, schließlich könnte ein Kunde ja mehrfach in beiden Jahren etwas gekauft haben.

```
1  SELECT DISTINCT kunde_id
2      FROM rechnung
3      WHERE
4          YEAR(datum) = '2012' AND kunde_id IN (
5  1, 2, 3)
6  ;
```

Schritt 3 – Einbau der Unterabfrage in die Oberanweisung:
In das Klammerpaar von IN() wird jetzt die Unterabfrage eingebaut.

```
1  mysql> SELECT DISTINCT kunde_id
2      -> FROM rechnung
3      -> WHERE
4      -> YEAR(datum) = '2012' AND kunde_id I
5  N
6      -> (
7      -> SELECT DISTINCT kunde_id
8      -> FROM rechnung
9      -> WHERE YEAR(datum) = '2011'
10     -> )
11     -> ;
12
13  +-----+
14  | kunde_id |
15  +-----+
16  | 1 |
```

In [Bild 14.1](#) ist der Umbau grafisch dargestellt.



Aufgabe 14.2: Was müssten Sie weglassen, um einen INTERSECT ALL zu emulieren?

Aufgabe 14.3: Welche Adressen sind sowohl Rechnungs- als auch Lieferadresse?

14.3 Die Differenzmenge

Die Zeilen, die in einer, aber nicht in einer anderen Tabelle vorkommen, nennt man die Differenzmenge zweier Tabellen. Wie bei INTERSECT unterscheiden sich MySQL und MariaDB² diesbezüglich; MySQL kennt keinen entsprechenden Operator, aber Sie können das Verhalten mit einer Unterabfrage emulieren.

Anders als bei der Vereinigung oder dem Durchschnitt zweier Tabellen ist hier die Reihenfolge der Tabellen wichtig. Ziehen Sie von allen Hauptstädten der Welt die europäischen ab, erhalten Sie alle außereuropäischen Hauptstädte. Beispiel: Ziehen von allen europäischen Hauptstädten alle Hauptstädte der Welt ab, erhalten Sie eine leere Menge!

14.3.1 Mit EXCEPT



SQL:2023

PostgreSQL/MariaDB T-SQL

SELECT <i>auswahl</i>	SELECT <i>auswahl</i>	SELECT <i>auswahl</i>
[EXCEPT [ALL]	[EXCEPT [ALL DISTINCT]	[EXCEPT
SELECT <i>auswahl</i>]*	SELECT <i>auswahl</i>]*	SELECT <i>auswahl</i>]*
;	;	;

Ich möchte Folgendes wissen: Welche Adressen sind keine Lieferadressen?

```

1  oshop=# SELECT strasse, hnr, lkz, plz, ort
   FROM adresse
2  oshop=# EXCEPT
3  oshop=# SELECT strasse, hnr, lkz, plz, ort
4  oshop=# FROM
5  oshop=# kunde INNER JOIN adresse ON liefer
   _adresse_id = adresse_id
6  oshop=# ;
7      strasse | hnr | lkz | plz | ort
8  -----+-----+-----+-----+
9  Hochstrasse | 4a  | DE  | 44879 | Bochum
10 Stockstraße | 1   | AL  | 45689 | Buckelstadt
11 Industriegebiet | 8   | DE  | 44878 | Bochum
12 Brandyschloss | 1   | BL  | 57990 | Bockenbur
   g
13 Auf der Feste | 1   | GO  | 54786 | Minas Tir
   ith
14 Bocklandstraße | 4   | BL  | 57996 | Krickloc
   h
15 Baradur      | 1   | MO  | 62519 | Lugburz
16 Brandyschloss | 1a  | BL  | 57990 | Bockenbu

```

```

rg
17 Beutelhaldenweg | 5 | AL | 67676 | Hobbing
en
18 Maggots Weg | 2 | BL | 57980 | Maggots Hof
19 Letztes Haus | 4 | ER | 87567 | Bruchtal
20 (11 Zeilen)

```

Bitte beachten Sie, dass der Beutelhaldenweg 1 fehlt.

Wie bei UNION werden Dubletten hier unterdrückt. Mit EXCEPT ALL würden ggf. vorhandene Dubletten ausgegeben werden. Die Anforderungen an die einzelnen SELECT ist wie bei einem UNION. Alle SELECT müssen die gleiche Anzahl von Spalten mit den gleichen Reihenfolgen von Datentypen haben.

14.3.2 Mit Unterabfragen

Auch den fehlenden EXCEPT können Sie in MySQL emulieren. Als Beispiel wollen wir alle Neukunden des Jahres 2012.

```

SELECT ident1 FROM rest1 EXCEPT SELECT ident2 FROM rest2

```

```

SELECT DISTINCT ident1
FROM rest1
WHERE ident1 NOT IN (SELECT ident2 FROM rest2)

```

Bild 14.2 Umbau von EXCEPT in MySQL und MariaDB

Schritt 1 –Unterabfrage ermitteln:Wir brauchen alle Kundennummern, die vor 2012 eine Rechnung hatten:

```

1 SELECT DISTINCT kunde_id
2   FROM rechnung
3   WHERE YEAR(datum) < 2012

```

4 ;

Schritt 2 – Bau der Oberanweisung ohne Unterabfrage: Die sieht fast gleich aus wie beim INTERSECT. Der einzige Unterschied ist das NOT vor dem IN:

```
1  SELECT DISTINCT kunde_id
2  FROM rechnung
3  WHERE
4  YEAR(datum) = 2012 AND kunde_id NOT IN (1,
2, 3);
```

Schritt 3 – Einbau der Unterabfrage in die Oberanweisung: Wie beim INTERSECT wird die Unterabfrage in die Klammerung des IN() eingefügt.

```
1  mysql> SELECT DISTINCT kunde_id
2      -> FROM rechnung
3      -> WHERE
4      -> YEAR(datum) = '2012' AND kunde_id N
OT IN
5      -> (
6      -> SELECT DISTINCT kunde_id
7      -> FROM rechnung
8      -> WHERE YEAR(datum) < 2012
9      -> )
10     -> ;
11  +-----+
12  | kunde_id |
13  +-----+
14  | 2 |
15  | 3 |
16  | 5 |
17  +-----+
```

In [Bild 14.2](#) ist der Umbau grafisch dargestellt.



Aufgabe 14.4: Was müssen Sie hier weglassen, um den Befehl `EXCEPT ALL` zu emulieren?

14.4 UNION, INTERSECT und EXCEPT ... versteh' ich nicht!

Ich werde nach einer formalen Definition versuchen, anhand einer Grafik die jeweilige Operation zu verdeutlichen.

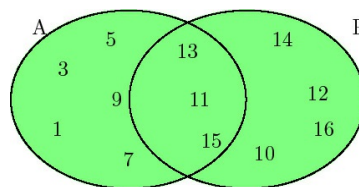


Bild 14.3 UNION

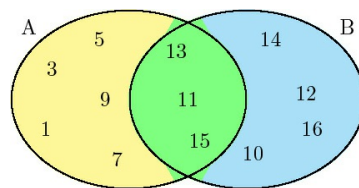


Bild 14.4 INTERSECT

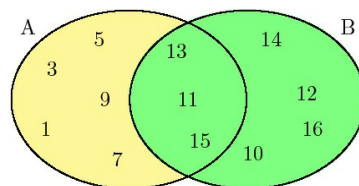


Bild 14.5 EXCEPT



Definition 54: UNION

Die Menge aller Zeilen $z \in U$ nennt man *UNION* oder *Vereinigung* von A und B , wenn Folgendes gilt: $U = \{z | z \in A \vee z \in B\}$. Kurzschreibweise: $U = A \cup B$

Die Vereinigung (siehe [Bild 14.3](#)) liefert mir also alle Zeilen, egal aus welcher Tabelle sie stammen, zurück.

In unserem Beispiel wäre $U = \{1, 3, 5, 7, 9, 10, 11, 12, 13, 14, 15, 16\}$. Bei einer Vereinigung kann es vorkommen, dass es Zeilen gibt, die sowohl in der Tabelle A als auch in der Tabelle B enthalten sind. Mit der Option `ALL` werden diese Zeilen mehrfach ausgegeben, ansonsten nicht.

Wenn wir uns jetzt noch einmal die Ergebnisse in [Abschnitt 14.1](#) anschauen, dann sind dies immer alle Zeilen, die durch die `SELECT` vor und nach dem `UNION` erzeugt wurden.



Definition 55: INTERSECT

Die Menge aller Zeilen $z \in I$ nennt man *INTERSECT* oder *Schnittmenge* von A und B , wenn Folgendes gilt: $I = \{z | z \in A \wedge z \in B\}$. Kurzschreibweise: $I = A \cap B$

Die Schnittmenge (siehe [Bild 14.4](#)) liefert mir nur die Zeilen, die sowohl in der Tabelle A als auch in der Tabelle B vorkommen. In unserem Beispiel wäre $I = \{11, 13, 15\}$. Auch hier können mehrere Zeilen in beiden Tabellen vorkommen. Die Option `ALL` liefert jede Zeile ggf. mehrfach.

Wenn wir uns das Ergebnis in [Abschnitt 14.2](#) anschauen, kommt dort nur die Zeile vor, die sowohl vor dem `INTERSECT`

als auch danach durch die SELECT erzeugt wurde.



Definition 56: EXCEPT

Die Menge aller Zeilen $z \in E$ nennt man *EXCEPT* oder *Differenzmenge* von A ohne B , wenn Folgendes gilt: $E = \{z | z \in A \wedge z \notin B\}$. Kurzschreibweise: $E = A \setminus B$

Die Differenzmenge (siehe [Bild 14.5](#)) liefert mir nur die Zeilen, die in der Tabelle A , aber nicht in der Tabelle B vorkommen. In unserem Beispiel wäre $E = \{1,3,5, 7,9\}$. Auch hier können mehrere Zeilen in Tabelle A vorkommen. Die Option ALL liefert jede Zeile ggf. mehrfach.

In [Abschnitt 14.3](#) werden nur die Zeilen ausgegeben, die ausschließlich durch das SELECT vor dem INTERSECT ermittelt wurden.



Hinweis: Anders als bei UNION und INTERSECT ist die Reihenfolge hier bedeutsam:

$$E_1 = A \setminus B = \{1,3,5,7,9\}, \text{ aber } E_2 = B \setminus A = \{10,12,14,16\}$$

Zur Beruhigung möchte ich anmerken, dass der UNION häufiger vorkommt, aber INTERSECT und EXCEPT relativ selten. Natürlich kommt es auf die Anwendung an, aber für *stinknormale* Geschäftsprozesse und Auswertungen brauchen Sie diese fast nie.

¹ MariaDB verfügt seit der Version 10.3.0 über den INTERSECT.

² MariaDB verfügt seit der Version 10.3.0 über den EXCEPT.

15 Bedingungslogik



Fallunterscheidungen fallen einem nur von Fall zu Fall ein!

- Vertiefendes
 - Problem und Lösung
 - CASE mit Test auf Gleichheit
 - CASE mit beliebigen Bedingungen



Die Quelltexte dieses Kapitels stehen in der Datei `listing12.sql` (siehe [Listing 28.12](#), [Listing 28.52](#) und [Listing 28.37](#)).

15.1 Warum ein CASE?

Wir haben zwei Kundenarten: Geschäftskunden und Privatkunden. Diese Information ist in einem ENUM der Tabelle `kunde` abgelegt:

```
1 mysql> SELECT kunde_id, nachname, vorname,
```

```

art
2  -> FROM kunde;
3  +-----+-----+-----+-----+
----+
4  | kunde_id | nachname | vorname | art |
5  +-----+-----+-----+-----+
----+
6  | 1 | Gamdschie | Samweis | prv |
7  | 2 | Beutlin | Frodo | prv |
8  | 3 | Beutlin | Bilbo | prv |
9  | 4 | Telcontar | Elessar | prv |
10 | 5 | Earendillionn | Elrond | gesch |
11 | 6 | Eichenschild | Thorin | unb |
12 [...]
13 | 22 | Tuk | Peregrin | unb |
14 +-----+-----+-----+-----+
----+
15 22 rows in set (0.00 sec)

```

Die abkürzende Schreibweise des ENUM ist nicht selbsterklärend oder präsentabel. Wir wollen, dass die Wörter Privatkunde, Geschäftskunde und Unbekannt ausgegeben werden. Eine Möglichkeit wäre die Anlage einer temporären Tabelle und einer Verknüpfung:

```

1  mysql> CREATE TEMPORARY TABLE tmp_art (
2      -> art VARCHAR(255),
3      -> k_art VARCHAR(255)
4      -> );
5
6  mysql> INSERT INTO tmp_art
7      -> VALUES
8      -> ('prv', 'Privatkunde')
9      -> ,('gesch', 'Geschäftskunde')
10     -> ,('unb', 'Unbekannt');
11
12  mysql> SELECT
13     -> kunde_id, nachname, vorname, k_art

```

```

14      -> FROM
15      -> kunde INNER JOIN tmp_art USING(art)
16      ;
17      +-----+-----+-----+---
-----+
18      | kunde_id | nachname | vorname | k_art |
19      +-----+-----+-----+---
-----+
20      | 1 | Gamdschie | Samweis | Privatkunde |
21      | 2 | Beutlin | Frodo | Privatkunde |
22      | 3 | Beutlin | Bilbo | Privatkunde |
23      | 4 | Telcontar | Elessar | Privatkunde |
24      | 5 | Earendilionn | Elrond | Geschäftskun
de |
25      | 6 | Eichenschild | Thorin | Unbekannt |
26      [...]
27      | 22 | Tuk | Peregrin | Unbekannt |
28      +-----+-----+-----+---
-----+
29      22 rows in set (0.00 sec)

```

Klappt ja prima, aber geht das nicht einfacher? Klar – mit vielen UNIONS und einer schönen Unterabfrage:

```

1  mysql> SELECT kunde_id, nachname, vorname,
2         k_art
3         -> FROM
4         -> kunde INNER JOIN
5         -> (
6         -> SELECT 'prv' AS art, 'Privatkunde'
AS k_art
7         -> UNION
8         -> SELECT 'gsch' AS art, 'Geschäftskun
de' AS k_art
9         -> UNION
10        -> SELECT 'unb' AS art, 'Unbekannt' AS
k_art

```

```

10      -> ) t
11      -> USING(art)
12      -> ;
13
14  +-----+-----+-----+-----+
15  | kunde_id | nachname | vorname | k_art |
16  +-----+-----+-----+-----+
17  | 1 | Gamdschie | Samweis | Privatkunde |
18  | 2 | Beutlin | Frodo | Privatkunde |
19  | 3 | Beutlin | Bilbo | Privatkunde |
20  | 4 | Telcontar | Elessar | Privatkunde |
21  | 5 | Earendilionn | Elrond | Geschäftskun
22  | 6 | Eichenschild | Thorin | Unbekannt |
23  [...]
24  | 22 | Tuk | Peregrin | Unbekannt |
25  +-----+-----+-----+-----+
26  22 rows in set (0.01 sec)

```

Warum soll das einfacher sein? Es muss keine temporäre Tabelle angelegt werden. Auch der separate INSERT INTO entfällt. Die Unterabfrage in den [Zeilen 4 bis 10](#) erzeugt durch UNION eine *virtuelle Tabelle*, die durch die Vergabe der Spaltenalias auch spaltenweise angesprochen werden kann. Der INNER JOIN verwendet hier kein Primär-Fremdschlüsselpaar, sondern die Textgleichheit in der Spalte art.

Sie glauben, das war's? Nö:

```

1  mysql> SELECT kunde_id, nachname, vorname,
2      -> CASE art
3      -> WHEN 'prv' THEN 'Privatkunde'
4      -> WHEN 'gsch' THEN 'Geschäftskunde'
5      -> ELSE 'Unbekannt'

```

```

6      -> END AS k_art
7      -> FROM
8      -> kunde
9      -> ;
10  +-----+-----+-----+-----+
-----+
11  | kunde_id | nachname | vorname | k_art |
12  +-----+-----+-----+-----+
-----+
13  | 1 | Gamdschie | Samweis | Privatkunde |
14  | 2 | Beutlin | Frodo | Privatkunde |
15  | 3 | Beutlin | Bilbo | Privatkunde |
16  | 4 | Telcontar | Elessar | Privatkunde |
17  | 5 | Earendilionn | Elrond | Geschäftskun
de |
18  | 6 | Eichenschild | Thorin | Unbekannt |
19  [...]
20  | 22 | Tuk | Peregrin | Unbekannt |
21  +-----+-----+-----+-----+
-----+
22  22 rows in set (0.01 sec)

```

In [Zeile 2](#) beginnt eine Fallunterscheidung. Der Inhalt der Spalte `art` wird untersucht, indem er mit einer Textkonstanten verglichen wird. Trifft der Vergleich zu, wird dieser Teil ausgeführt. Trifft keine der Bedingungen zu, wird der ELSE in [Zeile 5](#) ausgeführt. Es ist guter Programmierstil, immer einen ELSE zu haben, damit die Abfrage robust gegenüber unbekannten Situationen ist.

15.2 Einfacher CASE



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

CASE {*spaltenname*|*ausdruck*}

```
WHEN {spaltenname|ausdruck} THEN anweisungen
[WHEN {spaltenname|ausdruck} THEN anweisungen]*
[ELSE anweisungen]
END
```

Der {*spaltenname|ausdruck*} hinter dem CASE wird immer auf Gleichheit mit dem {*spalten-name|ausdruck*} hinter dem WHEN getestet. Unser einführendes Beispiel ist somit ein einfacher CASE. Die Spalte *art* wird auf Gleichheit mit der Textkonstanten getestet.

Die Ausgabe der Währungen soll durch das Währungssymbol ersetzt werden:

```
1  mysql> SELECT artikel_id, bezeichnung,
2          -> CASE waehrung
3          -> WHEN 'EUR' THEN LPAD(CONCAT(FORMAT(
einzelpreis, 2, 'de_DE'), ' €'), 7, ' ')
4          -> WHEN 'USD' THEN LPAD(CONCAT(FORMAT(
einzelpreis, 2, 'de_DE'), ' $'), 7, ' ')
5          -> ELSE '?????'
6          -> END AS preis
7          -> FROM artikel
8          -> ;
9  +-----+-----+-----+
10 | artikel_id | bezeichnung | preis |
11 +-----+-----+-----+
12 | 3001 | Papier (100) | 2,32 € |
13 | 3005 | Tinte (gold) | 56,26 € |
14 | 3006 | Tinte (rot) | 6,26 € |
15 | 3007 | Tinte (blau) | 4,17 € |
16 | 3010 | Feder | 5,05 € |
17 | 7856 | Silberzwiebel | 0,52 € |
18 | 7863 | Tulpenzwiebel | 3,42 € |
```

```

19 | 9010 | Schaufel | 15,10 $ |
20 | 9015 | Spaten | 20,10 € |
21 +-----+-----+-----+

```

Die Funktion `FORMAT()` bastelt mir den Preis ein wenig nach meinem Geschmack um. Der erste Parameter ist die zu formatierende Zahl. Der zweite Parameter gibt die Anzahl der Nachkommastellen an, und der dritte optionale Parameter formatiert Länderspezifisches, und zwar hier das Komma anstelle des Punkts. Mit `LPAD()` Sorge ich mit von links aufgefüllten Leerzeichen für eine maximal sieben Stellen lange rechtsbündige Ausgabe.

Auch bei diesem Beispiel habe ich ein `ELSE` eingebaut. Was würde eigentlich passieren, wenn es kein solches `ELSE` gäbe, aber kein passender Eintrag gefunden wird? Probieren wir es aus:

```

1  mysql> UPDATE artikel SET waehrung = 'XYZ'
  WHERE artikel_id = 3001;
2
3  mysql> SELECT artikel_id, bezeichnung,
4             -> CASE waehrung
5             -> WHEN 'EUR' THEN LPAD(CONCAT(FORMAT(
einzelpreis, 2, 'de_DE'), ' €'), 7, ' ')
6             -> WHEN 'USD' THEN LPAD(CONCAT(FORMAT(
einzelpreis, 2, 'de_DE'), ' $'), 7, ' ')
7             -> END AS preis
8             -> FROM artikel ;
9  +-----+-----+-----+
-+
10 | artikel_id | bezeichnung | preis |
11 +-----+-----+-----+
-+
12 | 3001 | Papier (100) | NULL |
13 [...]
14 +-----+-----+-----+

```

–+

Was hätte auch sonst dort stehen sollen? Einzige Alternative wäre eine Fehlermeldung gewesen. Das Problem ist aber nun, dass es nicht mehr unterscheidbar ist, ob hier schon vorher NULL stand oder nicht. Es ist deshalb immer besser, über das ELSE ein unmissverständliches Signal abzusetzen: *Hier stimmt etwas nicht!*

15.3 Searched CASE



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

CASE

WHEN *bedingung* THEN *anweisungen*

[WHEN *bedingung* THEN *anweisungen*]*

[ELSE *anweisungen*]

END

Der Unterschied in der Syntax ist nicht so groß, aber die dadurch entstehenden Möglichkeiten. Hinter dem CASE steht nichts. Stattdessen ist die gesamte Abfrage in die *bedingung* hinter dem WHEN verlagert worden. Die Bedingung kann nun beliebig kompliziert sein. Sie kann sogar eigene Unterabfragen enthalten.

Aber erst einmal langsam; wir fangen mit einem einfachen Beispiel an. Wir wollen wissen, ob ein Eintrag in der Tabelle *beitrag* eine Antwort ist oder nicht.

```
1 mysql> SELECT beitrags_id,
```



```

2      -> CASE
3      -> WHEN bezug_beitrag_id > 1 THEN 'Ant
wort'
4      -> WHEN bezug_beitrag_id <= 1 THEN 'Ke
ine Antwort'
5      -> ELSE '??????'
6      -> END AS Typ
7      -> FROM beitrag
8      -> ORDER BY beitrag_id
9      -> ;
10  +-----+-----+
11  | beitrag_id | Typ |
12  +-----+-----+
13  | 1 | Keine Antwort |
14  | 2 | Keine Antwort |
15  | 3 | Antwort |
16  | 4 | Antwort |
17  | 5 | Antwort |
18  | 6 | Keine Antwort |
19  +-----+-----+

```

Hinter dem CASE fehlt ein Spaltenname oder ein Ausdruck ([Zeile 2](#)). Hinter dem WHEN wird als Ersatz für den Test auf Gleichheit eine komplette Bedingung angegeben ([Zeile 3](#)).

Jetzt wollen wir das Ganze etwas erweitern. Wir wollen die Nachrichten sehen, und falls es sich um eine Antwort handelt, soll der referenzierte Beitrag angegeben werden. Eine Sonderstellung hat der Beitrag 1. Er dient als Dummy-Referenz für Beiträge, die keine Antworten sind.

```

1  mysql> SELECT beitrag_id,
2      -> CASE
3      -> WHEN bezug_beitrag_id > 1
4      -> THEN CONCAT(nachricht, ' Antwort au
f ', bezug_beitrag_id, ': >',
5      -> (
6      -> SELECT nachricht

```

```

7      -> FROM beitrags b2
8      -> WHERE b2.beitrags_id = b1.bezug_beit
rag_id
9      -> )
10     -> ) -- Ende CONCAT
11     -> WHEN bezug_beitrags_id <= 1 THEN nac
hricht
12     -> ELSE '???????'
13     -> END AS Inhalt
14     -> FROM beitrags b1
15     -> WHERE beitrags_id > 1
16     -> ;
17  +-----+-----+
-----+
18  | beitrags_id | Inhalt |
19  +-----+-----+
-----+
20  | 2 | Der Lieferservice ist super. |
21  | 3 | Das finde ich auch. Antwort auf 2: >
Der Lieferservice ist super. |
22  | 4 | Aber ein wenig langsam. Antwort auf
2: >Der Lieferservice ist[...] |
23  | 5 | Finde ich nicht. Antwort auf 4: >Abe
r ein wenig langsam. |
24  | 6 | Angebot könnte besser sein. |
25  +-----+-----+
-----+

```

In [Zeile 15](#) wird der Beitrag 1 ausgefiltert. Der erste Fall in [Zeile 4](#) tritt ein, wenn der Datensatz eine Antwort ist. In der entsprechenden Unterabfrage wird die Nachricht ermittelt, auf die geantwortet wurde. Der zweite Fall in [Zeile 11](#) gibt mir nur den Nachrichtentext der aktuellen Nachricht aus, da er keine Antwort auf irgendeinen anderen Beitrag ist.

Dadurch, dass jeder Fall seine eigene Bedingung hat, kann es vorkommen, dass für mehr als ein WHEN die Bedingung wahr wird; so könnte dort die gleiche Bedingung zweimal

vorkommen. Stellt sich doch die Frage: Werden dann beide Fälle ausgeführt oder nur der erste?

```
1  SELECT beitrags_id,  
2      CASE  
3      WHEN bezug_bei trags_id >= 0 THEN 'Antwort  
,  
4      WHEN bezug_bei trags_id >= 1 THEN 'Keine A  
ntwort'  
5      ELSE '??????'  
6      END AS Typ  
7      FROM beitrags  
8      ORDER BY beitrags_id  
9  ;
```



Aufgabe 15.1: Überlegen Sie sich vorher, was für eine Ausgabe Sie erwarten. Wie sieht die Ausgabe aus, wenn nach der ersten Bedingung, die TRUE ist, die Fallunterscheidung abbricht, und wie sieht die Ausgabe aus, wenn jede Bedingung geprüft wird?

Der SELECT erzeugt folgende Ausgabe:

```
1  +-----+-----+  
2  | beitrags_id | Typ |  
3  +-----+-----+  
4  | 1 | Antwort |  
5  | 2 | Antwort |  
6  | 3 | Antwort |  
7  | 4 | Antwort |  
8  | 5 | Antwort |  
9  | 6 | Antwort |  
10 +-----+-----+
```

Bitte denken Sie darüber nach, welche Aufgabe der CASE hat. Der Inhalt *einer* Spaltenausgabe soll festgelegt werden. Würde *er jede* Bedingung, die wahr ist, ausführen, hätte er mehr als einen Inhalt für eine Spalte. Das ist nicht nur fachlich bedenklich, sondern auch nicht in Tabellenform darstellbar.



Hinweis: Die Fallunterscheidung bricht mit der ersten Bedingung ab, die das Ergebnis TRUE hat.

15.4 Fallbeispiele

15.4.1 Lagerbestand überprüfen

Für jeden Artikel soll ausgegeben werden, ob noch genügend davon im Lager vorhanden ist. Zuerst ein Blick ins Lager:

```
1  mysql> SELECT * FROM lagerbestand;
2  +-----+-----+-----+-----+
---+-----+
3  | artikel_id | menge_mindest | menge_aktue
ll | deleted |
4  +-----+-----+-----+-----+
---+-----+
5  | 3001 | 1000.000000 | 5000.000000 | 0 |
6  | 3005 | 200.000000 | 250.000000 | 0 |
7  | 3006 | 200.000000 | 250.000000 | 0 |
8  | 3007 | 200.000000 | 149.000000 | 0 |
9  | 3010 | 100.000000 | 3.000000 | 0 |
10 | 7856 | 100.000000 | 500.000000 | 0 |
11 | 7863 | 100.000000 | 400.000000 | 0 |
12 | 9010 | 10.000000 | 30.000000 | 0 |
```

```

13 | 9015 | 10.000000 | 25.000000 | 0 |
14 +-----+-----+-----+-----+
---+-----+

```

Die Artikelbezeichnung werden wir über einen INNER JOIN zur Tabelle artikel ermitteln müssen. Ob der Lagerbestand ausreicht, kann anhand der Mengenspalten ermittelt werden. Ist die aktuelle Menge kleiner oder gleich der Mindestmenge, dann muss der Artikel nachbestellt werden, ansonsten nicht. Die Sortierung hat sich am Bedarf orientiert. Der Artikel, der die Mindestmenge am weitesten unterschreitet, steht oben.

```

1  mysql> SELECT a.bezeichnung,
2          -> CASE
3          -> WHEN l.menge_aktuell <= l.menge_min
dest
4          -> THEN 'Artikel nachbestellen'
5          -> ELSE 'Bestand ausreichend'
6          -> END AS lagerstand
7          -> FROM lagerbestand l INNER JOIN arti
kel a USING(artikel_id)
8          -> ORDER BY l.menge_aktuell / l.menge_
mindest
9          -> ;
10 +-----+-----+-----+-----+
11 | bezeichnung | lagerstand |
12 +-----+-----+-----+-----+
13 | Feder | Artikel nachbestellen |
14 | Tinte (blau) | Artikel nachbestellen |
15 | Tinte (gold) | Bestand ausreichend |
16 | Tinte (rot) | Bestand ausreichend |
17 | Spaten | Bestand ausreichend |
18 | Schaufel | Bestand ausreichend |
19 | Tulpenzwiebel | Bestand ausreichend |
20 | Papier (100) | Bestand ausreichend |
21 | Silberzwiebel | Bestand ausreichend |
22 +-----+-----+-----+-----+

```

15.4.2 Kundengruppen ermitteln

Wir wollen unseren Kunden unterschiedliche Rabatte einräumen. Wer weniger als zwei Rechnungen hat, gilt als *Kleinkunde*, wer drei hat, als *Guter Kunde* und wer mehr als drei hat, als *Premiumkunde*.

```
1  mysql> SELECT DISTINCT k.kunde_id, k.nachname, k.vorname,
2      -> CASE
3      -> WHEN 3 <
4      -> (
5      -> SELECT COUNT(rechnung_id) FROM rechnung r
6      -> WHERE r.kunde_id = k.kunde_id
7      -> ) THEN 'Premiumkunde'
8      -> WHEN 2 <
9      -> (
10     -> SELECT COUNT(rechnung_id) FROM rechnung r
11     -> WHERE r.kunde_id = k.kunde_id
12     -> ) THEN 'Guter Kunde'
13     -> ELSE 'Kleinkunde'
14     -> END kundenart
15     -> FROM
16     -> rechnung r RIGHT OUTER JOIN kunde k
17     -> USING(kunde_id)
18     -> ;
19  +-----+-----+-----+-----+
20  | kunde_id | nachname | vorname | kundenart |
21  +-----+-----+-----+-----+
22  | 3 | Beutlin | Bilbo | Kleinkunde |
23  | 17 | Brandybock | Saradoc | Kleinkunde |
```

```

24  | 5 | Earendillion | Elrond | Guter Kunde
    |
25  | 6 | Eichenschild | Thorin | Kleinkunde |
26  | 1 | Gamdschie   | Samweis | Premiumkunde |
27  | 10 | Goldwert    | Hanna  | Kleinkunde |
28  | 16 | Guld        | Menegilda | Kleinkunde |
29  [...]
30  | 22 | Tuk         | Peregrin | Kleinkunde |
31  +-----+-----+-----+-----+
    -----+

```



Aufgabe 15.2: Sind die beiden Unterabfragen in den [Zeilen 4](#) und [9](#) korrelierende oder nicht korrelierende Unterabfragen?

Die Unterabfragen ermitteln die Anzahl der Rechnungen, die ein Kunde hat. Da es sich um eine skalare Unterabfrage handelt, kann das Ergebnis direkt mit einer Zahl verglichen werden. Falls keine der beiden Bedingungen zutrifft, muss es sich um einen Kleinkunden handeln.



Aufgabe 15.3: Warum wird hier ein RIGHT OUTER JOIN verwendet? Was will ich damit erreichen?

Wenn ich so recht darüber nachdenke, ist eine Kategorisierung nach den Rechnungsbeträgen sinnvoller, als nach der Anzahl der Rechnungen zu gehen.

```

1  mysql> SELECT k.kunde_id, k.nachname, k.vorname,
2             -> (
3             -> SELECT SUM(a.einzelpreis * rp.menge

```

```

)
4      -> FROM rechnung r INNER JOIN rechnung
_position rp USING(rechnung_id)
5      -> INNER JOIN artikel a USING(artikel_
id)
6      -> WHERE r.kunde_id = k.kunde_id
7      -> ) rechnungswert,
8      -> CASE
9      -> WHEN
10     -> kunde_id NOT IN (SELECT kunde_id FR
OM rechnung)
11     -> THEN 'Kleinkunde'
12     -> WHEN
13     -> (
14     -> SELECT SUM(a.einzelpreis * rp.menge
) rechnungswert
15     -> FROM rechnung r INNER JOIN rechnung
_position rp USING(rechnung_id)
16     -> INNER JOIN artikel a USING(artikel_
id)
17     -> WHERE r.kunde_id = k.kunde_id
18     -> ) BETWEEN 0 AND 300 THEN 'Kleinkund
e'
19     -> WHEN
20     -> (
21     -> SELECT SUM(a.einzelpreis * rp.menge
) rechnungswert
22     -> FROM rechnung r INNER JOIN rechnung
_position rp USING(rechnung_id)
23     -> INNER JOIN artikel a USING(artikel_
id)
24     -> WHERE r.kunde_id = k.kunde_id
25     -> ) BETWEEN 300 AND 1000 THEN 'Guter
Kunde'
26     -> ELSE 'Prämiumkunde'
27     -> END AS kundenart
28     -> FROM kunde k
29     -> ORDER BY

```



```

30      -> (
31      -> SELECT SUM(a.einzelpreis * rp.menge
32      -> ) rechnungswert
33      -> FROM rechnung r INNER JOIN rechnung
34      -> _position rp USING(rechnung_id)
35      -> INNER JOIN artikel a USING(artikel_
36      -> id)
37      -> WHERE r.kunde_id = k.kunde_id
38      -> ) DESC
39      -> ;

```

kunde_id	nachname	vorname	rechnungswert	kundenart
5	Earendillion	Elrond	1212.55000000000000	Prämiumkunde
1	Gamdschie	Samweis	761.11000000000000	Guter Kunde
3	Beutlin	Bilbo	115.85000000000000	Kleinkunde
2	Beutlin	Frodo	80.70000000000000	Kleinkunde
4	Telcontar	Elessar	NULL	Kleinkunde
[...]				
22	Tuk	Peregrin	NULL	Kleinkunde

Spätestens jetzt fällt einem auf, dass die Unterabfragen zur Ermittlung der Umsätze immer die gleichen sind ([Zeilen 3, 14, 21](#) und [31](#)). Das war zwar auch schon bei der ersten Variante so, fiel aber nicht ins Gewicht¹. Eine sinnvolle Maßnahme wäre hier das Vorschalten einer temporären Tabelle.

```

1  mysql> CREATE TEMPORARY TABLE tmp_umsatz
2      -> SELECT kunde_id, SUM(a.einzelpreis
* rp.menge) rechnungswert
3      -> FROM rechnung r INNER JOIN rechnung
_rechnung rp USING(rechnung_id)
4      -> INNER JOIN artikel a USING(artikel_
id)
5      -> GROUP BY kunde_id
6      -> ;
7
8  mysql> SELECT k.kunde_id, k.nachname, k.vo
rname, tmp.rechnungswert,
9      -> CASE
10     -> WHEN
11     -> tmp.kunde_id IS NULL THEN 'Kleinkun
de'
12     -> WHEN
13     -> rechnungswert BETWEEN 0 AND 300 THE
N 'Kleinkunde'
14     -> WHEN
15     -> rechnungswert BETWEEN 300 AND 1000
THEN 'Guter Kunde'
16     -> ELSE 'Prämiumkunde'
17     -> END AS kundenart
18     -> FROM kunde k LEFT JOIN tmp_umsatz t
mp USING(kunde_id)
19     -> ORDER BY rechnungswert DESC
20     -> ;
21  +-----+-----+-----+-----+
-----+
22  | kunde_id | nachname | vorname | rechnung
swert | kundenart |
23  +-----+-----+-----+-----+
-----+
24  | 5 | Earendilionn | Elrond | 1212.5500000
00000 | Prämiumkunde |
25  | 1 | Gamdschie | Samweis | 761.1100000000
00 | Guter Kunde |

```

```

26 | 3 | Beutlin | Bilbo | 115.8500000000000 |
   |   | Kleinkunde |
27 | 2 | Beutlin | Frodo | 80.7000000000000 |
   |   | Kleinkunde |
28 | 7 | Brandybock | Gormadoc | NULL | Klein
   |   | kunde |
29 [...]
30 | 18 | Tuk | Esmeralda | NULL | Kleinkunde
   |
31 +-----+-----+-----+-----+
   +-----+-----+-----+

```

In [Zeile 1](#) wird eine temporäre Tabelle aus einem SELECT heraus definiert. Diese Technik haben wir jetzt schon einige Male verwendet, und sie sollte mittlerweile zum Handwerkszeug dazu gehören. Der Inhalt der temporären Tabelle tmp_umsatz entspricht genau den oben mehrfach ausgeführten Unterabfragen; sie wird lediglich um die kunde_id erweitert, damit man die temporäre Tabelle verknüpfen kann.

Jetzt wird die Auswertung durchgeführt, indem zuerst die Kundentabelle mit der temporären Tabelle verknüpft wird ([Zeile 18](#)). Da der Rechnungswert pro Kunde nun schon vorliegt, muss dieser nicht n -fach berechnet werden, sondern kann durch eine Verknüpfung sofort in den Bedingungen ([Zeilen 10](#) und [12](#)), der Sortieranweisung ([Zeile 19](#)) und der Spaltenausgabe ([Zeile 8](#)) verwendet werden.



Aufgabe 15.4: Warum muss in [Zeile 18](#) ein LEFT OUTER JOIN verwendet werden? Welcher Datensatz würde in der Ausgabe bei einem INNER JOIN fehlen?

Hier hat sich die Anlage einer temporären Tabelle wirklich

hier hat sich die Anlage einer temporären Tabelle wirklich gelohnt, aber ... können Sie sich noch an CTEs in [Abschnitt 11.5](#) erinnern? Es geht noch einfacher:

```
1  WITH tmp_umsatz (kunde_id, rechnungswert)
2  AS (
3  SELECT kunde_id, SUM(a.einzelpreis * rp.me
nge) rechnungswert
4  FROM rechnung r INNER JOIN rechnung_positi
on rp USING(rechnung_id)
5  INNER JOIN artikel a USING(artikel_id)
6  GROUP BY kunde_id
7  )
8  SELECT k.kunde_id, k.nachname, k.vorname,
tmp.rechnungswert,
9      CASE
10     WHEN tmp.kunde_id IS NULL THEN 'Kleinkun
de'
11     WHEN rechnungswert BETWEEN 0 AND 300 THE
N 'Kleinkunde'
12     WHEN rechnungswert BETWEEN 300 AND 1000
THEN 'Guter Kunde'
13     ELSE 'Premiumkunde'
14     END AS kundenart
15     FROM kunde k LEFT JOIN tmp_umsatz tmp US
ING(kunde_id)
16     ORDER BY rechnungswert DESC
17  ;
```

Diese Lösung ist ein gutes Beispiel für den Einsatz von CTEs. Man spart sich das explizite Erstellen einer temporären Tabelle und kann trotzdem auf dynamisch ermittelte Inhalte mehrfach zugreifen.

15.4.3 Aktive Lieferanten ermitteln

Wir wollen wissen, welche Lieferanten noch aktiv sind und welche nicht. Dazu reicht es aus nachzuschauen, ob der Primärschlüssel des Lieferanten in `artikel_nm_lieferant` vorkommt. Es ist irrelevant, ob der Primärschlüsselwert mehrfach vorkommt, da hier nur zwischen aktiv und inaktiv unterschieden werden soll.

```
1  mysql> SELECT l.firmenname,
2          -> CASE
3          -> WHEN
4          -> l.lieferant_id IN
5          -> (SELECT lieferant_id FROM artikel_n
m_lieferant) THEN 'aktiv'
6          -> ELSE 'inaktiv'
7          -> END AS status
8          -> FROM lieferant l
9          -> ORDER BY firmenname;
10 +-----+-----+
11 | firmenname | status |
12 +-----+-----+
13 | Bürohengst GmbH | aktiv |
14 | Gartenbedarf AllesGrün | aktiv |
15 | Office International | inaktiv |
16 +-----+-----+
```

15.4.4 Aufgaben



Aufgabe 15.5: Geben Sie für alle Zeilen in der Tabelle `account` klartextlich an, ob es sich um einen Administrator handelt oder nicht.

Aufgabe 15.6: Geben Sie für alle Zeilen der Tabelle `adresse` klartextlich an, in welchem Land die Adresse

liegt: AL = Auenland, DE = Deutschland, ER = Eriador, GO = Gondor, MO = Mordor.

Aufgabe 15.7: Es sollen drei Zustände ermittelt werden:

1. Die Mindestmenge ist erreicht oder unterschritten: Artikel sofort nachbestellen.
2. Die aktuelle Menge ist 130 % oder weniger als die Mindestmenge: Artikel bald nachbestellen.
3. Sonst: Bestand ausreichend.

Vergleichen Sie Ihr Ergebnis mit meinem:

```
+-----+-----+
-----+
| bezeichnung | lagerstand |
+-----+-----+
-----+
| Feder | Artikel sofort nachbestellen |
| Tinte (blau) | Artikel sofort nachbestel
len |
| Tinte (gold) | Artikel bald nachbestelle
n |
| Tinte (rot) | Artikel bald nachbestellen
|
| Spaten | Bestand ausreichend |
| Schaufel | Bestand ausreichend |
| Tulpenzwiebel | Bestand ausreichend |
| Papier (100) | Bestand ausreichend |
| Silberzwiebel | Bestand ausreichend |
+-----+-----+
-----+
```

Aufgabe 15.8: Geben Sie eine Namensliste für Kunden aus, die um die Angabe erweitert wird, ob ein Inkassoverfahren anhängig ist.

Aufgabe 15.9: Geben Sie eine Namensliste für Kunde aus, die angibt, ob er im Forum angemeldet ist.

Aufgabe 15.10: Teilen Sie die Warengruppen danach ein, wie oft diese in Rechnungen vorkommen. Es soll drei Gruppen geben: *oft*, *selten* und *nie*. Ermitteln Sie selbst anhand einer Auswertung mögliche Abgrenzungen.

Aufgabe 15.11: Geben Sie für alle Artikel den Mindestbestand im Lager aus. Falls es diesen Artikel noch nicht im Lager gibt, soll der Text *Keine Lagerdaten vorhanden* ausgegeben werden.

¹ Ich meine hier natürlich syntaktisch. Vom Laufzeitverhalten will ich erst gar nicht sprechen, werden doch die korrelierenden Unterabfragen n -fach wiederholt.

16 Ansichtssache



Frei nach Karl Marx: Die Ansicht bestimmt das Bewusstsein.

- Grundkurs
 - Anlegen einer Ansicht mit `CREATE VIEW`
 - Löschen einer Ansicht mit `DROP VIEW`
 - Ändern einer Ansicht mit `ALTER VIEW`
 - Unterschiede zwischen Tabelle und Ansicht
- Vertiefendes
 - Verarbeitungsstrategien einer Ansicht
 - Ändern einer Ansicht mit `CREATE OR REPLACE`
 - Selektions- und Verbundansicht

Die Quelltexte dieses Kapitels stehen in der Datei `listing13.sql` (siehe [Listing 28.13](#), [Listing 28.53](#) und [Listing 28.38](#)).

16.1 Was ist eine Ansicht?

Wir haben in den vorangegangenen Kapiteln immer wieder temporäre Tabellen eingesetzt. Manchmal, weil es nicht anders ging, manchmal, um die Komplexität der Auswahl zu verringern, und manchmal, um die Performance zu steigern. Temporäre Tabellen haben die Daten einer Auswahl abgespeichert und anderen Auswertungen zur Verfügung gestellt.

Ein Nachteil temporärer Tabellen ist, dass sie nur für diese eine Verbindung existieren und danach gelöscht werden. Ein weiterer Nachteil ist: Die Daten können veralten. Eine Analyse muss vorab ergeben, wie lange die Daten einer temporären Tabelle seriöserweise verwendet werden können. Handelt es sich um Bewegungsdaten, ist die Verwendbarkeit einer temporären Tabelle vielleicht schon nach kurzer Zeit nicht mehr gegeben. Eine Ansicht (VIEW) hingegen speichert nicht die Daten einer Auswahl, sondern die Konstruktionsvorschrift einer Auswahl. Ruft man die Ansicht ab, wird die Konstruktionsvorschrift so aufgerufen, als ob diese direkt eingegeben wurde (*Na ja, nicht ganz, aber dazu später mehr ...*).

16.1.1 Wie lege ich eine Ansicht an?

 SQL:2023	MySQL/MariaDB PostgreSQL	T-SQL
CREATE	CREATE [OR REPLACE]	CREATE
VIEW	VIEW	VIEW
<i>viewname[(spaltenliste)]</i>	<i>viewname[(spaltenliste)]</i>	<i>viewname[(spaltenliste)]</i>
AS	AS	AS

```
SELECT auswahl
```

```
;
```

```
SELECT auswahl
```

```
;
```

```
SELEC
```

```
;
```

Wir wollen beispielsweise eine Adressliste für alle Kunden als Ansicht zur Verfügung stellen.

```
1  mysql> CREATE VIEW
2      -> view_kundenrechnungsadresse (id, na
chname, vorname, strasse, ort)
3      -> AS
4      -> SELECT
5      -> kunde_id, nachname, vorname,
6      -> CONCAT_WS(' ', strasse, hnr),
7      -> CONCAT_WS(' ', lkz, plz, ort)
8      -> FROM
9      -> kunde INNER JOIN adresse ON rechnun
g_adresse_id = adresse_id
10     -> WHERE kunde.deleted = 0
11     -> ;
12
13 mysql> SELECT * FROM view_kundenrechnungsad
resse;
14  +-----+-----+-----+-----+
-----+-----+
15  | id | nachname | vorname | strasse | ort
|
16  +-----+-----+-----+-----+
-----+-----+
17  | 1 | Gamdschie | Samweis | Beutelhaldenwe
g 5 | AL 67676 Hobbingen |
18  | 2 | Beutlin | Frodo | Beutelhaldenweg 1
| AL 67676 Hobbingen |
19  [...]
20  | 21 | Lochner | Rufus | Maggots Weg 2 | B
L 57980 Maggots Hof |
```

```

21  | 22 | Tuk | Peregrin | Stockstraße 1 | AL
    45689 Buckelstadt |
22  +-----+-----+-----+-----+-----+-----+
    -----+-----+-----+-----+
23  21 rows in set (0.00 sec)

```

Ab der [Zeile 3](#) wird angegeben, was für ein SELECT ausgeführt wird, wenn man diese Ansicht abfragt. Der SELECT der Ansicht ist eine eigenständig ausführbare Anweisung. Er kann genauso einfach oder kompliziert sein wie jeder andere SELECT; er kann sogar wiederum auf Ansichten aufbauen. Die Ansicht kapselt den SELECT, um dann für weitere Auswertungen zur Verfügung zu stehen.

In der [Zeile 2](#) steht hinter dem Schlüsselwort VIEW der *viewname* und anschließend eine *spaltenliste*. Diese Spaltenliste ist nur eine Umbenennung der Spalten aus dem SELECT. Natürlich können hier auch Ausdrücke stehen, die die Spalten aufbereiten, aber meistens stehen hier nur Umbenennungen. In unserem Fall ist das nötig, da wir sonst hässliche Spaltennamen wie CONCAT_WS(' ', strasse, hnr) hätten.



Aufgabe 16.1: Legen Sie eine zweite VIEW namens `view_wurstbrot` an. Lassen Sie dabei einfach die Spaltenliste weg und führen Sie dann `SELECT * FROM view_wurstbrot` aus.

In [Zeile 10](#) werden alle als gelöscht markierten Zeilen aussortiert. In unserem Fall ist das Ergebnis das gleiche wie ohne die WHERE-Klausel. Wir erkennen aber hier schon eine wesentliche Anwendung der VIEW. Werden für die Tabellen und deren Standardauswertungen Ansichten erstellt, die

immer mit einer WHERE-Klausel auf die `deleted`-Spalten ausgestattet sind, können Sie sich in den weiteren Befehlen diese Abfrage sparen.



Hinweis: Der Name einer Ansicht sollte einer Namenskonvention folgend immer mit dem Präfix `view_` beginnen.

In MySQL und MariaDB gibt es kein zu `SHOW TABLES` analoges `SHOW VIEWS`. Vielmehr werden die Ansichten fast immer wie Tabellen behandelt. Damit Sie die Ansichten von den Tabellen unterscheiden können, ist eine solche Namenskonvention sehr sinnvoll. Ein weiterer Grund ist, dass Sie mit einer Ansicht eben *nicht* alles wie mit einer Tabelle machen können. Um es erst gar nicht zu versuchen, sollte schon am Namen erkennbar sein, dass es sich um eine Ansicht handelt. Selbst bei Systemen, die ein Analogon zu `SHOW VIEWS` haben, wie beispielsweise PostgreSQL mit `\dv`, empfiehlt sich die Namenskonvention, damit Sie in einem `SELECT` unterscheiden können, ob die Datenquelle eine Tabelle oder eine Ansicht ist.



Hinweis: MySQL und MariaDB haben kein `SHOW VIEWS`, aber Sie können die Ansichten einer Datenbank wie folgt ermitteln: `SHOW FULL TABLES IN datenbankname WHERE TABLE_TYPE = 'VIEW';`

```
1  mysql> SHOW FULL TABLES IN oshop WHERE
2  TABLE_TYPE = 'VIEW';
3  +-----+
4  | Tables_in_oshop | Table_type |
5  +-----+
```

```

5 | view_artikel_aktiv | VIEW |
6 | view_artikel_verfuegbar | VIEW |
7 | view_kundenrechnungsadresse | VIEW |
8 | view_lagerbestand_aktiv | VIEW |
9 | view_lagerbestand_artikelbezeichnung
  | VIEW |
10 | view_tmp | VIEW |
11 | view_tmp_eins | VIEW |
12 | view_tmp_zwei | VIEW |
13 +-----+
--+-----+
14 8 rows in set (0.00 sec)

```

Folgt man der Namenskonvention oben, so stehen diese bei einem `SHOW TABLES` wenigstens untereinander:

```

1 mysql> SHOW TABLES;
2 +-----+
3 | Tables_in_oshop |
4 +-----+
5 | account |
6 [...]
7 | rechnung_position |
8 | view_kundenrechnungsadresse |
9 | warengruppe |
10 +-----+

```

16.1.2 Wie wird eine Ansicht verarbeitet?

InMySQL/MariaDB können Sie die Art undWeise, wie eine Ansicht intern verarbeitet wird, festlegen oder festlegen lassen. Schauen wir uns dazu eine erweiterte Version des Befehls an.



MySQL/MariaDB

```
CREATE [OR REPLACE]
[ALGORITHM = {UNDEFINED|MERGE|TEMPTABLE}]
VIEW viewname[(spaltenliste)]
AS
SELECT auswahl
;
```

Die Option ALGORITHM bedeutet:

- MERGED: Die Auswahl, die hinter der Ansicht steht, wird in den verwendeten Befehl so eingebaut, als ob man ihn direkt dort hinein programmiert hätte. Dies erlaubt es dem internen Optimierer, ggf. den SELECT zu berücksichtigen.
- TEMPTABLE: Das Ergebnis der Ansicht wird in eine – nicht sichtbare – temporäre Tabelle abgespeichert, und erst dieses Ergebnis wird in der verwendenden Anweisung benutzt.
- UNDEFINED: MySQL/MariaDB entscheiden selbst darüber, welche der beiden Varianten verwendet wird. Wird keine Angabe zum Algorithmus gemacht, wird wie bei UNDEFINED verfahren.

Als Beispiel nehmen wir eine einfache Ansicht:

```
1 CREATE VIEW view_artikel_aktiv
2     AS
3     SELECT *
4     FROM artikel
5     WHERE deleted = 0;
```

Das entscheidende Kriterium für die Verwendung von MERGED ist, ob zwischen den Ergebnissen der Ansicht und dem Rest der Anweisung eine 1 : 1-Verknüpfung hergestellt werden kann. Bei Gruppenbildungen mit GROUP BY, DISTINCT oder anderen Mengenoperationen wie beispielsweise UNION geht die direkte Verknüpfung der Datensätze verloren. Mit EXPLAIN können Sie sich die Entscheidung von MySQL bzw. MariaDB anzeigen lassen:

```

1  mysql> SELECT FLOOR(artikel_id / 1000) gru
ppe, AVG(einzelpreis)
2      -> FROM view_artikel_aktiv
3      -> GROUP BY gruppe
4      -> ;
5  +-----+-----+
6  | gruppe | AVG(einzelpreis) |
7  +-----+-----+
8  | 3      | 14.8120000000    |
9  | 7      | 1.9700000000     |
10 | 9      | 17.6000000000    |
11 +-----+-----+

```

In diesem Beispiel werden die Durchschnittspreise nach Warengruppen gruppiert ermittelt.

```

1  mysql> EXPLAIN SELECT wie oben\G
2  ***** 1. row *****
3  id: 1
4  select_type: SIMPLE
5  table: artikel
6  partitions: NULL
7  type: ALL
8  possible_keys: PRIMARY,idx_artikel_bezeich
nung
9  key: NULL
10 key_len: NULL

```

```
11          ref: NULL
12          rows: 9
13      filtered: 11.11
14  Extra: Using where; Using temporary
```

Unter Extra wird angegeben, dass eine temporäre Tabelle verwendet wurde. Als Alternative schauen wir uns jetzt eine Einschränkung der Datensätze auf einen bestimmten Nummernbereich an:

```
1  mysql> SELECT artikel_id, bezeichnung
2      -> FROM
3      -> view_artikel_aktiv
4      -> WHERE
5      -> artikel_id BETWEEN 7000 AND 9000
6      -> ;
7  +-----+-----+
8  | artikel_id | bezeichnung |
9  +-----+-----+
10 | 7856 | Silberzwiebel |
11 | 7863 | Tulpenzwiebel |
12 +-----+-----+
```

EXPLAIN teilt mir lediglich mit, dass hier die WHERE-Klausel verwendet wird. Die Ansicht wird somit mit der verwendenden Anweisung verschmolzen.

```
1  mysql> EXPLAIN SELECT wie oben\G
2  ***** 1. row *****
3          id: 1
4  select_type: SIMPLE
5      table: artikel
6  partitions: NULL
7      type: range
8 possible_keys: PRIMARY
9          key: PRIMARY
```



```
10      key_len: 4
11      ref: NULL
12      rows: 2
13      filtered: 11.11
14      Extra: Using where
```

Stellt sich doch die Frage, warum man etwas angeben kann, was doch MySQL oder MariaDB eigenständig entscheiden. Na, weil wir den Durchblick haben:



Hinweis: Wird das Ergebnis einer Ansicht mehrfach verwendet, sollten Sie `ALGORITHM=TEMPTABLE` setzen, da diese temporäre Tabelle im Cache gehalten wird und daher wieder verwendet werden kann.

Erzeugen wir uns die gleiche Ansicht, allerdings um die Angabe `ALGORITHM=TEMPTABLE` erweitert:

```
1  CREATE ALGORITHM=TEMPTABLE VIEW view_artikel_aktiv_tmp
2      AS
3      SELECT * FROM artikel
4      WHERE deleted = 0;
5  ;
```

Führen wir nun zweiMal einen `SELECT` mit unterschiedlicher `WHERE`-Klausel, aber der neuen Ansicht aus:

```
1  SELECT artikel_id, bezeichnung
2      FROM view_artikel_aktiv_tmp
3      WHERE artikel_id BETWEEN 7000 AND 8000
4  ;
5
6  SELECT artikel_id, bezeichnung
7      FROM view_artikel_aktiv_tmp
```

```
8 WHERE artikel_id BETWEEN 3000 AND 4000
9 ;
```

Jetzt liegt das Ergebnis der Ansicht in einer unsichtbaren temporären Tabelle im Cache. Ein EXPLAIN auf die letzte Abfrage bestätigt uns, dass dieser Cache-Inhalt verwendet wird.

```
1 mysql> EXPLAIN SELECT wie oben WHERE artik
2 el_id BETWEEN 3000 AND 4000\G
3 ***** 1. row *****
4 id: 1
5 select_type: PRIMARY
6 table: <derived2> Verweis auf die
7 abgeleitete Tabelle mit der id: 2
8 partitions: NULL
9 type: ALL
10 possible_keys: NULL
11 key: NULL
12 key_len: NULL
13 ref: NULL
14 rows: 2 ←- Schätzung! MariaDB lief
15 ert 5.
16 filtered: 100.00
17 Extra: NULL
18 ***** 2. row *****
19 id: 2
20 select_type: DERIVED
21 table: artikel
22 partitions: NULL
23 type: range
24 possible_keys: PRIMARY
25 key: PRIMARY
26 key_len: 4
27 ref: NULL
28 rows: 5
```

26 filtered: 11.11
27 Extra: Using where

Die temporäre Tabelle wird als Basis für die neue Auswertung mit der veränderten WHERE-Klausel verwendet. Ohne die Angabe ALGORITHM=TEMPTABLE würde hier der gleiche EXPLAIN ausgegeben werden wie bei der Verwendung von view_artikel_aktiv.

16.1.3 Wie lösche ich eine Ansicht?

 SQL:2023	MySQL/MariaDB PostgreSQL	T-SQL
DROP VIEW	DROP [IF EXISTS] VIEW	DROP [IF EXISTS] VIEW
<i>viewname</i> [, <i>viewname</i>]*	<i>viewname</i> [, <i>viewname</i>]* [CASCADE RESTRICT]	<i>viewname</i> [, <i>viewname</i>]*
;	;	;

```
1  mysql> SHOW TABLES;
2  +-----+
3  | Tables_in_oshop |
4  +-----+
5  | account |
6  [...]
7  | rechnung_position |
8  | view_artikel_aktiv |
```

```

 9  | view_artikel_aktiv_tmp |
10  | view_kundenrechnungsadresse |
11  | warengruppe |
12  +-----+
13  20 rows in set (0.00 sec)
14
15  mysql> DROP VIEW view_artikel_aktiv_tmp;
16
17  mysql> SHOW TABLES;
18  +-----+
19  | Tables_in_oshop |
20  +-----+
21  | account |
22  [...]
23  | rechnung_position |
24  | view_artikel_aktiv |
25  | view_kundenrechnungsadresse |
26  | warengruppe |
27  +-----+
28  19 rows in set (0.00 sec)

```

So weit, so einfach. Was passiert aber, wenn eine Ansicht Teil einer anderen Ansicht ist? Oder was passiert, wenn die Tabelle gelöscht wird, die Teil der Ansicht ist? Fragen über Fragen ...

Bauen wir uns schnell ein Beispiel für die Frage zusammen:
Was passiert, wenn ich eine Tabelle lösche, die in einer Ansicht verwendet wird?

```

1  mysql> CREATE DATABASE tmp1;
2  mysql> USE tmp1;
3  mysql> CREATE TABLE a (i INT);
4  mysql> CREATE TABLE b (i INT);
5  mysql> CREATE VIEW ab
6      -> AS
7      -> SELECT * FROM a INNER JOIN b USING(
i)
8      -> ;

```

```

9
10
11 mysql> SHOW TABLES;
12 +-----+
13 | Tables_in_tmp1 |
14 +-----+
15 | a |
16 | ab |
17 | b |
18 +-----+
19
20 mysql> DROP TABLE a;
21
22 mysql> SHOW TABLES;
23 +-----+
24 | Tables_in_tmp1 |
25 +-----+
26 | ab |
27 | b |
28 +-----+

```

Das ist überraschend! Die Ansicht ab ist nicht gelöscht worden. Der Versuch, mit dieser Ansicht zu arbeiten, schlägt allerdings fehl:

```

1 mysql> SELECT * FROM ab;
2 ERROR 1356 (HY000): View 'tmp1.ab' references invalid table(s) or column(s) or function(s) or definer/invoke
r of view lack rights to use them

```

Wenn Sie vorab klären möchten, ob eine Ansicht einsatzfähig ist, verwenden Sie CHECK TABLE:

```

1 mysql> CHECK TABLE ab\G
2 ***** 1. row *****
3
4 Table: tmp1.ab

```

```

4 Op: check
5 Msg_type: Error
6 Msg_text: View 'tmp1.ab' references invalid table(s) or column(s) or function(s) or definer/invoke
er of view lack rights to use them
7 ***** 2. row *****
*****
8 Table: tmp1.ab
9 Op: check
10 Msg_type: error
11 Msg_text: Corrupt

```

Analog sieht es in T-SQL aus:

```

1 [...]
2 1> SELECT * FROM ab;
3 2> GO
4 ai bi
5 -----
6
7 (0 rows affected)
8 1> DROP TABLE a;
9 2> GO
10 1> SELECT * FROM ab;
11 2> GO
12 Msg 208, Level 16, State 1, Server b43f45f9b05d, Procedure ab, Line 3
13 Invalid object name 'a'.
14 Msg 4413, Level 16, State 1, Server b43f45f9b05d, Line 1
15 Could not use view or function 'ab' because of binding errors.

```

In anderen Systemen werden solche Abhängigkeiten anders behandelt. In PostgreSQL beispielsweise erhält man eine Fehlermeldung:

```

1 tmp1=# \dt

```

```

2 List of relations
3 Schema | Name | Type | Owner
4 -----+-----+-----+-----
5 public | a | table | postgres
6 public | b | table | postgres
7
8 tmp1=# \dv
9 List of relations
10 Schema | Name | Type | Owner
11 -----+-----+-----+-----
12 public | ab | view | postgres
13
14
15 tmp1=# DROP TABLE a;
16 psql:listing13.sql:81: ERROR: cannot drop
table a because other objects depend on it
17 DETAIL: view ab depends on table a
18 HINT: Use DROP ... CASCADE to drop the dep
endent objects too.

```

Folgen wir dem Hinweis von PostgreSQL und erweitern den DROP TABLE:

```

1 tmp1=# DROP TABLE a CASCADE;
2 NOTICE: drop cascades to view ab
3 DROP TABLE
4
5 tmp1=# \dv
6 Did not find any relations.
7
8 tmp1=# \dt
9      List of relations
10 Schema | Name | Type | Owner
11 -----+-----+-----+-----
12 public | b | table | postgres

```

Die Ansicht verschwindet mit, da diese ein abhängiges Objekt von Tabelle a ist. In MySQL und MariaDB wird die Option

CASCADE ignoriert.



Aufgabe 16.2: Und was ist bei einer Ansicht, die auf einer Ansicht basiert? Bauen Sie einen ähnlichen Versuch auf und interpretieren Sie das Ergebnis.

Das Ergebnis lässt sich schon erahnen und führt zu folgendem Hinweis:



Hinweis: Werden in MySQL oder MariaDB Tabelle oder Ansichten, die in anderen Ansichten verwendet werden, gelöscht, erfolgt kein Weiterreichen dieser Löschoperation an die enthaltenden Ansichten.

16.1.4 Wie ändere ich eine Ansicht?

Tabellen können mit `ALTER TABLE` in fast allen Parametern verändert werden. Analoges gilt auch für die Ansicht. Bis auf den Namen¹ können alle Parameter/Optionen neu bestimmt werden. Es gibt grundsätzlich zwei Syntaxvarianten.



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

```
ALTER VIEW viewname  
    view_spezifikation ;
```

In den meisten Fällen wird aber mit dem obigen `OR REPLACE`

bzw. OR ALTER-Syntaxt des CREATE VIEW gearbeitet.

Somit wird nicht eine vorhandene Ansicht verändert – wie es bei einem ALTER TABLE der Fall ist –, sondern es wird eine neue Ansicht mit den veränderten Parametern geschaffen.



Hinweis: Ansichten sind nicht besonders wartungsstabil. Ändert sich die Tabellenstruktur einer beteiligten Tabelle (z.B. werden Spalten hinzugefügt oder gelöscht), wird dies nicht automatisch an die Ansicht weitergeleitet. Die Ansicht muss dann gelöscht und neu erstellt werden. Das Gleiche gilt für beteiligten Ansichten.

16.2 Anwendungsgebiet: Vereinfachung

Eine wichtige Vereinfachung ist schon erwähnt worden. Wir haben im Rahmen der Diskussion um die Löschweitergabe (siehe [Abschnitt 2.3.1](#) und den ersten Hinweis in [Kapitel 3](#)) die Spalte `deleted` als Löschkennzeichen eingeführt. Jetzt wäre es aber extrem nervig, bei jeder Auswertung in die WHERE-Klausel ein `AND deleted = 0` einzubauen, schließlich muss auch noch die Klammerung oder Auswertungsreihenfolge beachtet werden. Es ist somit eine gute Idee, für die Tabellen entsprechende Ansichten zu erstellen:

```
1 CREATE OR REPLACE
2 ALGORITHM = TEMPTABLE
3 VIEW view_artikel_aktiv
4 AS
5     SELECT * FROM artikel
6     WHERE deleted = 0;
```

Diese Art der Ansicht hat einen Namen:



Definition 57: Selektionsansicht

Eine Ansicht, deren Aufgabe das Herausfiltern von Zeilen ist, wird *Selektionsansicht* genannt.

Bei der Definition von `view_artikel_aktiv` drängen sich doch sofort Fragen auf:

1. *Warum ALGORITHM = TEMPTABLE?*
Die Ansicht ist eine Auswertung der Tabelle `artikel` ohne irgendeine weitere Einschränkung oder Aufbereitung. Mit anderen Worten, es ist recht wahrscheinlich, dass das Ergebnis der Ansicht relativ oft verwendet wird, und zwar jedes Mal dann, wenn auf die bereinigten Artikeldaten zugegriffen wird.
2. *Warum SELECT * ?*
Unter der Annahme, dass `deleted` nur 0 oder 1 sein kann, haben wir mit `deleted = 0` immer den gleichen Wert in dieser Spalte stehen. Wäre es da nicht besser, diese Spalte gänzlich zu unterdrücken? Dagegen sprechen zwei Argumente: Zum einen müsste jedes Mal, wenn sich die Spalten in der Tabelle `artikel` ändern, die Ansicht angepasst werden. Zum anderen können Ansichten auch in UPDATE-Anweisungen verwendet werden. Und eine solche Anwendung kann das *Löschen* sein. Da dabei der Spalte `deleted` der Wert 1 zugewiesen werden muss, sollte diese Spalte sichtbar bleiben.
3. *Ist es wirklich sinnvoll, für alle Tabellen eine solche Ansicht zu erzeugen?*
Ein Frage wie diese lässt sich nicht mit *Ja* oder *Nein*

beantworten. Es ist offen, ob diese Ansichten immer gebraucht werden, aber das Erzeugen ist auch nicht teuer, und dann hat man sie schon mal fertig, wenn man sie braucht.



Aufgabe 16.3: Erstellen Sie für die Tabellen account, adresse, artikel, bank, bankverbindung, beitrags, bestellung, bestellung_position, kunde, lagerbestand, lieferant, rechnung, rechnung_position und warengruppe die entsprechende Selektionsansicht.

Weitere Ansichten ergeben sich in der Regel aus den Anforderungen der Präsentationsschicht. Werden beispielsweise in einer Listbox alle Artikelnummern und -bezeichnungen angezeigt, die noch im Lager ausreichend vorhanden sind, so könnten Sie aus der Funktionsschicht einen solchen SELECT absetzen. Sie hätte dann in der Funktionsschicht einen etwas komplizierteren SELECT, den Sie dann auch immer dort warten müsste. Anders, wenn eine VIEW vorhanden ist:

```
1  mysql> CREATE VIEW view_artikel_verfügbar
2      -> AS
3      -> SELECT artikel_id, bezeichnung
4      -> FROM artikel INNER JOIN lagerbestan
d USING(artikel_id)
5      -> WHERE menge_aktuell >= menge_mindes
t
6      -> ORDER BY artikel_id;
7
8  mysql> SELECT * FROM view_artikel_verfügba
r;
9  +-----+-----+
```

```

10 | artikel_id | bezeichnung |
11 +-----+-----+
12 | 3001 | Papier (100) |
13 | 3005 | Tinte (gold) |
14 | 3006 | Tinte (rot) |
15 | 7856 | Silberzwiebel |
16 | 7863 | Tulpenzwiebel |
17 | 9010 | Schaufel |
18 | 9015 | Spaten |
19 +-----+-----+

```

Eine weitere, oft gewünschte Vereinfachung ist, dass man die *Standardverknüpfungen* schon vorwegnimmt. Dazu gehören Kunde – Rechnungsadresse, Kunde – Lieferadresse, Kunde – Rechnung, Kunde – Bestellung, Rechnung – Positionen, Bestellung – Positionen, Bankverbindung – Bankname, Lagerbestand – Artikelname.

Fangen wir mit der letzten an:

```

1  mysql> CREATE OR REPLACE VIEW view_lagerbe
stand_artikelbezeichnung
2      -> AS
3      -> SELECT
4      -> l.artikel_id, l.menge_mindest, l.me
nge_aktuell, a.bezeichnung
5      -> FROM
6      -> view_lagerbestand_aktiv l INNER JOI
N view_artikel_aktiv a
7      -> USING(artikel_id)
8      -> ;
9
10 mysql> SELECT * FROM view_lagerbestand_art
ikelbezeichnung;
11 +-----+-----+-----+-----+
---+-----+
12 | artikel_id | menge_mindest | menge_aktue
ll | bezeichnung |

```

```

13  +-----+-----+-----+
---+-----+
14  | 3001 | 1000.000000 | 5000.000000 | Papie
r (100) |
15  | 3005 | 200.000000 | 250.000000 | Tinte (
gold) |
16  | 3006 | 200.000000 | 250.000000 | Tinte (
rot) |
17  | 3007 | 200.000000 | 149.000000 | Tinte (
blau) |
18  | 3010 | 100.000000 | 3.000000 | Feder |
19  | 7856 | 100.000000 | 500.000000 | Silberz
wiebel |
20  | 7863 | 100.000000 | 400.000000 | Tulpenz
wiebel |
21  | 9010 | 10.000000 | 30.000000 | Schaufel
|
22  | 9015 | 10.000000 | 25.000000 | Spaten |
23  +-----+-----+-----+
---+-----+

```

Auch diese Art von Ansicht hat einen Namen:



Definition 58: Verbundansicht

Eine Ansicht, deren Aufgabe die Verknüpfung von Tabellen ist, wird *Verbundansicht* genannt.



Aufgabe 16.4: Erstellen Sie die verbleibenden Verbundansichten, die wir oben aufgelistet haben.

Ich halte es für eine gute Idee, Auswertungen aus der

Funktionsschicht der Anwendung herauszunehmen und in einer Ansicht zu kapseln. In der Anwendung müssen dann nur noch Befehle wie `SELECT * FROM view_xyz WHERE ...` abgesetzt werden. Oft sind in Projekten Entwickler mit unterschiedlichen Skills eingesetzt. Ein guter Java-Programmierer muss nicht unbedingt auch ein guter SQL-Programmierer sein.

Ein gewaltiger Vorteil ist auch, dass eine Änderung der Abfrage zu dieser Auswertung – beispielsweise aus Gründen der Performance – keine Änderung im Quelltext der Funktionsschicht zur Folge hätte.

Zum Schluss noch eine kleine Aufgabe zum Selbsttest:



Aufgabe 16.5: Bei der Besprechung darüber, wie man die Ergebnisse einer Verknüpfung wiederverwenden kann, sind wir am Beispiel Bankeinzug in [Abschnitt 11.2.3](#) auf eine Hierarchie von Auswertungen gekommen. Bauen Sie diese Hierarchie mithilfe von Ansichten nach. Überprüfen Sie, ob Sie das gleiche Ergebnis bekommen.

16.3 Anwendungsgebiet: Datenschutz

In der Literatur finden Sie oft einen Hinweis darauf, dass man mithilfe der Ansicht Daten vor den Anwendern verbergen kann. Soll beispielsweise die Bürokraft die Gehaltsangaben eines Mitarbeiters nicht, aber alle anderen Angaben sehen können, so könne man eine entsprechende Ansicht bilden, die die sensiblen Spalten ausblendet.



Definition 59: Projektionsansicht

Eine Ansicht, deren Aufgabe das Herausfiltern von Spalten ist, wird *Projektionsansicht* genannt.

Das Argument halte ich in der so vorgetragenen Form für Der Anwender arbeitet gar nicht direkt auf Ebene der Datenbank, sondern auf Ebene der Anwendung. Er kann doch nur sehen und verändern, was die Anwendung zulässt.

16.4 Grenzen einer Ansicht

Die MySQL- und MariaDB-Referenz schränkt Ansichten an vielen Stellen ein². Hier einige davon:

- Es dürfen keine Unterabfragen in der FROM-Klausel verwendet werden.

```
1 mysql> CREATE VIEW v AS SELECT bezeichnung
FROM (SELECT * FROM artikel) a;
2 ERROR 1349 (HY000): View's SELECT contains
a subquery in the FROM clause
```

- Es dürfen keine System- oder Benutzervariablen verwendet werden.

```
1 mysql> SET @b = 3000;
2 Query OK, 0 rows affected (0.00 sec)
3
4 mysql> CREATE VIEW v AS SELECT * FROM arti
kel WHERE artikel_id > @b;
5 ERROR 1351 (HY000): View's SELECT contains
a variable or parameter
```

- Es dürfen keine temporären Tabellen verwendet werden. Auch kann keine temporäre Ansicht erzeugt werden.

```
1  mysql> CREATE TEMPORARY TABLE tmp SELECT *
   FROM artikel;
2  Query OK, 18 rows affected (0.16 sec)
3  Records: 18 Duplicates: 0 Warnings: 0
4
5  mysql> CREATE VIEW view_tmp AS SELECT arti
   kel_id FROM tmp;
6  ERROR 1352 (HY000): View's SELECT refers t
   o a temporary table 'tmp'
```

- Das ORDER BY der Ansicht wird vom ORDER BY der
enthaltenden Anweisung überlagert.

```
1  mysql> CREATE VIEW view_tmp
2  -> AS
3  -> SELECT nachname FROM kunde ORDER BY nac
   hname DESC;
4
5  mysql> SELECT * FROM view_tmp;
6  +-----+
7  | nachname |
8  +-----+
9  | Tuk      |
10 [...]
11 | Beutlin  |
12 +-----+
13
14 mysql> SELECT * FROM view_tmp ORDER BY nac
   hname;
15 +-----+
16 | nachname |
17 +-----+
18 | Beutlin  |
19 [...]
20 | Tuk      |
```


- Nicht jede Ansicht ist veränderbar.

Den letzten Punkt müssen wir uns mal genauer anschauen.
Zunächst zwei Beispiele:

```

1  CREATE VIEW view_tmp_eins
2  AS
3      SELECT nachname FROM kunde
4  ;
5
6  CREATE VIEW view_tmp_zwei
7  AS
8      SELECT rechnung_id, COUNT(*) anzahl
9      FROM rechnung_position
10     GROUP BY rechnung_id
11 ;

```

Die erste Ansicht liefert mir einfach nur die Nachnamen der Kunden und die zweite zu jeder Rechnung die Anzahl der Positionen. Und jetzt kommt's:

```

1  mysql> UPDATE view_tmp_eins
2      -> SET nachname = 'Streicher'
3      -> WHERE nachname = 'Telcontar'
4  ;
5  Query OK, 1 row affected (0.03 sec)
6  Rows matched: 1 Changed: 1 Warnings: 0
7
8  mysql> UPDATE view_tmp_zwei
9      -> SET anzahl = 2
10     -> WHERE anzahl > 2;
11  ERROR 1288 (HY000): The target table view_
    tmp_zwei of the UPDATE is not updatable

```

Der erste UPDATE hat tadellos funktioniert. Ein SELECT würde mir das bestätigen. Aber der zweite lieferte eine hässliche

Fehlermeldung: Die Ansicht ist nicht *updateable*³. Wenn ich der Anzahl einen neuen Wert zuweisen möchte, erkenne ich, dass es überhaupt keine Spalte einer Zeile gibt, in den ich diesen neuen Wert hineinschreiben könnte. Der Wert wird ja durch Nachzählen ermittelt. Würde ich jetzt die Anzahl auf 2 setzen, was sollte er dann tun? Alle Positionen bis auf zwei löschen? Ein UPDATE auf diese Zeile ist also sinnlos.

```
mysql> UPDATE view_tmp_zwei SET rechnung_id =  
1 WHERE rechnung_id > 1;  
ERROR 1288 (HY000): The target table view_tmp_  
zwei of the UPDATE is not updatable
```

Man könnte meinen, dass dieser UPDATE eigentlich klappen müsste, da die *rechnung_id* das Gruppierungselement ist. Aber auch hier erscheint die Fehlermeldung. Warum? Weil es eben keine 1 : 1-Verknüpfung mehr zwischen den *Zeilen der Originaltabelle* und den *Zeilen der Ansicht* mehr gibt. Das erscheint als ein sehr formales Argument, weil man sich hier vorstellen könnte, dass die Aktualisierung hätte klappen können⁴.



Definition 60: Veränderbare Ansicht

Eine Ansicht kann dann durch INSERT, UPDATE oder DELETE verändert werden, wenn zwischen den Werten der Ansicht und den Ausgangsdaten eine 1 : 1-Verknüpfung besteht.

Das heißt, dass bei *view_tmp_eins* auch INSERT- und DELETE-Operationen möglich sind. Zunächst das Einfügen zweier neuer Kundennachnamen:

```
1 mysql> INSERT INTO view_tmp_eins VALUES ('
```

```

Wurst'), ('Brot');
2 Query OK, 2 rows affected (0.04 sec)
3 Records: 2 Duplicates: 0 Warnings: 0
4
5 mysql> SELECT * FROM view_tmp_eins;
6 +-----+
7 | nachname |
8 +-----+
9 | Beutlin  |
10 [...]
11 | Brot    |
12 [...]
13 | Wurst   |
14 +-----+
15 24 rows in set (0.00 sec)

```

Die neuen Kunden sind auch tatsächlich in der Tabelle kunde angekommen und nicht in irgendeiner ominösen neuen Tabelle gelandet, wie Sie leicht mit einem SELECT auf kunde bestätigen können. Jetzt wollen wir das Löschen versuchen:

```

1 mysql> DELETE FROM view_tmp_eins WHERE nac
hname IN ('Wurst', 'Brot');
2 Query OK, 2 rows affected (0.04 sec)

```

Auch hier sind die Zeilen in der Tabelle kunde gelöscht worden.



Hinweis: Bei Änderungen in einer veränderbaren Ansicht gelten die gleichen Regeln, als ob Sie die Daten direkt in den Tabellen einfügen, ändern oder löschen. Besonders die Fremdschlüssel- und die NOT NULL-Constraints sind hier zu beachten.

Wann ist eine Ansicht denn nicht mehr veränderbar, also wann geht die geforderte 1 : 1-Verknüpfung verloren? Diese Frage hat einige einfache und einige kompliziertere Antworten.

- Zunächst ist klar, dass die Ansicht nicht mehr aktualisierbar ist, wenn Aggregatfunktionen (siehe [Abschnitt 26.2.3](#)) vorkommen.
- Enthält eine Ansicht eine nichtveränderbare Ansicht, ist diese auch nicht veränderbar.
- Nach einem DISTINCT (siehe [Abschnitt 10.4](#)) können einzelne Zeilen nicht mehr der ursprünglichen Tabelle zugeordnet werden; analog das GROUP BY (siehe [Abschnitt 12.2](#)).
- Theoretisch wäre ein UNION machbar, aber dazu müsste der Verweis auf die ursprüngliche Tabelle bei einem UNION mit verwaltet werden. Somit ist auch dann keine Veränderung mehr möglich.
- Durch die Verwendung einer Unterabfrage (siehe [Kapitel 13](#)).
- Die Spaltenwerte werden aus Konstanten oder Ausdrücken ermittelt. Bei Konstanten gibt es keine Zeilen geschweige denn Tabellen, die Sie verändern könnten. Bei Ausdrücken müsste die Eindeutigkeit und Existenz des Inversen des Ausdrucks gegeben sein, was natürlich nicht der Fall ist.

Aber: Werden bei einem UPDATE nur die Spalten verändert, die sich nicht aus einem Ausdruck herleiten oder durch Konstanten bestimmt sind, ist eine Aktualisierung möglich.

- ALGORITHM=TEMPTABLE erzeugt von einer Auswertung eine temporäre Tabelle. Diese ist eine Kopie der ursprünglichen Daten. Daher sind Änderungen auf dieser ggf. gefährlich.

Besonders unübersichtlich wird es, wenn Sie eine Verbundansicht haben. Zunächst muss ein `INNER JOIN` verwendet werden. Grundsätzlich wird die Tabelle bei der Verwendung von `OUTER JOIN` oder `UNION` unveränderbar.

Auf einer Verbundansicht sind `INSERT`, `UPDATE` möglich, wenn sich die neuen oder geänderten Spaltenwerte nur auf *eine* Tabelle der Verknüpfung beziehen und keine der obigen Bedingungen verletzt werden und ist `DELETE` grundsätzlich nicht möglich.

So viel zu Ansichten.

¹ In PostgreSQL geht auch das mit `ALTER TABLE viewname RENAME TO`.

² Ich kann nicht sagen, ob diese Einschränkungen vom SQL-Sprachstandard oder den technischen Grenzen von MySQL und MariaDB herrühren.

³ engl.: aktualisierbar

⁴ Leider entzieht es sich meiner Kenntnis, ob diese Einschränkung MySQL-/MariaDB-spezifisch ist. In PostgreSQL gibt es auch entsprechende Einschränkungen, die aber anders motiviert sind.

17 Exkurs NoSQL



Die Wiese des Nachbarn ist immer grüner als die eigene.

- Was ist NoSQL?
- JSON-Objekte
- Arbeiten mit Collections
- Arbeiten mit dem SQL-Datentyp JSON



Die Quelltexte dieses Kapitels stehen in den Dateien `mysql/listing13NoSQL.sql` (siehe [Listing 28.15](#)) und `mysql/listing13NoSQL.js` (siehe [Listing 28.14](#)). Die Bestellungen liegen in `mysql/lstRechnung01.json` und der Beispielwarenkorb liegt in `mysql/lstWarenkorb01.json`.

Das Kapitel kann kein NoSQL¹-Buch ersetzen; ich möchte aber dieses für die Datenbank-Praxis so wichtige Thema hier eingeführt haben. Anwendungsgebiete gibt es genug und für die Prüfungen wird es auch immer bedeutender, Grundkenntnisse darin zu besitzen. Leider muss ich viele Themen, die typischerweise mit NoSQL verbunden werden wie

beispielsweise *Big Data* oder *räumliche Daten*, völlig außen vor lassen. Für eine tiefere Einführung empfehle ich daher die Bücher [[EFH+](#)] und [[Har](#)].

Aber was ist NoSQL eigentlich? Mithilfe von NoSQL werden Daten in nichttabellarischer Form abgespeichert und mit SQL oder einer Client-Sprache (JavaScript, PHP, .NET usw.) bearbeitet und ausgewertet. Da es noch keine NoSQL-Normierung oder etablierte Konvention gibt, ist NoSQL extrem proprietär und ich kann hier nur auf die im MySQL Server implementierten Features eingehen. Als Client-Sprache verwende ich JavaScript.

17.1 Vorbereitung der MySQL-Shell

Normalerweise werden die NoSQL-Anweisungen von einer Client-Anwendung erteilt. Die entsprechenden Funktionen stehen Ihnen in den passenden Connectoren (Python, JavaScript, .NET etc.) zur Verfügung. Da wir mit der erweiterten interaktiven MySQL-Shell arbeiten wollen, müssen wir auf eine dieser Clientsprachen schalten. Dies ist mit dem X-Plugin von MySQL 8.0 und der erweiterten MySQL-Shell (`mysqlsh`) möglich. Leider ist die erweiterte MySQL-Shell noch nicht in unserer Dockerinstanz (siehe [Abschnitt 4.3.1](#)) installiert, also frisch ans Werk.

Innerhalb der Docker-Bash wird nun `mysqlsh` gestartet:

```
1  bash-4.4# mysqlsh -uroot -p
2  Cannot set LC_ALL to locale en_US.UTF-8: N
o such file or directory
3  Please provide the password for 'root@/var
%2Frun%2Fmysqld%2Fmysqld.sock': *****
**
```

```
4  MySQL Shell 8.0.33
5
6  Copyright (c) 2016, 2023, Oracle and/or it
s affiliates.
7  Oracle is a registered trademark of Oracle
Corporation and/or its affiliates.
8  Other names may be trademarks of their res
pective owners.
9
10 Type '\help' or '\?' for help; '\quit' to
exit.
11 Creating a Classic session to 'root@/var%2
Frun%2Fmysqld%2Fmysqld.sock'
12 Fetching schema names for auto-completion.
.. Press ^C to stop.
13 Your MySQL connection id is 10
14 Server version: 8.0.33 MySQL Community Ser
ver - GPL
15 No default schema selected; type \use <sch
ema> to set one.
16 MySQL localhost JS > \q
17 Bye!
18 bash-4.4#
```

Falls Sie den MySQL Server unter Windows installiert haben, müssen Sie eine PowerShell öffnen und `mysqlsh` eintippen.

```
1  Windows PowerShell
2  [...]
3  PS C:\Users\adams> mysqlsh -uroot -p
4  Please provide the password for 'root@loca
localhost': *****
5  Save password for 'root@localhost'? [Y]es/
[N]o/Ne[v]er (default No): y
6  MySQL Shell 8.0.33
7
8  Copyright (c) 2016, 2023, Oracle and/or it
s affiliates.
```



```
9 Oracle is a registered trademark of Oracle
  Corporation and/or its affiliates.
10 Other names may be trademarks of their res
    pective owners.
11
12 Type '\help' or '\?' for help; '\quit' to
    exit.
13 Creating a session to 'root@localhost'
14 Fetching schema names for auto-completion.
    .. Press ^C to stop.
15 Your MySQL connection id is 9 (X protocol)
16 Server version: 8.0.33 MySQL Community Ser
    ver - GPL
17 No default schema selected; type \use <sch
    ema> to set one.
18 MySQL localhost:33060+ ssl JS > \q
19 Bye!
20 PS C:\Users\adams>
```

17.2 Datenmodellierung des Warenkorbs

Ich möchte einen Warenkorb modellieren. Dabei soll der online zusammengestellte Warenkorb nicht in eigenen Tabellen wie beispielsweise bestellung abgelegt werden – dies wäre mit Kanonen auf Spatzen schießen. Vielmehr möchte ich, dass jeder Kunde einen Warenkorb hat, der einfach verarbeitet werden kann, ohne dass dabei jedes Mal ein JOIN oder ein GROUP BY mit Aggregatfunktionen über die ganzen Tabellen abgesetzt werden muss. Klären wir also zunächst, in welchem Format wir die Daten speichern werden.

17.2.1 JavaScript Object Notation (JSON)

In [Abschnitt 2.1.1](#) haben wir uns verschiedene Datenstrukturen angeschaut. Üblicherweise werden bei NoSQL die Daten als JSON-Dokumente² (siehe [[Int17](#)]) abgelegt und man benötigt Funktionen, mit deren Hilfe man diese Dokumente auslesen kann.

Wie auch bei XML handelt es sich um eine einfache Auszeichnungssprache für nicht binäre Daten. Dabei können auch Hierarchien oder Mengen flexibel und bedarfsgerecht genutzt werden. JSON selbst ist eine Teilmenge von JavaScript bzw. ECMA und muss einen durch `eval()` ausführbaren JavaScript-Quelltext abbilden.

Schauen wir uns das Beispiel aus [Abschnitt 2.1.1](#) mal in JSON an. Grundbaustein von JSON sind Attribute, denen man Werte zuweist; beide werden durch Doppelpunkt getrennt.

```
1  "artikel_id":"7856"
```

Mehrere Attribute können durch Kommata getrennt zu Objekten zusammengefasst werden. Ganze Objekte werden durch geschweifte Klammern begrenzt. Dabei fungiert der Objektname wie ein Attributname und wird daher ebenfalls durch einen Doppelpunkt von seiner Attributliste getrennt. Hier ein Artikelobjekt:

```
1  "artikel":{
2      "artikelnummer":"7856",
3      "artikelname":"Silberzwiebel",
4      "einzelpreis":6.15
5  }
```

Objekte enthalten einfache Attributwerte beispielsweise die Artikelnummer oder wiederum Objekte mit Attributlisten. So können diese beliebig tief verschachtelt werden. Die

Bestellposition enthält beispielsweise ein Artikelobjekt und eine Mengenangabe:

```
1  "position":{
2    "artikel":{
3      "artikelnummer":"7856",
4      "artikelname":"Silberzwiebel",
5      "einzelpreis":6.15,
6      "währung":"EUR"
7    },
8    "menge":15
9  }
```

Mengen von Objekten können in Arrays³ abgelegt werden; sie sind durch rechteckige Klammern markiert. In unserer Bestellung kommen mehrere Bestellpositionen vor:

```
1  "positionen":[
2    {
3      "position":{
4        "artikel":{
5          "artikelnummer":"7856",
6          "artikelname":"Silberzwiebel",
7          "einzelpreis":6.15,
8          "währung":"EUR"
9        },
10       "menge":15
11     },
12     {
13       "position":{
14         "artikel":{
15           "artikelnummer":"7863",
16           "artikelname":"Tulpenzwiebel",
17           "einzelpreis":32.90,
18           "währung":"EUR"
19         },
20         "menge":10
21       }
```

```

22     }
23   },
24   {
25     "position":{
26       "artikel":{
27         "artikelnummer":"9015",
28         "artikelname":"Spaten",
29         "einzelpreis":19.90,
30         "währung":"EUR"
31       },
32       "menge":1
33     }
34   }
35 ]

```

Dies soll für eine minimale Einführung reichen. Eine vertiefende finden Sie unter [[Hel19](#)]. Der gesamte JSON-Quelltext ist in der Datei `lstRechnung01.json` abgelegt. Um Ihren JSON-Quelltext vorab überprüfen zu lassen, finden Sie unter <https://jsonformatter.curiousconcept.com/> einen Quelltext-Validator.

17.2.2 Struktur unseres JSON-Dokuments

Nach kurzer Reflexion kommen wir zu dem Ergebnis, dass der Warenkorb einer Bestellung sehr ähnlich ist. Im Wesentlichen besteht der Warenkorb aus folgenden Inhalten:

- **Kundennummer:** Da jeder Kunde nur einen Warenkorb haben kann, reicht hier eine Kundennummer und es muss keine Information wie eine Bestellnummer miterfasst werden.
- **Warenkorbpositionen:** Artikelnummer, Artikelbezeichnung, Artikelpreis, Währung, Anzahl und Gesamtpreis

- Der Warenwert des Warenkorbs soll nach Währung getrennt berechnet werden: Wert und Währung.



Aufgabe 17.1: Transformieren Sie die Inhalte von der ersten Bestellung von Kundennummer 12345 aus `lstRechnung01.json` in ein entsprechendes JSON-Dokument mit dem Namen `lstWarenkorb01.json`.

Das Ergebnis könnte so aussehen:

```
1  {
2    "warenkorb":{
3      "kunde_id":"12345",
4      "positionen":[
5        {
6          "position":{
7            "artikel":{
8              "artikel_id":"7856",
9              "artikelname":" Silberzwiebel"
10             },
11            "einzelpreis":6.15
12          },
13          "menge":15,
14          "gesamtpreis":92.25,
15          "währung":"EUR"
16        },
17        {
18          "position":{
19            "artikel":{
20              "artikel_id":"7863",
21              "artikelname":" Tulpenzwiebel"
22             },
23            "einzelpreis":32.90
24          },
25          "menge":10,
26          "gesamtpreis":329.00,
27          "währung":"EUR"
28        }
29      ]
30    }
31  }
```

```

24         "menge":10,
25         "gesamtpreis":329.0,
26         "währung":"EUR"
27     }
28 },
29 {
30     "position":{
31         "artikel":{
32             "artikel_id":"9015",
33             "artikelname":"Spaten",
34             "einzelpreis":19.90
35         },
36         "menge":1,
37         "gesamtpreis":19.90,
38         "währung":"EUR"
39     }
40 }
41 ],
42 "warenwerte":[
43     {
44         "EUR":441.15
45     }
46 ]
47 }
48 }

```



Aufgabe 17.2: Wenn Sie an die Normalisierungen denken (siehe [Abschnitt 2.4](#)), was ist bezüglich der Angaben gesamtpreis und warenwerte zu sagen?

Sie sehen hier ein häufiges Phänomen bei NoSQL-Datenbanken: Man verzichtet auf Redundanzfreiheit um einer besseren Performance willen. Die Erhaltung der

Datenkonsistenz muss dann vom Programmierer überwacht werden.

17.3 NoSQL: MySQL mit JavaScript-Client

So, jetzt haben wir alles beisammen: die erweiterte MySQL-Shell und das Datenmodell. Jetzt wollen wir mal das Zusammenspiel zwischen MySQL und JavaScript betrachten.

Alle JSON-Dokumente werden in einer *Collection* abgelegt. Dabei bekommt jedes Dokument einen automatisch oder manuell zugewiesenen Schlüssel mit dem festen Namen `_id`. Die Funktion dieses Schlüssels entspricht dem eines Primärschlüssels ([Definition 11](#)) in Tabellen; nur so kann bequem auf einzelne Dokumente in der Collection zugegriffen werden. Für unseren Anwendungsfall bietet es sich an, die Kundennummer als Schlüssel zu verwenden, da jeder Kunde immer nur einen Warenkorb haben kann.

Starten wir also die erweiterte MySQL-Shell mit `mysqlsh` und gehen unsere ersten vorsichtigen Schritte in JavaScript. Zunächst ein paar Vorbereitungen: In [Zeile 1](#) wird ein Objekt erstellt, welches alle Verbindungsparameter enthält; Sie müssten hier ggf. Anpassungen vornehmen. Mit `require()` stellen wir uns in [Zeile 3](#) die MySQL-JavaScript-Umgebung zur Verfügung. Hier – wie auch bei zukünftigen Lösungen – werden die Arbeiten in Funktionen gekapselt. Mithilfe des Schlüsselwortes `function` leiten wir die Funktionsdefinition ab [Zeile 5](#) ein.

Innerhalb der Funktion holen wir uns in [Zeile 6](#) mit `getSchema()` eine Referenz auf die Datenbank und in [Zeile 7](#) versuchen wir eine ggf. vorhandene Collection mit

dropCollection() zu löschen. Jetzt wird in [Zeile 8](#) die Collection angelegt und per return dem Aufrufer zur Verfügung gestellt. Das war die Funktionsdefinition.

Jetzt wollen wir das Ding auch benutzen: [Zeile 11](#) öffnet eine Verbindung/Sitzung zum Server und liefert mir eine Referenz darauf zurück. Diese übergeben ich anschließend der Funktion als Parameter. Mit getCollections() lasse ich mir in [Zeile 13](#) alle in der Datenbank angelegten Collections anzeigen: In unserem Fall gibt es nur eine.

Und zum Schluss: *Tür zu!* In [Zeile 18](#) wird die Sitzung mit close() beendet und dadurch werden serverseitig die Ressourcen wieder freigegeben.

```
1  MySQL JS > var connect = { host: 'localhost', port: 33060,
2                    -> user: 'root', password: 'weinschlauchX20' };
3  MySQL JS > var mysqlx = require('mysqlx');
4  MySQL JS >
5  MySQL JS > function warenkorbCollectionAnlegen(mySession) {
6                    -> var db0shop = mySession.getSchema('oshop');
7                    -> db0shop.dropCollection('warenkörbe');
8                    -> return db0shop.createCollection('warenkörbe');
9                    -> }
10
11 MySQL JS > var session = mysqlx.getSession(connect);
12 MySQL JS > warenkorbCollectionAnlegen(session);
13 MySQL JS > session.getSchema('oshop').getCollections();
14 [
```



```
15         <Collection:warenkörbe>
16     ]
17 MySQL JS >
18 MySQL JS > session.close();
```



Hinweis: Um Platz zu sparen, verkürze ich den Prompt von MySQL JS > auf JS >. Ebenso verzichte ich zukünftig auf die Zeilen, mit denen ich die Verbindung zum Server herstelle ([Zeilen 3 bis 11](#)), die Datenbank auswähle ([Zeile 6](#)) und die Collection ermittle bzw. anlege ([Zeilen 13 bis 8](#)). Auch das Schließen der Sitzung möchte ich nicht jedes Mal ([Zeile 18](#)) angeben.

17.3.1 Anlegen eines Warenkorbs

Als Erstes wird allen Kunden ein leerer Warenkorb zugewiesen. Dieser sollte folgende Struktur haben:

```
1  {
2      "warenkorb": {
3          "knr": "12345",
4          "positionen": [],
5          "warenwerte": []
6      }
7  }
```

Mit `getCollection()` besorgen wir uns eine Referenz auf die Collection. Über diese Referenz ermitteln wir eine weitere Referenz, und zwar auf die Kundentabelle ([Zeile 7](#)). In [Zeile 4](#) löscht zunächst `remove()` alle schon vorhandenen Warenkörbe (nicht die Dokumente!). Mit der Methode `select()` wird jetzt die Kundennummer aller Kunden

ermittelt und einer Ergebnisvariablen zugewiesen ([Zeile 8](#)). Danach kopiert uns `fetchAll()` das Ergebnis in ein Array und wir können mit der *foreach*-Variante von JavaScript `for...in` alle Arrayelemente durchwandern ([Zeile 11](#)). In [Zeile 13](#) wird der leere Warenkorb gebastelt und mit `add()` der Collection hinzugefügt. Das Resultat sind lauter leere Warenkörbe ([Zeile 25](#)).

```
1  JS > function warenkorbLeeren(mySession) {
2      -> var db0shop = mySession.getSchema('os
hop');
3      -> var collWarenkörbe = db0shop.getColle
ction('warenkörbe');
4      -> collWarenkörbe.remove("_id>0").execut
e();
5      ->
6      -> // Für alle Kunden einen leeren Waren
korb anlegen.
7      -> var tabKunde = db0shop.getTable("kund
e");
8      -> var alleKunden = tabKunde.select("kun
de_id").execute();
9      ->
10     -> var zeilen = alleKunden.fetchAll();
11     -> for (index in zeilen) {
12     -> var zeile = zeilen[index];
13     -> collWarenkörbe.add({
14     ->   _id: zeile.kunde_id,
15     ->   warenkorb: {
16     ->     kunde_id: zeile.kunde_id,
17     ->     positionen: new Array(),
18     ->     warenwerte: new Array()
19     ->   }
20     -> }).execute();
21     -> }
22     -> }
23     ->
```

```

24 JS > warenkorbLeeren(session);
25 JS > session.getSchema('oshop').getCollection('warenkörbe').find();
26 {
27     "_id": 1,
28     "warenkorb": {
29         "kunde_id": 1,
30         "positionen": [],
31         "warenwerte": []
32     }
33 }
34 [...]
35 22 documents in set (0.0004 sec)

```

17.3.2 Inhalte des Warenkorbs anlegen

Gehen wir davon aus, dass ein Kunde auf der Client-Oberfläche folgende Dinge tut: Der Kunde selektiert einen Artikel, legt eine Anzahl fest und fügt dies dem Warenkorb hinzu.

Um die Warenkorbpositionen einzufügen, müssen wir auf die Artikeltable zugreifen. Dazu wird zunächst in einem String der SELECT vorformuliert ([Zeile 5](#)). Haben Sie das Fragezeichen am Ende des Strings bemerkt? Dies ist ein Platzhalter für die *Parameterbindung* (*Parameter Binding*). Sie können den Mechanismus gut in [Zeile 10](#) erkennen. Durch die Methode `bind()` in [Zeile 10](#) wird das Fragezeichen durch den Wert von `myArtikelId` ersetzt. Die eigentlichen Methoden, die den SQL-Befehl ausführen, sind `sql()` und `execute()`.

Mit `fetchOne()` wird die erste Zeile der Ergebnismenge einer Variablen zugewiesen. Bei jedem Aufruf der Methode springt ein interner Zeiger immer eine Zeile in der Ergebnismenge weiter. So können Sie – beispielweise in einer `while`-Schleife – die gesamte Ergebnismenge abarbeiten. Da wir nur einen

Artikel ausgewählt haben, liefert mir `fetchOne()` die erste und einzige Zeile der Ergebnismenge.

Und wie kann nun auf die Spalten der Ergebniszeile zugegriffen werden? In JavaScript ist die Ergebniszeile ein Objekt und daher kann auf die Spalten per Punktnotation zugegriffen werden, wie z.B. in [Zeile 12](#).

Ab [Zeile 11](#) werden die Positionenobjekte gebastelt und mit den Methoden `modify()` und `arrayInsert()` in [Zeile 24](#) zurückgeschrieben. Dabei kommt eine Variante in der Syntax der Parameterbindung vor. In der Anweisung wird dem Parameternamen ein Doppelpunkt vorangestellt und anschließend kommt der Parametername – hier `param`. Durch die Vergabe von Namen könnten so für die Anweisung mehrere Parameterbindungen definiert werden.

In [Zeile 32](#) können Sie sehen, wie mithilfe des Schlüssels `_id` und der Methode `find()` gezielt auf ein bestimmtes Dokument der Collection zugegriffen wird.

```
1  JS > function warenkorbPositionEinfügen(my
Session,myKundeId,myArtikelId,myMenge) {
2      -> var dbOshop = mySession.getSchema('os
hop');
3      -> var collWarenkörbe = dbOshop.getColle
ction('warekörbe');
4      ->
5      -> var sql = "SELECT";
6      -> sql += " artikel_id, bezeichnung, ein
zelpreis, waehrung";
7      -> sql += " FROM oshop.artikel";
8      -> sql += " WHERE artikel_id=?";
9      ->
10     -> var artikel=mySession.sql(sql).bind(myA
rtikelId).execute().fetchOne();
11     -> var objArtikel = {
12     -> artikel_id: artikel.artikel_id,
```

```

13     -> artikelname: artikel.bezeichnung,
14     -> preis: artikel.einzelpreis
15     -> };
16     ->
17     -> var objPosition = {
18     -> artikel: objArtikel,
19     -> menge: myMenge,
20     -> gesamtpreis: myMenge * artikel.einzel
preis,
21     -> währung: artikel.waehrung
22     -> };
23     ->
24     -> collWarenkörbe.modify('_id=:param').a
rrayInsert('warenkorb.positionen[0]', {
25     -> position: objPosition
26     -> }).bind('param', myKundeId).execute()
;
27     -> }
28     ->
29 JS > warenkorbPositionEinfügen(session, 2,
3007, 10);
30 JS > warenkorbPositionEinfügen(session, 2,
9010, 5);
31 JS > warenkorbPositionEinfügen(session, 2,
3010, 2);
32 JS > session.getSchema('oshop').getCollect
ion('warenkörbe').find("_id=2");

```

Das Ergebnis der letzten Anweisung ist:

```

1  {
2    "_id":2,
3    "warenkorb":{
4      "kunde_id":2,
5      "positionen":[
6        {
7          "position":{
8            "menge":2,

```

```
9         "artikel":{
10             "preis":"5.050000",
11             "artikel_id":3010,
12             "artikelname":"Feder"
13         },
14         "währung":"EUR",
15         "gesamtpreis":10.1
16     },
17 },
18 {
19     "position":{
20         "menge":5,
21         "artikel":{
22             "preis":"15.100000",
23             "artikel_id":9010,
24             "artikelname":"Schaufel"
25         },
26         "währung":"USD",
27         "gesamtpreis":75.5
28     }
29 },
30 {
31     "position":{
32         "menge":10,
33         "artikel":{
34             "preis":"4.170000",
35             "artikel_id":3007,
36             "artikelname":"Tinte(blau)"
37         },
38         "währung":"EUR",
39         "gesamtpreis":41.7
40     }
41 },
42 ],
43 "warenwerte":[
44
45 ]
46 }
```

```
47  }
48  1 document in set (0.0006 sec)
```



Hinweis: Mithilfe von Parameterbindung wird ein Weg der SQL-Injection verhindert (siehe [Abschnitt 25.3](#)). Der Parameter selbst kann dann keinen SQL-Code mehr enthalten, da der Parameter intern entsprechend analysiert und präpariert ist.

Was mich am bisherigen Status stört, ist, dass der Warenwert des Warenkorbs nicht berechnet wird. Bauen wir also eine Funktion, die den Warenwert und den Gesamtpreis neu berechnet. Wir können diese dann nach dem Einfügen oder Ändern von Daten aufrufen. Spannend sind die beiden Methoden `hasOwnProperty()` ([Zeile 8](#)) und `keys` ([Zeile 15](#)). Beide Methoden entspringen der Technik *reflection* in der objektorientierten Programmierung. Dabei wird ein System so programmiert, dass es zur Laufzeit seine eigenen Eigenschaften kennt oder anderen mitteilt. Genauer an einem Beispiel: Ein Dreieck-Objekt kennt nicht nur seine Seitenlängen, die dann mit Methoden wie `GetSeiteA()` einem anderen mitgeteilt werden können, sondern kann einem anderen auch mitteilen, dass es eine Seite A hat.

`hasOwnProperty()` testet, ob der übergebene Wert ein Attribut der Klasse ist und `keys()` liefert mir alle Schlüssel eines Arrays oder einer Objektliste.

```
1  function aktualisiereGesamtpreisWarenwert(
myWarenkorb) {
2      var ergebnis = Object();
3      var posindizes = Object.keys(myWarenkorb
.warenkorb.positionen);
```

```

4    posindizes.forEach(function(posindex) {
// Positionen durchwandern
5        var myPos = myWarenkorb.warenkorb.posi
tionen[posindex].position;
6        // gesamtpreis der Position berechnen
7        myPos.gesamtpreis = myPos.artikel.prei
s * myPos.menge;
8        if (ergebnis.hasOwnProperty(myPos.währ
ung)) {
9            ergebnis[myPos.währung] += myPos.gesam
tpreis;
10       } else {
11           ergebnis[myPos.währung] = myPos.gesa
mtpreis;
12       }
13   });
14   var warenwert = new Array();
15   for (index in Object.keys(ergebnis)) {
16       var währung = Object.keys(ergebnis)[in
dex];
17       var wert = ergebnis[währung];
18       warenwert.push({ [währung]: wert });
19   }
20
21   myWarenkorb.warenkorb.warenwerte = waren
wert;
22   return myWarenkorb;
23 }

```



Aufgabe 17.3: Versuchen Sie in Prosa den Quelltext zu interpretieren. Was wird beispielsweise in [Zeile 8](#) getestet und warum?

Jetzt können wir diese Methode geschickt einbauen (siehe [Zeile 24](#)):


```

1  function warenkorbPositionEinfügen(mySession, myKundeId, myArtikelId, myMenge) {
2      var db0shop = mySession.getSchema('oshop');
3      var collWarenkörbe = db0shop.getCollection('warenkörbe');
4      var sql = "SELECT";
5      sql += " artikel_id, bezeichnung, einzelpreis, waehrung";
6      sql += " FROM oshop.artikel";
7      sql += " WHERE artikel_id=?";
8      var artikel = mySession.sql(sql).bind(myArtikelId).execute().fetchOne();
9      var objArtikel = {
10         artikel_id: artikel.artikel_id,
11         artikelname: artikel.bezeichnung,
12         preis: artikel.einzelpreis
13     };
14     var objPosition = {
15         artikel: objArtikel,
16         menge: myMenge,
17         gesamtpreis: myMenge * artikel.einzelpreis,
18         währung: artikel.waehrung
19     };
20     collWarenkörbe.modify('_id=:param').arrayInsert('warenkorb.positionen[0]', {
21         position: objPosition
22     }).bind('param', myKundeId).execute();
23     var wk = collWarenkörbe.find("_id=:param").bind('param', myKundeId).execute().fetchOne();
24     wk = aktualisiereGesamtpreisWarenwert(wk);
25     collWarenkörbe.modify('_id=:param').set('$', wk).bind('param', myKundeId).execute();
26 }

```



Aufgabe 17.4: Wie können Sie sich das Ergebnis anschauen? Programmieren Sie es und überprüfen Sie, ob der Gesamtpreis und die Warenwerte stimmen. Das Ergebnis müsste so aussehen:

```
1  {
2    "_id":2,
3    "warenkorb":{
4      "kunde_id":2,
5      "positionen":[
6        {
7          "position":{
8            "artikel":{
9              "artikel_id":3010,
10             "artikelname":"Feder",
11             "preis":"5.050000"
12           },
13           "gesamtpreis":10.1,
14           "menge":2,
15           "währung":"EUR"
16         }
17       },
18       {
19         "position":{
20           "artikel":{
21             "artikel_id":9010,
22             "artikelname":"Schaufel" ,
23             "preis":"15.100000"
24           },
25           "gesamtpreis":75.5,
26           "menge":5,
27           "währung":"USD"
28         }
29       },
30     }
```

```

31         "position":{
32             "artikel":{
33                 "artikel_id":3007,
34                 "artikelname":"Tinte ( blau)",
35                 "preis":"4.170000"
36             },
37             "gesamtpreis":41.7,
38             "menge":10,
39             "währung":"EUR"
40         }
41     },
42 ],
43 "warenwerte":[
44     {
45         "EUR":51.8
46     },
47     {
48         "USD":75.5
49     }
50 ]
51 }
52 }
```

17.3.3 Inhalte des Warenkorbs auswerten

Für die Darstellung auf der WEB-Oberfläche soll der Inhalt des Warenkorbs ausgewertet werden.

Beispiel: Zu den Artikeln sollen die Warengruppen ausgegeben werden

Um zu den Artikeln die Warengruppen ausgeben zu können, bedienen wir uns des guten alten SQL. Der SELECT ermittelt für alle Artikel die Warengruppe, sodass ich das Ergebnis in [Zeile](#)

[31](#) als Nachschlage-Tabelle nutzen kann. Aber erst mal zur Ausführung des Befehls. Anders als oben wird der SELECT hier nicht über eine Tabellenreferenz abgesetzt, sondern in [Zeile 15](#) direkt über die Session. Die Session stellt dafür die Methode `sql()` zur Verfügung. Die Abfrage liefert mir mehr als eine Zeile zurück, die ich auch direkt komplett verarbeiten möchte, sodass ich nicht wie oben `fetchOne()`, sondern `fetchAll()` verwende. Diese Ergebnismenge wird ab [Zeile 17](#) in ein Array konvertiert, da damit leichter zu arbeiten ist. Der Rest sollte selbsterklärend sein ;-)

```
1  JS > function printWarenkorbMitWarengruppe
n(mySession, myKundeId) {
2      -> var dbOshop = mySession.getSchema('os
hop');
3      -> var collWarenkörbe = dbOshop.getColle
ction('warenkörbe');
4      ->
5      -> var select = "SELECT";
6      -> select += " artikel.artikel_id,";
7      -> select += " GROUP_CONCAT(warengruppe.
bezeichnung) 'warengruppen'";
8      -> select += " FROM";
9      -> select += " oshop.artikel INNER JOIN
oshop.artikel_nm_warengruppe";
10     -> select += " USING(artikel_id)";
11     -> select += " INNER JOIN";
12     -> select += " oshop.warengruppe";
13     -> select += " USING(warengruppe_id)";
14     -> select += " GROUP BY artikel.artikel_
id";
15     -> var alleArtikel = mySession.sql(select)
.execute().fetchAll();
16     ->
17     -> var arrArtikel = new Array();
18     -> for (index in alleArtikel) {
19     -> var zeile = alleArtikel[index];
```

```

20     -> arrArtikel.push({
21     -> artikel_id: zeile["artikel_id"],
22     -> warengruppen: zeile["warengruppen"]
23     -> });
24     -> }
25     ->
26     -> var wk = collWarenkörbe.find("_id=:pa
ram").bind('param', myKundeId).execute ().fetc
hOne();
27     -> var n = 1;
28     -> var items = Object.keys(wk.warenkorb.
positionen);
29     -> items.forEach(function(item) {
30     -> var myPos = wk.warenkorb.positionen[i
tem].position;
31     -> var wg = arrArtikel.find(function(ele
ment) {
32     -> return element.artikel_id == myPos.ar
tikel.artikel_id
33     -> });
34     -> print('Warenkorbposition ' + n++ + ':
');
35     -> print(' ' + myPos.artikel.artikel_id)
;
36     -> print(', ' + myPos.artikel.artikelnam
e);
37     -> print(', "' + wg.warengruppen + '"');
38     -> print(', ' + myPos.artikel.preis);
39     -> print(', ' + myPos.menge);
40     -> print(', ' + myPos.gesamtpreis);
41     -> print(', ' + myPos.währung + "\n");
42     -> });
43     -> }
44     ->
45 JS > printWarenkorbMitWarengruppen(session
, 2);
46 Warenkorbposition 1: 3010, Feder, "Bürobed
arf", 5.050000, 2, [...]

```

```
47 Warenkorbposition 2: 9010, Schaufel, "Gart
enbedarf,Werkzeug",[...]
48 Warenkorbposition 3: 3007, Tinte (blau), "
Bürobedarf", 4.1700[...]
```

Beispiel: Anzahl der Artikel pro Warengruppe

Eigentlich wäre es doch ganz nützlich, wenn die Warengruppen direkt bei den Artikeln stehen würden. Auch dies lässt sich nun einfach mit einer Mischung aus SQL und JavaScript bewerkstelligen:

```
1  function erweiternUmWarengruppen(mySession
, myKundeId) {
2      var db0shop = mySession.getSchema('oshop
');
3      var collWarenkörbe = db0shop.getCollecti
on('warenkörbe');
4      var wk = collWarenkörbe.find("_id=:param
").bind('param', myKundeId).execute(). fetchOn
e();
5
6      var items = Object.keys(wk.warenkorb.pos
itionen);
7      items.forEach(function(item) {
8          var myPos = wk.warenkorb.positionen[it
em].position;
9
10         var select = "SELECT";
11         select += " warengruppe_id, bezeichnun
g";
12         select += " FROM";
13         select += " oshop.artikel_nm_warengrup
pe";
14         select += " INNER JOIN";
15         select += " oshop.warengruppe";
```

```

16         select += " USING(warengruppe_id)";
17         select += " WHERE artikel_id = ?";
18
19         var wgs = mySession.sql(select).bind(myPos.artikel.artikel_id).execute().fetchAll();
20         arr = new Array();
21         for (index in wgs) {
22             var zeile = wgs[index];
23             arr.push({
24                 warengruppe_id: zeile.warengruppe_id
25             ,
26                 bezeichnung: zeile.bezeichnung
27             });
28         myPos.artikel.warengruppen = arr;
29     });
30     collWarenkörbe.modify('_id=:param').set(
31     '$', wk).bind('param', myKundeId).execute ();
32     return wk;
33 }

```

Die JSON-Struktur wird damit erweitert und nach dem Aufruf sieht sie dann wie folgt aus:

```

1  [...]
2  {
3      "position":{
4      "artikel":{
5          "artikel_id":9010,
6          "artikelname":"Schaufel",
7          "preis":"15.100000",
8          "warengruppen":[
9              {
10                 "bezeichnung":"Gartenbedarf",
11                 "warengruppe_id":3
12             },
13             {

```

```

14         "bezeichnung": "Werkzeug",
15         "warengruppe_id": 4
16     }
17 ]
18 },
19 "gesamtpreis": 75.5,
20 "menge": 5,
21 "währung": "USD"
22 }
23 },
24 [...]

```

Ich habe bereits oben erwähnt, dass in NoSQL-Lösungen oft auf Redundanzfreiheit verzichtet wird. Vielmehr stattet man die Entitäten mit möglichst vielen Informationen aus, sodass bei einer Verarbeitung eben keine Tabellen mehr verknüpft werden müssen; die Information ist ja in unmittelbarer Nähe vorhanden. Wir können die Vorteile hier gut beobachten, indem wir auf Basis der erweiterten JSON-Struktur ermitteln, wie oft eine Warengruppe im Warenkorb vertreten ist:

```

1  JS > function getAnzahlWarengruppe(mySession, myKundeId) {
2      -> var ergebnis = Object();
3      -> var posindizes = Object.keys(wk.warenkorb.positionen);
4      -> posindizes.forEach(function(posindex)
5      {
6          -> var myPos = wk.warenkorb.positionen[posindex].position;
7          -> var wgs = myPos.artikel.warengruppen;
8          -> var wgindizes = Object.keys(wgs);
9          -> wgindizes.forEach(function(wgindex) {
10             -> if (ergebnis.hasOwnProperty(wgs[wgindex].bezeichnung)) {
11                 -> ergebnis[wgs[wgindex].bezeichnung]++;
12             } else {

```



```

12     -> ergebnis[wgs[wgindex].bezeichnung] =
13     1;
14     -> }
15     -> })
16     -> });
17     -> return ergebnis;
18     -> }
19     ->
20 JS >
21 JS > getAnzahlWarengruppe(session, 2);
22 {
23     "Bürobedarf": 2,
24     "Gartenbedarf": 1,
25     "Werkzeug": 1
26 }

```

17.3.4 Inhalte des Warenkorbs verändern

Beispiel: Änderung der Mengenangabe

Der Kunde ändert über den Client die Mengenangabe im Warenkorb. Diese Änderung soll nun im JSON-Dokument abgelegt werden. Nichts ist einfacher als das. Im Grunde werden folgende Schritte unternommen: finde die richtige Warenkorbposition zur Artikelnummer, lege den neuen Wert dort ab und lasse Gesamtpreis und Warenkorbwerte neu berechnen.

```

1 JS > function mengenÄndern(mySession, myKundeId, myArtikelId, myMenge) {
2     -> var db0shop = mySession.getSchema('shop');
3     -> var collWarenkörbe = db0shop.getColle

```

```

ction('warenkörbe');
4    -> var wk = collWarenkörbe.find("_id=:pa
ram").bind('param', myKundeId).execute ().fetc
hOne();
5    ->
6    -> for (index in Object.keys(wk.warenkor
b.positionen)) {
7    -> if (wk.warenkorb.positionen[index].po
sition.artikel.artikel_id == myArtikelId) {
8    -> wk.warenkorb.positionen[index].positi
on.menge = myMenge;
9    -> }
10   -> }
11   -> wk = aktualisiereGesamtpreisWarenwert
(wk);
12   -> collWarenkörbe.modify('_id=:param').s
et('$', wk).bind('param', myKundeId). execute(
);
13   ->
14   -> return wk;
15   -> }
16   ->
17 JS > mengenÄndern(session, 2, 3010, 0);
18 JS > mengenÄndern(session, 2, 3007, 1);
19 JS > mengenÄndern(session, 2, 9010, 2);

```

Beispiel: Löschen einer Warenkorbposition

Der Kunde 2 löscht beispielsweise die Warenkorbposition mit der Artikelnummer 3007. Hier können wir komplett auf SQL verzichten. Die Warenkorbposition wird im JSON-Dokument gesucht und mit `arrayDelete()` in [Zeile 9](#) entfernt.

```

1 JS > function warenkorbPositionLöschen(myS
ession, myKundeId, myArtikelId) {
2    -> var db0shop = mySession.getSchema('os

```

```

hop');
3    -> var collWarenkörbe = db0shop.getCollection('warenkörbe');
4    -> var wk = collWarenkörbe.find("_id=:param").bind('param', myKundeId).execute().fetchOne();
5    ->
6    -> for (index in Object.keys(wk.warenkorb.positionen)) {
7    -> if (wk.warenkorb.positionen[index].position.artikel.artikel_id == myArtikelId) {
8    -> var str = 'warenkorb.positionen[' + index + ']';
9    -> collWarenkörbe.modify('_id=:param').arrayDelete(str).bind('param', myKundeId).execute(); {
10   -> wk = collWarenkörbe.find("_id=:param").bind('param', myKundeId).execute().fetchOne();
11   -> wk = aktualisiereGesamtpreisWarenwert(wk);
12   -> }
13   -> }
14   ->
15   -> return wk;
16   -> }
17   ->
18 JS > warenkorbPositionLöschen(session, 2, 3007);

```

17.4 NoSQL: klassisches SQL mit JSON-Funktionen

In MySQL stehen zwei Möglichkeiten zur Verfügung, JSON-Objekte zu verarbeiten: die Dokumentensammlung, die wir

oben verwendet haben, und Tabellenspalten vom Datentyp JSON. Da es für jeden Kunden nur einen Warenkorb geben soll, bietet es sich auch an, diesen als JSON-Spalte in der Tabelle kunde zu speichern. Die Tabelle kunde muss somit um die entsprechende Spalte erweitert werden:

```
1 ALTER TABLE kunde
2     ADD warenkorb JSON AFTER COLUMN art
3 ;
```

17.4.1 Anlegen eines Warenkorbs

Wir haben vorbereitend jedem Kunden die Spalte warenkorb zur Verfügung gestellt. Diese Spalte sollte zu Beginn einen leeren Warenkorb – wie in [Abschnitt 17.3.1](#) modelliert – enthalten. Wie bauen wir einen solchen Warenkorb nun auf? Imersten Schritt sollte nach dem Anlegen eines neuen Kunden in dieser Spalte ein leerer Warenkorb zugewiesen werden⁴.

MySQL stellt Funktionen für JSON-Dokumente zur Verfügung, die ich hier nicht ausführlich besprechen kann. Die meisten ergeben sich aber selbsterklärend aus dem Kontext. Wir wollen erstmal allen vorhandenen Kunden einen leeren Warenkorb zuweisen:

```
1 mysql> UPDATE kunde
2     -> SET warenkorb = JSON_OBJECT(
3     -> 'warenkorb',
4     -> JSON_OBJECT(
5     -> 'knr', kunde_id,
6     -> 'positionen', JSON_ARRAY(),
7     -> 'warenwerte', JSON_ARRAY()
8     -> )
9     -> )
10    -> ;
```

```

11 Query OK, 22 rows affected (0.19 sec)
12 Rows matched: 22 Changed: 22 Warnings: 0
13
14 mysql> SELECT kunde_id, warenkorb FROM kunde;
15 +-----+-----+
16 | kunde_id | warenkorb |
17 +-----+-----+
18 | 1 | {"warenkorb": {"knr": 1, "positionen": [], "warenwerte": []}} |
19 | 2 | {"warenkorb": {"knr": 2, "positionen": [], "warenwerte": []}} |
20 [...]
21 | 21 | {"warenkorb": {"knr": 21, "positionen": [], "warenwerte": []}} |
22 | 22 | {"warenkorb": {"knr": 22, "positionen": [], "warenwerte": []}} |
23 +-----+-----+

```

Das Ergebnis ist recht vielversprechend, aber nun die Funktionen im Einzelnen:

- **JSON_OBJECT():** Diese Funktion bekommt eine beliebig lange, aber gerade Anzahl von Parametern übergeben. Der ungerade Parameter wird dabei immer als *Schlüssel* und der gerade als *Wert* interpretiert, formatiert und als ein gültiges JSON-Objekt verpackt. Eine leere Klammer liefert ein leeres Objekt.
- **JSON_ARRAY():** Diese Funktion bekommt eine beliebige Anzahl von Parametern übergeben und liefert diese als ein JSON-Array zurück. Eine leere Parameterliste liefert ein leeres Array.

17.4.2 Inhalte des Warenkorbs anlegen

Beispiel: Kunde 2 bestellt zehnmal Artikel 3007.

Ab [Zeile 1](#) werden die Vorgaben in Variablen gespeichert. Dies macht den Quelltext erheblich flexibler. In [Zeile 4](#) wird der vorhandene Warenkorb des Kunden ebenfalls in einer Variablen abgelegt und anschließend wird in [Zeile 5](#) das Array mit den Positionen extrahiert.

In [Zeile 7](#) wird die neue Position aus den Tabellen und den Vorgabevariablen ermittelt, damit sie in [Zeile 25](#) an das Positionsarray angehängt und in die Kundentabelle zurückgeschrieben werden kann.

```
1  mysql> SELECT 2 INTO @kunde_id; -- Vorgabe
n
2  mysql> SELECT 3007 INTO @artikel_id;
3  mysql> SELECT 10 INTO @menge;
4  mysql> SELECT warenkorb FROM kunde WHERE k
unde_id = @kunde_id INTO @wk;
5  mysql> SELECT JSON_EXTRACT(@wk, "$.warengo
rb.positionen");
6
7  mysql> SELECT
8      -> JSON_OBJECT(
9      -> 'position', JSON_OBJECT(
10     -> 'artikel', JSON_OBJECT(
11     -> 'artikel_id', CAST(artikel_id AS CHAR
),
12     -> 'bezeichnung', bezeichnung,
13     -> 'einzelpreis', einzelpreis
14     -> ),
15     -> 'menge', @menge,
16     -> 'gesamtpreis', @menge * einzelpreis,
17     -> 'währung', waehrung
```

```

18     -> )
19     -> )
20     -> FROM artikel
21     -> WHERE artikel_id = @artikel_id
22     -> INTO @position
23     -> ;
24
25 mysql> UPDATE kunde
26     -> SET warenkorb = JSON_ARRAY_APPEND(
27     -> @wk,
28     -> "$.warenkorb.positionen",
29     -> CAST(@position AS JSON)
30     -> )
31     -> WHERE kunde_id = @kunde_id
32     -> ;

```

Das Ergebnis lässt sich aufgehübscht leicht interpretieren:

```

1  mysql> SELECT JSON_PRETTY(warenkorb) FROM
kunde WHERE kunde_id = @kunde_id\G
2  ***** 1. row *****
*****
3  JSON_PRETTY(warenkorb): {
4      "warenkorb": {
5          "kunde_id": 2,
6          "positionen": [
7              {
8                  "position": {
9                      "menge": 10,
10                     "artikel": {
11                         "artikel_id": 3007,
12                         "bezeichnung": "Tinte (blau)",
13                         "einzelpreis": 4.17
14                     },
15                     "gesamtpreis": 41.7,
16                     "währung": "EUR"
17                 }
18             }

```

```

19         ],
20         "warenwerte": []
21     }
22 }

```

Und was ist neu?

- **JSON_EXTRACT()**: Diese Funktion liefert ein JSON-Objekt oder einen JSON-Wert aus einem Dokument. Dazumuss sie wissen, wo das Objekt oder der Wert im Dokument liegt. Deshalb hat diese Funktion drei Parameter: das JSON-Dokument, den Pfad im Dokument zum Wert und den Schlüssel des Objektes oder des Wertes, der zurückgeliefert werden soll. Alternativ kann der Operator `->` verwendet werden. Will man auch noch die Anführungsstriche loswerden: `->>`
- **JSON_ARRAY_APPEND()**: Diese Funktion hängt ein neues Element einem Array an. Dazu muss sie wissen, wo das Array im Dokument abgelegt ist. Deshalb hat diese Funktion drei Parameter: das JSON-Dokument, den Pfad zum Wert im Dokument und das neue Arrayelement.
- **JSON_PRETTY()**: Diese Funktion liefert das JSON-Dokument in einer für Menschen gut lesbaren Form zurück.



Aufgabe 17.5: Erweitern Sie den Quelltext so, dass Kunde 2 zweimal Artikel 3010 und fünfmal Artikel 9010 bestellt. Das Ergebnis müsste so aussehen:

```

mysql> SELECT warenkorb FROM kunde WHERE k
unde_id = @kunde_id\G
***** 1. row *****
*****
warenkorb: {"warenkorb": {"kunde_id": 2, "
positionen": [{"position": {"menge":

```



```

10, "artikel": {"artikel_id": 3007, "
bezeichnung": "Tinte (blau)", "
    einzelpreis": 4.17}, "gesamtpreis": 4
1.7, "währung": "EUR"}}, {"position":
{"menge": 2, "artikel": {"artikel_id"
: 3010, "bezeichnung": "Feder", "
    einzelpreis": 5.05}, "gesamtpreis": 1
0.1, "währung": "EUR"}}, {"position":
{"menge": 5, "artikel": {"artikel_id"
: 9010, "bezeichnung": "Schaufel", "
    einzelpreis": 15.1}, "gesamtpreis": 7
5.5, "währung": "USD"}}], "warenwerte"
: []}}

```

Was wir bisher schön vergessen haben, ist die Berechnung der Warenwerte des Warenkorbs. Das möchte ich auf später verschieben.

17.4.3 Inhalte des Warenkorbs auswerten

Für die Darstellung auf der WEB-Oberfläche soll der Inhalt des Warenkorbs ausgewertet werden. Da wir hier nur mit SQL-Befehlen arbeiten wollen, müssen wir das JSON-Dokument immer wieder in Tabellenform bringen, um dieses auswerten zu können. Hier hilft die Funktion `JSON_TABLE()`:

```

1  mysql> SELECT 2 INTO @kunde_id;
2  mysql> SELECT warenkorb FROM kunde WHERE k
unde_id = @kunde_id INTO @wk;
3  mysql>
4  mysql> SELECT @kunde_id AS 'kunde_id', wk.
*
5      -> FROM
6      -> JSON_TABLE(

```

```

7      -> @wk,
8      -> '$.warenkorb.positionen[*]',
9      -> COLUMNS (
10     -> posnr FOR ORDINALITY,
11     -> artikel_id INT PATH '$.position.art
12     -> artikel_id',
13     -> einzelpreis DECIMAL(12,2) PATH '$.p
14     -> position.artikel.einzelpreis',
15     -> menge INT PATH '$.position.menge',
16     -> währung CHAR(3) PATH '$.position.wä
17     -> hrung'
18     -> )
19     -> ) AS wk
20     -> ;

```

	kunde_id	posnr	artikel_id	einzelpreis	menge	währung
21	2	1	3007	4.17	1	EUR
22	2	2	3010	5.05	0	EUR
23	2	3	9010	15.10	10	USD

Man kann hier gut sehen, wie das JSON-Dokument durchwandert wird und die Inhalte auf Tabellenspalten verteilt werden. Die Angabe FOR ORDINALITY in [Zeile 10](#) ist so eine Art Mini-AUTO_INCREMENT und liefert mir eine eindeutige Zeilennummer.

Beispiel: Anzahl der Artikel pro Warengruppe

Wenn Sie einmal verstanden haben, wie das mit JSON_TABLE() funktioniert, ist eigentlich alles wie gehabt. In

[Zeile 16](#) wird das JSON-Dokument in eine Tabelle überführt. Danach wird diese in der [Zeile 17](#) und [Zeile 18](#) mit den entsprechenden Tabellen der *n:m*-Verknüpfung verknüpft.

```
1  mysql> SELECT 2 INTO @kunde_id;
2  mysql> SELECT warenkorb FROM kunde WHERE k
unde_id = @kunde_id INTO @wk;
3
4  mysql> SELECT bezeichnung AS 'warengruppe'
, COUNT(*) AS 'anzahl'
5      -> FROM
6      -> JSON_TABLE(
7      -> @wk,
8      -> '$.warenkorb.positionen[*]',
9      -> COLUMNS (
10     -> posnr FOR ORDINALITY,
11     -> artikel_id INT PATH '$.position.art
ikel.artikel_id',
12     -> einzelpreis DECIMAL(12,2) PATH '$.p
osition.artikel.einzelpreis',
13     -> menge INT PATH '$.position.menge',
14     -> währung CHAR(3) PATH '$.position.wä
hrung'
15     -> )
16     -> ) AS wk
17     -> INNER JOIN artikel_nm_warengruppe U
SING(artikel_id)
18     -> INNER JOIN warengruppe USING(wareng
ruppe_id)
19     -> GROUP BY bezeichnung
20     -> ;
21  +-----+-----+
22  | warengruppe | anzahl |
23  +-----+-----+
24  | Bürobedarf | 2 |
25  | Gartenbedarf | 1 |
26  | Werkzeug | 1 |
27  +-----+-----+
```

Richtig schön wird das erst, wenn das Erstellen der Tabelle mithilfe einer CTE (siehe [Abschnitt 11.5](#)) erfolgt.

Man erkennt deutlich, dass die Abfrage leichter zu lesen und zu interpretieren ist:

```
1  mysql> WITH wk AS (  
2      -> SELECT * FROM JSON_TABLE(  
3      -> @wk,  
4      -> '$.warenkorb.positionen[*]'  
5      -> COLUMNS (  
6      -> posnr FOR ORDINALITY,  
7      -> artikel_id INT PATH '$.position.art  
ikel.artikel_id',  
8      -> einzelpreis DECIMAL(12,2) PATH '$.p  
osition.artikel.einzelpreis',  
9      -> menge INT PATH '$.position.menge',  
10     -> währung CHAR(3) PATH '$.position.wä  
hrung',  
11     -> )  
12     -> ) AS t  
13     -> )  
14     -> SELECT bezeichnung AS 'warengruppe'  
, COUNT(*) AS 'anzahl'  
15     -> FROM  
16     -> wk INNER JOIN artikel_nm_warengrupp  
e USING(artikel_id)  
17     -> INNER JOIN warengruppe USING(wareng  
ruppe_id)  
18     -> GROUP BY bezeichnung  
19     -> ;  
20  +-----+-----+  
21  | warengruppe | anzahl |  
22  +-----+-----+  
23  | Bürobedarf | 2 |  
24  | Gartenbedarf | 1 |  
25  | Werkzeug | 1 |  
26  +-----+-----+
```

17.4.4 Inhalte des Warenkorbs verändern

Wie werden Werte verändert? Grundsätzlich ist die größte Schwierigkeit – nach dem Berechnen des neuen Wertes –, die Stelle im JSON-Dokument zu finden, wo der Wert geändert werden soll. Schließlich können wir nicht einfach wie bei Tabellen über Spaltennamen direkt darauf zugreifen, sondern wir müssen im Dokumentenbaum den Pfad finden, der zur richtigen Position führt.

Beispiel: Kunde 2 ändert bei Artikel 3007 die Anzahl auf 1.

In [Zeile 6](#) wird die Artikelnummer im Warenkorb gesucht. Da diese nur einmal im Warenkorb vorkommen kann, liefert sie uns die richtige Position. Danach wird in [Zeile 8](#) eine fröhliche Bastelarbeit betrieben. Ziel ist, innerhalb der zur Artikelnummer passenden Position die Mengenangabe zu finden⁵. Dazu wird die Position im String gesucht, wo die Artikelnummer angegeben wird. In der Ebene darüber ist dann die Mengenangabe. Zum Schluss wird die neue Mengenangabe mit `JSON_REPLACE()` an die richtige Position geschrieben (siehe [Zeile 31](#)).

```
1  mysql> SELECT 2 INTO @kunde_id;
2  mysql> SELECT 3007 INTO @artikel_id;
3  mysql> SELECT 1 INTO @menge;
4  mysql> SELECT warenkorb FROM kunde WHERE k
unde_id = @kunde_id INTO @wk;
5
6  mysql> SELECT JSON_SEARCH(@wk, 'one', @art
ikel_id) INTO @path;
```

```

7
8  mysql> SELECT CONCAT( -- Bastelarbeiten
9      -> SUBSTRING(
10     -> @path,
11     -> 2,
12     -> 22+LOCATE('.', LEFT(@path, 13))
13     -> ),
14     -> '.position.menge'
15     -> )
16     -> INTO @path_to_pos;
17
18  mysql> UPDATE kunde
19     -> SET warenkorb = JSON_REPLACE(@wk, @
path_to_pos, @menge)
20     -> WHERE kunde_id = @kunde_id
21     -> ;
22
23  mysql> SELECT JSON_PRETTY(warenkorb) FROM
kunde WHERE kunde_id = @kunde_id\G
24  ***** 1. row *****
*****
25  JSON_PRETTY(warenkorb): {
26      "warenkorb": {
27          "kunde_id": 2,
28          "positionen": [
29              {
30                  "position": {
31                      "menge": 1, ←Hier
32                      "artikel": {
33                          "artikel_id": "3007",
34                          "bezeichnung": "Tinte (blau)",
35                          "einzelpreis": 4.17
36                      },
37                      "währung": "EUR",
38                      "gesamtpreis": 41.7
39                  }
40              },
41  [...]
```

Und was ist neu?

- `JSON_SEARCH()`: Diese Funktion liefert den Pfad zu einem Suchstring zurück und hat drei Parameter: das JSON-Dokument, die Angabe 'one' oder 'all' und den gesuchten String. Bei 'one' wird der erste Fund zurückgeliefert und bei 'all' alle gefundenen Stellen.
- `JSON_REPLACE()`: Diese Funktion ersetzt einen vorhandenen Wert durch einen neuen. Auch diese Funktion hat drei Parameter: das JSON-Dokument, den Pfad im Dokument und den neuen Wert.



Hinweis: Böses Foul: Die Funktion `JSON_SEARCH()` findet nur Strings; numerische Werte werden nicht gefunden. Ein Workaround ist mir nicht bekannt!



Aufgabe 17.6: Erweitern Sie den Quelltext so, dass auch der Artikel 3010 0-mal und 9010 zehnmal bestellt wird. Das Ergebnis müsste so aussehen:

```
mysql> SELECT warenkorb FROM kunde WHERE k
unde_id = @kunde_id\G
***** 1. row *****
*****
warenkorb: {"warenkorb": {"kunde_id": 2, "
positionen": [{"position": {"menge":
    1, "artikel": {"artikel_id": "3007",
"bezeichnung": "Tinte (blau)", "
    einzelpreis": 4.17}, "gesamtpreis": 4
1.7, "währung": "EUR"}}, {"position":
    {"menge": 0, "artikel": {"artikel_id"
: "3010", "bezeichnung": "Feder", "
    einzelpreis": 5.05}, "gesamtpreis": 1
```

```
0.1, "währung": "EUR"}}, {"position":  
  {"menge": 10, "artikel": {"artikel_id  
": "9010", "bezeichnung": "Schaufel",  
  "einzelpreis": 15.1}, "gesamtpreis":  
75.5, "währung": "USD"}}], "warenwerte  
": []}}
```

Aktualisierung von Gesamtpreis und Warenwert

Und jetzt sollte es auch kein Problem mehr sein, die oben ausgesparten Aktualisierungen (Gesamtpreis und Warenwert) zu ermitteln.

```
1  mysql> SELECT 2 INTO @kunde_id;  
2  mysql> SELECT 3007 INTO @artikel_id;  
3  mysql> SELECT 1 INTO @menge;  
4  
5  mysql> SELECT warenkorb FROM kunde WHERE k  
unde_id = @kunde_id INTO @wk;  
6  mysql> SELECT JSON_SEARCH(@wk, 'one', @art  
ikel_id) INTO @path;  
7  mysql> SELECT CONCAT(  
8      -> SUBSTRING(  
9      -> @path,  
10     -> 2,  
11     -> 22+LOCATE('.', LEFT(@path, 13))  
12     -> ),  
13     -> '.position.menge'  
14     -> )  
15     -> INTO @path_to_pos;  
16  
17  mysql> SELECT CONCAT(  
18     -> SUBSTRING(  
19     -> @path,
```



```

20      -> 2,
21      -> 22+LOCATE('.', LEFT(@path, 13))
22      -> ),
23      -> '.position.gesamtpreis'
24      -> )
25      -> INTO @path_to_gesamtpreis;
26
27  mysql> SELECT '$.warenkorb.warenwerte' INT
0 @path_to_warenwerte;
28
29  mysql> CREATE TEMPORARY TABLE wk
30      -> SELECT * FROM JSON_TABLE(
31      -> @wk,
32      -> '$.warenkorb.positionen[*]',
33      -> COLUMNS (
34      -> posnr FOR ORDINALITY,
35      -> artikel_id INT PATH '$.position.art
ikel.artikel_id',
36      -> einzelpreis DECIMAL(12,2) PATH '$.p
osition.artikel.einzelpreis',
37      -> menge INT PATH '$.position.menge',
38      -> währung CHAR(3) PATH '$.position.wä
hrung'
39      -> )
40      -> ) AS t
41      -> ;
42
43  mysql> SELECT einzelpreis*menge
44      -> FROM wk
45      -> WHERE artikel_id = @artikel_id
46      -> INTO @gesamtpreis
47      -> ;
48
49  mysql> SELECT JSON_ARRAYAGG(warenwert)
50      -> FROM
51      -> (SELECT JSON_OBJECT(währung, SUM(ei
nzelpreis*menge)) warenwert
52      -> FROM wk

```

```

53      -> GROUP BY währung) AS a
54      -> INTO @warenwerte
55      -> ;
56
57  mysql> UPDATE kunde
58      -> SET warenkorb = JSON_REPLACE(
59      -> @wk,
60      -> @path_to_pos, @menge,
61      -> @path_to_gesamtpreis, @gesamtpreis,
62      -> @path_to_warenwerte, CAST(@warenwer
63  te AS JSON)
64      -> )
65      -> WHERE kunde_id = @kunde_id
66      -> ;

```



Aufgabe 17.7: Überprüfen Sie mit `JSON_PRETTY()` das Ergebnis und versuchen Sie den Quelltext in Prosa zu formulieren. Machen Sie dabei keinen auf *SQL→Deutsch-Übersetzer*, sondern formulieren Sie die Aufgabe und den Zweck der jeweiligen Anweisung. Warum – beispielsweise – verwende ich hier eine temporäre Tabelle?

17.4.5 Inhalte des Warenkorbs löschen

Gehen wir davon aus, dass ein Kunde auf der Client-Oberfläche folgende Dinge tut: Der Kunde löscht eine Warenkorbposition. Gelöscht wird mit den Funktionen `JSON_REMOVE()`, `JSON_REPLACE()` oder `JSON_SET()`.

In [Zeile 6](#) wird wieder die Stelle im JSON-Dokument gesucht, wo die Information steht. Dabei suche ich den ersten Punkt hinter dem Arrayelement und schneide den Rest ab.

Anschließend ersetze ich die Mengenangabe und speichere das Ergebnis zurück in die Tabelle.

```
1  mysql> SELECT 2 INTO @kunde_id;
2  mysql> SELECT 9010 INTO @artikel_id;
3  mysql> SELECT warenkorb FROM kunde WHERE k
unde_id = @kunde_id INTO @wk;
4  mysql> SELECT JSON_SEARCH(@wk, 'one', @art
ikel_id) INTO @path;
5
6  mysql> SELECT SUBSTRING(@path, 2, 22+LOCAT
E('.', LEFT(@path, 13)))
7      -> INTO @path_to_pos;
8
9  mysql> UPDATE kunde
10      -> SET warenkorb = JSON_REMOVE(@wk, @p
ath_to_pos)
11      -> WHERE kunde_id = @kunde_id
12      -> ;
```

Die Warenkorbposition ist verschwunden:

```
1  JSON_PRETTY(warenkorb): {
2      "warenkorb": {
3          "kunde_id": 2,
4          "positionen": [
5              {
6                  "position": {
7                      "menge": 1,
8                      "artikel": {
9                          "artikel_id": "3007",
10                         "bezeichnung": "Tinte ( blau)"
11                     },
12                     "einzelpreis": 4.17
13                 },
14                 "währung": "EUR",
15                 "gesamtpreis": 41.7
16             }
17         ]
18     }
19 }
```

```

16     },
17     {
18         "position": {
19             "menge": 2,
20             "artikel": {
21                 "artikel_id": "3010",
22                 "bezeichnung": "Feder",
23                 "einzelpreis": 5.05
24             },
25             "währung": "EUR",
26             "gesamtpreis": 10.1
27         }
28     },
29 ],
30 "warenwerte": []
31 }
32 }

```



Aufgabe 17.8: Programmieren Sie, dass der gesamte Warenkorb des Kunden geleert wird.

Was ist neu?

- **JSON_SET():** Diese Funktion erstellt einen neuen Inhalt oder ersetzt einen vorhandenen durch einen neuen. Sie ist damit sehr mit **JSON_REPLACE()** verwandt. Im Gegensatz zu dieser wird entweder ein vorhandener Wert ersetzt oder es wird ein neues Wert angelegt. Sie benötigt drei Parameter: das JSON-Dokument, den Pfad zum Wert und den neuen Wert.
- **JSON_REMOVE():** Diese Funktion löscht das ganze Element aus dem Dokument. Es wird also nicht der Inhalt eines Elements geleert, sondern das ganze Element wird entfernt. Sie hat zwei Parameter: das JSON-Dokument

und den Pfad zum Element.

¹ Abkürzung für: **Not only SQL**

² Abkürzung für: **JavaScript Object Notation**

³ Eigentlich sind es Listen. Arrays bestehen aus gleichartigen Objekten, aber in vielen modernen Programmiersprachen – besonders bei nicht typisierten – sind Arrays und Listen Synonyme.

⁴ Das riecht geradezu nach einem Trigger (siehe [Kapitel 22](#)).

⁵ In einem Python- oder JavaScript- Client gestaltet sich dies viel eleganter. Aber was soll's, es klappt.

TEIL V

Anweisungen kapseln

18 Locking



Bitte nicht alle gleichzeitig; immer der Reihe nach.

- Grundkurs
 - Zeilen-, Seiten- und Tabellensperre



Die Quelltexte dieses Kapitels stehen in der Datei `listing14.sql` (siehe [Listing 28.16](#) und [Listing 28.39](#)).

Eine Datenbank wird in vielen Fällen für Anwendungen, die auf mehreren Clients laufen, entwickelt. Denken Sie an ein Online-Forum, einen Internetshop, ein Zeiterfassungssystem etc. Dabei kommt es oft vor, dass zwei Anwendungen gleichzeitig auf eine Datenbank, eine Tabelle, eine Zeile oder einen Spalteninhalt zugreifen. Tun beide dies nur lesend, haben Sie erst mal kein Problem. Tun es beide oder auch nur eine von ihnen auch schreibend, haben Sie ein Problem. Konkret: Über eine offene Sitzung unseres Online-Shops werden fünf Spaten bestellt und über eine andere drei Spaten. Kein Problem, oder?

```

1  mysql> SELECT * FROM lagerbestand WHERE ar
artikel_id = 9015;
2  +-----+-----+-----+
---+-----+
3  | artikel_id | menge_mindest | menge_aktue
ll | deleted |
4  +-----+-----+-----+
---+-----+
5  | 9015 | 10.000000 | 25.000000 | 0 |
6  +-----+-----+-----+
---+-----+
7
8  mysql> UPDATE lagerbestand
9  -> SET menge_aktuell = menge_aktuell - 5 W
HERE artikel_id = 9015;
10
11 mysql> UPDATE lagerbestand
12 -> SET menge_aktuell = menge_aktuell - 3 W
HERE artikel_id = 9015;
13
14 mysql> SELECT * FROM lagerbestand WHERE ar
tikel_id = 9015;
15 +-----+-----+-----+
---+-----+
16 | artikel_id | menge_mindest | menge_aktue
ll | deleted |
17 +-----+-----+-----+
---+-----+
18 | 9015 | 10.000000 | 17.000000 | 0 |
19 +-----+-----+-----+
---+-----+

```

Dieser Selbstbetrug geht aber davon aus, dass diese beiden Aktualisierungen nacheinander ausgeführt werden. Aber warum sollten die beiden Sitzungen nicht fast gleichzeitig diese Aktualisierungen durchführen?

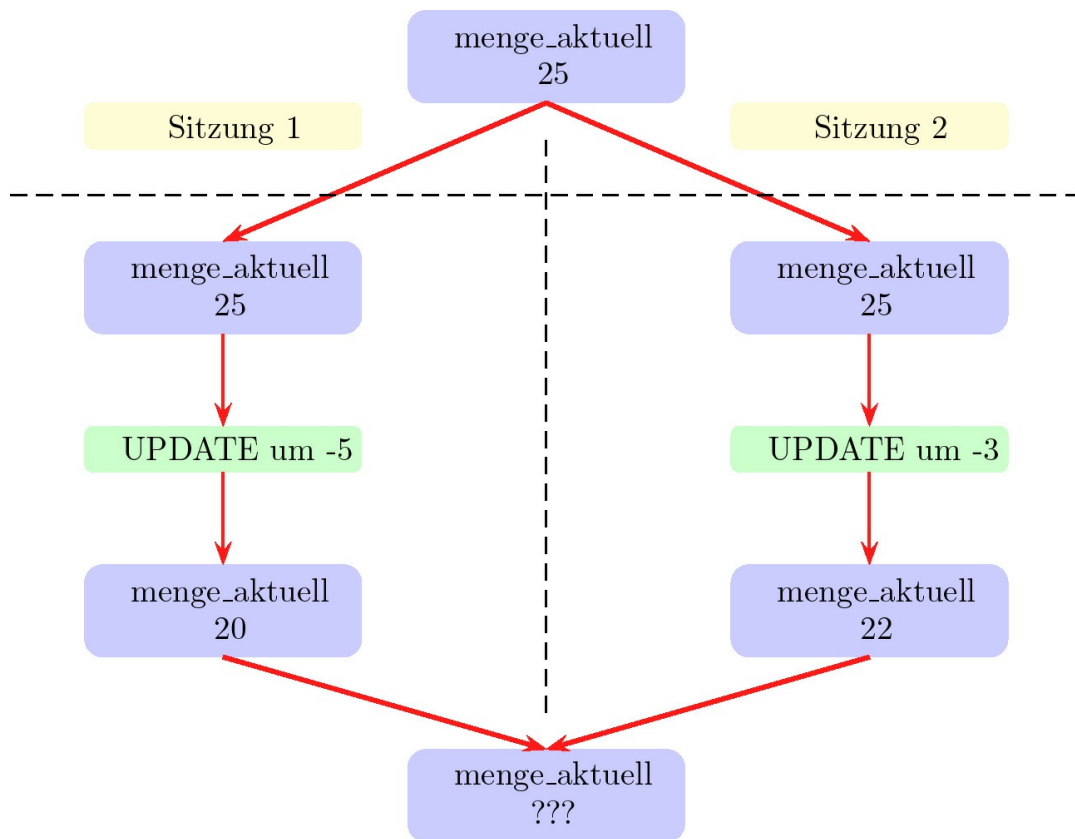


Bild 18.1 Was gilt jetzt?

In [Bild 18.1](#) wird das Phänomen des *lost update* verdeutlicht. Beide Sitzungen melden fast gleichzeitig, dass sie eine Aktualisierung des Lagerbestands machen wollen. Beide erhalten den aktuellen Lagerbestand zwecks Veränderung. Beide verändern ihre Bestände und wollen das Ergebnis in die Tabelle zurückschreiben. Egal, wer zuerst wegschreibt, das Ergebnis ist falsch!

Wir brauchen einen Mechanismus, der einer Sitzung sagt, dass sie einen Moment auf die andere warten soll. Diesen Mechanismus nennt man *sperr*¹. Derzeit werden drei verschiedene Sperrstrategien verwendet:

- *Tabellensperre (table lock)*: Die gesamte Tabelle ist für die Dauer einer Änderung für andere gesperrt.

- *Seitensperre (page lock)*: Falls nicht die gesamte Tabelle betroffen ist, werden die Daten einer Tabelle in der Regel *seitenweise* eingelesen. Eine Seite hat meist eine Größe von 2 bis 16 KBytes. Alle Zeilen der Tabelle, die in der entsprechenden Seite sind, werden dabei für andere gesperrt. Die Zeilen dieser Seite müssen nicht in der Tabelle hintereinander liegen.
- *Zeilensperre (row lock)*: Die einzelne oder die betreffenden Zeilen werden für andere gesperrt.



Hinweis: In MySQL und MariaDB unterstützt die InnoDB bzw. XtraDB die Zeilen- und MyISAM die Tabellensperre.



Hinweis: MySQL, MariaDB, PostgreSQL und T-SQL unterscheiden sich bei den nachfolgenden Befehlen oft in der Syntax und den möglichen Optionen. Daher möchte ich mich auf eine minimale MySQL-Syntax beschränken.

Normalerweise sollten Sie das Sperren der Engine bzw. dem Server überlassen. Dort sind Anwendungsregeln definiert, wann welche Sperre einzusetzen ist. Sie können in MySQL oder MariaDB aber auch eine Tabelle explizit mit `LOCK TABLES` sperren²:



MySQL/MariaDB

`LOCK TABLES`

`tabellenname [[AS] alias] sperrtyp[, ta
 bellennamen [[AS] alias] sperrtyp]*`

```
;
sperrtyp:
    READ [LOCAL] | [LOW_PRIORITY] WRITE
```

Der *sperrtyp* READ sagt aus, dass ich aus dieser Tabelle etwas lesen möchte. Alle anderen Sitzungen können diese Tabelle ebenfalls noch lesen, ja sogar auch eine READ-Sperre auslösen. Keine der Sitzungen, auch nicht die, welche diese Sperrung zuerst ausgelöst hat, kann jetzt in die Tabelle schreiben. Wird die Option LOCAL angegeben, können andere Sitzungen neue Datensätze einfügen, sofern diese nicht zu einem Konflikt mit der Leseoperation führen. In MySQL setzt die InnoDB READ mit READ LOCAL gleich.

Der zweite *sperrtyp* WRITE erlaubt es, dass die Sitzung, die die Sperre durchführt, auch in die Tabelle schreiben darf. Alle anderen Sitzungen müssen warten, bis die Sperre wieder aufgehoben wird. Selbst lesende Zugriffe sind jetzt nicht mehr erlaubt. Die Option LOW_PRIORITY kann verwendet werden, wenn der Schreibzugriff lange dauert und die Serverressourcen für Zugriffe auf andere Tabellen gebraucht werden.

Die Sperrung wird mit UNLOCK TABLES aufgehoben.



MySQL/MariaDB

```
UNLOCK TABLES
```

```
;
```



Hinweis: Wenn in einer Sitzung Tabellen gesperrt wurden, kann in dieser Sitzung nur auf diese Tabellen

zugegriffen werden.

Äh, wie bitte? Das wollen wir uns mal genauer anschauen. Zuerst wird die Tabelle `artikel` gesperrt ([Zeile 2](#)). Eine Auswertung der Tabelle ist problemlos möglich ([Zeile 7](#)). Der Versuch, auf eine andere Tabelle zuzugreifen, schlägt dann aber fehl ([Zeile 16](#)).

Entweder sperrt man gleich mehrere Tabellen in [Zeile 2](#), oder man gibt die Tabelle wieder frei ([Zeile 20](#)).

```
1  mysql> LOCK TABLES
2      -> artikel WRITE
3      -> ;
4  Query OK, 0 rows affected (0.00 sec)
5
6  mysql> SELECT COUNT(*)
7      -> FROM artikel
8      -> ;
9  +-----+
10 | COUNT(*) |
11 +-----+
12 | 9 |
13 +-----+
14
15 mysql> SELECT COUNT(*)
16 -> FROM warengruppe
17 -> ;
18 ERROR 1100 (HY000): Table 'warengruppe' wa
s not locked with LOCK TABLES
19
20 mysql> UNLOCK TABLES
21 -> ;
22 Query OK, 0 rows affected (0.00 sec)
23
24 mysql> SELECT COUNT(*)
```

```

25  -> FROM warengruppe
26  -> ;
27  +-----+
28  | COUNT(*) |
29  +-----+
30  | 4 |
31  +-----+

```



Hinweis: Zum Locking gehört auch das Problem des *Deadlocks*. Mehr dazu in [Abschnitt 19.5](#).



Aufgabe 18.1: Erweitern Sie das Experiment so, dass nun mehrere Tabellen gleichzeitig gelockt werden, und testen Sie dann den Zugriff auf die gesperrten und ungesperrten Tabellen.

¹ engl.: *locking* oder *to lock*

² Da es keinen SQL:2023-Standard zum Locking gibt, sind zwar die Strategien vergleichbar, aber die einzelnen Befehle und ihre Parameter oft sehr unterschiedlich. In den PostgreSQL-Varianten der Quelltextlistings habe ich – soweit sinnvoll – die MySQL- bzw. MariaDB-Beispiele überführt, um einen syntaktischen Vergleich zu ermöglichen.

19 Transaktionen



Wo zwei oder drei auf meinem Server versammelt sind, da ist die Transaktion mitten unter ihnen.

- Grundkurs
 - Was ist eine Transaktion?
 - Deadlock
 - Start einer Transaktion mit `START TRANSACTION`
 - Impliziter Start einer Transaktion
 - Transaktion mit `COMMIT` abschließen
 - Transaktion mit `ROLLBACK` abschließen
- Vertiefendes
 - ACID-Test
 - Isolationsebene `READ UNCOMMITTED`
 - Isolationsebene `READ COMMITTED`
 - Isolationsebene `REPEATABLE READ`
 - Isolationsebene `SERIALIZABLE`
 - Transaktion aus einem C#-Client heraus



Die Quelltexte dieses Kapitels stehen in der Datei `listing15.sql` (siehe [Listing 28.17](#) und [Listing 28.40](#)).

19.1 Das Problem

Die Auswertungen und Änderungen, die wir bisher durchgeführt haben, sind mehr oder weniger atomar gewesen. Sie standen für sich alleine und hatten kaum Abhängigkeiten mit anderen Aufgaben. Dies ist aber nur der einfache Fall. Oft sind SQL-Anweisungen inhaltlich zusammenhängend und müssen zwingend *ganz oder gar nicht* ausgeführt werden.

Beispiel: Wir legen über eine Oberfläche den neuen Artikel Säge an. Gleichzeitig können wir über Checkboxes angeben, dass er zu den Warengruppen Gartenbedarf und Werkzeug gehört. Wenn wir jetzt auf OK drücken, müssen zwei Dinge passieren: In der Tabelle `artikel` muss der neue Artikel angelegt und in der Tabelle `artikel_nm_warengruppe` müssen die beiden Verknüpfungen erstellt werden.

Beide Anweisungen sollten kein Problem darstellen, oder?

```
1  mysql> INSERT INTO artikel
2      -> (bezeichnung, einzelpreis, waehrung
3      -> VALUES
4      -> ('Säge', 17.85, 'EUR');
5
6  mysql> INSERT INTO artikel_nm_warengruppe
7      -> VALUES
8      -> (3, ???)
9      -> ,(4, ???);
```

Da ist auf einmal eine Frage aufgetaucht. Da wir im INSERT auf die Tabelle `artikel` keine `artikel_id` vorgegeben haben, greift hier der `AUTO_INCREMENT`. Wir wissen aber nicht, welche Nummer dabei vergeben wurde. Wie soll man dann den zweiten INSERT ausführen?

Da hilft die MySQL-/MariaDB-Funktion `LAST_INSERT_ID()`. Sie liefert mir den Wert, der zuletzt durch ein `AUTO_INCREMENT` erzeugt wurde.

```
1  mysql> INSERT INTO artikel
2      -> (bezeichnung, einzelpreis, waehrung
3      -> VALUES
4      -> ('Säge', 17.85, 'EUR');
5
6  mysql> SET @id = LAST_INSERT_ID();
7
8  mysql> INSERT INTO artikel_nm_warengruppe
9      -> VALUES
10     -> (3, @id)
11     -> ,(4, @id);
```

Man stelle sich aber mal vor, dass diese Anweisungen nicht *alle* ausgeführt werden. Ursachen kann es viele geben. Wenn wir uns im ersten Befehl vertippt haben, könnte das Ergebnis so aussehen:

```
1  mysql> INSERT ITO artikel[label={lst15_003
2      -> (bezeichnung, einzelpreis, waehrung
3      -> VALUES
4      -> ('Säge', 17.85, 'EUR');
5  ERROR 1064 (42000): You have an error in y
our SQL syntax; check the manual that correspo
```



```

nds to your MySQL server version for the right
syntax to use near 'artikel
6
7  mysql> SET @id = LAST_INSERT_ID();
8
9  mysql>
10 mysql> INSERT INTO artikel_nm_warengruppe
11      -> VALUES
12      -> (3, @id)
13      -> ,(4, @id);
14 ERROR 1452 (23000): Cannot add or update a
   child row: a foreign key constraint fails ('o
shop'. 'artikel_nm_warengruppe', CONSTRAINT 'a
rtikel_nm_warengruppe_ibfk_2' FOREIGN KEY ('ar
tikel_id') REFERENCES 'artikel' ('artikel_id')
)

```

Ein Haufen Fehlermeldungen, und glücklicherweise verhindert der Constraint, dass unsinnige Werte in die Hilfstabelle eingefügt werden.

Nehmen wir ein anderes Beispiel: Sam Gamdschie hat einen Spaten bestellt, und dieser wird jetzt versendet. Dann müssen zwei Dinge passieren: Erstens muss ein Spaten aus dem Lagerbestand entfernt und zweitens der Status der Bestellung auf versendet gesetzt werden.

```

1  UPDATE lagerbestand
2      SET menge_aktuell = menge_aktuell - 1
3      WHERE artikel_id = 9015;
4
5  UPDATE bestellung
6      SET status = 'versendet'
7      WHERE bestellung_id = 1;

```

Was würde passieren, wenn eine der beiden Anweisungen fehlschlägt und die andere trotzdem ausgeführt wird? In

einem Fall hätten wir einen Fehlbestand im Lager, und im anderen Fall würden wir vermutlich die Bestellung noch einmal versenden. Beides bedeutet einen wirtschaftlichen Schaden.

19.2 Was ist eine Transaktion?



Hinweis: SQL:2023, MySQL, MariaDB, PostgreSQL und T-SQL unterscheiden sich bei den nachfolgenden Befehlen oft in der Syntax und den möglichen Optionen. Ich beschränke mich daher auf eine minimale (MySQL-)Syntax.

Eine Transaktion beginnt automatisch, wenn die Sitzung im AUTOCOMMIT-Modus ist oder der Befehl `START TRANSACTION` abgesendet wird. Beendet wird diese im AUTOCOMMIT-Modus nach jedem einzelnen Befehl bzw. bei `START TRANSACTION` durch `COMMIT` oder `ROLLBACK`.



SQL:2023/PostgreSQL	MySQL/MariaDB	T-SQL

<code>START TRANSACTION</code>	<code>START TRANSACTION BEGIN TRANSACTION [WORK]</code>	<code>BEGIN TRANSACTION</code>

<code>[isolationsebene]</code>		

<code>;</code>		

	<code>;</code>	

		<code>;</code>

isolationsebene:

READ UNCOMMITTED	READ COMMITTED	REPEATABLE READ
SERIALIZABLE		

COMMIT	COMMIT	COMMIT

;	;	;

ROLLBACK	ROLLBACK	ROLLBACK

;	;	;

Bevor ich auf verschiedene Aspekte der Theorie eingehe, möchte ich den grundsätzlichen Ablauf einer Transaktion anhand eines Schaubildes verdeutlichen (siehe [Bild 19.1](#)).

Der Ablauf sieht einigermaßen kompliziert aus, ist er aber nicht. Zuerst muss klargestellt werden, dass eine Transaktion zwar auf dem Server ausgeführt, aber durch einen Client gesteuert wird.

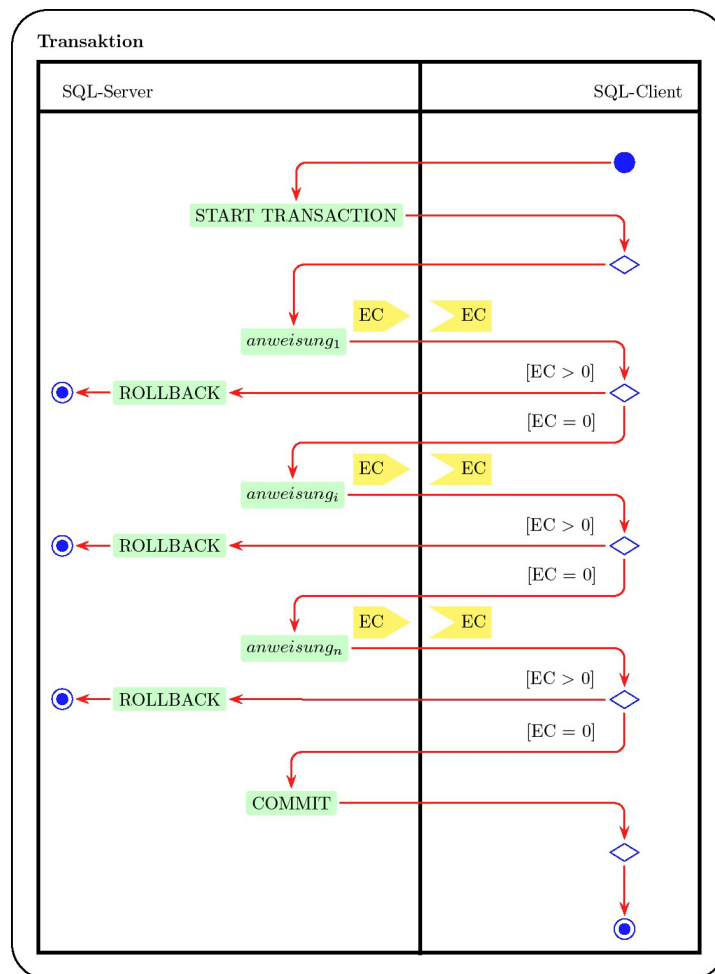


Bild 19.1 Ablauf einer Transaktion

- **Phase Transaktionsbeginn:** Der Client baut eine Verbindung auf und sendet den Befehl START TRANSACTION an den Server. Dieser richtet daraufhin eine Transaktionsumgebung ein. Was das genau ist, erkläre ich später. Im Prinzip läuft es aber darauf hinaus, dass Sie zu dem Datenbankzustand zurückkehren können, der bei START TRANSACTION bestand.
- **Phase Transaktionsausführung:** Der Client sendet die erste Anweisung an den Server. Diese ist meist eine datenverändernde, denn bei Transaktionen kommt es in der Regel auf datenverändernde Anweisungen an. Der Server antwortet mit einer Exception oder einem ERROR

CODE (EC). Der Client entscheidet anhand der Fehlermeldung¹, ob er weitermachen möchte oder die Verarbeitung abgebrochen werden soll.

- Will er abbrechen, so sendet er einen ROLLBACK an den Server. Der Server stellt den ursprünglichen Zustand wieder her, und die Transaktion ist beendet.
- Will er nicht abbrechen, so sendet er die nächste Anweisung an den Server.

- **Phase Transaktionsabschluss:** Alle Anweisungen sind wunschgemäß ausgeführt worden. Der Client hat dies überprüft und sendet deshalb ein COMMIT an den Server. Erst jetzt werden die Änderungen tatsächlich in die Tabellen überführt.

Was soll das heißen, erst jetzt werden die Änderungen in die Tabellen überführt? Wo waren sie denn vorher? Das lässt sich nun nicht so einfach beantworten. Hier brauchen wir ein wenig Theorie.



Definition 61: Transaktion

Einen Anweisungsblock, der den ACID-Anforderungen entspricht, nennt man *Transaktion*.

ACID ist die Abkürzung für *Atomicity* (Atomarität), *Consistency* (Konsistenz), *Isolation* und *Durability* (Dauerhaftigkeit).

- *Atomicity* (Atomarität): Der altgriechische Ursprung des Wortes soll ausdrücken, dass etwas nicht weiter aufteilbar ist². Bezüglich der Transaktionen ist damit gemeint, dass man die Anweisungen der Transaktion nicht aufteilen kann. Entweder werden alle vollständig ausgeführt und umgesetzt oder keine³. Durch die Atomarität erscheint

eine Transaktion nach außen wie eine Anweisung.

- *Consistency (Konsistenz)*: Die Datenbank ist immer in einem konsistenten Zustand. Wird die Verarbeitung der Transaktion unter- oder abgebrochen, sind die Inhalte der Tabellen immer noch fachlich und technisch integer. Wie wird das hier erreicht? Durch das COMMIT. Würde die Verarbeitung der Transaktion nicht bis zum Ende durchlaufen, erfolgt kein COMMIT. Wenn die Datenbank vor der Transaktion in einem konsistenten Zustand war, so wird sie das nach einen ROLLBACK auch wieder sein. Nicht verhindert werden kann, dass innerhalb der Transaktion durch falsche Programmierung ein nicht konsistenter Zustand erreicht wird.
- *Isolation*: Da die Konsistenz der Datenbank während der Ausführung einer Transaktion nicht garantiert wird, sollten andere auch nicht auf diesem inkonsistenten Zustand arbeiten. Aus diesem Grund sind die Ergebnisse der Anweisungen innerhalb der Transaktion bis zum COMMIT für andere unsichtbar. Wie *unsichtbar* hier *unsichtbar* ist, wird mit den sogenannten *Isolationsebenen* (s.u.) festgelegt.
- *Durability (Dauerhaftigkeit)*: Während die Ergebnisse der Anweisungen innerhalb der Transaktion *in der Schwebe sind*, sollten nach einem COMMIT die Änderungen wirklich und tatsächlich abgespeichert sein. Ein Systemabsturz oder Ähnliches sollte das Vorhandensein der Daten nicht mehr gefährden. Auch gibt es verschiedene Grade von Dauerhaftigkeit.



Hinweis: Unterschiede MySQL/MariaDB und SQL:2023:

- Damit Sie in MySQL oder MariaDB Transaktionen verwenden können, muss die Engine

transaktionsfähig sein. Die InnoDB bzw. XtraDB ist transaktionsfähig.

- In SQL:2023 wird eine neue Transaktion implizit nach Abschluss einer Transaktion gestartet.
- In MySQL oder MariaDB wird eine neue Transaktion implizit nach Abschluss einer Transaktion gestartet, wenn der AUTOCOMMIT-Modus ausgeschaltet ist.
- Ist in MySQL oder MariaDB der AUTOCOMMIT-Modus eingeschaltet, wird eine Transaktion mit `START TRANSACTION` begonnen. Während der Transaktion bleibt der AUTOCOMMIT-Modus bis zu einem `COMMIT` oder einem `ROLLBACK` ausgeschaltet.

19.3 Isolationsebenen

Eine Forderung im ACID war die Isolation. Dabei werden die Zwischenstände einer Transaktion vor anderen Transaktionen bzw. Sessions verborgen, um zu gewährleisten, dass diese immer nur einen konsistenten Datenzustand vorfinden.

Wenn Sie nichts eingestellt haben, wird Ihr Server vermutlich im Modus AUTOCOMMIT laufen. Das bedeutet, dass jede Anweisung implizit von einem COMMIT abgeschlossen wird. Wir haben es dann mit lauter Transaktionen mit nur einer Anweisung zu tun. Um herauszufinden, ob meine Session im AUTOCOMMIT-Modus läuft, müssen wir den Inhalt der Variablen AUTOCOMMIT auslesen:

```
SHOW VARIABLES LIKE 'AUTOCOMMIT';
```

Die Werte ON und 1 haben die gleiche Bedeutung. Um mit den Transaktionen starten zu können, sollte der erste Befehl der Session den AUTOCOMMIT-Modus ausschalten oder der Beginn einer Transaktion explizit mit `START TRANSACTION` eingeleitet werden:

```
SET AUTOCOMMIT = OFF;
```

Jetzt gilt, dass wir immer in einer Transaktion sind, bis wir `COMMIT` oder `ROLLBACK` anweisen. Allerdings wird dann sofort eine neue Transaktion gestartet. Bei nicht transaktionsfähigen Engines wird der Wert der Variablen `AUTOCOMMIT` ignoriert.



Hinweis: Es gibt Anweisungen, die implizit eine Transaktion beenden. Dies sind meist Befehle der DDL (siehe [Abschnitt 26.4.1](#)) wie `ALTER TABLE`.

Die Isolationsebene wird mit diesem Befehl festgelegt:



MySQL/MariaDB

```
SET [GLOBAL|SESSION] TRANSACTION ISOLATION
    LEVEL isolationsebene
;
isolationsebene:

    READ UNCOMMITTED|READ COMMITTED|REPEATABLE
    READ|SERIALIZABLE
```

Mit Hilfe der Optionen `GLOBAL` und `SESSION` können Sie die Lebensdauer und Reichweite der Isolationsebene festlegen.

Wenn Sie die Option GLOBAL verwenden, werden alle Sitzungen, die danach gestartet werden, mit der angegebenen Isolationsebene versehen. Alle aktuell laufenden bleiben davon unberührt. Anders bei SESSION: Nur in der laufenden Sitzung werden für alle nachfolgenden Transaktionen die Isolationsebene bestimmt. Interessanterweise hat das Weglassen der beiden Optionen eine eigene Bedeutung: Die angegebene Isolationsebene wird nur auf die *eine* nachfolgende Transaktion angewendet – deshalb darf diese Variante auch nicht innerhalb einer Transaktion angewendet werden.

Welcher Isolationsebene serverseitig eingestellt ist, können Sie wie folgt herausfinden:

```
SHOW VARIABLES LIKE '%ISOLATION%';
```

19.3.1 READ UNCOMMITTED

Es handelt sich eigentlich nicht um eine Isolationsebene, da die anderen Transaktionen oder Sessions den Zwischenstand lesen können. Da dabei auch inkonsistente Zustände auftreten können, wird dieser auch *dirty read* genannt.⁴

SESSION 1

```
1 START TRANSACTION;
2
3
4 SELECT artikel_id, menge_aktuell
5 FROM lagerbestand
6 WHERE artikel_id = 9015;
7 +-----+-----+
8 | artikel_id | menge_aktuell |
9 +-----+-----+
```

```

10 | 9015 | 17.000000 |
11 +-----+-----+
12
13 UPDATE lagerbestand
14 SET menge_aktuell = menge_aktuell - 1
15 WHERE artikel_id = 9015;
16
17 SELECT artikel_id, menge_aktuell
18 FROM lagerbestand
19 WHERE artikel_id = 9015;
20 +-----+-----+
21 | 9015 | 16.000000 |
22 +-----+-----+
23
24 ROLLBACK;
25
26 SELECT artikel_id, menge_aktuell
27 FROM lagerbestand
28 WHERE artikel_id = 9015;
29 +-----+-----+
30 | 9015 | 17.000000 |
31 +-----+-----+

```

SESSION 2

```

1 SET SESSION TRANSACTION ISOLATION LEVEL
2 READ UNCOMMITTED;
3
4 SELECT artikel_id, menge_aktuell
5 FROM lagerbestand
6 WHERE artikel_id = 9015;
7 +-----+-----+
8 | artikel_id | menge_aktuell |
9 +-----+-----+
10 | 9015 | 17.000000 |
11 +-----+-----+
12
13

```

```

14
15
16
17 SELECT artikel_id, menge_aktuell
18 FROM lagerbestand
19 WHERE artikel_id = 9015;
20 +-----+-----+
21 | 9015 | 16.000000 |
22 +-----+-----+
23
24
25
26 SELECT artikel_id, menge_aktuell
27 FROM lagerbestand
28 WHERE artikel_id = 9015;
29 +-----+-----+
30 | 9015 | 17.000000 |
31 +-----+-----+

```

Um dieses Beispiel nachzubauen, sollten Sie `listing14.sql` ausgeführt und zwei Instanzen des MySQL oder MariaDB Clients gleichzeitig vor sich haben. In Session 2 wird dafür gesorgt, dass diese einen *dirty read* auf die Datenbank durchführen kann, und bei beiden Sessions wird der Datenbestand überprüft.

In Session 1 wird ab [Zeile 13](#) der Lagerbestand der Spaten aktualisiert. Bitte beachten Sie, dass dies in der zweiten Session nicht passiert. In [Zeile 17](#) wird jetzt in beiden Sessions wieder der Lagerbestand überprüft. Beide geben nun das gleiche Ergebnis aus, obwohl in Session 1 noch kein COMMIT erfolgt ist. Session 2 arbeitet also möglicherweise mit inkonsistenten Daten.

An diesem Beispiel können Sie auch den Effekt von ROLLBACK beobachten. In Session 1 [Zeile 24](#) wird der ROLLBACK angewiesen. Eine anschließende Überprüfung zeigt, dass in

beiden Sessions wieder der ursprüngliche Zustand hergestellt wurde.

Es gibt eigentlich keinen guten Grund, das READ UNCOMMITTED zu verwenden. In [SZT+09], Seite 9, steht, dass ein *dirty read* keinen nennenswerten Performancevorteil bietet und deshalb in der Praxis selten eingesetzt wird.

19.3.2 READ COMMITTED

Bei dieser Isolationsebene werden nur die Daten in einer Transaktion berücksichtigt, die durch andere mit einem COMMIT bestätigt wurden.

SESSION 1

```
1 START TRANSACTION;
2
3
4 SELECT artikel_id, menge_aktuell
5 FROM lagerbestand
6 WHERE artikel_id = 9015;
7 +-----+-----+
8 | artikel_id | menge_aktuell |
9 +-----+-----+
10 | 9015 | 17.000000 |
11 +-----+-----+
12
13 UPDATE lagerbestand
14 SET menge_aktuell = menge_aktuell - 1
15 WHERE artikel_id = 9015;
16
17 SELECT artikel_id, menge_aktuell
18 FROM lagerbestand
19 WHERE artikel_id = 9015;
20 +-----+-----+
21 | 9015 | 16.000000 |
```

```

22 +-----+-----+
23
24 COMMIT;
25
26 SELECT artikel_id, menge_aktuell
27 FROM lagerbestand
28 WHERE artikel_id = 9015;
29 +-----+-----+
30 | 9015 | 16.000000 |
31 +-----+-----+

```

SESSION 2

```

1 SET SESSION TRANSACTION
2 ISOLATION LEVEL READ COMMITTED;
3
4 SELECT artikel_id, menge_aktuell
5 FROM lagerbestand
6 WHERE artikel_id = 9015;
7 +-----+-----+
8 | artikel_id | menge_aktuell |
9 +-----+-----+
10 | 9015 | 17.000000 |
11 +-----+-----+
12
13
14
15
16
17 SELECT artikel_id, menge_aktuell
18 FROM lagerbestand
19 WHERE artikel_id = 9015;
20 +-----+-----+
21 | 9015 | 17.000000 |
22 +-----+-----+
23
24
25

```

```

26 SELECT artikel_id, menge_aktuell
27 FROM lagerbestand
28 WHERE artikel_id = 9015;
29 +-----+-----+
30 | 9015 | 16.000000 |
31 +-----+-----+

```

In Session 2, [Zeile 1](#) wird die Isolationsebene auf READ COMMITTED gesetzt. Dadurch erscheinen in [Zeile 21](#) in Session 1 und Session 2 unterschiedliche Werte. In Session 1 wird mit dem veränderten und in Session 2 mit dem ursprünglichen Lagerbestand gearbeitet.

Erst nach dem COMMIT der Session 1 in [Zeile 24](#) geben beide Sitzungen wieder das gleiche – nun aktualisierte – Ergebnis aus.

Der Nachteil dieser Isolationsebene ist aber auch zu erkennen. Die gleiche Anweisung führt zu unterschiedlichen Ergebnissen. Die Ergebnisse sind somit nicht wiederholbar (*repeatable*), was zu unerwünschten Effekten innerhalb einer Transaktion führen kann.



Aufgabe 19.1: Dies führt dazu, dass in unterschiedlichen Sitzungen COUNT (*) unterschiedliche Ergebnisse ermitteln könnte, obwohl diese auf der gleichen Tabelle arbeiten. Versuchen Sie, unterschiedliche COUNT ()-Ergebnisse in zwei verschiedenen Sitzungen zu erzielen.

19.3.3 REPEATABLE READ

Das Ziel dieser Isolationsebene ist, dass in einer Transaktion

Das Ziel dieser Isolationsebene ist, dass in einer Transaktion, die selbst die Daten nicht verändert, die gleiche Auswertung das gleiche Ergebnis liefert. In MySQL und MariaDB ist diese Isolationsebene der Standard, wenn Sie eine Transaktion starten.

SESSION 1

```
1
2
3
4 START TRANSACTION;
5
6 SELECT artikel_id, menge_aktuell
7 FROM lagerbestand
8 WHERE artikel_id = 9015;
9 +-----+-----+
10 | artikel_id | menge_aktuell |
11 +-----+-----+
12 | 9015      | 16.000000    |
13 +-----+-----+
14
15 UPDATE lagerbestand
16 SET menge_aktuell = menge_aktuell - 1
17 WHERE artikel_id = 9015;
18
19 COMMIT;
20
21 SELECT artikel_id, menge_aktuell
22 FROM lagerbestand
23 WHERE artikel_id = 9015;
24 +-----+-----+
25 | artikel_id | menge_aktuell |
26 +-----+-----+
27 | 9015      | 15.000000    |
28 +-----+-----+
```

SESSION 2

```

1 SET SESSION TRANSACTION
2 ISOLATION LEVEL REPEATABLE READ;
3
4 START TRANSACTION;
5
6 SELECT artikel_id, menge_aktuell
7 FROM lagerbestand
8 WHERE artikel_id = 9015;
9 +-----+-----+
10 | artikel_id | menge_aktuell |
11 +-----+-----+
12 | 9015 | 16.000000 |
13 +-----+-----+
14
15
16
17
18
19
20
21 SELECT artikel_id, menge_aktuell
22 FROM lagerbestand
23 WHERE artikel_id = 9015;
24 +-----+-----+
25 | artikel_id | menge_aktuell |
26 +-----+-----+
27 | 9015 | 16.000000 |
28 +-----+-----+
29
30 COMMIT;
31
32 SELECT artikel_id, menge_aktuell
33 FROM lagerbestand
34 WHERE artikel_id = 9015;
35 +-----+-----+
36 | artikel_id | menge_aktuell |
37 +-----+-----+
38 | 9015 | 15.000000 |

```


In Session 2 wird in [Zeile 1](#) die Isolationsebene auf REPEATABLE READ eingestellt. Dadurch werden die Daten ab [Zeile 4](#) für diese Transaktion eingefroren. Selbst die COMMIT-Anweisung in Session 1 ([Zeile 19](#)) führt nicht dazu, dass in beiden Transaktionen die gleichen Werte gelten. Erst der COMMIT in Session 2 ([Zeile 30](#)) macht die Veränderungen von Session 1 in Session 2 sichtbar.

Somit ist gewährleistet, dass beide Sessions/Transaktionen mit in sich konsistenten Datenbeständen arbeiten.

19.3.4 SERIALIZABLE

Das Ziel der Isolationsebene REPEATABLE READ ist, dass innerhalb einer Transaktion die Daten nicht von außen verändert werden können. Trotzdem sind sogenannte *phantom reads* möglich. Nehmen wir mal an, dass in Session 1 neue Zeilen mit COMMIT in die Tabelle `artikel` hinzugefügt werden und in Session 2 Auswertungen auf der Tabelle stattfinden. Dann werden in Session 2 diese neuen Zeilen berücksichtigt.

Um dieses zu verhindern, sperrt die Isolationsebene SERIALIZABLE alles, was in einer Transaktion verwendet wird, für andere Transaktionen.

SESSION 1

```

1
2
3
4 START TRANSACTION;
5
6 SELECT COUNT(*) FROM artikel ;

```

```

7 +-----+
8 | COUNT(*) |
9 +-----+
10 | 10 |
11 +-----+
12
13 INSERT INTO artikel
14 VALUES (1 , 'X' , 0.0 , 'EUR' , 0) ;
15 ERROR 1205 (HY000) :
16 Lock wait timeout exceeded ;
17 try restarting transaction
18
19 INSERT INTO artikel
20 VALUES (1 , 'X' , 0.0 , 'EUR' , 0) ;
21
22 SELECT COUNT(*) FROM artikel ;
23 +-----+
24 | COUNT(*) |
25 +-----+
26 | 11 |
27 +-----+
28
29 COMMIT;
30
31 SELECT COUNT(*) FROM artikel ;
32 +-----+
33 | COUNT(*) |
34 +-----+
35 | 11 |
36 +-----+

```

SESSION 2

```

1 SET SESSION TRANSACTION
2 ISOLATION LEVEL SERIALIZABLE ;
3
4 START TRANSACTION;
5

```

```

6 SELECT COUNT(*) FROM artikel ;
7 +-----+
8 | COUNT(*) |
9 +-----+
10 | 10 |
11 +-----+
12
13
14
15
16
17
18 COMMIT;
19
20
21
22 SELECT COUNT(*) FROM artikel ;
23 +-----+
24 | COUNT(*) |
25 +-----+
26 | 10 |
27 +-----+
28
29
30
31 SELECT COUNT(*) FROM artikel ;
32 +-----+
33 | COUNT(*) |
34 +-----+
35 | 11 |
36 +-----+

```

Zu Beginn greifen beide Sessions auf die gesamte Tabelle `artikel` zu. In Session 1 wird in [Zeile 13](#) versucht, einen neuen Datensatz einzufügen. Aber die Tabelle ist wegen der Einstellung der Isolationsebene auf `SERIALIZABLE` in Session 2 auch für Session 1 blockiert. Session 1 wartet auf eine

Freigabe. Da diese nicht in angemessener Frist erfolgt ist, erscheint ein Timeout-Fehler in [Zeile 15](#).

Jetzt wird in Session 2 in [Zeile 18](#) mit COMMIT die Tabelle freigegeben. Nun kann Session 1 erfolgreich den Datensatz einfügen. Bitte beachten Sie, dass der neue Datensatz in Session 2 noch nicht bekannt ist; erst nach dem COMMIT in [Zeile 29](#). Dies hängt damit zusammen, dass Session 1 seine Änderungen noch nicht durch ein COMMIT freigegeben hatte.



Hinweis: Wegen dieser doch sehr restriktiven Sperrvorgänge ist diese Isolationsebene sehr anfällig für Performanceverluste und Timeout-Fehler und sollte nur sehr überlegt eingesetzt werden.

In MySQL, MariaDB und anderen Datenbanksystemen wie Oracle und PostgreSQL wird diese Isolationsebene auch nicht zur Vermeidung eines *phantom read* benötigt, da das Multi-Version Concurrency Control (MVCC) eingesetzt wird. Dabei wird – vereinfacht gesagt – für jede Transaktion ein *snapshot* der Tabellen erstellt. Jede Transaktion arbeitet dann auf ihrem *snapshot*. MVCC steht nur bei READ COMMITTED und REPEATABLE READ und nicht bei READ UNCOMMITTED und SERIALIZABLE zur Verfügung.

Tabelle 19.1 SQL-Isolationsebenen

Isolationsebene	dirty read	nonrepeatable read	phantom read
READ UNCOMMITTED	möglich	möglich	möglich
	unmöglich	möglich	möglich

READ COMMITTED			
READ COMMITTED (MVCC)	unmöglich	möglich	unmöglich
REPEATABLE READ	unmöglich	unmöglich	möglich
REPEATABLE READ (MVCC)	unmöglich	unmöglich	unmöglich
SERIALIZABLE	unmöglich	unmöglich	unmöglich

19.4 Fallbeispiel in C#

Das obige Beispiel mit dem neuen Artikel Säge wollen wir aus einer C#-Anwendung heraus durchführen. Ich verzichte hier auf eine Beschreibung der MySQL-API und vertraue darauf, dass die meisten Befehle selbsterklärend sind. Zunächst brauche ich ein Feld, in dem ich die derzeit offene Verbindung zum Server ablege ([Zeile 10](#)). In `Main()` werden dann nur drei Methoden aufgerufen, die die Details erledigen.

Listing 19.1 Transaktion, Teil 1

```

6  using MySql.Data.MySqlClient;
7
8  namespace TransaktionBsp {
9      class Program {
10         static MySqlConnection mysqlConnection
            = new MySqlConnection();//
11     }

```

```

12      static void Main(string[] args) {
13          Start();
14          InsertArtikel("Sge", 14.85, "EUR");
15          Stopp();
16      }

```

Die Methode `Start()` öffnet eine Verbindung zum Server und stellt diese Verbindung anderen Aktionen über das Feld `mysqlConnection` zur Verfügung. Die Variable `strConnectionString` enthält die Verbindungsparameter zum Server.

Listing 19.2 Transaktion, Teil 2

```

18      static void Start() {
19          String strConnectionString = "";
20          strConnectionString += "server=127.0.0
    .1;uid=root";
21          strConnectionString += ";password=XXX;
    database=oshop;charset=utf8";
22          try {
23              mysqlConnection.ConnectionString = s
    trConnectionString;
24              mysqlConnection.Open();
25          }
26          catch (Exception ex) {
27              Console.WriteLine(ex.Message);
28          }
29      }

```

Das Analogon ist die Methode `Stopp()`. Sie ist nur eine Wrapper-Methode für die Methode `Close()`:

Listing 19.3 Transaktion, Teil 3

```

31      static void Stopp() {
32          mysqlConnection.Close();
33      }

```

Und jetzt die Transaktion: Im try-Block werden die SQL-Anweisungen aufgebaut und an den Server geschickt. Geht dabei etwas schief (ein SQL-Befehl ist falsch geschrieben, ein Constraint meldet sich etc.), wird vom Server eine Exception ausgelöst, die im catch-Block verarbeitet wird. Dort wird ein ROLLBACK zum Server gesendet ([Zeile 72](#)), damit keine inkonsistenten oder halbfertigen Umbauten zurückbleiben. Wird vom Server kein Fehler gemeldet, werden die Ergebnisse der Anweisungen mit COMMIT ([Zeile 65](#)) bestätigt.

Listing 19.4 Transaktion, Teil 4

```
35 static void InsertArtikel(string strArtike
l, double dPreis, string strWaehrung) {
36     MySqlCommand command = new MyS
qlCommand();
37     command.Connection = mysqlConne
ction;
38     long artikel_id = 0;
39     try {
40         // Befehl 1
41         command.CommandText = "START
TRANSACTION";
42         command.Prepare();
43         command.ExecuteNonQuery();
44         // Befehl 2
45         command.CommandText = "INSER
T INTO artikel (bezeichnung, einzelpreis,
waehrung) VALUES (@bezeichnung, @einzel
preis, @waehrung)";
46         command.Prepare();
47         command.Parameters.AddWithVa
lue("@bezeichnung", strArtikel);
48         command.Parameters.AddWithVa
lue("@einzelpreis", dPreis);
49         command.Parameters.AddWithVa
lue("@waehrung", strWaehrung);
50         command.ExecuteNonQuery();
51         artikel_id = command.LastInse
rtedId;
52         // Befehl 3
```

```

53         command.CommandText = "INSERT INTO a
rtikel_nm_warengruppe VALUES ( @warengruppe_id
, @artikel_id)";
54         command.Prepare();
55         command.Parameters.Clear();
56         command.Parameters.AddWithValue("@wa
rengruppe_id", 3);
57         command.Parameters.AddWithValue("@ar
tikel_id", artikel_id);
58         command.ExecuteNonQuery();
59         // Befehl 4
60         command.Parameters["@warengruppe_id"
].Value = 4;
61         command.Parameters["@artikel_id"].Va
lue = artikel_id;
62         command.ExecuteNonQuery();
63         // Befehl 5
64         command.Parameters.Clear();
65         command.CommandText = "COMMIT"; // A
lles in Ordnung
66         command.Prepare();
67         command.ExecuteNonQuery();
68     }
69     catch (Exception ex) {
70         Console.WriteLine(ex.Message);
71         command.Parameters.Clear();
72         command.CommandText = "ROLLBACK"; //
Alles wieder auf Anfang
73         command.Prepare();
74         command.ExecuteNonQuery();
75     }
76 }

```

19.5 Deadlock

Ein besonders Problem tritt immer dann auf, wenn sich Sperren/Transaktionen gegenseitig blockieren.

SESSION 1

```
1 START TRANSACTION;
2
3 UPDATE artikel
4 SET einzelpreis = 2.32
5 WHERE artikel_id = 3001;
6
7 UPDATE artikel
8 SET einzelpreis = 56.26
9 WHERE artikel_id = 3005;
```

SESSION 2

```
1 START TRANSACTION;
2
3 UPDATE artikel
4 SET einzelpreis = 56.26
5 WHERE artikel_id = 3005;
6
7
8
9
10
11 UPDATE artikel
12 SET einzelpreis = 2.32
13 WHERE artikel_id = 3001;
```

Ab [Zeile 5](#) werden zwei Zeilen gesperrt. Session 1 sperrt den Artikel 3001 und Session 2 Artikel 3005. Session 1 versucht, in [Zeile 9](#) anschließend auch auf Artikel 3005 zuzugreifen. Da dieser noch gesperrt ist, wartet die Session auf Freigabe durch Session 2. Warten Sie lange genug, erhält man folgende Fehlermeldung:

```
ERROR 1205 (HY000): Lock wait timeout exceeded  
; try restarting transaction
```

Wir wollen diese Fehlermeldung aber gar nicht und führen in Session 2 in [Zeile 13](#) ein UPDATE auf Artikel 3001 aus. Dieser ist aber von Session 1 gesperrt. Somit warten beide auf die Freigabe durch einen anderen. Und das täten sie bis in alle Ewigkeit, wenn nicht seitens des Servers versucht wird, Deadlocks zu identifizieren und unterbrechen. MySQL liefert jedenfalls unmittelbar nach [Zeile 13](#) die Fehlermeldung:

```
ERROR 1213 (40001): Deadlock found when trying  
to get lock; try restarting transaction
```



Definition 62: Deadlock

Ist die Freigabe einer Sperrung A von einer Sperrung B abhängig, die von der Freigabe der Sperrung A abhängt, spricht man von einem Deadlock.

¹ Ich habe hier vereinfachend $EC > 0$ angenommen. Das ist nicht immer sinnvoll.

² altgr. *átomos*: das Unzerschneidbare, von *á-* für *un-* und *témnein* bedeutet *schneiden*.

³ Mithilfe von *snapshots* lässt sich diese strenge Forderung etwas aufweichen.

⁴ Hier und bei den anderen Isolationsebenen reduziere ich die dargestellte Bildschirmausgabe der beiden Sessions, um ein wenig Platz zu sparen.

20 Prozeduren



Endlich wieder zu Hause: schöne prozedurale, imperative Programmierung.

- Grundkurs

- Anlegen mit `CREATE PROCEDURE`
- Das `DELIMITER`-Problem
- Löschen mit `DROP PROCEDURE`
- Variablen: lokale Variable, Sitzungsvariable, globale Variable
- Parameter: `IN`, `OUT`, `INOUT`
- Verzweigung mit `IF`
- Fallunterscheidung mit `CASE`
- `WHILE`-Schleife
- Aufruf mit `CALL`

- Vertiefendes

- `LOOP`-Schleife
- `REPEAT`-Schleife
- Transaktion innerhalb einer Prozedur
- `CURSOR`



Die Quelltexte dieses Kapitels stehen in der Datei `mysql/listing16.sql` (siehe [Listing 28.18](#)). Sie brauchen auch die beiden Dateien `mysql/blz_20120305.csv` und `mysql/lstBestellungen.xml`.



Hinweis: Anders als in den vorherigen Kapiteln wird hier auf eine Gegenüberstellung von SQL:2023 und MySQL oder MariaDB verzichtet. Prozeduren werden so unterschiedlich von den DBS-Anbietern spezifiziert, dass ich hier sinnvollerweise nur eine Sprachimplementierung vorstellen werde: die von MySQL. Die Grundelemente Aufruf, Parameter, Verzweigung, Schleife und Cursor kommen aber überall vor, sodass sich das erworbene Verständnis übertragen lässt.

Um einen syntaktischen Vergleich zu ermöglichen, habe ich die entsprechenden Beispiele auch in PostgreSQL erstellt und Ihnen zum Download zur Verfügung gestellt.

20.1 Einstieg und Variablen

Manche Aufgaben lassen sich nicht in einem SQL-Befehl unterbringen. Das kann inhaltliche Gründe haben, wie wir sie beispielsweise bei den Transaktionen kennengelernt haben (siehe [Kapitel 19](#)), aber auch technische.

In vielen Fällen wird dazu auf der Client-Seite eine passende PHP-, Java- oder C#-Methode erstellt, die die Aufgabe als Ganzes löst. Man kann aber auch wollen, dass diese Aufgabe auf dem SQL Server gelöst werden soll. Ein Grund könnten hohes Datenvolumen und der damit verbundene Datenverkehr sein.

Wir bleiben bei dem Beispiel, einen Artikel mit seiner Warengruppe einzufügen. Dazu brauchen wir im ersten Schritt nur zwei Informationen: den Artikelnamen und die Warengruppe.



MySQL/MariaDB

```
CREATE PROCEDURE prozedurname ([parameter]  
    [, parameter]*)  
    anweisungsblock  
;  
parameter:  
    [IN | OUT | INOUT] parametername typ  
typ:  
    jeder gültige MySQL Datentyp (siehe Abschnitt 26.1)
```

Ein Parameter wird mit einer Art (siehe [Tabelle 20.1](#)), einem Typ (siehe [Abschnitt 26.1](#)) und einem Namen festgelegt. Der Name sollte selbsterklärend sein. Um die Übersicht nicht zu verlieren, verwende ich immer ein Präfix.

Tabelle 20.1 Parameterart

Art	Präfix	Beschreibung
IN	i	

		Der Parameter ist eine lokale Variable. Der Wert des Parameters wird beim Prozeduraufruf übergeben. Jede Wertänderung innerhalb der Prozedur bleibt außerhalb der Prozedur unbekannt. Diese Parameterart ist die Vorbelegung.
OUT	o	Beim Aufruf wird zu diesem Parameter kein Wert angegeben. Der Parameter wird innerhalb der Prozedur mit einem neuen Wert versehen. Dieser neue Wert kann anschließend außerhalb der Prozedur weiterverarbeitet werden.
INOUT	io	Wie bei IN kann beim Aufruf dem Parameter ein Wert mitgegeben werden. Der Wert des Parameters wird innerhalb der Prozedur verwendet, abgeändert und wieder in diesen Parameter abgelegt. Dieser geänderte Wert kann wie bei OUT nach dem Prozeduraufruf weiterverwendet werden.



Hinweis: Parameter sollten vor ihrer Verwendung immer formal und fachlich plausibilisiert werden.

Formal bedeutet hier, dass er einen passenden Datentyp hat; gegebenenfalls müssen diese konvertiert werden. Fachlich meint, dass die Übergabe innerhalb des Wertebereichs liegt. Der Anweisungsblock beginnt immer mit dem Schlüsselwort BEGIN und schließt mit END ab. Jetzt haben wir schon

genügend Informationen gesammelt, um uns an das erste Prozedurfragment zu wagen:

```
1 CREATE PROCEDURE insert_artikel
2 (
3     IN iArtikelname VARCHAR(255),
4     IN iWarengruppe VARCHAR(255)
5 )
6 BEGIN
7 INSERT INTO artikel (bezeichnung) VALUES (
iArtikelname);
```

Und schon ist Schluss :-(. Wir haben hier ein Problem, das wir zuerst lösen müssen, bevor es weitergehen kann. Die Anweisung in [Zeile 7](#) endet mit einem Semikolon, mit was auch sonst? Aber ein Semikolon bedeutet doch, dass die Anweisung abgeschlossen/fertig ist. Wir erhalten also eine unschöne Fehlermeldung und sind erst einmal ratlos.

Das Problem ist, dass die eigentliche Anweisung, die SQL ausführen soll, CREATE PROCEDURE ist. Innerhalb dieser Anweisung tauchen nun weitere Anweisungen auf, die ebenfalls mit einem Semikolon enden. Um diese Mehrdeutigkeit zu umgehen, können Sie das Zeichen zum Abschluss eines Befehls mit DELIMITER festlegen.



Hinweis: Achten Sie bei der Wahl des DELIMITER darauf, dass dieses Zeichen nicht anderweitig innerhalb der Prozedur verwendet wird.

Wählten wir beispielsweise die öffnende Klammer oder ein Komma, würde die Verwirrung sich nur steigern. Beliebte DELIMITER sind ; oder // oder \$\$ oder ^.

Bevor wir die Prozedur weiter programmieren, fällt uns ein, dass wir den durch `AUTO_INCREMENT` neu erzeugten Primärschlüsselwert erfahren müssen. Hierzu haben Sie schon die Funktion `LAST_INSERT_ID()` kennengelernt. Diesen Wert wollen wir in einer lokalen Variablen ablegen, um ihn später weiterzuverwenden.

Lokale Variablen werden mit dem Schlüsselwort `DECLARE` bekannt gemacht und können die bekannten Datentypen haben. Mit dem Zusatz `DEFAULT` können Sie ihnen einen Startwert mitgeben. Wie bei den Übergabeparametern verwende ich auch hier immer gerne ein Präfix: `v`.



MySQL/MariaDB

```
DECLARE variablenname datentyp [DEFAULT  
vorbelegung]  
;
```

Wird ein `DEFAULT` angegeben, ist dies der Startwert der Variablen. Oben ist erwähnt, dass es sich um eine lokale Variable handelt, aber was ist das eigentlich?



Definition 63: Lokale Variable

Ist eine Variable nur innerhalb des Anweisungsblocks bekannt, in dem sie deklariert wurde, nennt man diese eine *lokale Variable*.

Dadurch, dass eine Variable lokal ist, kommt man sich mit den lokalen Variablen anderer Prozeduren selbst bei Namensgleichheit nicht ins Gehege. Die meisten Variablen in

Prozeduren sind entweder Übergabeparameter oder lokal deklarierte Variablen. Aber sorry, was sind noch mal Übergabeparameter?



Definition 64: Übergabeparameter

Übergabeparameter vom Typ IN sind lokale Variablen, deren Wert von außerhalb der Prozedur beim Aufruf festgelegt wird. Sind sie vom Typ OUT, sind es ebenfalls lokale Variablen, aber der Wert der Variablen bleibt nach dem Verlassen der Prozedur erhalten. INOUT kann beides. Genaueres entnehmen Sie [Tabelle 20.1](#).

Der Wert einer Variablen wird entweder durch SET oder ein SELECT . . . INTO festgelegt.



MySQL/MariaDB

SET *variablenname* = *wert* ;

SELECT *selectausdruck* INTO *variablenname* ;

Wenn es lokale Variablen gibt, gibt es dann auch was anderes? Klar, die globalen Variablen. Dabei werden zwei Stufen der *Globalität* unterschieden: *sitzungsglobal* und *serverglobal*.



Definition 65: Sitzungsglobale Variablen

Ist eine Variable bzw. sind ihre Wertänderungen allen Prozeduren, Funktionen und Eingaben *einer* Sitzung bekannt, so wird diese *Sitzungsvariable* oder *session*

variable genannt. Sie wird mit `SESSION` oder `@@session.` gekennzeichnet.

Der Wert kann innerhalb einer Sitzung¹ verändert werden. Diese Veränderung ist nur innerhalb dieser Sitzung bekannt. Andere Sitzungen kennen die Variable nicht oder haben immer noch den alten Wert. Viele Sitzungsvariablen werden beim Start einer Sitzung schon vom Server eingerichtet. Eine davon haben wir bei den Transaktionen (siehe [Kapitel 19](#)) kennengelernt: `AUTOCOMMIT`.



Definition 66: Serverglobale Variable

Ist eine Variable bzw. sind ihre Wertänderungen allen Prozeduren, Funktionen und Eingaben in *allen* Sitzungen bekannt, so wird diese *globale Variable* genannt. Sie wird mit `GLOBAL` oder `@@global.` gekennzeichnet.

Eine Änderung ist allen anderen Sitzungen bekannt. Auch hier stellt der Server schon viele zur Verfügung. Als Beispiel sei hier `CONNECT_TIMEOUT` genannt.



Hinweis: Variablen, die durch Wertzuweisung deklariert werden, haben das Präfix `@`.

Zurück zu unserem Beispiel: Zu Beginn unserer Programmierung wird als Erstes der neue Begrenzer mit `DELIMITER` definiert. Ich bevorzuge das `//`. Bitte vergessen Sie nicht: Wenn irgendetwas schief läuft und Sie mit dem Client

[illegible]

```

1  DELIMITER //
2  CREATE PROCEDURE insert_artikel
3  (
4      IN iArtikelname VARCHAR(255),
5      IN iWarengruppe VARCHAR(255)
6  )
7
8  BEGIN
9
10 DECLARE v_artikel_id INT DEFAULT 0;
11 DECLARE v_warengruppe_id INT DEFAULT 0;
12
13
14 INSERT INTO artikel
15 (bezeichnung)
16 VALUES (iArtikelname);
17
18 SET v_artikel_id = LAST_INSERT_ID(); -- Variable setzen
19
20 SELECT warengruppe_id
21     INTO v_warengruppe_id -- Oder so
22     FROM warengruppe
23     WHERE bezeichnung = iWarengruppe;
24
25 INSERT INTO artikel_nm_warengruppe -- Eint
rag Hilfstabelle
26     (warengruppe_id, artikel_id)
27     VALUES
28     (v_warengruppe_id, v_artikel_id);
29
30 END// -- Anderer DELIMITER
31 DELIMITER ;

```

Mit der Anweisung SET wird in [Zeile 18](#) der Inhalt einer Variablen mit einem neuen Wert versehen. Dabei können rechts vom Gleichheitszeichen Konstanten, Ausdrücke oder Funktionsaufrufe stehen, sofern diese skalar sind und einen Wert zurückliefern, der zum Datentyp der Variablen passt.

Das Ergebnis einer Auswahl mit SELECT kann ebenfalls in einer Variablen abgelegt werden. Nach dem SELECT wird durch das Schlüsselwort INTO in [Zeile 21](#) die Zielvariable festgelegt. In [Zeile 30](#) kommt unser neuer Begrenzer zum Einsatz. Die Semikolons des Anweisungsblocks sind von SQL-Interpreter bis jetzt ignoriert worden. Das // schließt den CREATE ab, und die Prozedur wird übernommen. Jetzt ist es wichtig, den Begrenzer wieder auf das Semikolon zu setzen, damit er für nachfolgende Befehle wieder gilt.

Sicherlich haben Sie auch bei LAST_INSERT_ID() die Stirn gerunzelt, oder? Immerhin könnten mehrere Sessions gleichzeitig in die Tabelle artikel neue Zeilen einfügen. Stellen Sie sich vor, ein anderer Anwender würde in der Zeitspanne zwischen dem Einfügen und dem Aufruf von LAST_INSERT_ID() ebenfalls einen Datensatz einfügen. Stellt sich doch die Frage, welchen Wert liefert jetzt diese Funktion?

Ein Blick ins Handbuch verschafft Klarheit. Jede Session merkt sich nur die ID, die in dieser erstellt wurde. Werden in anderen Sessions andere IDs erstellt, sind die in meiner nicht durch LAST_INSERT_ID() ermittelbar. Oder positiv ausgedrückt: Es ist garantiert, dass diese Funktion den zuletzt erzeugten AUTO_INCREMENT-Wert meiner Session liefert.

Rufen wir die Prozedur mal auf:

```
1  mysql> CALL insert_artikel ('Schlauch', 'G
artenbedarf');
2
3  mysql> SELECT artikel_id, bezeichnung FROM
```

```

artikel ORDER BY artikel_id;
4  +-----+-----+
5  | artikel_id | bezeichnung |
6  +-----+-----+
7  [...]
8  | 9017 | Schlauch |
9  +-----+-----+
10
11 mysql> SELECT warengruppe_id, bezeichnung
FROM warengruppe;
12 +-----+-----+
13 | warengruppe_id | bezeichnung |
14 +-----+-----+
15 [...]
16 | 3 | Gartenbedarf |
17 [...]
18 +-----+-----+
19
20 mysql> SELECT * FROM artikel_nm_warengrupp
e;
21 +-----+-----+
22 | warengruppe_id | artikel_id |
23 +-----+-----+
24 [...]
25 | 3 | 9017 |
26 +-----+-----+

```

Eine Prozedur kann mit `DROP PROCEDURE` gelöscht werden.



MySQL/MariaDB

```

DROP PROCEDURE [IF EXISTS] prozedurname
;

```

Mit `SHOW PROCEDURE STATUS` kann eine Liste der verfügbaren

Prozeduren eingesehen werden; wir schränken die Ausgabe auf unsere neue Prozedur ein:

```
1  mysql> SHOW PROCEDURE STATUS LIKE 'insert_
artikel'\G
2  ***** 1. row *****
*****
3          Db: oshop
4          Name: insert_artikel
5          Type: PROCEDURE
6          Definer: root@localhost
7          Modified: 2023-07-13 17:40:04
8          Created: 2023-07-13 17:40:04
9          Security_type: DEFINER
10         Comment:
11 character_set_client: utf8mb4
12 collation_connection: utf8mb4_0900_ai_ci
13 Database Collation: utf8mb4_german2_ci
```

20.2 Verzweigung

20.2.1 Einfache Verzweigung mit IF

Da sind wir mal mutig:

```
1  mysql> CALL insert_artikel ('Fleischwurst'
, 'Lebensmittel');
2  ERROR 1452 (23000): Cannot add or update a
child row: a foreign key constraint
      fails ('oshop`.`artikel_nm_warengruppe`,
CONSTRAINT `
      artikel_nm_warengruppe_ibfk_1` FOREIGN K
EY ('warengruppe_id') REFERENCES `
      warengruppe` ('warengruppe_id'))
```

Das ist hässlich. Der Artikel Fleischwurst ist zwar angelegt worden, aber der Eintrag in der Tabelle `artikel_nm_warengruppe` ist fehlgeschlagen.



Aufgabe 20.1: Finden Sie anhand der Fehlermeldung die Fehlerursache heraus.

Um diesen Fehler in Zukunft zu vermeiden, wollen wir unsere Prozedur um eine Plausibilitätsüberprüfung der Übergabeparameter erweitern.



Definition 67: Plausibilitätsüberprüfung

Eine *Plausibilitätsüberprüfung* ist der Test eines Übergabeparameters oder einer Anwendereingabe auf fachliche Richtigkeit oder technische Unbedenklichkeit.

Wie bei jeder guten Analyse sollten wir die fachlichen Regeln und die daraus abgeleiteten Handlungen vor der Programmierung notieren:

- Ist die Warengruppenbezeichnung nicht vorhanden, soll eine neue Warengruppe angelegt werden.
- Ist die Artikelbezeichnung nicht vorhanden, soll ein neuer Artikel angelegt werden.
- Ist die Artikelbezeichnung schon vorhanden, soll überprüft werden, ob es schon eine Zuordnung des Artikels zu dieser Warengruppe gibt.
 - Es gibt keine Zuordnung, dann soll eine neue Zuordnung in die Hilfstabelle eingetragen werden.

- Es gibt eine Zuordnung, dann brauchen wir nichts zu tun.

Wir brauchen also ein Sprachkonstrukt, welches uns festzulegen erlaubt, ob ein Anweisungsblock überhaupt ausgeführt wird.



Definition 68: Verzweigung

Wenn ein Anweisungsblock nur abhängig von einer Bedingung (siehe [Definition 34](#)) ausgeführt wird, wird er *Verzweigung* genannt.

In MySQL und MariaDB wird für eine Verzweigung das IF verwendet. Der Aufbau eines IF ist wie folgt:



MySQL/MariaDB

```
IF bedingung THEN  
    anweisungsblock1 [  
ELSE  
    anweisungsblock2]  
END IF;
```

Der *anweisungsblock1* wird nur ausgeführt, wenn die Bedingung TRUE oder einen Wert $\neq 0$ liefert. Der *anweisungsblock2* wird nur ausgeführt, wenn *anweisungsblock1* nicht ausgeführt wird; der ELSE-Teil ist optional.

Zurück zu unserem oben beschriebenen Plan. Es ist nicht

möglich, eine Prozedur nur im Anweisungsblock zu ändern. Sie muss mit DROP PROCEDURE gelöscht und mit CREATE PROCEDURE wieder angelegt werden:

```
1  DROP PROCEDURE IF EXISTS insert_artikel;  
2  
3  DELIMITER //  
4  CREATE PROCEDURE insert_artikel  
5  (  
6  IN iArtikelname VARCHAR(255),  
7  IN iWarengruppe VARCHAR(255)  
8  )
```

Die Variable `v_artikel_id` wird den neuen oder gegebenenfalls schon vorhandenen Primärschlüsselwert des Artikels enthalten. Das Gleiche gilt für `v_warengruppe_id`. In `v_count_zuordnung` wird die Anzahl der schon vorhandenen Einträge in der Hilfstabelle bezüglich des Artikels und der Warengruppe abgelegt. Erwartet werden die Werte 1 oder 0.

```
10 BEGIN  
11 DECLARE v_artikel_id INT DEFAULT -1;  
12 DECLARE v_warengruppe_id INT DEFAULT -1;  
13 DECLARE v_count_zuordnung INT DEFAULT 0;
```

In `iWarengruppe` steht die Warengruppe, die als Parameter beim Aufruf übergeben wird. Diese wird nun innerhalb der WHERE-Klausel dazu verwendet, den passenden Primärschlüsselwert zu ermitteln. Das Ergebnis wird in `v_warengruppe_id` abgespeichert. In [Zeile 23](#) wird mit IF überprüft, ob ein Datensatz vorhanden war. War er nicht vorhanden, wird er neu angelegt und der neue Primärschlüsselwert mithilfe von `LAST_INSERT_ID()` ermittelt. In beiden Fällen steht der Primärschlüsselwert der Warengruppe in der lokalen Variablen `v_warengruppe_id`: entweder der schon vorhandene oder der neue.

```

16  -- Teste, ob Warengruppe vorhanden ist
17  SELECT warengruppe_id
18      INTO v_warengruppe_id
19      FROM warengruppe
20      WHERE bezeichnung = iWarengruppe;
21
22  -- Wenn nein, füge eine ein
23  IF v_warengruppe_id < 0 THEN -- Einfache V
    erzweigung
24      INSERT INTO warengruppe (bezeichnung) VA
    LUES (iWarengruppe);
25      SET v_warengruppe_id = LAST_INSERT_ID();
26  END IF;

```

Analog wird bei dem Artikel vorgegangen. Zuerst wird versucht, den gegebenenfalls schon vorhandenen Primärschlüsselwert zu ermitteln. Dieser wird auf < 0 getestet, und falls die Bedingung TRUE liefert, wird ein neuer Artikel mit der übergebenen Bezeichnung angelegt.

```

28  -- Teste, ob artikel vorhanden ist
29  SELECT artikel_id
30      INTO v_artikel_id
31      FROM artikel
32      WHERE bezeichnung = iArtikelname;
33
34  -- Wenn nein, teste auf füge ein
35  IF v_artikel_id < 0 THEN
36      INSERT INTO artikel (bezeichnung) VALUES
        (iArtikelname);
37      SET v_artikel_id = LAST_INSERT_ID();
38  END IF;

```

Jetzt wird in der Hilfstabelle anhand der ermittelten Primärschlüsselwerte ein Eintrag gesucht, genauer: die Anzahl der passenden Einträge. Liefert die Bedingung `v_count_zuordnung <= 0` den Wert TRUE, muss eine Zeile in

die Hilfstabelle eingefügt werden. Der Hinweis in [Zeile 53](#) ist nur für eine Warnung da.

```
40  -- Test, ob Zuordnung schon vorhanden
41  SELECT COUNT(*)
42      INTO v_count_zuordnung
43      FROM artikel_nm_warengruppe
44      WHERE artikel_id = v_artikel_id AND ware
ngruppe_id = v_warengruppe_id;
45
46  -- Wenn nein, füge eine hinzu
47  IF v_count_zuordnung <= 0 THEN
48      INSERT INTO artikel_nm_warengruppe
49      (warengruppe_id, artikel_id)
50      VALUES
51      (v_warengruppe_id, v_artikel_id);
52  ELSE
53      SELECT 'Zuordnung schon vorhanden'; -- H
inweis
54  END IF;
55  END//
56  DELIMITER ;
```

Ein neuer Versuch:

```
40  mysql> CALL insert_artikel ('Fleischwurst'
, 'Lebensmittel');
41  mysql> CALL insert_artikel ('Fleischwurst'
, 'Lebensmittel');
42  +-----+
43  | Zuordnung schon vorhanden |
44  +-----+
45  | Zuordnung schon vorhanden |
46  +-----+
```



Aufgabe 20.2: Testen Sie alle Fälle mit eigenen Beispielen durch und überprüfen Sie die Ergebnisse in den Tabellen.

20.2.2 Mehrfache Verzweigung mit CASE

Für die Fallunterscheidung gibt es zwei Varianten: eine, die eine Wertübereinstimmung ermittelt, und eine, die Bedingungen prüft.

Hier die Syntax für die Wertübereinstimmung: Hinter dem CASE steht ein irgendwie gearteter Container für Werte, z.B. ein Variablenname. Hinter dem WHEN wird ein konkreter Wert angegeben. Stimmen die Werte überein, wird der hinter dem THEN stehende Anweisungsblock ausgeführt.



MySQL/MariaDB

```
CASE spaltenname|ausdruck|variablenname
  WHEN wert THEN anweisungsblock
  [WHEN wert THEN anweisungsblock]*
  [ELSE anweisungsblock]
END CASE;
```

Und nun die Syntax für die Bedingung: Hinter jedem WHEN steht eine Bedingung. Ist diese TRUE, wird der Anweisungsblock hinter dem THEN ausgeführt. Die Formulierung der Bedingung erfolgt wie bei der Verzweigung mit IF (siehe [Abschnitt 20.2.1](#)).



MySQL/MariaDB

CASE

WHEN *bedingung* THEN *anweisungsblock*

[WHEN *bedingung* THEN *anweisungsblock*]*

[ELSE *anweisungsblock*]

END CASE;

Ich möchte durch eine Prozedur abhängig vom Übergabeparameter unterschiedliche Umsatzberichte erstellen lassen. Ist der Übergabeparameter eine 1, soll der totale Umsatz anhand der Tabelle rechnung ermittelt werden. Ist er 2, wird der Umsatz pro Jahr, bei 3 pro Monat berechnet. Das Ergebnis wird in der Tabelle umsatzzahlen abgelegt.

```
1  DELIMITER //
2  CREATE PROCEDURE umsatzreport
3  (
4  IN iZeitraum INT
5  )
6  BEGIN
7  DROP TABLE IF EXISTS umsatzzahlen;
8
9  CASE iZeitraum
10     WHEN 1 THEN -- Total
11         CREATE TABLE umsatzzahlen
12             SELECT 'Total', SUM(einzelpreis * meng
e) 'Umsatz'
13             FROM
14                 rechnung_position INNER JOIN artikel
15                 USING(artikel_id)
16                 INNER JOIN rechnung USING(rechnung_id)
16                 ;
```

```

17     WHEN 2 THEN -- Pro Jahr
18     CREATE TABLE umsatzzahlen
19     SELECT YEAR(datum) 'Jahr', SUM(einzelpre
is * menge) 'Umsatz'
20     FROM
21     rechnung_position INNER JOIN artikel U
SING(artikel_id)
22     INNER JOIN rechnung USING(rechnung_i
d)
23     GROUP BY 'Jahr'
24     ORDER BY 'Jahr' DESC
25     ;
26     WHEN 3 THEN -- Pro Monat
27     CREATE TABLE umsatzzahlen
28     SELECT YEAR(datum) 'Jahr' , MONTH(datum)
'Monat'
29     , SUM(einzelpreis * menge) 'Umsatz'
30     FROM
31     rechnung_position INNER JOIN artikel U
SING(artikel_id)
32     INNER JOIN rechnung USING(rechnung_id)
33     GROUP BY 'Jahr', 'Monat'
34     ORDER BY 'Jahr' DESC, 'Monat' DESC
35     ;
36     ELSE -- So was sollte es immer geben
37     CREATE TABLE umsatzzahlen
38     SELECT 'Unbekannte Berichtsart', iZeitra
um;
39 END CASE;
40 END//
41 DELIMITER ;

```

Schauen wir uns die Ausgaben an:

```

1  mysql> CALL umsatzreport(1); SELECT * FROM
umsatzzahlen;
2  +-----+-----+
3  | Total | Umsatz |

```

```

4  +-----+-----+
5  | Total | 2170.2100000000000 |
6  +-----+-----+

```

Der Übergabeparameter ist 1. Beim ersten WHEN in [Zeile 10](#) wird der Inhalt von iZeitraum genau auf diesen Wert überprüft. Die anschließende Anweisung erzeugt den Bericht. Würden mehrere Anweisungen auszuführen sein, muss dieser mit BEGIN...END; umschlossen sein.

```

1  mysql> CALL umsatzreport(2); SELECT * FROM
umsatzzahlen;
2  +-----+-----+
3  | Jahr | Umsatz |
4  +-----+-----+
5  | 2012 | 1615.8000000000000 |
6  | 2011 | 554.4100000000000 |
7  +-----+-----+

```

Jetzt wird der Fall in [Zeile 17](#) ausgeführt. Dabei werden die Umsatzdaten nach der Jahresangabe gruppiert, summiert und sortiert. Analoges geschieht in [Zeile 26](#) bei 3:

```

1  mysql> CALL umsatzreport(3); SELECT * FROM
umsatzzahlen;
2  +-----+-----+-----+
3  | Jahr | Monat | Umsatz |
4  +-----+-----+-----+
5  | 2012 | 4 | 1328.4000000000000 |
6  | 2012 | 3 | 287.4000000000000 |
7  | 2011 | 1 | 554.4100000000000 |
8  +-----+-----+-----+

```

Wird ein Parameterwert der Prozedur übergeben, für den keine Wertübereinstimmung gefunden wird, wird der ELSE-Teil ab [Zeile 36](#) ausgeführt:

```

1  mysql> CALL umsatzreport(0); SELECT * FROM
umsatzzahlen;
2
3  +-----+-----+
4  | Unbekannte Berichtsart | iZeitraum |
5  +-----+-----+
6  | Unbekannte Berichtsart | 0 |
7  +-----+-----+

```



Hinweis: Die erste Wertübereinstimmung oder die erste zutreffende Bedingung wird ausgeführt. Anschließend wird die Fallunterscheidung verlassen.

Das bedeutet, dass die nachfolgenden Bedingungen nicht mehr überprüft werden.

```

1  DELIMITER //
2  CREATE PROCEDURE b()
3  BEGIN
4  DECLARE vZeitraum INT ;
5  SET vZeitraum = 2;
6  CASE
7  WHEN vZeitraum = 2 THEN SELECT 'A';
8  WHEN vZeitraum > 0 THEN SELECT 'B';
9  WHEN vZeitraum < 2 THEN SELECT 'C';
10 END CASE;
11 END//
12 DELIMITER ;
13
14 mysql> CALL b();
15 +----+
16 | A |
17 +----+
18 | A |

```




Aufgabe 20.3: Hier hätte man vielleicht auch folgende Ausgabe erwarten können. Warum?

```
+---+
|  A  |
+---+
|  A  |
+---+
```

```
+---+
|  B  |
+---+
|  B  |
+---+
```

20.3 Schleifen

Anweisungen werden oft mehrfach oder besser gesagt wiederholt ausgeführt. Denken Sie an eine Performancemessung, die bei wachsender Datenmenge eigentlich immer das Gleiche tut.



Definition 69: Schleife

Ein Anweisungsblock, der sich abhängig von einer Bedingung wiederholt, wird *Schleife* genannt.

In der Programmierung kennt man viele Arten von Schleifen: kopf- oder fußgesteuerte, annehmende oder abweisende, offene oder geschlossene Schleifen.



Definition 70: Kopf- oder fußgesteuerte Schleife

Wird bei einer Schleife *vor* der Ausführung des Anweisungsblocks die Bedingung überprüft, so spricht man von einer *kopfgesteuerten* Schleife. Wird der Anweisungsblock ausgeführt und *danach* die Bedingung überprüft, so spricht man von einer *fußgesteuerten* Schleife.

Aus der Definition folgt, dass bei einer fußgesteuerten Schleife der Anweisungsblock mindestens einmal ausgeführt wird; bei einer kopfgesteuerten ist das nicht garantiert. In Sprachen wie C entspricht die kopfgesteuerte einer `while`-Schleife und die fußgesteuerte einer `do/while`-Schleife.



Definition 71: Annehmende oder abweisende Schleife

Wird bei einer Schleife der Anweisungsblock ausgeführt, *bis* eine Bedingung wahr ist, wird dies *abweisende* Schleife genannt. Wird der Anweisungsblock wiederholt, *solange* eine Bedingung wahr ist, wird sie *annehmende* Schleife genannt.

Abweisende Schleifen sind aus der Mode gekommen. Heute werden fast nur noch annehmende verwendet. Die `repeat/until`-Schleife in Delphi ist eine abweisende Schleife.



Definition 72: Offene oder geschlossene Schleife

Liegt die Anzahl der Wiederholungen im Prinzip fest, so spricht man von einer *geschlossenen* Schleife. Ist die Anzahl der Wiederholungen unbestimmt, von einer *offenen*.

Alle Arten von Zählschleifen wie die `for`-Schleife in C# sind geschlossene Schleifen. Offene Schleifen werden meist als `while`-Schleife spezifiziert.

20.3.1 LOOP-Schleife



MySQL/MariaDB

```
[labelname:]LOOP  
    anweisungsblock  
END LOOP[ labelname];
```

Die LOOP-Schleife ist eine offene Schleife. Da es keine Bedingung gibt, die den Abbruch oder die Fortsetzung der Schleife steuert, läuft sie im Prinzip ohne Ende durch – ist also eine Endlosschleife. In der Anwendungsentwicklung haben Endlosschleifen durchaus ihre Berechtigung, so auch bei SQL-Prozeduren, beispielsweise bei der Abarbeitung von Cursor (siehe [Abschnitt 20.5](#)).

Innerhalb der LOOP-Schleife – aber nicht nur dort – kommen daher folgende Schlüsselwörter zum Einsatz:

- **LEAVE:** Der Anweisungsblock oder die Schleife werden

verlassen. Dieses Schlüsselwort kann innerhalb eines BEGIN...END-Anweisungsblocks oder in einem Schleifenkonstrukt wie LOOP, REPEAT oder WHILE verwendet werden. Die Schleife wird durch LEAVE beendet, und die erste Anweisung – falls vorhanden – nach der Schleife wird ausgeführt. Gibt es keine nachfolgende Anweisung oder keinen Anweisungsblock, in den man zurückkehren kann, ist die Prozedur beendet. Handelt es sich um eine verschachtelte Schleife², wird die darüberliegende Schleife fortgesetzt³. Der Anweisungsblock oder die Schleife muss ein Label haben.

- **ITERATE:** Die Schleife beginnt mit einem neuen Durchlauf (Iteration). Dieses Schlüsselwort kann innerhalb eines Schleifenkonstrukts wie LOOP, REPEAT oder WHILE verwendet werden. Egal an welcher Stelle ich mich im Anweisungsblock der Schleife befinde, die Prozedur springt zur ersten Anweisung der Schleife oder bei kopfgesteuerten Schleifen zur Überprüfung der Schleifenbedingung⁴. Die Schleife muss ein Label haben.

Da mir keine sinnvollen Beispiele für eine Endlosschleife eingefallen sind, hier ein sinnloses. Wobei, so sinnlos vielleicht doch nicht, werden hier doch das Verhalten und der Wertebereich von einer ganzzahligen Variablen visualisiert.

```
1  DELIMITER //
2  CREATE PROCEDURE warte_auf_ueberlauf()
3  BEGIN
4  DECLARE vInt TINYINT;
5
6  SET vInt = 0;
7      LOOP -- Schleifenstart
8          SELECT vInt;
9          SET vInt = vInt + 1;
10     END LOOP; -- Schleifenende
```

```

11  END;
12  //
13  DELIMITER ;
14
15  CALL warte_auf_ueberlauf();

```

Eine wichtige Beobachtung ist, dass der Zähler bei 127 aufhört, anstatt sich weiter zu erhöhen! Es gibt also keinen klassischen Überlauf oder eine Exception wie bei C# oder Java.⁵



Hinweis: Die Ausführung kann auf dem MySQL oder MariaDB Client durch STRG+C unterbrochen werden.

Ich möchte auf diese Art und Weise herausfinden, was der höchste und was der kleinste Wert einer INT-Variablen ist. In [Zeile 9](#) wird die Schleife mit einem Aufkleber⁶ versehen. In [Zeile 13](#) kann daher der LEAVE genau sagen, *was* er verlassen möchte. Analog arbeitet die zweite Schleife. Der Abbruch einer Schleife mit LEAVE wird immer dann ausgelöst, wenn sich die Werte der Variablen nicht mehr erhöhen oder erniedrigen.

```

1  DELIMITER //
2  CREATE PROCEDURE min_max()
3  BEGIN
4      DECLARE vInt TINYINT;
5      DECLARE vMin INT;
6      DECLARE vMax INT;
7
8      SET vInt = 0;
9      maxsuche: LOOP -- Jetzt mit Label
10         SET vMax = vInt;
11         SET vInt = vInt + 1;
12         IF vMax = vInt THEN
13             LEAVE maxsuche; -- Verlasse Label

```

```

14     END IF;
15 END LOOP maxsuche;
16
17 SET vInt = 0;
18   minsuche: LOOP
19     SET vMin = vInt;
20     SET vInt = vInt - 1;
21     IF vMin = vInt THEN
22       LEAVE minsuche;
23     END IF;
24 END LOOP minsuche;
25
26 SELECT 'MIN = ', vMin, 'MAX = ', vMax;
27 END;
28 //
29
30 DELIMITER ;

```



Aufgabe 20.4: Führen Sie `CALL min_max();` aus. Warum dürfen `vMin` und `vMax` nicht den gleichen Datentyp verwenden wie `vInt`?

20.3.2 WHILE-Schleife



MySQL/MariaDB

```

[labelname:]WHILE schleifenbedingung DO
    anweisungsblock
END WHILE[labelname];

```

Die WHILE-Schleife ist eine offene, kopfgesteuerte und annehmende Schleife. Und jetzt kommt's:

- Jede fußgesteuerte Schleife kann durch eine kopfgesteuerte simuliert werden.
- Jede abweisende Schleife kann durch eine annehmende simuliert werden.
- Jede geschlossene Schleife kann durch eine offene simuliert werden.

Mit anderen Worten, ich kann jede Art von Schleife bauen, wenn ich eine kopfgesteuerte, annehmende und offene Schleife in einer Programmiersprache zur Verfügung stelle. Und genau das ist eine WHILE-Schleife. In den meisten Fällen werden daher bei Prozeduren WHILE-Schleifen verwendet. Zuerst ein kleines numerisches Beispiel: *das Sieb des Eratosthenes*⁷.

```
1  DELIMITER //
2  CREATE PROCEDURE sieb
3  (
4  iGrenze BIGINT UNSIGNED
5  )
6  BEGIN
7  DECLARE vZahl BIGINT UNSIGNED DEFAULT 2; -
- Start mit 2
8  DECLARE vPosition BIGINT UNSIGNED DEFAULT
2;
9
10 -- Aufbau der Tabelle
11 DROP TABLE IF EXISTS zahlenstrahl;
12 CREATE TABLE zahlenstrahl
13 (
14 position BIGINT UNSIGNED,
15 zahl BIGINT UNSIGNED,
16 PRIMARY KEY(position)
17 );
```

```

18
19 WHILE vZahl <= iGrenze DO
20     INSERT INTO zahlenstrahl VALUES (vZahl,
vZahl);
21     SET vZahl = vZahl + 1;
22 END WHILE;
23
24 -- Sieb
25     WHILE vPosition < CEILING(SQRT(iGrenze))
DO
26     SELECT zahl INTO vZahl FROM zahlenstrahl
WHERE position = vPosition;
27     IF vZahl > 0 THEN
28     UPDATE zahlenstrahl SET zahl = 0
29     WHERE (position > vPosition) AND (zahl %
vZahl = 0);
30     END IF;
31     SET vPosition = vPosition + 1;
32     END WHILE;
33
34 -- Bereinigen der Tabelle
35 DELETE FROM zahlenstrahl WHERE zahl = 0;
36 END//
37 DELIMITER ;

```

Jetzt kann das Sieb aufgerufen werden. Wenn Sie die Idee des Algorithmus nachlesen wollen, sei Ihnen [[MOP06](#)] empfohlen.

```

1  mysql> CALL sieb(20);
2
3  mysql> SELECT zahl FROM zahlenstrahl ORDER
BY zahl;
4  +-----+
5  | zahl |
6  +-----+
7  | 2 |
8  | 3 |
9  | 5 |

```


10		7	
11		11	
12		13	
13		17	
14		19	
15	+-----+		

Ein besseres Beispiel: Ein häufiges Problem ist, dass eine umfangreiche Aktion eine Tabelle für andere Aktionen blockiert. Wenn wir beispielsweise die Bankleitzahlen importieren, ist die Tabelle bank für alle anderen Schreib- und vielleicht sogar Leseoperationen nicht verfügbar. Ist diese Nichtverfügbarkeit eine Frage von einer halben bis ganzen Sekunde, könnte man ein solches Locking noch hinnehmen. Vergehen mehrere Sekunden, riskieren Sie einen Timeout-Fehler.

Eine Lösungsstrategie ist, die umfangreiche Aktion in Teilaktionen zu zerlegen und so anderen Aktionen Zeit für Auswertungen zu lassen. Bauen wir uns eine passende Beispielumgebung:

```

1  DROP DATABASE IF EXISTS tmpSchleife;
2  CREATE DATABASE tmpSchleife CHARACTER SET
utf8mb4;
3
4  USE tmpSchleife;
5
6  CREATE TABLE bank_quelle
7  (
8    blz CHAR(8), merkmål CHAR(1), bezeichnung
VARCHAR(255),
9    plz CHAR(5), ort VARCHAR(255), kurz VARCHA
R(255),
10   pan CHAR(5), bic VARCHAR(255), sm CHAR(2),
11   ds CHAR(6), kz CHAR(1), blzl CHAR(1),
12   blzn CHAR(8)

```

```

13 )
14 ;
15
16 SET GLOBAL local_infile = ON;
17 LOAD DATA LOCAL INFILE 'blz_20120305.csv'
18 INTO TABLE bank_quelle
19 FIELDS TERMINATED BY ';'
20 LINES TERMINATED BY '\n'
21 ;
22 SET GLOBAL local_infile = OFF;

```

Wir haben jetzt eine Tabelle, die ungefähr 19.600 Zeilen umfasst, und diese will ich in eine andere Tabelle kopieren. Dabei sollen die Adressdaten in ein eigenes Feld zusammengefasst werden.

```

1 CREATE TABLE bank_ziel
2 (
3 blz CHAR(8),
4 bezeichnung VARCHAR(255),
5 adresse TEXT
6 )
7 ;

```

Der erste Lösungsansatz ist ein normaler, uns bekannter INSERT:

```

1 INSERT INTO bank_ziel (blz, bezeichnung, a
dresse)
2 SELECT blz, bezeichnung, CONCAT(plz, ' ',
ort) FROM bank_quelle;

```

Wie oben schon erwähnt, ist der Nachteil dieser Lösung, dass die Tabelle bank_ziel für die Dauer der Ausführung keiner anderen Aktion zur Verfügung steht. Hier kann eine Prozedur Abhilfe schaffen:

```

1  DELIMITER //
2  CREATE PROCEDURE kopiere_blz
3  (
4  iAnzahl INT UNSIGNED -- Anzahl der Zeilen
pro Kopiervorgang
5  )
6  BEGIN
7  DECLARE vOffset INT UNSIGNED; -- Aktueller
Startpunkt
8  DECLARE vCount INT UNSIGNED; -- Anzahl der
zu kopierenden Zeilen
9
10 SELECT COUNT(*) FROM bank_quelle INTO vCou
nt;
11
12 SET vOffset = 0;
13 WHILE vOffset < vCount DO -- Start der Sch
leife
14 SELECT 'vOffset ', vOffset; -- Nur zum Guc
ken
15
16     INSERT INTO bank_ziel (blz, bezeichnung,
    adresse)
17     SELECT blz, bezeichnung, CONCAT(plz, ' ',
    , ort)
18     FROM bank_quelle
19     LIMIT vOffset, iAnzahl;
20     SET vOffset = vOffset + iAnzahl; -- Neue
r Startwert
21 END WHILE; -- Ende der Schleife
22 END//
23 DELIMITER ;
24
25 CALL kopiere_blz(1000);

```

In [Zeile 13](#) beginnt die Schleife. Hinter dem Schlüsselwort WHILE wird die Schleifenbedingung formuliert. Das

Schlüsselwort DO startet den Anweisungsblock der Schleife. In [Zeile 21](#) endet der Anweisungsblock der Schleife, und die Prozedur springt wieder zur [Zeile 13](#). Erst, wenn die Schleifenbedingung nicht TRUE ergibt, wird mit der Zeile nach [Zeile 21](#) weitergemacht.

Der Übergabeparameter `iAnzahl` der Prozedur legt fest, wie viele Zeilen pro Schleifendurchlauf von der Tabelle `bank_quelle` zur Tabelle `bank_ziel` kopiert werden. Die Variable `vOffset` enthält die Information darüber, mit welcher Zeile der Kopiervorgang beginnen soll. In `vCount` wird die Anzahl der Zeilen von `bank_quelle` abgelegt, damit die Schleife weiß, wann sie fertig ist.

Zuerst wird `vOffset` auf 0 gesetzt, damit der Kopiervorgang mit der ersten Zeile beginnt. In [Zeile 20](#) wird `vOffset` um die Anzahl der kopierten Zeilen erhöht. Die Schleifenbedingung überprüft, ob dieser Wert kleiner als die Gesamtanzahl der zu kopierenden Zeilen ist. Falls ja, muss die Schleife ausgeführt werden.

Die beiden Variablen `vOffset` und `iAnzahl` werden in einer LIMIT-Klausel dazu verwendet, genau die Zeilen aus der Quelltable auszuschneiden, die gerade kopiert werden sollen.



Aufgabe 20.5: Überlegen Sie sich eine geeignete Überprüfung des Kopiervorgangs und bauen Sie die Prozedur zu einer Transaktion um. Es soll ein ROLLBACK erfolgen, wenn der Kopiervorgang nicht erfolgreich war.



Hinweis: Eigentlich möchte man die beiden Tabellen für die Dauer einer Kopieraktion sperren und zwischenzeitlich den Zugriff für andere Tabellen erlauben. Sie könnten also vor [Zeile 16](#)

```
LOCK TABLES bank_quelle READ, bank_ziel  
WRITE;
```

und nach [Zeile 16](#)

```
UNLOCK TABLES;
```

einfügen. Sie erhalten dann aber folgende selbsterklärende Fehlermeldung:

```
ERROR 1314 (0A000): LOCK is not allowed in  
stored procedures
```

20.3.3 REPEAT-Schleife



MySQL/MariaDB

```
[labelname:]REPEAT
```

```
    anweisungsblock
```

```
UNTIL bedingung END REPEAT[ labelname];
```

Die REPEAT-Schleife ist eine abweisende, fußgesteuerte Schleife. Wie oben erwähnt, brauchen Sie diese Schleife nicht unbedingt, da Sie diese durch eine WHILE-Schleife simulieren können. *Hier ein numerisches Beispiel:* Es wird der größte gemeinsame Teiler nach Euklid⁸ berechnet⁹.

```

1  DELIMITER //
2  CREATE PROCEDURE ggt
3  (
4  iZahl1 INT UNSIGNED,
5  iZahl2 INT UNSIGNED
6  )
7  BEGIN
8  SET @m = iZahl1; -- Deklaration durch
9  SET @n = iZahl2; -- eine Zuweisung
10
11      REPEAT -- Schleifenstart
12          SET @t = @m % @n;
13          SET @m = @n;
14          SET @n = @t;
15  UNTIL @n = 0 END REPEAT; -- Schleifenende
16
17  SELECT CONCAT('GGT(',iZahl1,', ',iZahl2,')
18  = ') 'GgT()', @m;
19  END//
20  DELIMITER ;

```

Der Anweisungsblock zwischen [Zeile 11](#) und [15](#) wird mindestens einmal durchgeführt. Dann erfolgt die Überprüfung der Schleifenbedingung. Ergibt diese TRUE, wird die Schleife beendet, ansonsten startet ein neuer Schleifendurchlauf.

Ich persönlich finde abweisende Schleifen in der Analyse und bei der Fehlersuche verwirrend. Wenn es Ihnen genauso geht, vermeiden Sie diese. Als Beispiel wollen wir den größten gemeinsamen Teiler von 20 und 15 wissen.

```

1  CALL ggt(20, 15);
2  +-----+-----+
3  | GgT() | @m |
4  +-----+-----+
5  | GGT(20, 15) = | 5 |

```

20.4 Transaktion innerhalb einer Prozedur

Erinnern Sie sich noch an unser erstes Beispiel in [Abschnitt 20.1](#)? Es ging dabei um das Einfügen eines Artikels und einer Warengruppe. Was dabei noch fehlte, war der Einbau der Transaktion, denn schließlich soll unsere Prozedur nicht weniger können als die in [Kapitel 19](#) beschriebene Transaktion. Dazu müssen wir in dem Quelltext angeben, was passieren soll, falls ein Fehler aufgetreten ist.

```
1  DROP PROCEDURE insert_artikel;
2
3  DELIMITER //
4  CREATE PROCEDURE insert_artikel
5  (
6  IN iArtikelname VARCHAR(255),
7  IN iWarengruppe VARCHAR(255)
8  )
9
10 BEGIN
11 DECLARE v_artikel_id INT DEFAULT -1;
12 DECLARE v_warengruppe_id INT DEFAULT -1;
13 DECLARE v_count_zuordnung INT DEFAULT 0;
14
15 DECLARE EXIT HANDLER FOR SQLEXCEPTION -- Handle für Fehler
16 BEGIN
17     SELECT 'Wegen Fehler ROLLBACK';
18     ROLLBACK;
19 END;
20
```

```

21  START TRANSACTION;
22  [...]
23  COMMIT;
24
25  END//
26  DELIMITER ;

```

Wird in unserer Prozedur ein Fehler erzeugt, dann bricht die Verarbeitung ab und springt in den Anweisungsblock des Handle ab [Zeile 15](#). Anschließend wird die Prozedur verlassen, da der Handle mit `DECLARE EXIT` definiert wurde. Alternativ hätten wir auch `DECLARE CONTINUE` angeben können. Dann wird die Prozedur nicht verlassen, sondern die Verarbeitung nach Ausführung des Handle fortgesetzt.

MySQL und MariaDB kennen drei solcher Handles: `NOT FOUND`, `SQLWARNING` und `SQLEXCEPTION` (siehe [Tabelle 20.2](#)).

Alternativ kann der Zustand von `SQLSTATE` abgefragt werden:

```

1  [...]
2  DECLARE CONTINUE HANDLER FOR SQLSTATE '230
00'
3  [...]

```

Tabelle 20.2 SQL-HANDLER

Art des Handlers	Erklärung
SQLEXCEPTION	Wird ausgelöst, wenn ein Fehler eintritt. Dies sind alle SQLSTATE, die nicht mit 00, 01 oder 02 beginnen.
SQLWARNING	Wird ausgelöst, wenn eine Warnung erfolgt. Dies sind alle SQLSTATE, die mit 01 beginnen.

NOT FOUND	Wird ausgelöst, wenn ein <code>SELECT . . . INTO</code> kein Ergebnis findet, oder im Zusammenhang mit einem Cursor. Die entsprechenden <code>SQLSTATE</code> beginnen mit 02.

20.5 Cursor

Die Verarbeitung innerhalb der Prozeduren war bisher so, dass abhängig von irgendwelchen Bedingungen Teilmengen und Auswertungen von Tabellen verarbeitet wurden. Gerade in Prozeduren ergibt sich aber oft die Situation, dass Sie sich jeden einzelnen Datensatz anschauen müssen.

Mithilfe eines *Cursors*¹⁰ können wir in Zusammenarbeit mit einer Schleife für jede einzelne Datenzeile eine Entscheidung treffen. Für einen Cursor braucht man vier Befehle:

`DECLARE . . . FOR CURSOR`, `OPEN`, `FETCH` und `CLOSE`.



MySQL/MariaDB

```
DECLARE cursorname CURSOR FOR
    SELECT auswahl
;
```

Die *auswahl* darf kein `INTO` enthalten. Schließlich soll das Ergebnis der Auswahl erst durch den `FETCH` in einer Variablen landen.



MySQL/MariaDB

```
OPEN cursorname  
;
```

Mit dem OPEN wird die im DECLARE formulierte *auswahl* ausgeführt.



MySQL/MariaDB

```
FETCH cursorname  
    INTO variablenname[, variablenname]*  
;
```

Der erste FETCH liest die erste Zeile der Ergebnismenge der *auswahl*. Der Inhalt der ersten Zeile wird in die Variablen kopiert. Daher müssen genauso viele Variablen hinter dem INTO stehen, wie die Ergebnismenge Spalten hat. Anschließend wird ein interner Zeiger auf die nächste Zeile der Ergebnismenge verschoben.

Wird ein FETCH auf eine leere Ergebnismenge ausgeführt oder sind bereits alle Zeilen *gefetcht*, so wird die NOT FOUND-Exception ausgelöst (siehe [Tabelle 20.2](#)).



MySQL/MariaDB

```
CLOSE cursorname  
;
```

Der Cursor wird geschlossen, die Ergebnismenge verworfen.



Hinweis: In MySQL und MariaDB sind CURSOR schreibgeschützt, vorwärtsgerichtet und serverseitig.

- Schreibgeschützt: Sie arbeiten auf temporären Tabellen, sodass Änderungen nicht in den Tabellen übernommen werden.
- Vorwärtsgerichtet: Man kann immer nur zum nächsten Datensatz gehen und nicht zurück. Auch können keine Datensätze übersprungen werden.
- Serverseitig: Sie können nur innerhalb von Prozeduren verwendet werden. Man kann sie nicht in Client-Anwendungen erstellen und verwenden.

Als Beispiel soll hier die Aufbereitung eines Datenimports dienen. Unten sehen Sie eine XML-Datei, die Bestelldaten mit ihren Positionen enthält. Unglücklicherweise haben die Positionen keine Nummern, sodass wir den Import nicht direkt in die Tabellen `bestellung` und `bestellung_position` durchführen können. Durch unsere Prozedur soll jede Position eine Nummer zugewiesen bekommen. Erst wenn das alles passiert ist, werden die Daten den beiden Zieltabellen hinzugefügt.

```
1  <?xml version="1.0"?>
2  <bestellungen>
3    <bestellung>
4      <bestellung_id>1</bestellung_id>
5      <kunde_id>1</kunde_id>
6      <adresse_id>1</adresse_id>
7      <datum>2012-05-10 10:01</datum>
8      <positionen>
9        <position>
10          <artikel_id>3001</artikel_id>
11          <menge>1</menge>
```

```
12     </position>
13     <position>
14         <artikel_id>3005</artikel_id>
15         <menge>12</menge>
16     </position>
17     <position>
18         <artikel_id>3007</artikel_id>
19         <menge>123</menge>
20     </position>
21 </positionen>
22 </bestellung>
23 <bestellung>
24     <bestellung_id>2</bestellung_id>
25     <kunde_id>2</kunde_id>
26     <adresse_id>1</adresse_id>
27     <datum>2012-05-11 10:02</datum>
28     <positionen>
29     <position>
30         <artikel_id>9010</artikel_id>
31         <menge>2</menge>
32     </position>
33     <position>
34         <artikel_id>7856</artikel_id>
35         <menge>234</menge>
36     </position>
37 </positionen>
38 </bestellung>
39 <bestellung>
40     <bestellung_id>3</bestellung_id>
41     <kunde_id>2</kunde_id>
42     <adresse_id>1</adresse_id>
43     <datum>2012-05-12 10:03</datum>
44     <positionen>
45     <position>
46         <artikel_id>9015</artikel_id>
47         <menge>3</menge>
48     </position>
49 </positionen>
```

```
50     </bestellung>
51 </bestellungen>
```



Laden Sie die Prozedur mit SOURCE
importMitCursor01.sql. Achten Sie darauf, dass die
Datei lstBestellungen.xml im gleichen Verzeichnis
liegt, oder passen Sie die Pfadangaben vorher an.

Bevor wir die Prozedur ausführen können, müssen wir
Importtabellen anlegen und mit dem Inhalt der XML-Dateien
füllen. Ich verwende ein LOAD XML. Die Daten der Bestellungen
sind im Tag <bestellung> und die der Positionen in
<position> abgelegt. Mit dem ROWS IDENTIFIED BY kann so
der Import auf bestimmte Tags eingegrenzt werden. Da kein
Tag <status> in der XML-Datei vorkommt, wird pauschal mit
SET auf offen gesetzt.

```
1  USE oshop;
2  -- Anlegen und Befüllen der Zwischentabell
e
3  -- für die Bestellungen
4  DROP TABLE IF EXISTS tmp_bestellung;
5  CREATE TEMPORARY TABLE tmp_bestellung LIKE
bestellung;
6  SET GLOBAL local_infile = ON;
7  LOAD XML
8      LOCAL INFILE 'lstBestellungen.xml'
9      INTO TABLE tmp_bestellung
10     ROWS IDENTIFIED BY '<bestellung>'
11     SET status = 'offen';
12  SET GLOBAL local_infile = OFF;
13
14  -- Anlegen und Befüllen der Zwischentabell
e
```

```

15  -- für die Bestellungen
16      DROP TABLE IF EXISTS tmp_position;
17      CREATE TEMPORARY TABLE tmp_position LIKE
    bestellung_position;
18      ALTER TABLE tmp_position DROP PRIMARY KE
Y; -- Sonst Probleme!
19      ALTER TABLE tmp_position ADD position_id
    INT AUTO_INCREMENT PRIMARY KEY;
20  SET GLOBAL local_infile = ON;
21      LOAD XML
22      LOCAL INFILE 'lstBestellungen.xml'
23      INTO TABLE tmp_position
24      ROWS IDENTIFIED BY '<position>';
25  SET GLOBAL local_infile = OFF;

```

In [Zeile 18](#) wird der Primärschlüssel ausgeschaltet. In der Zeile darauf wird ein neuer angelegt, der mit einer laufenden Nummer die Positionen eindeutig unterscheidbar macht.



Aufgabe 20.6: Was würde passieren, wenn Sie die Anweisung in [Zeile 18](#) weglassen würden, und warum?

Schauen wir uns mal an, wie die Daten in den temporären Tabellen nach dem Import aussehen:

```

1  mysql> SELECT
2      -> bestellung_id, kunde_id, adresse_id
, datum, status
3      -> FROM tmp_bestellung;
4  +-----+-----+-----+-----+
-----+-----+
5  | bestellung_id | kunde_id | adresse_id |
datum | status |
6  +-----+-----+-----+-----+
-----+-----+

```

```

7   | 1 | 1 | 1 | 2012-05-10 19:45:00 | offen
|
8   | 2 | 2 | 1 | 2012-05-08 17:42:00 | offen
|
9   | 3 | 2 | 1 | 2012-05-02 16:00:01 | offen
|
10  +-----+-----+-----+-----+
-----+-----+
11  3 rows in set (0.00 sec)
12
13  mysql> SELECT
14  -> bestellung_id, position_nr, artikel_id,
    menge, position_id
15  -> FROM tmp_position;
16  +-----+-----+-----+-----+
--+-----+-----+
17  | bestellung_id | position_nr | artikel_id
    | menge | position_id |
18  +-----+-----+-----+-----+
--+-----+-----+
19  | 1 | 0 | 3001 | 5.000000 | 1 |
20  | 1 | 0 | 3005 | 10.000000 | 2 |
21  | 1 | 0 | 3008 | 100.000000 | 3 |
22  | 2 | 0 | 9010 | 1.000000 | 4 |
23  | 2 | 0 | 7856 | 100.000000 | 5 |
24  | 3 | 0 | 9015 | 1.000000 | 6 |
25  +-----+-----+-----+-----+
--+-----+-----+

```

Die Bestelldaten sind genauso importiert worden, wie wir uns das vorgestellt haben. Nur sehen wir sofort ein Problem: Die Bestellnummer kann so nicht übernommen werden.

Grundsätzlich ist auch davon auszugehen, dass importierte Primärschlüssel zu den schon bestehenden nicht einfach so hinzugefügt werden können.

In der Tabelle tmp_position ist die Positionsnummer immer 0, da wir keine Informationen über die Nummer der Position

innerhalb der Bestellung haben. Um später einfach eine bestimmte Position referenzieren zu können, hatten wir ja einen neuen Primärschlüssel angelegt. Das hat auch funktioniert, wie wir am Inhalt der Spalte `position_id` sehen können.

Die benötigte Prozedur wird nun angelegt, und Variablen werden deklariert.

- `vBId`: Der Cursor `curBestellung` durchwandert mit `FETCH` alle Zeilen der Tabelle `tmp_bestellung` und speichert den aktuellen Wert von `bestellung_id` ab.
- `vPId`: Der Cursor `curPosition` durchwandert mit `FETCH` alle Zeilen der temporären Tabelle `tmp_position` und speichert den aktuellen Wert von `position_id` ab.
- `vBestellungId`: Beim Einfügen der Bestellung in die Tabelle `bestellung` wird durch `AUTO_INCREMENT` eine neue Bestellnummer erzeugt. Diese wird per `LAST_INSERT_ID()` ermittelt und zugewiesen.
- `vPosNr`: Ein Zähler, der dazu verwendet wird, die Positionen pro Bestellung zu nummerieren.
- `vEndeHaupt`: Die Schleifenvariable der Hauptschleife. Bei 0 läuft die Schleife, bei 1 soll sie sich beenden.
- `vEndeInnere`: Die Schleifenvariable der inneren Schleife. Bei 0 läuft die Schleife, bei 1 soll sie sich beenden.
- `vWoBinIch`: Mithilfe dieser Variablen unterscheide ich im Handler, welcher Cursor die `NOT FOUND`-Exception ausgelöst hat (siehe [Tabelle 20.2](#)). Der Wert 1 kennzeichnet die Hauptschleife, 2 die innere Schleife.

```
27 DROP PROCEDURE IF EXISTS importBestellung;  
28 DELIMITER //  
29 CREATE PROCEDURE importBestellung()
```



```

30 BEGIN
31 DECLARE vBId INT;
32 DECLARE vPId INT;
33 DECLARE vBestellungId INT;
34 DECLARE vPosNr INT ;
35 DECLARE vEndeHaupt INT DEFAULT 0;
36 DECLARE vEndeInnere INT DEFAULT 0;
37 DECLARE vWoBinIch INT;

```

Ein Cursor wird mit DECLARE eingeleitet. Es folgen der Name des Cursors und die Schlüsselwörter CURSOR FOR. Jetzt kommt ein SELECT, der festlegt, was eigentlich ausgeführt wird, wenn ein Cursor mit OPEN geöffnet wird. Der Cursor curPosition enthält eine WHERE-Klausel, die nur die Positionen zulässt, die zur gerade aktiven Bestellung gehören.

```

39 DECLARE curBestellung CURSOR FOR
40 SELECT bestellung_id FROM tmp_bestellung;
41
42 DECLARE curPosition CURSOR FOR
43 SELECT position_id FROM tmp_position WHERE
    bestellung_id = vBId;

```

Anders als im Beispiel oben (siehe [Abschnitt 20.4](#)) wird hier etwas mehr im Handler zu tun sein. Daher brauchen wir einen Anweisungsblock, der mit BEGIN ... END; umklammert wird. Innerhalb des Handlers wird ermittelt, ob die Exception in der Hauptschleife durch ein FETCH auf curBestellung oder in der inneren Schleife durch eine Exception wegen eines FETCH auf curPosition ausgelöst wurde.

```

45 DECLARE CONTINUE HANDLER FOR SQLSTATE '020
00'
46 BEGIN
47     CASE vWoBinIch
48     WHEN 1 THEN SET vEndeHaupt = 1;
49     WHEN 2 THEN SET vEndeInnere = 1;

```

```
50      END CASE;  
51      END;
```

In [Zeile 53](#) wird der Cursor `curBestellung` geöffnet. Dabei wird der oben deklarierte `SELECT` ausgeführt. Anschließend wird die Schleifenvariable so gesetzt, dass die Schleife läuft. Die Hauptschleife geht von [Zeile 55](#) bis [89](#). In der ersten Zeile der Hauptschleife wird erst einmal in `vWoBinIch` markiert, dass wir in der Hauptschleife sind. Als Nächstes wird ein `FETCH` auf den Cursor ausgeführt. Dabei wird mit dem ersten Datensatz begonnen, der durch den `SELECT` entstanden ist. Bei jedem `FETCH` geht ein interner Zeiger zum nächsten Datensatz. Ist der `FETCH` nicht am Ende angekommen, steht in `vBId` die Bestellnummer, auf die der interne Zeiger gerade verweist. Ansonsten wird die `NOT FOUND`-Exception ausgelöst und der Handler ausgeführt. Da dieser von Typ `CONTINUE` ist, springt die Prozedur zur nächsten Zeile, die nach dem `FETCH` steht. Dort wird dann ermittelt, ob die Hauptschleife sich beenden soll oder nicht.

Ab [Zeile 62](#) wird die Bestellung in die Tabelle `bestellung` eingefügt, auf die der Cursor gerade zeigt. Dabei wird eine neue Bestellnummer vergeben. Diese müssen wir uns merken, da wir den Wert als Fremdschlüsselwert noch brauchen. Wichtig ist, dass wir hier den Zähler für die Position innerhalb der Bestellung wieder auf 1 setzen. Jetzt wird mit `OPEN` der zweite Cursor geöffnet. Dieser enthält alle Positionen, die zur aktuellen Bestellung gehören.

Die innere Schleife geht von [Zeile 72](#) bis [86](#). Als Erstes wird wieder in `vWoBinIch` vermerkt, dass wir in der inneren Schleife sind. Wie oben wird mit `FETCH` die nächste Position ermittelt oder die Exception ausgelöst. Wurde ein `position_id` ermittelt, werden die entsprechenden Daten ab [Zeile 79](#) in die Tabelle `bestellung_position` eingefügt. Bitte beachten Sie,

dass hierbei die neu ermittelte `bestellung_id` und die mitgezählte `vPosNr` verwendet werden.

Ist die innere Schleife zu Ende, wird der Cursor `curPosition` geschlossen, und die Hauptschleife startet eine neue Iteration. Wird die Hauptschleife beendet, wird der Cursor `curBestellung` geschlossen, und alles ist gut ;-).

```
53  OPEN curBestellung; -- Jetzt wird der SELE
CT ausgeführt
54      SET vEndeHaupt = 0;
55      hauptschleife: LOOP -- Start Hauptschlei
fe
56          SET vWoBinIch = 1; -- Ich bin in der Hau
ptschleife
57          FETCH curBestellung INTO vBId; -- Entwed
er Wert oder CONTINUE HANDLER
58          IF vEndeHaupt = 1 THEN
59              LEAVE hauptschleife;
60          END IF;
61
62          INSERT INTO bestellung -- Füge die neue
Bestellung ein
63              SELECT NULL, kunde_id, adresse_id, datum
, status, deleted
64              FROM tmp_bestellung
65              WHERE bestellung_id = vBId; -- Und zwar
nur die eine!
66
67          SET vBestellungId = LAST_INSERT_ID(); --
Neue Bestellnummer
68          SET vPosNr = 1; -- Position bei 1 starte
n
69          SET vEndeInnere = 0;
70          OPEN curPosition; -- Jetzt wird der SELE
CT ausgeführt
71
72      innereschleife: LOOP -- Start inneren Schl
```

```

eife
73     SET vWoBinIch = 2; -- Ich bin in der inn
    eren Schleife
74     FETCH curPosition INTO vPId; -- Entweder
    Wert oder CONTINUE HANDLER
75     IF vEndeInnere = 1 THEN
76     LEAVE innereschleife;
77     END IF;
78
79     INSERT INTO bestellung_position -- Posit
    ion mit neuen Bestell-/Posnr
80     SELECT vBestellungId, vPosNr, artikel_id
    , menge, deleted
81     FROM tmp_position
82     WHERE position_id = vPId;
83
84     SET vPosNr = 1 + vPosNr; -- Erhöhe Posit
    ionsnummer
85     ITERATE innereschleife;
86     END LOOP; -- Ende inneren Schleife
87     CLOSE curPosition; -- SchlieÙe den Curso
    r
88     ITERATE hauptschleife;
89     END LOOP; -- Ende Hauptschleife
90
91     CLOSE curBestellung; -- SchlieÙe den Curso
    r
92     END//
93     DELIMITER ;

```

Führen wir die Prozedur nun aus und betrachten das Ergebnis:

```

1  mysql> CALL importBestellung();
2
3  mysql> SELECT
4      -> bestellung_id, kunde_id, adresse_id
    , datum, status
5      -> FROM bestellung

```

```

6      -> ;
7  +-----+-----+-----+-----+
-----+-----+
8  | bestellung_id | kunde_id | adresse_id |
datum | status |
9  +-----+-----+-----+-----+
-----+-----+
10 [...]
11 | 7 | 1 | 1 | 2012-05-10 10:01:00 | offen
|
12 | 8 | 2 | 1 | 2012-05-11 10:02:00 | offen
|
13 | 9 | 2 | 1 | 2012-05-12 10:03:01 | offen
|
14 +-----+-----+-----+-----+
-----+-----+
15 9 rows in set (0.00 sec)
16
17 mysql> SELECT *
18      -> FROM bestellung_position
19      -> ;
20 +-----+-----+-----+-----+
-+-----+-----+
21 | bestellung_id | position_nr | artikel_id
| menge | deleted |
22 +-----+-----+-----+-----+
-+-----+-----+
23 [...]
24 | 7 | 1 | 3001 | 1.000000 | 0 |
25 | 7 | 2 | 3005 | 12.000000 | 0 |
26 | 7 | 3 | 3007 | 123.000000 | 0 |
27 | 8 | 1 | 9010 | 2.000000 | 0 |
28 | 8 | 2 | 7856 | 234.000000 | 0 |
29 | 9 | 1 | 9015 | 3.000000 | 0 |
30 +-----+-----+-----+-----+
-+-----+-----+
31 19 rows in set (0.01 sec)

```



Hinweis: Cursor sind nicht sehr schnell. Falls es irgendwie möglich ist, sollten Sie die üblichen INSERT, UPDATE, DELETE und SELECT mit sehr schlaue gebauten WHERE-Klauseln verwenden.

20.6 Aufgaben



Aufgabe 20.7: Machen Sie eine Performancemessung: Ein SELECT * auf die Tabelle bank versus durchwandern mit einem Cursor.

Aufgabe 20.8: Erstellen Sie eine Prozedur, die einer Bestellung eine neue Position hinzufügt. Die Übergabeparameter müssen plausibilisiert werden.

Aufgabe 20.9: Erweitern Sie diese Prozedur als Transaktion und berücksichtigen Sie auch den Lagerbestand! Einmal bei der Plausibilitätsüberprüfung und wobei noch?

Aufgabe 20.10: Erstellen Sie eine Prozedur, die aus einer Bestellung eine Position löscht. Die Positionsnummern sollen wieder richtig durchnummeriert sein. Die Übergabeparameter müssen plausibilisiert werden.

Aufgabe 20.11: Erstellen Sie eine Prozedur, die zwei Positionsnummern vertauscht. Die Übergabeparameter müssen plausibilisiert werden.

Aufgabe 20.12: Erstellen Sie eine Prozedur, die zu einer Bestellnummer eine passende Rechnung erstellt. Dabei

soll überprüft werden, ob es schon eine Rechnung zu der Bestellung gibt. Vergessen Sie mir nicht die Positionen!

Aufgabe 20.13: Erstellen Sie eine Prozedur, die in einer neuen Tabelle `revision` folgende Informationen ablegt: Liste der Warengruppen ohne Artikel, Liste der Artikel ohne Warengruppe, Liste der Adressen ohne Kunden oder Lieferanten, Liste der Lieferanten ohne Artikel und Liste der Artikel ohne Lieferanten. Die Tabelle soll nur zwei Spalten haben: einen Zeitstempel und einen Berichtstext.



Hinweis: Weitere Beispiele für Prozeduren finden Sie in [Kapitel 27](#) über die Experimente.

¹ engl: session

² Das ist eine Schleife innerhalb einer Schleife.

³ Für alle, die in der C-Sprachfamilie zu Hause sind, dürfte sich ein Vergleich mit dem `break`; anbieten.

⁴ Für alle, die in der C-Sprachfamilie zu Hause sind, dürfte sich ein Vergleich mit dem `continue`; anbieten.

⁵ Zumindest unter MariaDB; bei MySQL erscheint die Fehlermeldung: *ERROR 1264 (22003): Out of range value for column 'vInt' at row 1*

⁶ engl.: label

⁷ griechischer Gelehrter: * zwischen 276 und 273 v. Chr. in Kyrene; † um 194 v. Chr. in Alexandria.

⁸ Euklid von Alexandria; * ca. 360 v. Chr.; † ca. 280 v. Chr.; griechischer Mathematiker.

⁹ Bitte, das ist keine numerisch stabile und plausibilisierte Funktion, sondern nur ein Beispiel!

¹⁰ Abkürzung für: current set of records

21 Funktionen



Eigene Ergebnisse mit selbst erstellten Funktionen erzeugen.

- Vertiefendes
 - Anlegen mit `CREATE FUNCTION`
 - Werterückgabe mit `RETURN`
 - Funktionen mit `DROP FUNCTION` löschen



Die Quelltexte des Kapitels stehen in der Datei `mysql/listing17.sql` (siehe [Listing 28.19](#)).



MySQL/MariaDB

```
CREATE FUNCTION funktionsname ([parameter][,  
parameter]*)
```

```
    RETURNS typ
```

```
    anweisungsblock
```

```
;
```

```
parameter:
```

[IN|OUT|INOUT] *parametername typ*

typ:

jeder gültige MySQL-Datentyp (siehe [Abschnitt 26.1](#))

Funktionen ähneln in vielen Aspekten den Prozeduren. Wie diese kennen Funktionen folgende Komponenten: Namen, Übergabeparameter, lokale Variablen, Anweisungen, Verzweigungen, Schleifen und Cursor. Der wesentliche Unterschied ist, dass Funktionen *immer* einen Wert zurückliefern müssen.

Auch Prozeduren können Rückgaben liefern, aber die Betonung liegt auf *können*. Funktionen müssen Werte liefern. Im Falle eines Fehlers oder eines unbestimmten Ergebnisses wird üblicherweise NULL verwendet.

Ein einfaches Beispiel: Wir wollen mithilfe einer Funktion Preise prozentual verändern. Das Ergebnis soll auf zwei Stellen nach dem Komma gerundet werden.

```
1  DROP FUNCTION IF EXISTS preis_anpassen;
2  DELIMITER //
3  CREATE FUNCTION preis_anpassen
4  (
5      iPreisOriginal DECIMAL(14,6), -- Originalpreis
6      iProzentsatz DECIMAL(7,3) -- Anpassung in Prozent (-100-100)
7  )
8  RETURNS DECIMAL(14,6) DETERMINISTIC
9  BEGIN
10 DECLARE vPreisNeu DECIMAL(14,6); -- Neuer Preis
11
12     IF iPreisOriginal IS NULL THEN -- Test a
```

```

uf NULL
13     RETURN NULL;
14     END IF;
15     IF iProzentsatz IS NULL THEN -- Test auf
    NULL
16     RETURN NULL;
17 END IF;
18     IF iProzentsatz NOT BETWEEN -100 AND 100
    THEN -- Wertebereich Prozent
19     RETURN NULL;
20     END IF;
21     SET vPreisNeu = iPreisOriginal + ((iPrei
sOriginal * iProzentsatz) / 100.0);
22     SET vPreisNeu = ROUND(vPreisNeu, 2);
23     RETURN vPreisNeu;
24 END //
25 DELIMITER ;

```



Hinweis: Eine weitere Möglichkeit ist das Erstellen *user defined functions*, die als DLL-Dateien dem Server zur Verfügung gestellt werden. Das ist cool und interessant und kann im Detail in <https://dev.mysql.com/doc/extending-mysql/8.0/en/adding-functions.html> nachgelesen werden.

Sie können aus Funktionen keine Ausgaben beispielsweise mit SELECT erzeugen. Dies verhindert leider auch die Ausgabe von Warnungen oder Fehlermeldungen. Sie werden dazu *exception handle* verwenden müssen (siehe [Abschnitt 20.4](#)).

Eine Funktion kann mit DROP FUNCTION gelöscht werden.



MySQL/MariaDB

```
DROP FUNCTION [IF EXISTS] functionsname  
;
```

22 Auslöser



Domino Day für Fortgeschrittene: Wie eins zum anderen führt.

- Vertiefendes
 - Anlegen mit `CREATE TRIGGER`
 - Verfügbarkeit von `OLD` und `NEW`
 - Trigger mit `DROP TRIGGER` löschen



Die Quelltexte des Kapitels stehen in der Datei `mysql/listing18. sql` (siehe [Listing 28.20](#)).

22.1 Was ist das?



Definition 73: Trigger

Ein *Trigger* oder *Auslöser* ist ein benannter Anweisungsblock mit folgenden Eigenschaften:

- Der Triggername ist in der Datenbank einmalig.

- Der Trigger hat keine Übergabeparameter.
- Der Trigger liefert nichts zurück.
- Der Trigger ist mit einer Tabelle verbunden.
- Sie müssen festlegen, wann der Trigger automatisch aufgerufen wird: bei einem INSERT, UPDATE oder DELETE auf der verbundenen Tabelle.
- Sie müssen festlegen, ob der Trigger vor oder nach Ausführung von INSERT, UPDATE oder DELETE automatisch aufgerufen wird.

Trigger erinnern stark an Prozeduren (siehe [Kapitel 20](#)), da es benannte Anweisungsblöcke sind. Der Unterschied zu einer Prozedur ist, dass ein Trigger *automatisch* aufgerufen wird, wenn ein festgelegtes Ereignis bei einer Tabelle eintritt.

Man kann Trigger immer dann einsetzen, wenn das Ereignis auf der Tabelle zwingend mit einer Aktion verbunden werden muss. Natürlich kann dies auch durch die Programmierung im Client oder durch passende Prozeduren erreicht werden.

Trigger bieten sich hier aber als Alternativen an. Der Trigger wird mit `CREATE TRIGGER` angelegt.



MySQL/MariaDB

```
CREATE TRIGGER triggername
{BEFORE|AFTER}
{INSERT|UPDATE | DELETE}
ON tabellename
FOR EACH ROW
anweisungsblock
;
```

Mit `DROP TRIGGER` wird er wieder gelöscht.



MySQL/MariaDB

```
DROP TRIGGER [IF EXISTS] triggername
;
```

Trigger sind aber auch nicht ohne. Sie werden automatisch aufgerufen, was zwangsläufig dazu führt, dass man vergisst, dass sie aufgerufen werden. Stellen Sie sich vor, wir machen ein `UPDATE` auf einer Tabelle mit 100.000 Zeilen. Dann wird für jede Zeile, die verändert wird, der Anweisungsblock des Triggers aufgerufen. Das kann die Performance einer Anweisung dramatisch verschlechtern.



Hinweis: Achten Sie darauf, dass es keine Zirkelaufrufe gibt.

Was bedeutet das? In Tabelle 1 wird eine Zeile eingefügt. Dies löst einen Trigger aus, der in Tabelle 2 eine Zeile einfügt. Dies löst ein Trigger aus, der in Tabelle 3 etwas löscht. Dies löst einen Trigger aus, der in Tabelle 1 eine Zeile einfügt :-).

Innerhalb eines Triggers stehen bestimmte Werte zur Verfügung, die man oft bei der Triggerverarbeitung braucht. Bei einem `INSERT` kann über `NEW` der neu eingefügte Datensatz erreicht werden. Bei `UPDATE` steht `NEW` für den neuen Wert und `OLD` für den alten Wert einer Spalte. Bei `DELETE` gibt es nur ein `OLD`.

Tabelle 22.1 Verfügbarkeit von `NEW` und `OLD` in einem Trigger

Anweisung	NEW	OLD
INSERT	ja	nein
UPDATE	ja	ja
DELETE	nein	ja

22.2 Ein Beispiel für einen INSERT-Trigger

Wenn wir eine Bestellung anlegen, soll auch eine passende Rechnung erzeugt werden. Dadurch muss der Anwender die Bestelldaten nicht noch einmal manuell als Rechnungsdaten erfassen. Fangen wir mit den Bestelldaten an und machen anschließend die Positionen. In [Zeile 2](#) wird der Triggernamen festgelegt. Ich verwende dazu folgende Namenskonvention:



Hinweis: Namenskonvention bei einem Trigger:

tri_tabellenname_zeitpunkt_anweisung

In [Zeile 3](#) wird festgelegt, dass es ein Trigger ist, der *nach* einem INSERT in Tabelle bestellung ([Zeile 4](#)) erfolgt.

```

1  DELIMITER //
2  CREATE TRIGGER tri_bestellung_after_insert
-- ein schöner Name
3  AFTER INSERT -- wann?
4  ON bestellung -- bei wem?
5  FOR EACH ROW

```



```

6 BEGIN
7 INSERT INTO rechnung -- was?
8     SET
9         kunde_id = NEW.kunde_id,
10        bestellung_id = NEW.bestellung_id,
11        adresse_id = NEW.adresse_id,
12        datum = NEW.datum;
13 END//
14 DELIMITER ;

```

In [Zeile 7](#) steht der neue INSERT, der die Daten von den Bestellungen zu den Rechnungen kopiert. Dabei muss der Trigger die gerade neu eingefügten Daten kennen. Dazu steht das Schlüsselwort NEW zur Verfügung. Durch einen Punkt abgetrennt geben Sie den Spaltennamen an, in dem der neue Wert steht. Das ist schwieriger zu formulieren als zu verwenden. Schauen Sie in den Quelltext ;-).

Jetzt sind wir doch alle gespannt, ob alles klappt:

```

1  mysql> INSERT INTO
2      -> bestellung (kunde_id, adresse_id, d
atum)
3      -> VALUES (1, 1, NOW())
4      -> ;
5
6  mysql> SELECT bestellung_id, kunde_id, adr
esse_id, datum
7      -> FROM bestellung
8      -> ;
9  +-----+-----+-----+-----+
-----+
10 | bestellung_id | kunde_id | adresse_id |
datum |
11 +-----+-----+-----+-----+
-----+
12 [...]
13 | 10 | 1 | 1 | 2023-07-14 12:30:29 |

```

```
14  +-----+-----+-----+
-----+
```

Das war klar, in der Tabelle `bestellung` ist der INSERT angekommen. Jetzt stellt sich die spannende Frage, was in der Tabelle `rechnung` steht:

```
1  mysql> SELECT rechnung_id, kunde_id, beste
2         llung_id, adresse_id, datum
3         -> FROM rechnung\G
4         [...]
5         ***** 12. row *****
6         *****
7         rechnung_id: 12
8         kunde_id: 1
9         bestellung_id: 10
10        adresse_id: 1
11        datum: 2023-07-14 12:30:29
12  12 rows in set (0.00 sec)
```

Genau wie wir es wollten: Zur Bestellung ist eine passende Rechnung angelegt worden.



Aufgabe 22.1: Testen Sie das Ganze noch einmal mit einem INSERT, der gleich mehrere Bestellungen einfügt.

Aufgabe 22.2: Programmieren Sie den passenden Trigger zum Kopieren der Bestellpositionen zu Rechnungspositionen.

22.3 Ein Beispiel für einen UPDATE-

Trigger

Werden in einer Bestellung die Daten oder Positionen geändert, muss dies auch in der entsprechenden Rechnung nachgehalten werden. Zuerst merken wir uns in [Zeile 9](#) den Primärschlüsselwert der Bestellung, damit wir die passende Rechnung wiederfinden.

Anschließend wird für die entscheidenden Spalten überprüft, ob dort eine Änderung erfolgt ist. Dazu wird der alteWert mit dem neuen verglichen ([Zeile 11](#))¹.

```
1  DELIMITER //
```

```
2  CREATE TRIGGER tri_bestellung_after_update
```

```
3  AFTER UPDATE
```

```
4      ON bestellung
```

```
5      FOR EACH ROW
```

```
6      BEGIN
```

```
7          DECLARE vBestellungId INT;
```

```
8
```

```
9  SET vBestellungId = OLD.bestellung_id; --
```

Damit wir es wiederfinden

```
10
```

```
11  IF OLD.kunde_id != NEW.kunde_id THEN -- Nu
```

r, wenn nötig

```
12      UPDATE rechnung
```

```
13      SET kunde_id = NEW.kunde_id
```

```
14      WHERE bestellung_id = vBestellungId;
```

```
15  END IF;
```

```
16
```

```
17      IF OLD.adresse_id != NEW.adresse_id THEN
```

```
18      UPDATE rechnung
```

```
19      SET adresse_id = NEW.adresse_id
```

```
20      WHERE bestellung_id = vBestellungId;
```

```
21  END IF;
```

```
22
```

```
23      IF OLD.datum != NEW.datum THEN
```

```

24     UPDATE rechnung
25     SET datum = NEW.datum
26     WHERE bestellung_id = vBestellungId;
27     END IF;
28
29 END//
30 DELIMITER ;

```

Durch ein UPDATE lösen wir jetzt nicht nur einen Trigger aus, sondern mehrere. Alle Bestellungen des Kunden 1 hatten die adresse_id 1, bevor sie auf 2 geändert wurden.

```

1  mysql> UPDATE bestellung SET adresse_id =
2  WHERE kunde_id = 1;
3
3  mysql> SELECT bestellung_id, kunde_id, adr
esse_id, datum
4  -> FROM bestellung
5  -> WHERE kunde_id=1;
6  +-----+-----+-----+-----+
-----+
7  | bestellung_id | kunde_id | adresse_id |
datum |
8  +-----+-----+-----+-----+
-----+
9  | 1 | 1 | 2 | 2012-03-24 17:41:00 |
10 | 3 | 1 | 2 | 2011-01-15 16:43:00 |
11 | 4 | 1 | 2 | 2011-01-16 09:15:00 |
12 | 5 | 1 | 2 | 2011-01-16 09:16:00 |
13 | 7 | 1 | 2 | 2012-05-10 10:01:00 |
14 | 10 | 1 | 2 | 2023-07-14 12:30:29 |
15 +-----+-----+-----+-----+
-----+

```

Das UPDATE in der Tabelle bestellung war erfolgreich, und wie sieht es nun in der Tabelle rechnung aus?

```

1  mysql> SELECT rechnung_id, kunde_id, beste

```

```

1  lung_id, adresse_id
2  -> FROM rechnung
3  -> WHERE kunde_id = 1;
4  +-----+-----+-----+-----+
-----+
5  | rechnung_id | kunde_id | bestellung_id |
   | adresse_id |
6  +-----+-----+-----+-----+
-----+
7  | 1 | 1 | 1 | 2 |
8  | 3 | 1 | 3 | 2 |
9  | 4 | 1 | 4 | 2 |
10 | 5 | 1 | 5 | 2 |
11 | 12 | 1 | 10 | 2 |
12 +-----+-----+-----+-----+
-----+

```

Sie können jetzt sehen, dass nicht nur eine Zeile geändert wurde, sondern alle betroffenen Zeilen.



Aufgabe 22.3: Programmieren Sie den analogen Trigger für die Aktualisierung der Rechnungspositionen.

22.4 Ein Beispiel für einen DELETE-Trigger

Wird eine Bestellung gelöscht, werden vorher alle ihre Positionen gelöscht.²

```

1  DELIMITER //
2  CREATE TRIGGER tri_bestellung_before_delet

```

```

e
3     BEFORE DELETE
4     ON bestellung
5     FOR EACH ROW
6     BEGIN
7     DECLARE vBestellungId INT;
8     DECLARE vRechnungId INT;
9
10    SET vBestellungId = OLD.bestellung_id; -
- Damit wir es wiederfinden
11    SELECT rechnung_id
12        FROM rechnung
13        WHERE bestellung_id = vBestellungId
14        INTO vRechnungId;
15
16    DELETE FROM rechnung_position WHERE rechnung_id = vRechnungId;
17    DELETE FROM rechnung WHERE rechnung_id = vRechnungId;
18    DELETE FROM bestellung_position WHERE bestellung_id = vBestellungId;
19    END//
20    DELIMITER ;

```

Zuerst will ich den Ist-Zustand ermitteln:

```

1  mysql> SELECT
2      -> (
3      -> SELECT COUNT(*) FROM bestellung_pos
4      -> ) AS AnzPosBestellung,
5      -> (
6      -> SELECT COUNT(*)
7      -> FROM
8      -> rechnung_position INNER JOIN rechnung
9      -> USING(rechnung_id)
10     -> WHERE bestellung_id = 5
11     -> ) AS AnzPosRechnung

```

```

11      -> ;
12  +-----+-----+
13  | AnzPosBestellung | AnzPosRechnung |
14  +-----+-----+
15  | 4 | 4 |
16  +-----+-----+

```

Beide Tabellen haben vier Positionen zur Bestellung mit der Nummer 5. Wir können blind annehmen, dass es jeweils einen passenden Datensatz in den Tabellen `bestellung` und `position` gibt.



Aufgabe 22.4: Warum können wir das einfach so annehmen?

Tipp: Siehe [Definition 22](#) in [Abschnitt 2.3](#).

Auf zum munteren Löschen:

```

1  mysql> DELETE FROM bestellung WHERE bestel
lung_id = 5;
2
3  mysql> SELECT
4      -> (
5      -> SELECT COUNT(*) FROM bestellung WHE
RE bestellung_id = 5
6      -> ) AS AnzBestellung,
7      -> (
8      -> SELECT COUNT(*) FROM bestellung_pos
ition WHERE bestellung_id = 5
9      -> ) AS AnzPosBestellung,
10     -> (
11     -> SELECT COUNT(*) FROM rechnung WHERE
bestellung_id = 5
12     -> ) AS AnzRechnung,
13     -> (

```

```

14      -> SELECT COUNT(*)
15      -> FROM
16      -> rechnung_position INNER JOIN rechnu
ng USING(rechnung_id)
17      -> WHERE bestellung_id = 5
18      -> ) AS AnzPosRechnung
19      -> ;
20
21  +-----+-----+-----+-----+
-----+-----+
22  | AnzBestellung | AnzPosBestellung | AnzRe
chnung | AnzPosRechnung |
23  +-----+-----+-----+-----+
-----+-----+
24  | 0 | 0 | 0 | 0 |
25  +-----+-----+-----+-----+
-----+-----+

```

Alles leer geräumt und weggeputzt.



Aufgabe 22.5: Bauen Sie einen Trigger, der lediglich eine gelöschte Bestellposition auch in rechnung_position löscht.



Hinweis: Mit SHOW TRIGGERS kann eine Liste der aktuell gültigen Trigger eingesehen werden.

```

1  mysql> SHOW TRIGGERS\G
2  ***** 1. row *****
*****
3          Trigger: tri_bestellung_after_
insert

```



```

4          Event: INSERT
5          Table: bestellung
6          Statement: BEGIN
7      INSERT INTO rechnung
8  [...]
9  END
10         Timing: AFTER
11         Created: 2023-07-14 12:30:36.5
2
12         sql_mode: STRICT_TRANS_TABLES,NO_E
ENGINE_SUBSTITUTION
13         Definer: root@localhost
14 character_set_client: utf8mb4
15 collation_connection: utf8mb4_0900_ai_ci
16 Database Collation: utf8mb4_german2_ci
17 ***** 2. row *****
*****
18         Trigger: tri_bestellung_after_upda
te
19         Event: UPDATE
20         Table: bestellung
21         Statement: BEGIN
22     [...]
23     END
24         Timing: AFTER
25         Created: 2023-07-14 12:33:29.96
26         sql_mode: STRICT_TRANS_TABLES,NO_E
ENGINE_SUBSTITUTION
27         Definer: root@localhost
28 character_set_client: utf8mb4
29 collation_connection: utf8mb4_0900_ai_ci
30 Database Collation: utf8mb4_german2_ci
31 ***** 3. row *****
*****
32         Trigger: tri_bestellung_before_del
ete
33         Event: DELETE
34         Table: bestellung

```

```
35          Statement: BEGIN
36  [...]
37  END
38          Timing: BEFORE
39          Created: 2023-07-14 12:35:17.09
40          sql_mode: STRICT_TRANS_TABLES,NO_ENG
INE_SUBSTITUTION
41          Definer: root@localhost
42  character_set_client: utf8mb4
43  collation_connection: utf8mb4_0900_ai_ci
44  Database Collation: utf8mb4_german2_ci
45  3 rows in set (0.00 sec)
```

¹ Das ist natürlich aus Sicht der Performance Quatsch. Es wäre viel besser, alle relevanten Spalten in einer Anweisung zu aktualisieren, egal ob sich die Werte verändert haben. Ich wollte hier nur den Effekt von `OLD` und `NEW` darstellen.

² An welche *böse* Praxis erinnert Sie dieses Vorgehen?

23 Ereignisse



Einen Terminkalender für den SQL Server erstellen.

- Vertiefendes
 - Anlegen mit `CREATE EVENT`
 - Anzeigen von Terminen mit `SHOW EVENTS`
 - Konfigurieren des Terminkalenders
 - Termine mit `ALTER EVENT ... ENABLE` aktivieren
 - Termine mit `ALTER EVENT ... DISABLE` deaktivieren
 - Termine mit `DROP EVENT` löschen



Die Quelltexte des Kapitels stehen in der Datei `mysql/listing19.sql` (siehe [Listing 28.21](#)).

23.1 Wie lege ich ein Ereignis an?



Definition 74: Event

Ein *Event* oder *Ereignis* ist ein benannter Anweisungsblock, der aufgrund eines zeitlichen Ereignisses aufgerufen wird.

Durch ein Ereignis können Sie in periodischen Abständen oder einmalig zu einem bestimmten Zeitpunkt einen Anweisungsblock ausführen. Erinnern Sie sich noch daran, dass wir regelmäßig statistische Daten über unsere Bestellungen in Tabellen abgelegt haben (siehe [Abschnitt 20.2.2](#))? Wir haben dort eine Prozedur erstellt, die für einen bestimmten Zeitraum den Umsatz anhand der Rechnungsdaten ermittelte.

Den Aufruf dazu wollen wir jetzt automatisieren, indem wir ihn an ein bestimmtes zeitliches Ereignis binden. Dazu verwenden wir den CREATE EVENT-Befehl. Dieser sieht zwar im ersten Moment etwas kompliziert aus, ist aber im Prinzip sehr einfach und selbsterklärend.



MySQL/MariaDB

```
CREATE
EVENT [IF NOT EXISTS] eventname
ON SCHEDULE termin
DO
anweisungsblock
;
termin:
AT zeitpunkt [+ INTERVAL intervall]*
| EVERY intervall
[STARTS zeitpunkt [+ INTERVAL intervall]*]
[ENDS zeitpunkt [+ INTERVAL intervall]*]
```

```
intervall:  
zahl {YEAR|QUARTER|MONTH|WEEK|DAY|HOUR|MIN  
UTE|SECOND  
|YEAR_MONTH|DAY_HOUR|DAY_MINUTE|  
DAY_SECOND|HOUR_MINUTE|HOUR_SECOND|MINUTE_  
SECOND}
```

Man gibt an, zu welchem *termin* ein bestimmter *anweisungsblock* ausgeführt werden soll. Nehmen wir als erstes Beispiel, dass wir den totalen Umsatz wöchentlich ermitteln wollen.



Hinweis: Namenskonvention bei einem Ereignis:
event_aufgabe_termin

```
1 CREATE EVENT event_umsatz_woche  
2 ON SCHEDULE EVERY 1 WEEK  
3 DO CALL  
4 umsatzreport(1)  
5 ;
```

Jetzt wird jedeWoche der Totalumsatz ermittelt.



Hinweis: Mit `SHOW EVENTS` kann eine Liste der aktuell gültigen Ereignisse eingesehen werden.

```
1 mysql> SHOW EVENTS\G  
2 ***** 1. row *****  
*****  
3 Db: oshop
```

```

4          Name: event_umsatz_woche
5          Definer: root@localhost
6          Time zone: SYSTEM
7          Type: RECURRING
8          Execute at: NULL
9          Interval value: 2
10         Interval field: WEEK
11         Starts: 2023-07-14 12:47:14
12         Ends: NULL
13         Status: ENABLED
14         Originator: 1
15 character_set_client: utf8mb4
16 collation_connection: utf8mb4_0900_ai_ci
17 Database Collation: utf8mb4_german2_ci

```



Aufgabe 23.1: Erstellen Sie solche Ereignisse, dass der Jahresumsatz vierteljährlich und der Monatsumsatz monatlich (bezogen auf ein Jahr) erstellt wird.

Damit ein Ereignis überhaupt ausgeführt wird, muss der *event scheduler* aktiviert werden. Den aktuellen Status ermitteln Sie mit:

```

1  mysql> SHOW VARIABLES LIKE 'event%';
2  +-----+-----+
3  | Variable_name | Value |
4  +-----+-----+
5  | event_scheduler | OFF |
6  +-----+-----+

```

Tabelle 23.1 Bedeutung von event_scheduler

Wert		Bedeutung

0	OFF	Scheduler ist deaktiviert.
1	ON	Scheduler ist aktiviert und führt Ereignisse aus.
2	DISABLED	Scheduler ist aktiviert, führt aber keine Ereignisse aus.

Der *event scheduler* wird entweder global über die Konfigurationsdatei `my.ini` oder `my.cnf` gesteuert oder zur Laufzeit über `SET GLOBAL`:

```
1 SET GLOBAL event_scheduler = ON;
```

Machen wir doch mal einen kleinen Test:

```
1 DROP TABLE IF EXISTS a;
2 CREATE TABLE a
3 (
4     zeit DATETIME
5 );
6 DROP EVENT IF EXISTS event_zeit_sekunde;
7 CREATE EVENT event_zeit_sekunde
8 ON SCHEDULE EVERY 2 SECOND
9 DO
10 INSERT INTO a VALUES (NOW())
11 ;
12
13 -- Warte 20 Sekunden
14 SELECT * FROM a;
15 +-----+
16 | zeit |
17 +-----+
18 | 2023-07-14 12:47:14 |
19 | 2023-07-14 12:47:16 |
20 | 2023-07-14 12:47:18 |
```

21		2023-07-14 12:47:20	
22		2023-07-14 12:47:22	
23		2023-07-14 12:47:24	
24		2023-07-14 12:47:26	
25		2023-07-14 12:47:28	
26		2023-07-14 12:47:30	
27		2023-07-14 12:47:32	
28	+	-----	+

23.2 Wie werde ich Ereignisse wieder los?

Das hängt von der Art des Ereignisses ab:

- *Einmalig*: Wird ein Ereignis nur einmal zu einem Zeitpunkt ausgeführt, wird das Ereignis nach diesem Zeitpunkt automatisch entfernt.
- *Periodisch mit Endeangabe*: Ist der Endtermin erreicht, wird das Ereignis automatisch entfernt.
- *Periodisch ohne Endeangabe*: Wollen Sie das Ereignis nur deaktivieren und nicht löschen, können Sie `ALTER EVENT` verwenden, ansonsten `DROP EVENT`.

Man kann ein Ereignis mit `ALTER EVENT` deaktivieren:



MySQL/MariaDB

```
ALTER
  EVENT eventname
  [ENABLE|DISABLE]
;
```



```
1 ALTER EVENT event_umsatz_woche DISABLE;
```

Damit kann das Ereignis jederzeit wieder aktiviert werden, wenn der Grund für die Pause entfällt.



Hinweis: Bitte beachten Sie, dass alle Ereignisse, die Sie jetzt im Rahmen Ihrer Ausbildung anlegen, automatisch im Hintergrund auf dem Server laufen. Weisen Sie deshalb Übungsereignissen einen zeitnahen Endtermin zu.

Ein Ereignis kann komplett mit `DROP EVENT` gelöscht werden.



MySQL/MariaDB

```
DROP EVENT [IF EXISTS]
eventname
;
```

Also räume ich besser hier noch auf, bevor ich für heute Schlussmache:

```
1 DROP EVENT event_umsatz_woche;
```



Aufgabe 23.2: Sie müssen noch das Ereignis `event_zeit_sekunde` löschen und ggf. den *event scheduler* ausschalten.

TEIL VI

Anhänge

24 Datenbank administrieren

24.1 Daten sichern und wiederherstellen

24.1.1 Backup

Datensicherungen sind eine wichtige Sache. Sie ermöglichen nicht nur die Wiederherstellung von Daten nach einem Hardware-Problem. Auch wenn Sie versehentlich Daten gelöscht oder verändert haben, sind Sie über jede Sicherheitskopie froh.

Aus eigener Erfahrung kann ich Ihnen berichten, dass nach zerstörter referenzieller Integrität ohne alte Datensicherungen eine Reparatur mit vertretbarem Aufwand kaum möglich ist. Ihnen bleibt nur die manuelle Datenerhebung aus den Papierunterlagen.

So viel zur Motivation. Möglichkeiten zur Datensicherung gibt es viele. Eine ist das Sichern der Dateien im Datenverzeichnis des Servers:¹

```
1  bash-4.4# cp /var/lib/mysql/oshop/*.{frm,MYD,MYI} ./save/
2  bash-4.4# cp /var/lib/mysql/oshop/db.opt ./save/
```



Hinweis: Das Sichern auf Dateiebene kann nur bei MyISAM durchgeführt werden, da bei InnoDB-Tabellen die Dateien nicht zwingend den aktuellen Stand beinhalten.

Ich empfehle Ihnen die Sicherung mit `mysqldump`. Dieses Tool sichert die Daten als SQL-Skript, wobei binäre Daten – wie Bilder etc. – in hexadezimalen Zahlen abgelegt werden können. Empfohlen wird:² `mysqldump --opt --databases datenbankname > dateiname.sql`

Schauen wir uns an, was `--opt` bedeutet. Diese Option ist ein Synonym für folgende Optionen:

- `--add-drop-table`: Vor jedem `CREATE TABLE` wird die gegebenenfalls vorhandene Tabelle gelöscht.
- `--add-locks`: Die Tabellen werden vor dem Einfügen der Daten mit `LOCK TABLES` gesperrt und danach wieder mit `UNLOCK TABLES` freigegeben. Das Einfügen wird dadurch beschleunigt.
- `--create-options`: Alle MySQL-proprietären Tabellenoptionen bleiben erhalten. Alternativ könnte man mit `--compatible=` diverse SQL-Dialekte auswählen.
- `--disable-keys`: Vor dem Einfügen werden die Schlüssel deaktiviert. Nach dem Einfügen der Datensätze werden diese wieder aktiviert. Dadurch, dass die Schlüssel während des Einfügens nicht aktiv sind, geschieht das Einfügen schneller. Anschließend wird der entsprechende Index wieder aufgebaut. Wie schon in [Abschnitt 6.2.4](#) erwähnt, ist diese Option nur bei MyISAM wirksam.
- `--extended-insert`: Es wird nur ein `INSERT` mit vielen

Zeilen anstelle von vielen INSERTs mit nur einer Zeile verwendet. Auch diese Option beschleunigt das Einfügen.

- `--lock-tables`: Die Tabellen werden bei MyISAM mit `READ LOCAL` gesperrt. Für InnoDB wird stattdessen die Option `--single-transaction` empfohlen.
- `--quick`: Bei großen Datenmengen werden die Datensätze einzeln vom Server geholt. Ansonsten werden erst alle Datensätze gelesen und im Speicher gehalten.
- `--set-charset`: Der verwendete Zeichensatz wird durch `SET NAMES` am Anfang des Skripts angegeben.

Aus diesen Kommentaren und der Tatsache, dass wir binäre Daten haben (Tabelle `bid`), ergibt sich folgender Aufruf:

```
1 bash-4.4# mysqldump -uroot -p --add-drop-d
  atabase --add-locks --create-options --
  extended-insert --single-transaction --quick -
  -set-charset --hex-lob --
  databases oshop > ./save/oshop.sql
```

Die Option `--hex-lob` wandelt die binären Daten – das sind `BINARY`, `VARBINARY`, `BLOB` und `BIT` – in eine hexadezimale Schreibweise um.

Die Optionen können der besseren Übersicht wegen in eine Datei ausgelagert werden:

```
1 bash-4.4# mysqldump -uroot -p < ./oshop.op
  t > ./save/oshop.sql
```

Wenn Sie unter Linux/Unix einen CRON-Job erstellen möchten, der täglich eine Sicherung macht, empfehle ich folgende Zeile:³

```
1 bash-4.4# crontab -e
```

```
2 # Jeden Tag um 23:30 Uhr:
3 30 23 * * * /pfadzumskript/oshop.backup.sh
```

Das Shell-Skript⁴ sieht dann so aus:

```
1 #!/bin/sh
2 ## Einige Variablen
3 echo on
4 MyUSER="root"
5 MyPASS=""
6 MyDB="oshop"
7 MyOption="--add-drop-table --add-locks --c
reate-options --extended-insert --single-trans
action --quick --set-charset --hex-blob --data
bases"
8 MySQLDUMP="$(which mysqldump)"
9 MyGZIP="$(which gzip)"
10
11 ## Verzeichnisname anhand des Datums ermit
teln
12 ## und ggf. anlegen
13 MyNOW="$(date +"%Y-%m-%d")"
14 MyDEST="./save"
15 MyMBD="$MyDEST/$MyNOW"
16 [ ! -d $MyMBD ] && mkdir -p $MyMBD || :
17
18 ## Los geht's
19 MyFILE="$MyMBD/$MyDB.$MyNOW.sql.gz"
20 $MySQLDUMP -u $MyUSER -p$MyPASS $MyOption
$MyDB | $MyGZIP -9 > $MyFILE
```

Das Skript ist so aufgebaut, dass Sie eigentlich nur die My*-Variablen anpassen müssen, um es verwenden zu können.

Wenn Sie tabellenweise sichern wollen:

```
1 mysqldump --opt --tables -uroot -p oshop a
rtikel > artikel.sql
```

24.1.2 Restore

Wenn Sie das Backup mit dem obigen Shell-Skript erstellt haben, wird Ihnen beim Restore die gesamte Datenbank wiederhergestellt. Dabei werden alle gerade vorhandenen Tabellenstände etc. gelöscht.

Man kann diesen Dump auf zwei Arten wieder einspielen. In beiden wird der MySQL/MariaDB Client verwendet:

```
1 bash-4.4# zcat oshop.18-06-2012.sql.gz | mysql -uroot -p oshop
```

Mit `zcat` wird die Datei entpackt und auf die Standardausgabe umgeleitet. Das Pipe-Symbol `|` macht daraus einen Eingabestrom für den MySQL Client.

Eine andere Methode liefert der Aufrufparameter `-e`. Bitte beachten Sie dabei, dass die Datei entpackt sein muss.

```
1 bash-4.4# mysql -e "source ./oshop.18-06-2012.sql" -uroot -p oshop
```

Falls Sie die Tabellendaten im CSV-Format gesichert haben, können Sie für die Wiederherstellung auch `mysqlimport` verwenden. Er arbeitet im Prinzip wie ein `LOAD DATA INFILE`. Dieser erwartet, dass der Name der Sicherungsdatei vor der Endung der Tabellenname ist.

```
1 bash-4.4# mysqlimport -uroot -p --local oshop artikel.csv
```

24.2 Benutzerrechte

24.2.1 Benutzerrechte und Privilegien

Das Anmelden auf dem Server erfolgt in diesem Buch mit dem Benutzer `root`. Dieser darf alles machen und ist ideal für die SQL-Einführung, da keine Einschränkungen bezüglich der Befehle vorliegen.

Das ist aber nicht der Normalfall. Datenbankbenutzer sollten nur das tun dürfen, was sie für die Anwendung brauchen, und auch nur auf den DBMS-Objekten, die zur Anwendung gehören. Soll doch die Anwendung, die den Online-Shop betreibt, nicht auch auf anderen Datenbanken wie einer Vereinsverwaltung oder einem CMS wie Typo3 Datenmanipulationen vornehmen dürfen.

Für unseren Online-Shop wollen wir einen Benutzer haben, der eine Art *Administrator* für die Datenbanken darstellen soll, und einen Benutzer für die alltäglichen Arbeiten: `osshop_root` und `osshop_user`.

Stellt sich zunächst die Frage, was für Rechte kann ich eigentlich vergeben oder entziehen? Bei Dateisystemen sind dies in der Regel die Rechte Lesen, Anlegen, Ändern und Löschen in verschiedenen Varianten. So ähnlich ist es auch in SQL, wie Sie der [Tabelle 24.1](#) entnehmen können.

Die Rechte bestimmen, welchen SQL-Befehl der Benutzer ausführen darf. Man könnte jetzt annehmen, dass es somit für jeden Befehl ein solches Recht gibt, aber manche Rechte sind summarisch wie beispielsweise `CREATE USER`, der für mehrere Befehle steht.

SQL:2023 und MySQL/MariaDB unterscheiden sich in vielen Punkten. Zum einen kennen MySQL und MariaDB erheblich mehr Rechte als SQL:2023, zum anderen werden einige Befehle im Detail anders interpretiert. Die wichtigsten

Unterschiede zwischen MySQL bzw. MariaDB und SQL:2023 sind:

- In MySQL/MariaDB wird ein Benutzer über seinen Namen *und* seinen Hostnamen identifiziert:
'abc'@'192.168.41.%', 'abc'@'localhost' und 'abc'@'%' sind drei verschiedene Benutzer. SQL:2023 nimmt nur den Benutzernamen, hier also 'abc'.
- SQL:2023 kennt keine Benutzerrechte auf Ebene einer oder aller Datenbanken (global).
- In SQL:2023 sind die Benutzerrechte hierarchisch bezüglich des Rechte vergebenden Benutzers angeordnet. Wird ein Benutzer gelöscht, werden auch alle Rechte entzogen, die dieser anderen per GRANT zugewiesen hat. In MySQL/MariaDB wird das nur bei DROP USER gemacht.
- Wenn Sie in SQL:2023 eine Tabelle löschen, werden alle damit verbundenen Rechte entzogen.
- Wenn Sie in SQL:2023 ein Recht entziehen, werden alle Rechte, die aufgrund dieses Rechts erteilt wurden, ebenfalls entzogen.
- In MySQL und MariaDB werden Rechte nur durch DROP USER und REVOKE entzogen. Alternativ kann man die Systemtabellen mit den Benutzerrechten direkt manipulieren.
- In MySQL und MariaDB kann man das Recht INSERT auf bestimmte Spalten einer Tabelle einschränken, sodass ein Benutzer eine neue Zeile anlegen kann, wenn er nur Angaben zu diesen Spalten macht. Die anderen Spalten werden mit ihren Standardwerten aufgefüllt. Ist der MySQL bzw. MariaDB Server im strict-Modus gestartet, wird der INSERT abgewiesen.

Tabelle 24.1 Von MySQL unterstützte Benutzerrechte
(S=* MySQL proprietär, S=° von MySQL nicht unterstützt)

S	Benutzerrecht	Erlaubt oder entzieht ...
	ALL [PRIVILEGES]	... alle Rechte außer GRANT OPTION.
	ALTER	... das Recht auf ALTER TABLE.
*	ALTER ROUTINE	... das Recht, Prozeduren zu verändern oder zu löschen.
	CREATE	... das Recht, Schemata oder Tabellen zu erstellen.
*	CREATE ROUTINE	... das Recht, Prozeduren zu erstellen.
*	CREATE TABLESPACE	... alle Rechte bezüglich <i>tablespaces</i> und Logdateigruppen.
*	CREATE TEMPORARY TABLES	... das Recht, temporäre Tabellen anzulegen.
*	CREATE USER	... das Recht auf {CREATE DROP RENAME} USER und REVOKE ALL PRIVILEGES.
*	CREATE VIEW	... das Recht, Ansichten anzulegen und zu verändern.
	DELETE	... das Recht auf DELETE.
	DROP	

		... das Recht, Schemata, Tabellen und Ansichten zu löschen.
*	EVENT	... alle Rechte bezüglich Ereignissen.
*	EXECUTE	... das Recht, Prozeduren auszuführen.
*	FILE	... das Recht, Dateioperationen auf dem Server auszuführen.
*	GRANT OPTION	... das Recht, Benutzerrechte zuzuweisen oder zu entziehen.
*	INDEX	... das Recht, Indizes anzulegen oder zu löschen.
	INSERT	... das Recht auf INSERT.
*	LOCK TABLES	... das Recht auf LOCK TABLES für Tabellen, auf denen man das SELECT-Recht besitzt.
*	PROCESS	... das Recht auf SHOW PROCESSLIST.
*	PROXY	... das Recht, einen Proxy-Benutzer einzurichten.
°	REFERENCES	... das Recht einen Fremdschlüssel-Constraint einzurichten.
*	RELOAD	... das Recht auf alle FLUSH-Anweisungen.
*	REPLICATION CLIENT	... das Recht, Master- oder Slave-Server zu erfragen.

*	REPLICATION SLAVE	... das Recht, die Replikationsdaten vom Master auszulesen.
*	SELECT	... das Recht auf SELECT.
*	SHOW DATABASES	... das Recht, alle Datenbanken anzuzeigen.
*	SHOW VIEW	... das Recht auf SHOW CREATE VIEW.
*	SHUTDOWN	... das Recht, <code>mysqladmin shutdown</code> auszuführen.
*	SUPER	... das Recht, administrative Anweisungen wie <code>CHANGE MASTER TO</code> , <code>KILL</code> , <code>PURGE BINARY LOGS</code> , <code>SET GLOBAL</code> und <code>mysqladmin debug</code> auszuführen.
*	TRIGGER	... das Recht auf alle Trigger-Anweisungen.
	UPDATE	... das Recht auf UPDATE.
	USAGE	Synonym für <i>keine Rechte</i> .

24.2.2 Benutzer anlegen und Rechte zuweisen



Die Quelltexte des Kapitels stehen in der Datei `mysql/listing20.sql` (siehe [Listing 28.22](#)).

24.2.2.1 CREATE USER bzw. CREATE ROLE



SQL:2023 MySQL/MariaDB

CREATE ROLE CREATE USER

*benutzername**benutzerspezifikation*
[,*benutzerspezifikation*]*

;

;

benutzerspezifikation:

benutzername

[IDENTIFIED BY [PASSWORD] '*passwort*'

|IDENTIFIED WITH *auth_plugin* [AS
'*auth_string*']]

Um diesen Befehl verwenden zu können, muss man das CREATE USER- oder das INSERT-Recht auf der Datenbank mysql haben, sodass für jeden Benutzer eine Zeile in der Tabelle mysql.user eingefügt wird.

Wie oben schon erwähnt, setzt sich in MySQL der benutzername aus zwei Teilen zusammen: *name@host*. Der *host* kann ein Hostname wie www.myhost.de oder eine IP-Adresse sein. Beide Varianten können durch den Platzhalter % Hostfamilien festlegen. Möchten Sie beispielsweise allen

Benutzern innerhalb eines Subnetzwerks den Zugriff erlauben, geben Sie '192.168.41.%'⁵ ein.

Dem Benutzernamen kann ein Passwort mitgegeben werden. Dieses wird klartextlich in der Tabelle `mysql.user` abgelegt. Um zu vermeiden, dass dieses einfach ausgespäht werden kann, können Sie die Funktion `PASSWORD()` verwenden:

```
1  mysql> SELECT PASSWORD('86(2,Lphg')\G
2  ***** 1. row *****
3  PASSWORD('86(2,Lphg'): *A31AAF3968A1272A06
   DAC82B3D33646B1BD43A79
```

Wenn Sie ein Plugin zur Authentifizierung verwenden, können Sie dieses anstelle des Passworts angeben.



Hinweis: Falls Sie den Benutzer über eine Konsole/Shell anlegen, ist das klartextlich lesbare Passwort in der History ablegt (z.B. `~/.mysql_history`).

Legen wir jetzt unsere beiden Benutzer an:

```
1  mysql> CREATE USER
2      -> 'oshop_root'@'%' IDENTIFIED BY 'root'
3      -> , 'oshop_user'@'%' IDENTIFIED BY 'user'
4      -> ;
5  mysql> SHOW GRANTS FOR 'oshop_root'@'%\G
6  ***** 1. row *****
7  Grants for oshop_root@%: GRANT USAGE ON *.
* TO 'oshop_root'@'%' IDENTIFIED BY PASSWORD '
```

*81F5E21E35407D884A6CD4A731AEBFB6AF209E1B'



Aufgabe 24.1: Hinter dem Wort GRANT steht das Recht USAGE. Was bedeutet das noch mal?

Der Platzhalter im Hostnamen ist so gewählt, dass der Benutzer sich von allen Rechnern aus anmelden kann. Ein Anmeldeversuch sorgt für Ernüchterung:

```
1  bash-4.4# mysql -uoshop_root -proot
2  ERROR 1045 (28000): Access denied for user
   'oshop_root'@'localhost' (using password : YES)
```

Ein Anmeldeversuch von einem entfernten Rechner aus wäre übrigens erfolgreich gewesen. Hier kommt eine nicht offensichtliche Eigenschaft von MySQL und MariaDB zum Tragen. Der Hostname `localhost` wird in der Zugriffskontrolle besonders behandelt. Will man dem Benutzer wirklich jeden Zugriffsweg erlauben, muss man tatsächlich noch einmal den gleichen Benutzer mit dem `localhost` anlegen:

```
1  mysql> CREATE USER
2      -> 'oshop_root'@'localhost' IDENTIFIED
   BY 'root'
3      -> , 'oshop_user'@'localhost' IDENTIFIED
   BY 'user'
4      -> ;
5
6  mysql> exit
7  Bye
8  bash-4.4# mysql -uoshop_root -proot
9  [...]
10 mysql>
```

Jetzt klappt's auch mit dem localhost.



Hinweis: Ich habe hier die Passwörter root und user gewählt. Davon ist in der Praxis dringend abzuraten. Frei nach den Empfehlungen des BSI (siehe [\[Inf23\]](#)) sollten folgende Regeln gelten:

- Keine Wörter oder Wortkombinationen
- Mindestens zehn Zeichen lang
- Mindestens ein Sonderzeichen
- Keine Trivialpasswörter wie aa11 . . bb22 etc.

Mit DROP USER kann ein Benutzer wieder gelöscht werden.



MySQL/MariaDB

```
DROP USER [IF EXISTS] benutzerspezifikation[,  
benutzerspezifikation]*  
;
```

24.2.2.2 Rechte vergeben mit GRANT

Wir haben jetzt zwar zwei Benutzer, aber sie dürfen nichts. Die Ausgabe von SHOW GRANTS FOR hatte genau dies mit USAGE mitgeteilt. Mit GRANT können einem Benutzer neue Rechte zugewiesen werden. Hier eine gekürzte Version des Befehls:



SQL:2023, MySQL/MariaDB


```
GRANT benutzerrecht[, benutzerrecht]*  
    ON [objekttyp] privilegtiefe  
    TO benutzerspezifikation[, benutzerspezifikation]*  
    [WITH GRANT OPTION]  
;
```

SQL:2023 kennt die Objekttypen TABLE, DOMAIN, COLLATION, CHARACTER SET und TRANSLATION. MySQL und MariaDB kennen TABLE, FUNCTION und PROCEDURE.

SQL:2023 hat keine *privilegtiefe*; hier steht dann der Objektname wie beispielsweise der Tabellename. MySQL und MariaDB haben die Privilegtiefen *Global* (* und *. *), *Datenbank* (datenbankname.*), *Tabelle* (datenbankname.tabellename, tabellename) und *Routine* (datenbankname.routinename).

Soll der Benutzer anderen Benutzern Rechte vergeben können – natürlich nur solche, die er selber innehat –, so wird WITH GRANT OPTION ans Ende gesetzt.

Für den Benutzer oshop_root sind folgende Angaben nötig:

- *benutzerrecht*: Unser oshop_root soll alles dürfen, dazu können wir der [Tabelle 24.1](#) das ALL PRIVILEGES entnehmen.
- *objekttyp*: Ist optional, daher stellt sich die Frage, brauchen wir die Angabe? Nö. Wir wollen ja, dass er auf allen Objekten der Datenbank alles darf.
- *benutzerspezifikation*: Diese kann genauso umfänglich sein wie bei CREATE TABLE. Wäre der Benutzer noch nicht vorhanden, würde er mit dieser Spezifikation angelegt

werden. Da wir ihn schon angelegt haben, reichen hier die Benutzernamen

'osshop_root'@'%' und 'osshop_root'@'localhost'.

- WITH GRANT OPTION: Da er Rechte vergeben können soll, muss diese Option verwendet werden.

Das alles führt zu folgender Anweisung:

```
1  mysql> GRANT ALL PRIVILEGES
2      -> ON oshop.*
3      -> TO 'osshop_root'@'%', 'osshop_root'@'
localhost'
4      -> WITH GRANT OPTION;
5
6  mysql> SHOW GRANTS FOR 'osshop_root'@'%' \G
7  ***** 1. row *****
*****
8  Grants for oshop_root@%: GRANT USAGE ON *.
* TO 'osshop_root'@'%' IDENTIFIED BY PASSWORD '
*81F5E21E35407D884A6CD4A731AEBFB6AF209E1B'
9  ***** 2. row *****
*****
10 Grants for oshop_root@%: GRANT ALL PRIVILE
GES ON 'osshop'.* TO 'osshop_root'@'%' WITH GRAN
T OPTION
```

Probieren wir doch mal aus, ob er auch genau das darf oder nicht darf, was wir wollten:

```
1  bash-4.4# mysql -uosshop_root -proot
2  mysql> CREATE DATABASE wurst;
3  ERROR 1044 (42000): Access denied for user
'osshop_root'@'localhost' to database 'wurst'
4
5  mysql> CREATE TABLE oshop.wurst (a INT);
6
7  mysql> DROP TABLE oshop.wurst;
```

In [Zeile 2](#) wird versucht, eine neue Datenbank anzulegen. Dies scheitert, da der Benutzer `oshop_root` keine entsprechenden globalen Rechte besitzt. Das Anlegen einer Tabelle in [Zeile 5](#) und das anschließende Löschen derselben hingegen klappt prima.

Legen wir jetzt den Benutzer `ohop_user` an:

- *benutzerrecht*: Unser `oshop_user` soll die normalen Arbeiten auf den Tabellen durchführen dürfen. Dazu gehören das Einfügen neuer Daten mit `INSERT`, das Verändern und Löschen bestehender Daten mit `UPDATE` bzw. `DELETE`. Außerdem muss er Auswertungen mit `SELECT` durchführen dürfen.
- *objekttyp*: Ist optional, könnte aber `TABLE` lauten.
- *benutzerspezifikation*: Hier gilt das Gleiche wie beim Benutzer `oshop_root`.
- `WITH GRANT OPTION`: Der `oshop_user` soll keine weiteren Benutzer anlegen können, daher fehlt diese Angabe.

Zusammengefasst kommt folgende Anweisung heraus:

```
1  mysql> GRANT INSERT, UPDATE, DELETE, SELEC
T
2      -> ON TABLE oshop.*
3      -> TO 'oshop_user'@'%', 'oshop_user'@'
localhost'
4      -> ;
5
6  mysql> SHOW GRANTS FOR 'oshop_user'@'%' \G
7  ***** 1. row *****
*****
8  Grants for oshop_user@%: GRANT USAGE ON *.
* TO 'oshop_user'@'%' IDENTIFIED BY PASSWORD '
*D5D9F81F5542DE067FFF5FF7A4CA4BDD322C578F'
9  ***** 2. row *****
```

```
10 Grants for oshop_user@%: GRANT SELECT, INSERT, UPDATE, DELETE ON 'oshop'.* TO 'oshop_user'@'%'
```



Aufgabe 24.2: Testen Sie die Privilegien des neuen Benutzers: Darf er alles, was wir wollen, und werden ihm unerlaubte Aktionen verweigert?

24.2.2.3 Rechte entziehen mit REVOKE

Wenn Sie einem Benutzer ein oder mehrere Rechte entziehen wollen, brauchen Sie REVOKE.



SQL:2023, MySQL/MariaDB

```
REVOKE benutzerrecht[, benutzerrecht]*  
ON [objekttyp] privilegtiefe  
FROM benutzername[, benutzername]*  
[WITH GRANT OPTION]  
;
```

oder



SQL:2023, MySQL/MariaDB

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM  
FROM benutzername[, benutzername]*
```

;

Es ist letztlich eine Frage, was bequemer ist: eine Positivliste oder eine Negativliste. Gebe ich an, was dem Benutzer erlaubt ist, oder eher, was ihm verboten ist?

In unserem Fall soll der Benutzer `oshop_user` nicht löschen dürfen, da wir ja eigentlich das Löschkennzeichen verwenden wollen (siehe [Abschnitt 2.3.1](#)).

```
1  mysql> REVOKE DELETE
2          -> ON TABLE oshop.*
3          -> FROM 'oshop_user'@'%', 'oshop_user'
@'localhost'
4          -> ;
```



Aufgabe 24.3: Erlauben Sie dem Benutzer, in den Hilfstabellen einer $n:m$ -Verknüpfung zu löschen.

24.3 MySQL und MariaDB Engines

Eine Liste (Auszug) von Engines für MySQL und MariaDB. MariaDB-Engines sind kursiv, und der Default ist jeweils fett gedruckt.

- **ARCHIVE:** Wie der Name schon sagt, dient diese Engine dazu, Daten effizient zur Aufbewahrung abzuspeichern. Auswertungen und Datenänderungen werden nur sehr teuer unterstützt. Besonders bei großen Datenmengen ist diese Engine sehr vorteilhaft.

- *Aria*: Diese Engine ist eine Weiterentwicklung von MyISAM unter MariaDB.
- *Cassandra*: Eine NoSQL Engine.
- **BLACKHOLE**: Diese Engine ist ein *WRITE ONLY*-Baustein. Sie können zwar Daten in die Engine schreiben, aber nicht mehr auslesen :-(. Tatsächlich verwendet man diese Engine nur zu (Performance-)Testzwecken.
- **CSV**: Die Daten werden als CSV-Datei verwaltet. Man kann also direkt auf CSV-Dateien `SELECT` etc. ausführen. Man erspart sich damit lästiges Importieren oder Exportieren von CSV-Dateien. Der Nachteil ist eine geringe Performance und das Fehlen wichtiger Features.
- *CONNECT*: Zugriff auf Textdateien und Tabellen anderer RDBMS.
- **EXAMPLE**: Keine eigentliche Engine, sondern eine Einstiegshilfe für Programmierer neuer Engines.
- **FEDERATED**: Mithilfe dieser Engine greifen Sie auf Tabellen anderer MySQL Server zu.
- *FederatedX*: MariaDB-spezifische Erweiterung von **FEDERATED**.
- **InnoDB**: Das Arbeitspferd für die meisten nicht trivialen Anwendungen. Sie liefert fast alles, was man für die übliche Verarbeitung braucht. Wird bei MySQL im `CREATE TABLE` keine andere Engine angegeben, wird diese verwendet.
- **MEMORY**: Die Daten werden im Arbeitsspeicher abgelegt. Dies macht die Verarbeitung der Daten sehr schnell, was besonders für temporäre Tabellen von Vorteil sein kann.
Wichtig: Da die Daten nur im Arbeitsspeicher vorliegen, stehen sie beispielsweise nach einem Serverneustart

nicht mehr zur Verfügung!

- **MERGE:** Falls Sie Ihre Daten aus irgendeinem Grund nicht in einer Tabelle, sondern strukturgleich auf mehrere Tabellen verteilt haben, können Sie mit einer MERGE-Tabelle so tun, als arbeiteten Sie nur mit einer.
 - **MyISAM:** Zwar ist diese Engine nicht transaktionsfähig und auch andere Features fehlen, aber sie eignet sich sehr gut für die Auswertung großer, nicht zu verschachtelter Datenmengen.
 - **OQGRAPH:** Mit dieser Engine können baumartige Strukturen und komplexe Graphen abgebildet werden.
 - **XtraDB:** Eine MariaDB-spezifische Erweiterung/Verbesserung der InnoDB. Wird bei MariaDB im `CREATE TABLE` keine andere Engine angegeben, wird diese verwendet. **Wichtig:** Geben Sie bei MariaDB als Engine InnoDB an. Es wird automatisch eine XtraDB angelegt, obwohl immer InnoDB angezeigt wird.
-

¹ Sie müssen dazu Leserechte auf dem Verzeichnis besitzen.

² Beachten Sie, dass `mysqldump` im Suchpfad liegen muss. Gegebenenfalls müssen Sie im `bin`-Verzeichnis der Installation nachschauen!

³ Vergessen Sie nicht, für die Skripte das executable-Flag zu setzen.

⁴ Quelle: <http://serversupportforum.de/forum/sql/40542-mysql-dump-ueber-cronjob-2-von-3-laufen.html>

⁵ Selbstverständlich sind auch IPV6-Adressen möglich.

25 SQL-Injection

25.1 Das Problem



Auf YouTube habe ich die Angriffe und Abwehrmaßnahmen demonstriert:

<https://www.youtube.com/playlist?list=PL4aVL6eAcvHnfoZ0aVKzQFVA0AZK7bn2f>

Bevor ich die Angriffsmethoden und mögliche Abwehrmaßnahmen vorstelle, möchte ich Sie ein wenig für dieses Thema motivieren. In [[Inc20](#)] werden zwei markante Beispiele angegeben: ein Einbruch in die Datenbank des *Wall Street Journals* ([[Kir14](#)]) und der Diebstahl von rund 100 Mio \$ in Bitcoin aus E-Commerce-Systemen ([[Bam20](#)]). Nun könnten Sie denken, wann programmiere ich schon mal für einen berühmten Verlag oder für ein weltweites Bezahlungssystem?

Aber das ist der falsche Ansatz. Jede Art von illegaler Datenausspähung, -manipulation oder -löschung ist eine Katastrophe. Möchten Sie, dass Ihre Kreditkartendaten, Ihre Laborberichte oder die Vorstandsprotokolle leicht abgegriffen werden können? Und schon sind wir in Bereichen, wo es schon viel wahrscheinlicher wird, dass Sie an der Entwicklung

teilgenommen haben. Nehmen Sie das Problem also nicht auf die leichte Schulter.

In [Vij19] analysiert der Autor den Akamai-Report aus 2019 und zitiert: *The growth of SQLi as an attack vector over the last two years should concern website owners, [...] While every application attack vector is stable or growing, none are growing as quickly as SQLi.*¹ Darin kommen die Autoren zu dem Ergebnis, dass 2019 65,1% aller Webangriffe SQLi-Angriffe sind. Für 2021 stehen Injection-Angriffe auf Platz 3 der OWASP-Liste².

So, alle an Bord? Dann fangen wir mit was Spannendem an, einer Definition:



Definition 75: SQL-Injection (SQLi)

Wenn über ein Interface SQL-Quelltext an einen Interpreter zur Ausführung gelangt, obwohl dies nicht vorgesehen war, so spricht man von einer *SQL-Injection* (SQLi).

25.2 Beispiel: Suchmaske



Die Quelltexte des Kapitels stehen in den Dateien `mysql/listing21.sql` (siehe [Listing 28.23](#)), `mysql/index01.php` und `mysql/index02.php` (siehe [Listing 28.24](#) und [Listing 28.25](#)).

Ich habe ein kleines Beispiel zusammengestellt, welches Sie

Ich habe ein kleines Beispiel zusammengestellt, welches Sie in dashtdocs-Verzeichnis Ihrer XAMPP-Umgebung kopieren können; am besten in ein eigenes Unterverzeichnis – beispielsweise mit dem Namen buch. Starten Sie nun MySQL bzw. MariaBD und den Apache Web Server über das XAMPP Control Panel. Führen Sie `listing21.sql` aus – dadurch wird die Spieldatenbank `injection` mit den Tabellen `artikel` und `account` aufgebaut – und rufen Sie die URL <http://localhost/buch/index01.php> auf. Das Ergebnis sollte ungefähr so aussehen:

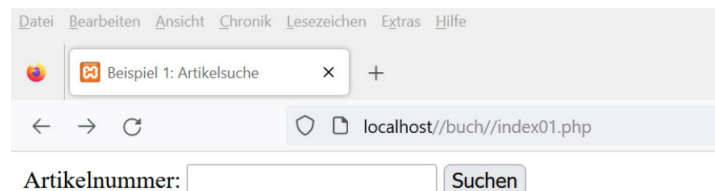


Bild 25.1 SQL-Injection: Leere Suchmaske

Werfen wir einen Blick in den Quelltext, damit Sie sehen können, dass ich nicht Hintertürchen eingebaut habe, sondern alles gute, alte PHP-Hausmannskost ist. Der Quelltext ist in zwei Bereiche aufgeteilt, ein reines HTML-Formular und einen PHP-Teil mit Datenbankabfrage und Ergebnisausgabe. Ich fange hinten mit dem HTML-Formular an:³

```
46 <!DOCTYPE HTML>
47 <html>
48 <head>
49 <title>Beispiel 1: Artikelsuche</title>
50 <meta name="author" content="Ralf Adams">
51 </head>
52 <body text="#000000" bgcolor="#FFFFFF">
53 <form action="http://localhost//buch//index01.php">
54     <label for="artikel_id">Artikelnummer: <
    /label>
```

```

55     <input type="text" id="artikel_id" name=
"artikel_id"> <!-- Eingabefeld -->
56     <button name="search" value="1">Suchen</
button> <!-- Submitbutton -->
57 </form>
58 </body>
59 </html>

```

Sie erkennen ein hübsches kleines Eingabeformular zur Suche von Artikelstammdaten. In [Zeile 55](#) wird ein Eingabefeld mit dem Namen `artikel_id` deklariert und in [Zeile 56](#) eine Schaltfläche, die den Wert 1 übergibt, falls sie angeklickt wurde. Gebe ich nun beispielsweise die Artikelnummer 3001 ein, werden per GET-Methode⁴ die Daten übermittelt. Jetzt kommt der obere Teil des Quelltextes ins Spiel, und den schauen wir uns etwas genauer an:

```

1  <?php
2  if (ISSET($_GET['search']) && $_GET['search'] == '1') { // Schaltfläche gedrückt?

```

In der ersten Zeile wird das PHP-Tag geöffnet. Wir werden später sehen, dass dies wieder geschlossen wird, also aus Sicht des HTML-Dokuments somit nur ein Tag unter vielen ist. Ab jetzt kann der PHP-Quelltext programmiert werden. Als Erstes werden in [Zeile 2](#) zwei Dinge überprüft. Einmal, ob die Variable `search` überhaupt übergeben wurde, und wenn ja, ob zum Zweiten diese den Wert 1 hat. Inhaltlich bedeutet dies, dass jemand den Button *Absenden* angeklickt hat. Nur in diesem Fall wird eine Verbindung zum Datenbankserver aufgebaut und die Verarbeitung durchgeführt.

```

3      $server = "localhost";
4      $user = "root";
5      $password = "weinschlauchX20!";
6      $dbname = "injection";

```

Um eine Verbindung zum Server aufzubauen, werden verschiedene Angaben benötigt: der *ConnectionString* bzw. die *Verbindungsparameter*. Üblicherweise werden diese in Variablen abgelegt und in eine externe Datei ausgelagert, was ich hier der Übersichtlichkeit halber unterlassen habe. Die Bedeutung der einzelnen Angaben muss ich hoffentlich nicht erklären ;-)

```
8      $conn = new mysqli($server, $user, $password, $dbname); // Verbindung aufbauen
9      if ($conn->connect_error) { // Fehler aufgetreten?
10         die("Connection failed: ".$conn->connect_error); // Fehlerausgabe
11     }
```

In [Zeile 8](#) werden die Verbindungsparameter dazu genutzt, den Server anzufunkeln, sich zu authentifizieren und die richtige Datenbank auszuwählen.⁵ In [Zeile 9](#) wird überprüft, ob ein Fehler aufgetreten ist, denn beim Verbindungsaufbau kann eine Menge schiefgehen (Timeout, falsches Passwort, Zahlendreher in der IP-Adresse, Server unten usw.), und deshalb sollte immer vor der Weiterverarbeitung überprüft werden, ob der Aufbau überhaupt geklappt hat. Falls ja, kann ich nun über die Variable `$conn` mit der Verbindung arbeiten.

```
13     if (ISSET($_GET['artikel_id'])) { // Variable vorhanden?
14         $artikel_id = $_GET['artikel_id'];
15         $sql = "SELECT * FROM artikel WHERE artikel_id=$artikel_id"; // SQL-Bau
16         echo "<p>$sql</p>"; // NUR FÜR UNS!
17     }
18     if (!$result = $conn->query($sql)) { // Query -> Server
19         die("Execution failed: ".$conn->error)
```

```
;
20 }
```

Zunächst wird wieder die Existenz einer Variablen überprüft, bevor sie verwendet wird ([Zeile 13](#)). Danach erfolgt in [Zeile 15](#) der Zusammenbau der Abfrage. Die `artikel_id` wird in der WHERE-Klausel passend platziert; wir kommen später auf diese wichtige Quelltextzeile zurück. Die Ausgabe des SELECT in [Zeile 16](#) ist auch nur für uns zu Demo-Zwecken!

In [Zeile 18](#) wird der SELECT zum Server gesendet und eine Referenz auf die Ergebnismenge zurückgeliefert. Scheitert dies, lassen wir uns wieder verbotenerweise die Fehlermeldung auf den Browser ausgeben.

```
22     if ($result->num_rows > 0) {
23         echo "<table border=\"1\">";
24         echo "<tr>";
25         echo "<th>Artikelnr</th>";
26         echo "<th>Bezeichnung</th>";
27         echo "<th>Einzelpreis</th>";
28         echo "<th>Währung</th><th>LKZ</th>";
29         echo "</tr>";
30         while($row = $result->fetch_assoc()) {
31             echo "<tr>";
32             echo "<td>".$row['artikel_id']."</td>";
33             echo "<td>".$row['bezeichnung']."</td>";
34             echo "<td>".$row['einzelpreis']."</td>";
35             echo "<td>".$row['waehrung']."</td>";
36             echo "<td>".$row['deleted']."</td>";
37             echo "</tr>";
38         }
39         echo "</table>";
40     }
```

```

41     echo "<p>".$result->num_rows." results</p>";
42     $conn->close(); // Verbindung schließen
43 }
44 else
45 ?>

```

Der Rest ist die Ausgabe der Artikelstammdaten in einer HTML-Tabelle und das Schließen der Verbindung zum Server ([Zeile 42](#)). In der letzten Zeile wird das PHP-Tag wieder geschlossen.

Jetzt wird es Zeit für etwas mehr Action: Was passiert, wenn ich 3001 eingebe?

SELECT * FROM artikel WHERE artikel_id=3001

Artikelnr	Bezeichnung	Einzelpreis	Währung	LKZ
3001	Papier (100)	2.320000	EUR	0

1 results

Artikelnummer:

Bild 25.2 SQL-Injection: Ergebnis bei 3001

Der Anwender gibt die gültige Artikelnummer 3001 ein. Diese wird an das PHP-Skript per GET übergeben. Anschließend wird in [Zeile 15](#) die Artikelnummer passend in die WHERE-Klausel eingebaut. Danach erfolgt das Absenden des Querys und die Ausgabe der einen Ergebniszeile als HTML-Tabelle. Ich erkenne, dass der SELECT genauso gebaut wurde, wie ich ihn brauchte und wollte. Dementsprechend enthält die Tabelle auch nur die Daten des gesuchten Artikels.

Und jetzt kommt es: Geben wir "0 OR TRUE" als Artikelnummer in das Suchfeld ein (siehe [Bild 25.3](#)).

```
SELECT * FROM artikel WHERE artikel_id=0 OR TRUE
```

Artikelnr	Bezeichnung	Einzelpreis	Währung	LKZ
3001	Papier (100)	2.320000	EUR	0
3005	Tinte (gold)	56.260000	EUR	0
3006	Tinte (rot)	6.260000	EUR	0
3007	Tinte (blau)	4.170000	EUR	0
3010	Feder	5.050000	EUR	0
7856	Silberzwiebel	0.520000	EUR	0
7863	Tulpenzwiebel	3.420000	EUR	0
9010	Schaufel	15.100000	USD	0
9015	Spaten	20.100000	EUR	0
9016	Schlauch	0.000000	EUR	0

10 results

Artikelnummer:

Bild 25.3 SQL-Injection: Ergebnis bei "0 OR TRUE"

Schauen Sie sich den SELECT oben im Bild an. Exakt dort, wo eben noch das harmlose 3001 stand, steht nun ein Fragment, das die erste Bedingung mit einem uninteressanten Vergleichswert abschließt und eine zweite Bedingung mit OR anfügt. Diese zweite Bedingung liefert immer TRUE, was in Zusammenhang mit der OR-Verknüpfung dazu führt, dass die Gesamtbedingung der WHERE-Klausel immer TRUE ergibt. Das Ganze würde auch bei einer äußerst schlecht programmierten Anmeldemaske funktionieren :-)

Betrachten Sie den Vorgang noch mal etwas genauer:

```
$artikel_id = $_GET['artikel_id'];  
0 OR TRUE  
$sql = "SELECT * FROM artikel WHERE artikel_id  
=$artikel_id";  
SELECT *  
FROM artikel  
WHERE artikel_id= 0 OR TRUE
```

Schon jetzt sollten wir alarmiert sein. Mit einem Schlag haben

Schon jetzt sollten wir alarmiert sein. Mit einem Schlag haben Sie allen Mitbewerbern auf dem Markt die komplette Preisliste offenbart. Es geht aber noch böser: Tragen wir mal "3001 UNION SELECT kunde_id, name, passwort, status, admin FROM account" ein (siehe [Bild 25.4](#)). Die Idee dahinter ist, dass die Ausgabe mit dem Inhalt jeder beliebigen Tabelle per UNION erweitert werden kann. Man muss nur die Anzahl der Spalten beachten. Hier nochmals im Detail der zusammengebaute SELECT:

```
$artikel_id = $_GET['artikel_id'];
3001 UNION SELECT kunde_id, name, passwort, st
atus, admin FROM account
$sql = "SELECT * FROM artikel WHERE artikel_id
=$artikel_id";
SELECT *
FROM artikel
WHERE artikel_id= 3001
UNION
SELECT *
kunde_id, name, passwort, status, admin
FROM account
```

Artikelnr	Bezeichnung	Einzelpreis	Währung	LKZ
3001	Papier (100)	2.320000	EUR	0
1	admin	45c6f0923e6a87454183b56c0935d253	aktiv	1
2	frodo	92cba3dc4cd1db0628fadeebfaa16ac8	aktiv	0
3	bilbo	dd509890e7bcece80b1a9c20ec4affb5	aktiv	0
5	elle	ee74f41a33032548bf6d315d7db3f1ee	aktiv	0

Bild 25.4 SQL-Injection: Ergebnis bei UNION mit der Tabelle account

Aber warum sollten wir uns nur auf Tabellen der Datenbank beschränken? Lassen sich nicht auch Systemtabellen ausspähen? Klar, geben Sie den markierten Bereich – ohne Umbrüche – in das Suchfeld ein, und wir erhalten die Ausgabe wie in [Bild 25.5](#):


```
SELECT *
FROM artikel
WHERE artikel_id= 3001
UNION
SELECT host, user, authentication_string, 0, 0
FROM mysql.user
WHERE
user LIKE '%oshop%' OR user LIKE '%root%'
```

Artikelnr	Bezeichnung	Einzelpreis	Währung	LKZ
3001	Papier (100)	2.320000	EUR	0
%	oshop_root	\$A\$005\$Y9x.@f1,]_qTIn!3#tmhIP8GU5X6sUpgeb[REDACTED]qjIeSdG7D	0	0
%	oshop_user	\$A\$005\$[REDACTED]	0	0
localhost	oshop_root	\$A\$005\$*QCh0Fsk%3&-Mcl/Av2O9qAt46x4fPUW[REDACTED]VnS9o8l5VmruD	0	0
localhost	oshop_user	\$A\$005\$H9dbYQ;5r Z ZS8e(RLC3k3b/k4jwVZAraZ[REDACTED]9GFy1VBk3WC	0	0
localhost	root	\$A\$005\$[REDACTED]ADEuy]SUmVAnMZIFqqFdOTVgIagY[REDACTED]KYfuSncL6bWLCIB	0	0

Bild 25.5 SQL-Injection: Ergebnis bei UNION mit der Tabelle `mysql.user`

Hallo! Was geht denn hier ab? Alle interessanten Benutzernamen werden mir mit den passenden Hash-Werten auf dem Silbertablett präsentiert. Da packen wir doch erst mal unsere Regenbogentabellen⁶ aus, und hoffen auf schwache Passwörter oder Zufallstreffer⁷.

Sind wir damit am Ende angekommen? Leider immer noch nicht. In manchen Fällen verlangen Benutzeraktionen, dass nicht ein SQL-Befehl, sondern mehrere hintereinander ausgeführt werden – die sogenannte Batchverarbeitung. In unserem Beispiel sehe ich spontan dafür keinen Bedarf, aber denken Sie beispielsweise daran, dass zu jedem Bestellvorgang auch immer sofort ein passender Rechnungsvorgang angelegt wird. Wenn dem so ist, wird nicht der Befehl `mysqli->query()`, sondern `mysqli->multi_query()` verwendet.

Rufen Sie also <http://localhost/buch/index02.php> auf und betrachten die [Zeile 18](#):

```
18    if (!$conn->multi_query($sql)) { // Mult
i-Query -> Server
19        die("Execution failed: ".$conn->error)
;
20    }
21    $result = $conn->store_result();
```

Die Variable `$sql` kann nun nicht nur einen SQL-Befehl enthalten, sondern mehrere durch Semikolon getrennte. Das können wir gleich mal für böse Streiche nutzen:

```
SELECT *
FROM artikel
WHERE artikel_id= 3001 ;
UPDATE artikel SET
SET einzelpreis=0.01
WHERE artikel_id=3001
;
```

Tja, da ist der Artikel aber schön billig geworden. Alternativ hätte ich mit `INSERT` einen neuen Artikel oder einen neuen SQL-User einfügen können. Solche Angriffe sind die schlimmsten. Sie sind still, bleiben lange unbemerkt und können erheblichen Schaden anrichten.

Okay, wir haben gesehen, dass man Schaden anrichten kann. Jetzt wird es Zeit, das Ganze zu systematisieren und zusammenzufassen, wobei ich hier nicht alle Szenarien durch Demos untermale, sondern manche nur aus der Erinnerung oder der Literatur reflektiere:⁸

- **In band injection:** Die Phrase *in band* umschreibt, dass für den Angriff selbst und das Abschöpfen des Angriffserfolges der gleiche Ein-/Ausgabekanal (Band)

verwendet wird. Unsere oben durchgeführten Angriffe waren beispielsweise In-Band-Angriffe. Der Angriff und das Abschöpfen der Ergebnisse erfolgte über die gleiche Weboberfläche. In-Band-Angriffe lassen sich in zwei Kategorien aufteilen.

- **Error based attack:** Der Angreifer versucht explizit, Fehlermeldungen zu provozieren. Werden diese – so wie ich es üblicherweise oben vorgemacht habe – für den Angreifer sichtbar ausgegeben, kann er die Fehlermeldung interpretieren. Er enthält Informationen über die eingesetzten SQL Server, PHP-Versionen, Datenbank-, Tabellen- oder Spaltennamen oder andere scheinbar unwichtige statistische Infos. Kennt der Angreifer beispielsweise den SQL Server und idealerweise auch die Versionsnummer des Servers, so könnte er in Exploit-Tabellen nachschauen, ob es vorgefertigte Angriffsszenarien für den Server gibt. Je älter die Serverversion ist, desto bedrohlicher wird die Anzahl der Angriffsmöglichkeiten. Mithilfe der Datenbank-, Tabellen- oder Spaltennamen können gezieltere SQL-Befehle für beispielsweise den Union-based-Angriff gebastelt werden.
- **Union based attack:** Der Name sagt schon alles über die Methode des Angriffs aus. Der SQL-Befehl wird geschickt um ein UNION erweitert. Kennt man beispielsweise durch einen Error-based-Angriff Tabellen- und Spaltennamen, lassen sich unerwünschte Auswertungen formulieren. Oft werden die Tabellen- und Spaltennamen aber auch einfach nur geraten. Es ist recht wahrscheinlich, dass es eine Tabelle mit einem Namen wie user oder account gibt. Unsere obigen Angriffe sind zum

größten Teil Union-based-Angriffe.

- **Inferential⁹ injection (blind injection):** Bei diesem Angriff werden erst mal keine Informationen über die interne Struktur der Datenbank verwendet. Vielmehr probiert man einfach verschiedene Techniken aus und beobachtet das Antwortverhalten. Auch hier werden zwei Kategorien unterschieden:

- **Boolean attack:** So wie wir es oben schon durchgeführt haben, wird die SQL-Abfrage um boolesche Ausdrücke wie `1=1` oder `1=2` erweitert. Reagiert das System unterschiedlich, können weitere Angriffe geplant werden. Sie fragen sich, wie ich das ausnutzen kann? Zum Beispiel mit folgender Abfrage:

```
3001 AND 'a'=SUBSTRING(DATABASE(),1,1) -- 1
```

So bekomme ich heraus, ob der erste Buchstabe des Datenbanknamens das kleine a ist. Die Kommentarzeichen sollen nur für einen syntaktisch korrekten Abschluss sorgen. Dann macht man weiter mit dem kleinen b usw., bis das System mit einer TRUE-Reaktion antwortet. Und dann, auf zum zweiten Buchstaben des Datenbanknamens. Übrigens sollten Sie mit der gleichen Technik vorab die Länge des Datenbanknamens ermitteln, damit Sie wissen, wann Sie aufhören können.

- **Time based attack:** Der Name leitet sich davon ab, dass man in den SQL-Befehl Funktionen wie `SLEEP()` injiziert. Probieren Sie das am obigen Beispiel mal aus: `3001-SLEEP(1)`. Sie werden feststellen, dass der Server frühestens nach einer Minute antwortet. Was ist der Benefit? Sie wissen nun, dass Sie die SQL-Abfrage um SQL-Funktionen erweitern können. Auch könnte die Verfügbarkeit bestimmter Funktionen auf

den SQL Server bzw. seine Versionsnummer hinweisen. Darüber hinaus könnten Sie mit Funktionen wie `BENCHMARK()` schlicht tote Last auf dem Server erzeugen.

- **Out of band injection:** Hierbei wird der Angriff über einen Kanal gestartet und das Ergebnis über einen ganz anderen abgeschöpft. Dabei spielen oft Funktionen wie `LOAD_FILE()` mit Features wie `INTO OUTFILE` zusammen. Bitte beachten Sie, dass bei einem schlecht konfigurierten Server, der weder die Datenquelle für die Funktion `LOAD_FILE()` noch die Datensenke für `INTO OUTFILE` eng einschränkt, jede erreichbare Datei ausgelesen und jede Ziel-IP-Adresse gewählt werden kann.

Sie meinen nun, dass der Aufwand doch erheblich ist und dieser sich nur bei wirklich wichtigen Zielen lohnt. Oder anders formuliert: Meine kleine Anwendung ist doch viel zu unbedeutend für diesen Aufwand. Da muss ich Sie enttäuschen. Viele Angriffsszenarien können hochgradig automatisiert durchgeführt werden. Programme wie *sqlmap* (<https://sqlmap.org/>) sind schon mit wenig technischem Vorwissen schnell einsetzbar. Sie sollten sich mit solchen Tools vertraut machen, da Sie diese hervorragend für Penetrationstests verwenden können.

25.3 Gegenmaßnahmen



Bild 25.6 Quelle <https://xkcd.com/327/>

25.3.1 Never Trust an Unknown Input



Hinweis: Unterziehen Sie Datenquellen immer einer Überarbeitung:

1. Maskieren/escapen Sie SQL-spezifische Sonderzeichen. Verwenden Sie dazu nach Möglichkeit schon vorhandene Bibliotheksfunktionen.
2. Nutzen Sie reguläre Ausdrücke, um nach SQL-Schlüsselwörtern zu suchen. Verweigern Sie anschließend die Weiterverarbeitung.
3. Nutzen Sie reguläre Ausdrücke, um die Eingabe zu plausibilisieren.

Never trust an unknown input! habe ich vor über 25 Jahren bei einem Kurs zur Sicherheitsprogrammierung kennengelernt. Damals ging es zwar um Staatsfeind Nr. 1, den Buffer Overflow, aber der Merksatz trifft auch in unserem Fall zu. Validieren Sie die Benutzereingaben. Bei SQLi könnten Sie beispielsweise über reguläre Ausdrücke nach SQL-Schlüsselwörtern oder

SQL-Sonderzeichen suchen. Auch bieten viele Serversprachen wie PHP Funktionen wie `mysqli->real_escape_string()` an, die SQL-Sonderzeichen maskieren oder escapen. Damit wären Ergänzungen wie `1=1` nicht mehr möglich. Unser Aufruf sähe dann wie folgt aus:

```
1  $_artikel_id = $conn->real_escape_string($
artikel_id);
2  $sql = "SELECT * FROM artikel WHERE artike
l_id=".$_artikel_id;
```

Unsere Eingabe – also die Artikelnummer – sollte eine einfache ganze Zahl sein. Das können Sie leicht mit einem regulären Ausdruck überprüfen:

```
1  $muster = /^[0-9]*$/;
2  if(preg_match($muster, $sql)) {
3  [...]
4  }
```

Im Netz (z.B. <https://regexlib.com>) finden Sie schon viele vorgefertigte reguläre Ausdrücke zu Postleitzahlen, Telefonnummern, E-Mail-Adressen, IBAN etc. Gerade weil reguläre Ausdrücke nicht so leicht zu verstehen sind, sollten Sie hier ruhig in erster Priorität auf Lösungen zurückgreifen, die schon von Tausenden Benutzern verwendet und damit auch getestet wurden, anstatt eigene zu basteln.

25.3.2 Trennung von Anweisungen und Daten



Hinweis: Trennen Sie Anweisungen von Daten entweder durch das Verwenden von Prozeduren/Funktionen oder Prepared Statements in Verbindung mit Parameter-bindung.

25.3.2.1 Verwenden Sie Prozeduren oder Funktionen

Bei Prozeduren (siehe [Kapitel 20](#)) oder Funktionen (siehe [Kapitel 21](#)) werden die Anweisungen auf dem SQL Server abgespeichert. Die Prozeduren oder Funktionen werden nach dem Erstellen mit CREATE vorkompiliert, liegen also schon im Binärformat (Objectcode) vor. Die Eingaben vom Anwender werden dann als Übergabeparameter in die Abfrage integriert. Die Parameter werden aber nicht mehr kompiliert, sodass SQL-Fragmente in den Parametern uninterpretiert bleiben.¹⁰

25.3.2.2 Verwenden Sie Prepared Statements

Die am häufigsten verwendete Technik ist das Prepared Statement. Dabei wird die eigentliche SQL-Anweisung schon in Objectcode überführt, und es werden Platzhalter für Parameter gelassen. Kurz vor der Ausführung werden die Platzhalter mit Daten gefüllt. Da nun aber keine Überführung mehr in Objectcode erfolgt, bleiben wie bei den Prozeduren SQL-Fragmente in den Parametern uninterpretiert. So könnte beispielsweise der Quelltext von `index01.php` umgeschrieben werden:

```
1  if (ISSET($_GET['artikel_id'])) {
2      $artikel_id = $_GET['artikel_id'];
3      $sql = $conn->prepare("SELECT * FROM art
ikel WHERE artikel_id=?");
4      $sql->bind_param("i", $artikel_id);
5  }
6  $sql->execute();
```



```

7  if (!$result = $sql->get_result()) {
8      die("Execution failed: ".$conn->error);
9  }

```

Die Methode `prepare()` wandelt den String in einen Objectcode um und das Fragezeichen am Ende dient als Platzhalter für Daten. Natürlich können auch mehrere Fragezeichen in WHERE-Klauseln verwendet werden, wenn Sie dies benötigen. Bitte beachten Sie, dass Sie nur Daten, aber keine Teile des SQL-Befehls selbst per Platzhalter dynamisch bestimmen können. So können weder der Befehl (hier `SELECT`), der Tabellename (hier `artikel`) noch der Spaltenname (hier `artikel_id`) parametrisiert werden.

In der nächsten Zeile wird der Parameter mit der Variablen verknüpft. Dabei steht das kleine `i` hier als Formatzeichen; es gibt an, dass dieser Parameter eine ganze Zahl ist. Ein kleines `s` stünde für ein String – also `CHAR`, `VARCHAR` oder `TEXT`. Hier ein Beispiel für eine Abfrage mit zwei Parametern:

```

1  if (ISSET($_GET['artikel_id']) && ISSET($_GET['bezeichnung'])) {
2      $artikel_id = $_GET['artikel_id'];
3      $bezeichnung = $_GET['bezeichnung'];
4      $sql = $conn->prepare("SELECT * FROM artikel WHERE artikel_id=? OR bezeichnung LIKE '%?%'");
5      $sql->bind_param("is", $artikel_id, $bezeichnung);
6  }
7  $sql->execute();
8  if (!$result = $sql->get_result()) {
9      die("Execution failed: ".$conn->error);
10 }

```

Mit `execute()` und `get_result()` erfolgt nun die weitere Verarbeitung der Ergebnismenge wie gewohnt.

25.3.3 Ich darf nur, was ich soll



Hinweis: Erstellen Sie SQL-User, die folgende Eigenschaften haben:

1. Der SQL-User kann nur auf solche Datenbankobjekte (Tabellen, Ansichten, Prozeduren etc.) zugreifen, die er für diese eine Aufgabe benötigt.
2. Der SQL-User bekommt nur die Privilegien (SELECT, UPDATE, CREATE etc.) zugewiesen, die er für diese eine Aufgabe benötigt.

Warum konnte ich bei meinen obigen Angriffen Daten anderer Tabellen, sogar Daten anderer Datenbanken einsehen? Weil es mir keiner verboten hatte. In der `index01.php` habe ich mich als `root` angemeldet, was offensichtlich eine superblöde Entscheidung war. Wenn ich Seiten/Masken habe, die nur bestimmte Recherchen durchführen sollen, sollte ich mich auch mit einem Benutzer anmelden, der sehr restriktiven Zugriff hat.

```
1 CREATE USER
2     'injection_artikel_read'@'localhost' IDE
NTIFIED BY 'supertollespassword'
3 ;
4
5 GRANT
6 SELECT
7 ON injection.artikel
8 TO 'injection_artikel_read'@'localhost'
9 ;
```

Nun muss nur noch der Connection-String angepasst werden:

```
1    $server = "localhost";  
2    $user = "injection_artikel_read";  
3    $password = "supertollespasswort";  
4    $dbname = "injection";
```

Versuchen Sie doch jetzt noch einmal auf `injection.account` oder `mysql.user` zuzugreifen.

25.3.4 Nichtssagende Fehlermeldungen



Hinweis: Geben Sie keine Fehlermeldung, die vom Web- oder Datenbankserver direkt kommt, für den Anwender sichtbar aus. Nutzen Sie lieber die vorhandenen Logging-Techniken.

In [Zeile 10](#) von `index01.php` wird mit `die()` eine Fehlermeldung auf dem Browser direkt ausgegeben. Dies sollten Sie in einer Produktivumgebung tunlichst unterlassen! Die Fehlermeldungstexte liefern nämlich Angreifern potenzielle Hebel wie Dateinamen, IP-Adressen usw.

Fangen Sie die Fehlermeldungen lieber ab und geben Sie nur allgemeine Hinweise aus wie: *Es ist zu einer technischen Störung gekommen. Das Technikteam ist schon verständigt worden. Versuchen Sie es später noch einmal.*

Viele Sprachen stellen Ihnen Funktionen oder Klassen zur Verfügung, mit deren Hilfe Sie die Fehlerprotokollierung gut managen können. So können Sie beispielsweise in PHP mithilfe der Funktion `error_log()` festlegen, ob die Meldungen eine E-Mail auslösen sollen oder in log-Dateien

landen oder andere Dienste bedient werden sollen. Unsere [Zeile 10](#) könnte also auch so aussehen:

```
10 error_log($conn->connect_error, 1, "admin@  
    injection.wurst");
```

Hier wird eine E-Mail mit dem Inhalt von `$conn->connect_error` an die Adresse `admin@injection.wurst` versendet.

¹ Siehe [\[Aka19\]](#). SQLi ist die Abkürzung von SQL injection.

² OWASP ist die Abkürzung von OpenWeb Application Security Project. Siehe [\[Fou22\]](#).

³ Natürlich muss ich hier auf eine HTML-/PHP-Einführung verzichten. Für HTML empfehle ich [\[SEL23\]](#), für PHP [\[The23a\]](#) und speziell für die mysqli-Schnittstelle [\[The23b\]](#).

⁴ Sie können die Übergabedaten in der URL sehen. Eine Übergabe per POST ist übrigens um keinen Deut sicherer!

⁵ Wie Sie erkennen können, verwende ich dieMySQLi-Bibliothek und nicht die allgemeinere PDO-Bibliothek. PDO kann auch auf andere als den MySQL Server zugreifen und bietet eine ODBC-Schnittstelle. Ich wollte aber explizit auf diesen Overhead verzichten und habe mich deshalb für MySQLi entschieden.

⁶ Regenbogentabellen sind effizient strukturierte, teilweise riesige Datensammlungen von Hash-Werten. Eine gute Einführung finden Sie in [\[Wik23q\]](#).

⁷ Was nicht gleichbedeutend mit *selten* ist. Im Gegenteil, durch geschicktes Raten kann die Wahrscheinlichkeit recht hoch geschraubt werden.

⁸ Quelle: [\[Dai20\]](#)

⁹ Inferential (engl.): herleitbar; sich aus Schlussfolgerungen ergebend

¹⁰ Das ist nicht immer und nicht bei allen SQL Servern so. Bitte informieren Sie sich in den entsprechenden technischen Dokumentationen oder in Online-Foren. Besonders kritisch wird es, wenn die Benutzereingaben nicht maskiert werden und schadhafter Objectcode übergeben wird.

26 SQL-Referenz

26.1 Datentypen

26.1.1 Numerische Datentypen

26.1.1.1 Ganze Zahlen

Tabelle 26.1 Ganzzahlige Datentypen (* = MySQL proprietär)

Typ	#Bytes		Minimum		Maximum
TINYINT(<i>l</i>)*	1	-2^7 =	-128	2^7 -1 =	127
SMALLINT(<i>l</i>)	2	-2^{15} =	-32.768	2^{15} -1 =	32.767
MEDIUMINT(<i>l</i>)*	3	-2^{23} =	-8.388.608	2^{23} -1 =	8.388.607
INT(<i>l</i>)	4		-2.147.483.648		2.147.483.647

		-2^{31} =		2^{31} -1 =	
BIGINT(<i>l</i>)*	8	-2^{63} ≈	$-9,223 \times 10^{18}$	2^{63} -1 ≈	9,223
TINYINT(<i>l</i>) UNSIGNED*	1	2^0 -1 =	0	2^8 -1 =	255
SMALLINT(<i>l</i>) UNSIGNED*	2	2^0 -1 =	0	2^{16} -1 =	65.53
MEDIUMINT(<i>l</i>) UNSIGNED*	3	2^0 -1 =	0	2^{24} -1 =	16.77
INT(<i>l</i>) UNSIGNED*	4	2^0 -1 =	0	2^{32} -1 =	4.294
BIGINT(<i>l</i>) UNSIGNED*	8	2^0 -1 =	0	2^{64} -1 ≈	18,44
BOOL*	1	0 =	FALSE	1 =	TRUE
ENUM*	2		0		65.53
SET*	1-8		-		-

BIT(l)*	1-8		0		2/-1

Bei den Datentypen TINYINT, SMALLINT, MEDIUMINT, INT und BIGINT steht das l für die Länge der Ausgabe. Diese Angabe ist optional und kann von der Anwendung verwendet oder ignoriert werden. Sie hat keine Auswirkung auf den Wertebereich.

Der Datentyp ENUM speichert eine Zahl ab. In den internen Infos über eine Tabelle wird eine Liste vorgehalten, die jeder Zahl eine passende Zeichenkette (String) zuordnet. Da ihm 2 Bytes zur Verfügung steht, können maximal 36.535 Zeichenketten verwendet werden. Weil Zeichenketten verwendet werden, kann man diesen einen Zeichensatz und eine Sortierung zuweisen.

Der Datentyp SET weist jedem Element einer Menge eine Zweierpotenz als Wert zu. Der Wert der Spalte in einer Zeile ergibt sich dadurch, welche Elemente der Menge zugewiesen wurden. Als konkrete Werte kann man NULL, einzelne Werte oder eine kommaseparierte Liste von Werten aus der Menge zuweisen. Somit können pro Spalte eine Menge von maximal 64 unterschiedlichen Elementen ($8\text{Bit} \times 8$) definiert werden. In jeder Zeile können einer SET-Spalte eine Werteliste mit maximal 255 Werten zugewiesen werden. *Ein Beispiel:*

```
mysql> CREATE TEMPORARY TABLE bla
-> (spalte SET('milch', 'kaffee', 'tee', 'wurst', 'marmelade'));
mysql> INSERT INTO bla
-> VALUES
-> ('kaffee,milch')
-> ,('tee,wurst,marmelade') ;
mysql> SELECT * FROM bla;
+-----+
| spalte |
```



```

+-----+
| milch,kaffee |
| tee,wurst,marmelade |
+-----+

```

Der Datentyp BIT wird für das Abspeichern einzelner Bit-Werte verwendet. Dabei gibt *l* die Anzahl der Bits an. Mit BIT(4) können also vier Bitwerte abgespeichert werden. Maximal ist dabei eine Länge von 64 möglich. Bei der Wertzuweisung wird ein String von 0en und 1en verwendet, denen ein kleines b vorangestellt wird: b'01011'.

26.1.1.2 Gebrochene Zahlen

Tabelle 26.2 Gebrochenzahlige Datentypen (* = MySQL proprietär)

Typ	#Bytes	Minimum	Maximum
FLOAT	4	$-3.402823466 \times 10^{+38}$ $+1.175494351 \times 10^{-38}$	-1.1754943 $+3.4028234$
REAL	4 8	wie FLOAT	wie FLOAT
DOUBLE	8	$-1.7976931349 \times 10^{+308}$ $+2.2250738585 \times 10^{-308}$	-2.2250738 $+1.7976931$
DOUBLE PRECISION	8	wie DOUBLE	wie DOUBLE
DECIMAL(<i>l</i> , <i>d</i>)	<i>l</i>	<i>l</i> = 1, <i>d</i> = 0	<i>l</i> = 65, <i>d</i> = 30
NUMERIC(<i>l</i> , <i>d</i>)	<i>l</i>	wie DECIMAL	wie DECIMAL

Alle gebrochenen Zahlen können zusätzlich den Wert 0

annehmen.

Der Datentyp REAL ist in MySQL ein Synonym für DOUBLE. Aktivieren Sie den SQL-Modus REAL_AS_FLOAT, so ist er ein Synonym für ein FLOAT. In der SQL:2023-Spezifikation wird verlangt, dass der REAL einen Wertebereich kleiner als DOUBLE PRECISION hat. Auch in anderen Programmiersprachen (Pascal-Familie) wird ein REAL als ein FLOAT verstanden. Vermeiden Sie den Datentyp FLOAT in MySQL oder MariaDB, da die Werte bei Berechnungen in DOUBLE umgewandelt werden. Verwenden Sie lieber direkt DOUBLE oder DECIMAL.

Die Datentypen DECIMAL und NUMERIC sind Festkommazahlen, die l viele Stellen insgesamt umfasst. Der Wertebereich von l liegt zwischen 1 und 65. Die Angabe d legt fest, wie viele Stellen hinter dem Komma gespeichert werden. Der Wertebereich von d ist 0 bis 30. Es muss gelten $l > d$. Die Angaben bestimmen den Wertebereich der Spalte. Innerhalb dieses Wertebereichs sind die Daten exakt und ohne Rundungsfehler. Tatsächlich werden die Daten als Zeichenketten abgespeichert. Es ist allerdings darauf zu achten, dass die Werte, die der Spalte zugewiesen werden, innerhalb der Festlegung liegen, ansonsten werden hier bei der Zuweisung Rundungsfehler erzeugt.

In SQL:2023 besteht ein Unterschied zwischen DECIMAL und NUMERIC. Bei NUMERIC werden exakt d Stellen hinter dem Komma abgespeichert. Bei DECIMAL werden mindestens d Stellen hinter dem Komma abgespeichert. Die Anzahl wird bei Bedarf verlängert und ist nur durch technische Grenzen eingeschränkt. In MySQL/MariaDB sind diese Datentypen Synonyme.

Der wesentliche Unterschied zwischen einem DECIMAL und einem DOUBLE ist, dass der DOUBLE einen großen Wertebereich relativ ungenau abdeckt (Breitenabdeckung) und der DECIMAL

innerhalb seines kleineren Wertebereichs exakt ist (Tiefenabdeckung).

26.1.2 Zeichen-Datentypen

Tabelle 26.3 Zeichen Datentypen(* = MySQL proprietär)

Typ	#Bytes	Maximale Länge
CHAR(<i>l</i>)	<i>l</i>	255
BINARY(<i>l</i>)	<i>l</i>	255
VARCHAR(<i>l</i>)	<i>l</i> +1 <i>l</i> +2	255 65.535
VARBINARY(<i>l</i>)	<i>l</i> +1 <i>l</i> +2	255 65.535
TINYTEXT*	<i>länge</i> +1	255
TEXT[<i>l</i>]*	<i>länge</i> +2	65.535
MEDIUMTEXT*	<i>länge</i> +3	16.777.215
LONGTEXT*	<i>länge</i> +4	4.294.967.295

Jeder Spalte mit einem Zeichen-Datentyp können ein Zeichensatz und eine Sortierung zugewiesen werden. Ohne Zusätze oder Zuweisung entsprechender Sortierungen wird nicht zwischen Groß- und Kleinschreibung unterschieden.

Bitte beachten Sie, dass bei Zeichensätzen, die mehr als ein Byte pro Zeichen verwenden, die effektive Zeichenlänge geringer ist. Ein CHAR(21) kann bei einer utf8-Kodierung im schlechtesten Fall nur sieben Zeichen aufnehmen.

Der Unterschied zwischen CHAR und VARCHAR ist, dass CHAR immer eine gleich lange Zeichenkette abspeichert und VARCHAR nur so viele, wie er braucht. Das Wort *wurstbrot* würde bei einem VARCHAR(15) nur 10 Zeichen (*länge* +1) verbrauchen, bei einem CHAR(15) genau 15.

Dem Datentyp TEXT kann eine Länge mitgegeben werden. Anhand dieser Länge ermittelt MySQL, welcher der Datentypen TINYTEXT, TEXT, MEDIUMTEXT oder LONGTEXT verwendet werden soll.

Will man einen Index auf eine der TEXT-Datentypspalten erstellen, muss eine Präfixlänge angegeben werden. Ansonsten erhält man die Fehlermeldung:

```
ERROR 1170 (42000): BLOB/TEXT column 'spaltenname' used in key specification without a key length
```

Der Inhalt einer Textspalte könnte beispielsweise 16 KB groß sein kann. Ein Index über einen solchen Schlüssel ist einfach *Schwachsinn* (siehe Experiment zur Indexselektivität in [Abschnitt 6.2.3](#))¹.

Die Datentypen BINARY und VARBINARY unterscheiden sich von CHAR bzw. VARCHAR dadurch, dass die Texte mal als *Zeichenfolge* und mal als *Bytefolge* interpretiert werden. Bei Zeichenfolgen werden die einzelnen Werte anhand ihrer Kodierung und Sortierung mit Zahlenwerten versehen und verarbeitet. Bei Bytefolgen wird jedes Byte als die Zahl interpretiert, die es in seiner Binärdarstellung hat. Daher wird anders sortiert – die *_bin*-Sortierungen der Zeichenkodierungen – und zwischen Groß- und Kleinschreibung unterschieden.

26.1.3 Datums- und Zeit-Datentypen

Datums- und Uhrzeitwerte sind für den Programmierer mühsam und fehlerträchtig. Zum einen gibt es viele verschiedene Kalender: julianisch, gregorianisch, muslimisch, jüdisch, chinesisch etc. Zum anderen gibt es innerhalb der Kalender die Zeitzone und darüber hinaus noch unterschiedliche Darstellungsformate.

MySQL/MariaDB verwenden den proleptischen gregorianischen Kalender: Beim Wechsel vom julianischen zum gregorianischen Kalender werden einige Tage (z.B. 11) übersprungen. Alle Datumswerte vor diesem Sprung sind julianisch, alle danach gregorianisch. Der proleptische gregorianische Kalender geht vereinfachend davon aus, dass es einen solchen Sprung nie gegeben hat und schon immer der gregorianische Kalender verwendet wurde.

Tabelle 26.4 Datum- und Uhrzeitdatentypen (Auszug, * = MySQL proprietär)

Typ	#Bytes	Minimum	Maximum
YEAR*	1	1901	2155
DATE	3	'1000-01-01'	'9999-12-31'
TIME[(n)]	3+n	'00:00:00.000000'	'23:59:59.999999'
TIME[(n)]	3+n	'-838:59:59.000000'	'838:59:59.999999'
DATETIME[(n)]*	8+n	'1000-01-01 00:00:00.000000'	'9999-12-31 23:59:59.999999'
TIMESTAMP[(n)]	8+n	'1970-01-01 00:00:01.000000'	'2038-01-19 03:14:07.999999'

Die Bedeutung der Datumsformatierungszeichen entnehmen Sie der [Tabelle 26.5](#) und die der Uhrzeitformatierungszeichen [Tabelle 26.6](#).

- YEAR hat das Format %Y.
- DATE hat das Format %Y-%m-%d.
- TIME: Entweder wird eine Uhrzeit oder eine Zeitdifferenz abgespeichert. Die Uhrzeit hat das Format %H-%i-%S[. %f], wobei der letzte Teil der optionale Sekundenbruchteil ist. Die Zeitdifferenz kann eine dreistellige Stundenangabe haben und auch negative Werte enthalten. Das Format ist nnn-%i-%S[. %f]. In beiden Fällen wird die Länge des Sekundenbruchteils durch *n* angegeben und kann maximal 6 sein. Die Speichergröße wird entsprechend angepasst.
- DATETIME: Eine Kombination aus Datum und Uhrzeit. Dieser Datentyp kann auch Sekundenbruchteile abbilden. Dazu wird hinter den Sekundenangaben durch einen Punkt getrennt der Sekundenbruchteil angegeben: %Y-%m-%d %H:%i:%S[. %f]. Die Länge des Sekundenbruchteils wird durch *n* angegeben und kann maximal 6 sein. Die Speichergröße wird entsprechend angepasst.
- TIMESTAMP: Wie DATETIME eine Kombination aus Datums- und Uhrzeitwerten. Intern wird der Wert aber nicht wie DATETIME als eine Zeichenkette, sondern als die Anzahl der Sekunden seit 01.01.1970 ab 00:00:00 Uhr abgespeichert. Zusammen mit seiner Länge von 8 Bytes ergibt sich daraus der Wertebereich. Dieser Datentyp kann auch Sekundenbruchteile abbilden. Dazu wird hinter den Sekundenangaben durch einen Punkt getrennt der Sekundenbruchteil angegeben: %Y-%m-%d

%H:%i:%S.%f]. Die Länge des Sekundenbruchteils wird durch *n* angegeben und kann maximal 6 sein. Die Speichergröße wird entsprechend angepasst. Die Zuweisung eines NULL-Werts setzt das Feld auf die aktuelle **Zeit des Servers**.

Tabelle 26.5 Formatierungszeichen (FZ) für Datumswerte

%FZ	Bedeutung
%a	abgekürzter Wochentag (Son, ..., Sat)
%b	abgekürzter Monat (Jan, ..., Dec)
%c	Monatszahl (0, ..., 12)
%D	Tageszahl des Monats mit englischem Suffix (0th, 1st, 2nd, 3rd, ...)
%d	zweistellige Tageszahl des Monats (00, ..., 31)
%e	Tageszahl des Monats (0, ..., 31)
%j	dreistellige Tageszahl des Jahres (001, ..., 366)
%M	Monatsname (January, ..., December)
%m	zweistellige Monatszahl (00, ..., 12)
%U	zweistellige Wochenzahl (00, ..., 53), wobei Sonntag der erste Wochentag ist
%u	zweistellige Wochenzahl (00, ..., 53), wobei Montag der erste Wochentag ist
%V	

	zweistellige Wochenzahl (01, ..., 53), wobei Sonntag der erste Wochentag ist; siehe %X
%v	zweistellige Wochenzahl (01, ..., 53), wobei Montag der erste Wochentag ist; siehe %x
%W	Wochentag (Sunday, ..., Saturday)
%w	Tageszahl der Woche (0=Sonntag, ..., 6=Samstag)
%X	vierstellige Jahreszahl einer Woche, wobei Sonntag der erste Wochentag ist; siehe %V
%x	vierstellige Jahreszahl einer Woche, wobei Montag der erste Wochentag ist; siehe %v
%Y	vierstellige Jahreszahl
%y	zweistellige Jahreszahl
%%	das % Zeichen

Tabelle 26.6 Formatierungszeichen (FZ) für Uhrzeitwerte

%FZ	Bedeutung
%f	Mikrosekunden (000000, ..., 999999)
%H	zweistellige Stunde in 24-Stunden-Anzeige (00, ..., 23)
%h	zweistellige Stunde in 12-Stunden-Anzeige (01, ..., 12)
%I	wie %h

%i	zweistellige Minuten (00, . . . , 59)
%k	Stunde in 24-Stunden-Anzeige (0, . . . , 23)
%l	Stunde in 12-Stunden-Anzeige (1, . . . , 12)
%p	AM oder PM
%r	Uhrzeit in 12-Stunden-Anzeige (%h:%m:%S AM oder PM)
%S	zweistellige Sekunden (00, . . . , 59)
%s	wie %S
%T	Uhrzeit in 24-Stunden-Anzeige (%H:%m:%S)

MySQL/MariaDB stellen die Funktion `DATE_FORMAT(datum, format)` zur Verfügung. Diese wandelt den Wert *datum* in ein frei definierbares Format um. Das Format wird mithilfe von Formatierungszeichen in *format* festgelegt. Eine Liste der verfügbaren Formatierungszeichen für Datumswerte finden Sie in [Tabelle 26.5](#).

Für die Uhrzeit gibt es die Funktion `TIME_FORMAT(zeit, format)`, welche analog wie `DATE_FORMAT()` arbeitet. Eine Liste der verfügbaren Formatierungszeichen für Uhrzeitwerte finden Sie in [Tabelle 26.6](#).

Einige Formatierungen sind schon passend vordefiniert. Dies betrifft länderspezifische Einstellungen oder die ISO 9075. Die vordefinierten Formate können mit `GET_FORMAT(wert, format)` ermittelt werden (siehe [Tabelle 26.7](#)).

Ein Beispiel:

```

10  mysql> SELECT
11      -> DATE_FORMAT(datum, GET_FORMAT(DATE,
12      'ISO')) ISO,
12      -> DATE_FORMAT(datum, GET_FORMAT(DATE,
13      'EUR')) EUR
13      -> FROM bestellung
14      -> ORDER BY EUR
15      -> LIMIT 6
16      -> ;
17  +-----+-----+
18  | ISO | EUR |
19  +-----+-----+
20  | 2012-04-01 | 01.04.2012 |
21  | 2012-05-10 | 10.05.2012 |
22  | 2012-05-11 | 11.05.2012 |
23  | 2012-05-12 | 12.05.2012 |
24  | 2011-01-15 | 15.01.2011 |
25  | 2011-01-16 | 16.01.2011 |
26  +-----+-----+

```

Tabelle 26.7 Vordefinierte Datums- und Uhrzeitformatierungen

Ausrufe	Ergebnis
GET_FORMAT(DATE, 'USA')	'%m.%d.%Y'
GET_FORMAT(DATE, 'JIS')	'%Y-%m-%d'
GET_FORMAT(DATE, 'ISO')	'%Y-%m-%d'
GET_FORMAT(DATE, 'EUR')	'%d.%m.%Y'
GET_FORMAT(DATE, 'INTERNAL')	'%Y%m%d'
GET_FORMAT(DATETIME, 'USA')	'%Y-%m-%d %H.%i.%s'

GET_FORMAT(DATETIME, 'JIS')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME, 'ISO')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME, 'EUR')	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME, 'INTERNAL')	'%Y%m%d%H%i%s'
GET_FORMAT(TIME, 'USA')	'%h:%i:%s %p'
GET_FORMAT(TIME, 'JIS')	'%H:%i:%s'
GET_FORMAT(TIME, 'ISO')	'%H:%i:%s'
GET_FORMAT(TIME, 'EUR')	'%H.%i.%s'
GET_FORMAT(TIME, 'INTERNAL')	'%H%i%s'

26.1.4 Binäre Datentypen

Binäre Datentypen werden für nicht zeichenbasierte Daten verwendet: Multimedia, pdf-Dateien, ausführbare Dateien, Signaturen etc. Sie können nicht so einfach verarbeitet werden wie die anderen Datentypen. Das Indizieren, Suchen und Ändern ist mit Besonderheiten verbunden. Diese sind für jeden DBMS-Anbieter proprietär, und Sie müssen in den Spezifikationen des Anbieters nachschauen, wie diese verarbeitet werden.

Tabelle 26.8 Binärdatentypen (Auszug, * = MySQL proprietär)

Typ	#Bytes	Kommentar
TINYBLOB*	länge+1	Ein Binärfeld mit einer maximalen Länge von 255 B.
BLOB*	länge+2	Ein Binärfeld mit einer maximalen Länge von 64 KB.
MEDIUMBLOB*	länge+3	Ein Binärfeld mit einer maximalen Länge von 16 MB.
LOB*	länge+4	Ein Binärfeld mit einer maximalen Länge von 4 GB.

Der Unterschied zwischen TEXT und BLOB ist analog dem von CHAR und BINARY. In [Abschnitt 7.2.4](#) wird gezeigt, wie Sie beispielsweise über C#-API ein Bild einfügen können, und in [Abschnitt 10.6.3](#) dann, wie Sie die Bilddaten auslesen und wieder in eine Datei umwandeln.

26.1.5 JSON

Was JSON ist und wie man es verwendet, habe ich schon in [Abschnitt 17.2.1](#) beschrieben. Eigentlich könnte man JSON-Dokumente beispielsweise in eine VARCHAR-Spalte speichern. Wird eine Spalte aber mit dem Datentyp JSON deklariert, stehen zwei wichtige Features zur Verfügung²:

- Beim Speichern wird automatisch überprüft, ob es sich um ein gültiges JSON-Dokument handelt. Wenn nicht, wird eine Fehlermeldung erzeugt.

```
10 mysql> CREATE TABLE t (a JSON);
11 Query OK, 0 rows affected (0.47 sec)
```

```

12
13 mysql> INSERT INTO t
14 -> SET a = '{"spalte":"wert"}'
15 -> ;
16 Query OK, 1 row affected (0.07 sec)
17
18 mysql> SELECT * FROM t;
19 +-----+
20 | a |
21 +-----+
22 | {"spalte": "wert"} |
23 +-----+
24 1 row in set (0.01 sec)
25
26 mysql> INSERT INTO t
27 -> SET a = '{"spalte"- "wert"}'
28 -> ;
29 ERROR 3140 (22032): Invalid JSON text: "Missing a colon after a name of object member."
   at position 9 in value for column 't.a'.

```

- Intern wird das JSON-Dokument strukturiert abgespeichert, sodass Verarbeitungs- und Auswertungsfunktionen erheblich beschleunigt verarbeitet werden können.

Eine ausführliche Anleitung bzgl. der JSON-Funktionen finden Sie hier: <https://dev.mysql.com/doc/refman/8.0/en/json.html>.

26.1.6 Räumliche Datentypen

Grundlage der räumlichen Datentypen sind die vom *Open Geospatial Consortium (OGC)* in OpenGIS festgelegten Regeln und Normierungen (siehe [The10]). Unter <https://itnext.io/playing-with-geometry-spatial-data-type-in-mysql-645b83880331> finden Sie eine nette Einführung. Eine

tiefergehende Einführung hier verbietet sich leider aus Platzgründen :-(

- POINT: Repräsentiert einen zweidimensionalen Punkt. Auf einer Karte kann ein Punkt ein Gebäude oder ein Unternehmen repräsentieren. Zwischen den Werten steht kein Komma: $(x\ y)$.
- MULTIPOINT: Eine Folge von Punkten. Auf einer Karte könnten dies die Filialen einer Sparkasse oder alle Haltestellen einer Buslinie sein. Die Punkte werden durch Kommata getrennt: $(x_1\ y_1, \dots, x_n\ y_n)$.
- LINESTRING: Eine durch Punkte definierte interpolierte Kurve. Auf einer Karte könnte dies der Weg einer Buslinie oder eine Stromtrasse sein: $(x_1\ y_1, \dots, x_n\ y_n)$.
- MULTILINESTRING: Eine Sammlung von LINESTRING-Kurven. Auf einer Karte könnten dies die Routen der Müllabfuhr oder ein Flusssystem sein. LINESTRING-Kurven werden durch runde Klammern definiert: $((x_{11}\ y_{11}, \dots, x_{1n}\ y_{1n}), \dots, (x_{m1}\ y_{m1}, \dots, x_{mn}\ y_{mn}))$.
- POLYGON: Ein Polygon ist eine geschlossene Fläche, die durch LINESTRING-Kurven definiert wird. Die Kurven dürfen sich nicht schneiden. In einer Karte können dies die Stadtgrenzen oder die Ufer eines Sees sein.
- MULTIPOLYGON: Eine Sammlung von POLYGON-Flächen. Diese müssen schnittmengenfrei sein. In einer Karte könnten dies die landwirtschaftlich genutzten Flächen einer Stadt oder die Fußgängerzonen sein.
- GEOMETRY: Dieser Datentyp kann alle Informationen einzelner räumlicher Datenobjekte beinhalten. Der Programmierer muss selbst durch geeignete Funktionen die Informationen für sich aufbereiten.

- **GEOMETRYCOLLECTION:** Dieser Datentyp kann alle Sammlungen von Informationen einzelner räumlicher Datenobjekte beinhalten. Der Programmierer muss selbst durch geeignete Funktionen die Informationen für sich aufbereiten.

26.1.7 Standardwerte

Tabelle 26.9 Datentypen (Auszug): Standardwerte

Typ <i>datentyp</i> NULL	Wert NULL	Kommentar Wenn die Spalte den Wert NULL enthalten darf, wird unabhängig vom Datentyp NULL eingetragen.
<i>datentyp</i> DEFAULT <i>wert</i>	<i>wert</i>	Wenn die Spalte einen Standardwert zugewiesen bekommen hat, wird dieser <i>wert</i> eingetragen.
<i>numerisch</i>	0	Alle numerischen Datentypen werden auf 0 oder 0.0 gesetzt.
<i>Zeichenketten</i>	' '	Alle zeichenbasierten Datentypen werden mit der leeren Zeichenkette initialisiert.
<i>Datum</i>	'0000-00-00'	Dies entspricht nicht dem Datum 01.01.0000 oder 01.01.1970.

YEAR	0000	Dies entspricht nicht dem Jahr 0 oder 1970.
TIME	'00:00:00'	Dies entspricht der ersten Sekunde eines Tages.

Wird eine neue Zeile eingefügt und zu einer Spalte keine Angabe gemacht, werden die Standardwerte eingetragen. Wurde ein DEFAULT – beispielsweise beim CREATE TABLE – angegeben, wird dieser verwendet. Ist kein eigener DEFAULT angegeben worden und ist in der Spalte der Wert NULL erlaubt, wird immer NULL verwendet. Ist NULL nicht erlaubt, werden die in [Tabelle 26.9](#) angegebenen Werte eingetragen.

26.1.8 Zusätze für Datentypen

Tabelle 26.10 Zusätze (Auszug, * = MySQL proprietär)

Typ	Kommentar
AS <i>ausdruck</i> * AUTO_INCREMENT*	Dieser Zusatz erzeugt analog einer Ansicht eine virtuelle Spalte. Dieser Zusatz ermöglicht einem ganzzahligen Feld die automatische Wertfindung. Dies eignet sich zur Generierung von Schlüsseln. Die Daten können in dieses Feld gelesen und auch geschrieben werden. Wenn aber ein Wert 0 oder NULL zugewiesen wird, wird der nächste Zahlenwert zugewiesen. Dieser Zusatz kann nur einmal pro Tabelle und nur dem Primärschlüssel zugewiesen werden.

BINARY*	Dieser Zusatz kann bei CHAR- und VARCHAR-Typen verwendet werden, um die Beachtung der Groß- und Kleinschreibung zu erzwingen.
CHECK <i>bedingung</i>	Mithilfe der <i>bedingung</i> kann die Domäne (siehe Definition 5) festgelegt werden.
DEFAULT <i>wert</i> *	Mit diesem Zusatz können Spalten Vorbelegungen zugewiesen werden. Falls einer Spalte kein Wert zugewiesen wird, wird automatisch der Wert <i>wert</i> zugewiesen.
FOREIGN KEY	Markiert, dass diese Spalte Teil eines Fremdschlüssel ist. Wird in MySQL von MyISAM nicht verarbeitet. InnoDB wertet den Constraint aus.
GENERATED <i>art</i> AS IDENTITY	Dieser Zusatz ermöglicht einem ganzzahligen Feld die automatische Wertfindung. Dies eignet sich zur Generierung von Schlüsseln. Es gibt zwei Varianten: <i>art</i> = ALWAYS erzeugt immer den neuen Wert und akzeptiert keine Werte von außen. <i>art</i> = BY DEFAULT erzeugt nur dann einen Wert, wenn er den Wert 0 oder NULL von außen bekommt (vergleichbar mit AUTO_INCREMENT). Dieser Zusatz kann nur einmal pro Tabelle und nur dem Primärschlüssel zugewiesen werden.

	Dieser Zusatz wird in dieser Form von MySQL nicht verarbeitet (s.u.).
GENERATED ALWAYS AS PRIMARY KEY*	Dieser Zusatz ermöglicht einem ganzzahligen Feld die automatische Wertfindung. Er erzeugt immer den neuen Wert und akzeptiert keine Werte von außen.
NULL*	Die Spalte darf auch keinen Attributwert haben. Dies ist nicht gleichbedeutend mit der Zahl 0 oder dem Leer-String!
NOT NULL	Spalten mit diesem Zusatz dürfen nicht leer sein.
PRIMARY KEY	Markiert, dass diese Spalte Teil des Primärschlüssels ist.
UNIQUE	Erzwingt die Schlüsseleigenschaft (siehe Definition 8) der Spalte.
UNSIGNED*	Bei ganzzahligen Zahlentypen steuert dieser Zusatz, ob nur positive oder auch negative Zahlen in der Spalte abgelegt werden können.

Die Zusätze sind datentypspezifisch. Wann welcher Zusatz möglich ist, entnehmen Sie bitte der Dokumentation Ihres DBMS.

Noch etwas zu *AS ausdrück*: Mithilfe dieses Zusatzes können Spalten einer Tabelle definiert werden, die sich aus Auswertungen und Berechnungen ergeben. Natürlich sind

diese dann *read only*³. Mithilfe dieser generierten Spalten könnte man häufig gebrauchte Varianten oder Auswertungen schon in die Tabelle programmieren und damit ggf. notwendige Ansichten sparen oder Abfragen vereinfachen:

```
10 MariaDB [test]> CREATE TABLE person (
11     -> vorname VARCHAR(255),
12     -> nachname VARCHAR(255),
13     -> listenname VARCHAR(255) AS (CON
CAT(nachname, ', ', vorname))
14     -> );
15 Query OK, 0 rows affected (0.04 sec)
16
17 MariaDB [test]> INSERT INTO person
18     -> SET
19     -> vorname = 'Heinz',
20     -> nachname = 'Bode'
21     -> ;
22 Query OK, 1 row affected (0.00 sec)
23
24 MariaDB [test]> SELECT * FROM person;
25 +-----+-----+-----+
26 | vorname | nachname | listenname |
27 +-----+-----+-----+
28 | Heinz  | Bode    | Bode, Heinz |
29 +-----+-----+-----+
30 1 row in set (0.00 sec)
31
32 MariaDB [test]> UPDATE person
33     -> SET
34     -> listenname = 'abc'
35     -> ;
36 Query OK, 0 rows affected, 1 warning (0.00
sec)
37 Rows matched: 1 Changed: 0 Warnings: 1
38
39 MariaDB [test]> SHOW WARNINGS\G
40 ***** 1. row *****
```

```

*****
41  Level: Warning
42  Code: 1906
43  Message: The value specified for computed
column 'listenname' in table 'person' ignored
44  1 row in set (0.00 sec)

```

Generierte Spalten lassen gut dafür nutzen, eigentlich etwas sperrige Inhalte – wie beispielsweise TEXT oder JSON – auszuwerten oder zu filtern. Besonders, da Sie diese Spalten indizieren können, ergeben sich viele Möglichkeiten der Performanceverbesserung.

26.2 Operatoren und Funktionen

26.2.1 Mathematische Operatoren

Operator	Name	Beispiel			
–	Unäres Minus	SELECT	–5;	→	–5
*	Multiplikation	SELECT	5 * 2;	→	10
/	Division	SELECT	5 / 2;	→	2.5
%	Modulo	SELECT	5 % 2;	→	1
MOD	Modulo	SELECT 5	MOD 2;	→	1

DIV	Ganzzahlige Division	SELECT 5	DIV 2;	→	2
+	Addition	SELECT	-5 + 2;	→	-3
-	Subtraktion	SELECT	-5 - 2;	→	-7
&	Bitweises AND	SELECT	25 & 8;	→	8
	Bitweises OR	SELECT	25 12;	→	29
^	Bitweises XOR	SELECT	25 ^ 12;	→	21
>>	Bitverschiebung nach rechts	SELECT	8 >> 1;	→	4
<<	Bitverschiebung nach links	SELECT	8 << 1;	→	16

Bei MySQL/MariaDB ergibt eine Division durch 0 NULL. Ist einer der Operanden NULL, ist das Ergebnis auch NULL.

Die Bitweisen-Operatoren arbeiten auf den Binärdarstellungen der Inhalte. Jedes einzelne Bit dieser Darstellung wird dann verknüpft bzw. verschoben. Die Rechts- bzw.

Linksverschiebung kann abhängig von der *little endian* bzw. *big endian* Architektur genau das gegenteilige Ergebnis liefern.

26.2.2 Mathematische Funktionen

Funktionen, die mit einem Sternchen (*) versehen sind, sind MySQL-proprietär.

ABS(x)	Absolutwert von x

```
SELECT ABS(-5), ABS(0), ABS(5);  
-> 5 0 5
```

ACOS(x)	Arcuscosinus von x

```
SELECT ACOS(1);  
-> 0
```

ASIN(x)	Arcussinus von x

```
SELECT ASIN(0.5);  
-> 0.5235987755982989
```

ATAN(x)	Arcustangens von x

```
SELECT ATAN(1);  
-> 0.7853981633974483
```

ATAN2(x, y), ATAN(x, y)	Arcustangens von xy

```
SELECT ATAN2(1, 0.5), ATAN(1, 0.5);  
-> 1.1071487177940904 1.1071487177940904
```

CEILING(x), CEIL(x)	Kleinste ganze Zahl $\geq x$

```
SELECT CEILING(-3.1), CEILING(-3.9), CEILING(3
```

```
.1), CEILING(3.9);  
-> -3 -3 4 4
```

*CONV(x,y,z)	Konvertiert x zur Basis y zur Basis z

```
SELECT CONV(255, 10, 16);  
-> FF
```

COS(x)	Cosinus von x

```
SELECT COS(0);  
-> 1
```

COT(x)	Cotangens von x

```
SELECT COT(1);  
-> 0.6420926159343308
```

CRC32(<i>ausdruck</i>)	Zyklische Redundanzprüfung von <i>ausdruck</i>

```
SELECT CRC32('wurstbrot');  
-> 524262623
```

DEGREES(x)	Konvertiert x von Bogenmaß nach Grad

```
SELECT DEGREES(PI());  
-> 180
```

EXP(x)	ex

```
SELECT EXP(1);  
-> 2.718281828459045
```

FLOOR(x)	Größte ganze Zahl $\leq x$

```
SELECT FLOOR(-3.1), FLOOR(-3.9), FLOOR(3.1), FLOOR(3.9);  
-> -4 -4 3 3
```

FORMAT(x,y)	Formatiert x anhand der Formatangaben in y

```
SELECT FORMAT(6235217.02562, 2, 'de_DE');  
-> 6.235.217,03
```

HEX(x)	Liefert die hexadezimale Schreibweise von x

```
SELECT HEX(255);  
-> FF
```

*LN(x)	Logarithmus naturalis von x

```
SELECT LN(2.718281828459045);  
-> 1
```

LOG(x,y)	Logarithmus von y zur Basis x

```
SELECT LOG(16, 65536);  
-> 4
```


LOG10(x)	Logarithmus zur Basis 10 von x

```
SELECT LOG10(1000);
-> 3
```

*LOG2(x)	Logarithmus zur Basis 2 von x

```
SELECT LOG2(256);
-> 8
```

MOD(x,y)	x Modulo y

```
SELECT MOD(27, 8);
-> 3
```

PI()	Die Konstante π

```
SELECT PI();
-> 3.141593
```

POWER(x,y), POW(x,y)	xy

```
SELECT POWER(2,8);
-> 256
```

RADIANS(x)	Konvertiert x von Grad nach Bogenmaß

```
SELECT RADIANS(180);
-> 3.141592653589793
```

RAND()	Zufallszahl $z \in [0.0, 1.0[$

```
SELECT RAND();
-> 0.43608484178416884
```

RAND(x)	Wie RAND(), aber mit seed x

```
SELECT RAND(1);
-> 0.40540353712197724
```

ROUND(x,y)	Rundet x auf y stellen

```
SELECT ROUND(1.49, 1);
-> 1.5
```

SIGN(x)	Vorzeichen von x

```
SELECT SIGN(-5), SIGN(0), SIGN(+5);
-> -1 0 1
```

SIN(x)	Sinus von x

```
SELECT SIN(PI()/2);
-> 1
```

SQRT(x)	

```
SELECT SQRT(16);
-> 4
```

TAN(x)	Tangens von x

```
SELECT TAN(PI()/4);  
-> 0.9999999999999999
```

Das Ergebnis müsste eigentlich 1 lauten. Rundungsfehler ist hier das Problem.

TRUNCATE(x,y)	Schneidet x nach der yten Stelle ab

```
SELECT TRUNCATE(2.367, 2);  
-> 2.36
```

26.2.3 Aggregatfunktionen

Die mit einem Sternchen (*) versehenen Funktionen sind MySQL-proprietär.

AVG(<i>ausdruck</i>)	Arithmetische Mittel der Werte in <i>ausdruck</i>

```
SELECT AVG(einzelpreis) FROM artikel;  
-> 11.6210000000
```

*AVG(DISTINCT <i>ausdruck</i>)	Arithmetische Mittel der unterschiedlichen Werte in <i>ausdruck</i>

```
SELECT AVG(DISTINCT einzelpreis) FROM artikel;  
-> 12.4533333333
```

*BIT_AND(<i>ausdruck</i>)	Liefert die binäre UND-Verknüpfung aller Werte in

	<i>ausdruck</i>

```
SELECT BIT_AND(artikel_id) FROM artikel WHERE
artikel_id % 3 = 0;
-> 566
```

Was bedeutet das? Alle artikel_id-Werte werden als 32Bit-Zahlen kodiert. Von jeder Zahl wird nun beispielsweise die 5. Position mit allen anderen 5. Positionen der anderen Zahlen per UND verknüpft. Das Ergebnis ist wiederum eine 32Bit-Zahl.

*BIT_OR(<i>ausdruck</i>)	Liefert die binäre ODER-Verknüpfung aller Werte in <i>ausdruck</i>

```
SELECT BIT_OR(artikel_id) FROM artikel WHERE a
rtikel_id % 3 = 0;
-> 16319
```

*BIT_XOR(<i>ausdruck</i>)	Liefert die binäre exklusive ODER-Verknüpfung aller Werte in <i>ausdruck</i>

```
SELECT BIT_XOR(artikel_id) FROM artikel WHERE
artikel_id % 3 = 0;
-> 13886
```

COUNT(*)	Anzahl der Zeilen unabhängig vom Inhalt

```
SELECT COUNT(*) FROM kunde;
-> 5
```


COUNT(<i>ausdruck</i>)	Anzahl der Zeilen, deren <i>ausdruck</i> nicht NULL ist

```
SELECT COUNT(liefer_adresse_id) FROM kunde;
-> 2
```

*COUNT(DISTINCT <i>ausdruck</i>)	Anzahl unterschiedlicher Inhalte in <i>ausdruck</i>

```
SELECT COUNT(DISTINCT rechnung_adresse_id) FROM kunde;
-> 4
```

*GROUP_CONCAT(<i>ausdruck</i>)	String, der sich aus dem Gruppierungsergebnis von <i>ausdruck</i> ergibt

```
SELECT kunde_id, GROUP_CONCAT(rechnung_id SEPARATOR ';')
FROM rechnung
GROUP BY kunde_id;
```

```
+-----+-----+
-----+
| kunde_id | GROUP_CONCAT(rechnung_id SEPARATOR ';') |
+-----+-----+
-----+
| 1 | 1;3;4;12 |
| 2 | 2;6 |
| 3 | 7;8 |
| 5 | 9;10;11 |
+-----+-----+
-----+
```


*JSON_ARRAYAGG(<i>ausdruck</i>)	Liefert das Ergebnis der Gruppierung als JSON-Array.

```
SELECT kunde_id, JSON_ARRAYAGG(rechnung_id)
FROM rechnung
GROUP BY kunde_id;
```

```
+-----+-----+
| kunde_id | JSON_ARRAYAGG(rechnung_id) |
+-----+-----+
| 1 | [1, 3, 4, 12] |
| 2 | [2, 6] |
| 3 | [7, 8] |
| 5 | [9, 10, 11] |
+-----+-----+
```

*JSON_OBJECTAGG(<i>ausdruck</i>)	Liefert das Ergebnis der Gruppierung als JSON-Objekt.

Dabei werden zwei Spalten oder Ausdrücke verwendet: Der erste ist der Objektname und der zweite liefert die Werte.

```
SELECT kunde_id, JSON_OBJECTAGG(rechnung_id, k
unde_id)
FROM rechnung
GROUP BY kunde_id;
```

```
+-----+-----+
-----+
| kunde_id | JSON_OBJECTAGG(rechnung_id, kunde
_id) |
+-----+-----+
-----+
| 1 | {"1": 1, "3": 1, "4": 1, "12": 1} |
| 2 | {"2": 2, "6": 2} |
| 3 | {"7": 3, "8": 3} |
```

```
| 5 | {"9": 5, "10": 5, "11": 5} |
+-----+-----+
-----+
```

MAX(<i>ausdruck</i>)	Liefert den größten Wert in <i>ausdruck</i>

```
SELECT MAX(einzelpreis) FROM artikel;
-> 55.700000
```

MIN(<i>ausdruck</i>)	Liefert den kleinsten Wert in <i>ausdruck</i>

```
SELECT MIN(einzelpreis) FROM artikel;
-> 0.510000
```

STDDEV_POP(<i>ausdruck</i>), *STDDEV(<i>ausdruck</i>), *STD(<i>ausdruck</i>)	STDDEV_POP(<i>ausdruck</i>) = SQRT(VAR_POP(<i>ausdruck</i>)) Liefert die Populationsstandardabweichung abhängig von <i>ausdruck</i>.

```
SELECT STDDEV_POP(einzelpreis) FROM artikel;
-> 15.7674414221
```

STDDEV_SAMP(<i>ausdruck</i>)	STDDEV_SAMP(<i>ausdruck</i>)=SQRT Liefert die Stichprobenstandardabweichung von <i>ausdruck</i>.

```
SELECT STDDEV_SAMP(einzelpreis) FROM artikel;
-> 16.6203425891
```


SUM(<i>ausdruck</i>)	Liefert die Summe von <i>ausdruck</i>

```
SELECT SUM(einzelpreis) FROM artikel;
-> 116.210000
```

*SUM(DISTINCT <i>ausdruck</i>)	Summe unterschiedlicher Werte von <i>ausdruck</i>

```
SELECT SUM(DISTINCT einzelpreis) FROM artikel;
-> 112.080000
```

VAR_POP(<i>ausdruck</i>), *VARIANCE(<i>ausdruck</i>)	Varianz unter der Annahme, dass alle Elemente der Menge in <i>ausdruck</i> einfließen. Mit anderen Worten: Die Stichprobe enthält die gesamte Menge.

```
SELECT VAR_POP(einzelpreis) FROM artikel;
-> 248.6122090000
```

VAR_SAMP(<i>ausdruck</i>)	Varianz unter der Annahme, dass nur eine Teilmenge in <i>ausdruck</i> einfließt. Mit anderen Worten: Die Stichprobe enthält die nicht gesamte Menge.

```
SELECT VAR_SAMP(einzelpreis) FROM artikel;
-> 276.2357877778
```

26.3 Bedingungen

26.3.1 Vergleichsoperatoren

Das Ergebnis einer Bedingung ist laut [Definition 34](#) in [Abschnitt 9.1](#) TRUE oder FALSE. Kann eine Bedingung nicht ermittelt werden oder ist eine der Teilbedingungen UNKNOWN oder kommt der Wert UNKNOWN in der Bedingung vor, ist das Ergebnis auch UNKNOWN.



Hinweis: Bei Kommazahlen – FLOAT, DOUBLE, DECIMAL – sind Rundungsfehler zu beachten.

Operator	Beispiele
=	$wert_1 = wert_2$ deleted = 0 $wert = artikel\_id$

Liefert TRUE, wenn die rechts und links vom Gleichheitszeichen stehenden Werte gleich sind, sonst FALSE. Haben $wert_1$ und $wert_2$ den Wert NULL, liefert der Vergleich ebenfalls NULL. Die Datentypen dürfen unterschiedlich sein. Vorsicht: Anders als bei C/C++ und verwandten Sprachen steht hier nur **ein** Gleichheitszeichen.

Operator	Beispiele
<=>	$wert_1 <=> wert_2$ deleted <=> 0 $wert <=> MAX(menge)$

Liefert TRUE, wenn die rechts und links von <=> stehenden Werte gleich sind, sonst FALSE. Haben $wert_1$ und $wert_2$ den

Wert NULL, liefert der Vergleich anders als = den Wert TRUE. Die Datentypen dürfen unterschiedlich sein.

Operator	Beispiele
<> !=	$wert_1 <> wert_2$ deleted != 0

Liefert TRUE, wenn die rechts und links vom Vergleichsoperator stehenden Werte ungleich sind, sonst FALSE. Die Datentypen dürfen unterschiedlich sein.

Operator	Beispiele
<	$wert_1 < wert_2$ menge < 10

Liefert TRUE, wenn der linke Wert kleiner als der rechte Wert ist. Bei numerischen Datentypen wird der Zahlenwert als Ordnungskriterium verwendet, bei Texten die festgelegte lexikalische Reihenfolge (siehe Sortierung auf [Abschnitt 5.2.5](#)) und nicht die Länge.

Operator	Beispiele
<=	$wert_1 <= wert_2$ menge <= 10

Kurzschreibweise von: $((wert_1 < wert_2) \text{ OR } (wert_1 = wert_2))$.

Operator	Beispiele
>	$wert_1 > wert_2$ menge > lagerbestand

Liefert TRUE, wenn der linke Wert größer als der rechte Wert ist. Bei numerischen Datentypen wird der Zahlenwert als Ordnungskriterium verwendet, bei Texten die festgelegte lexikalische Reihenfolge (siehe Sortierung auf [Abschnitt 5.2.5](#)) und nicht die Länge.

Operator	Beispiele
>=	$wert_1 \geq wert_2$ menge >= lagerbestand

Kurzschreibweise von: $((wert_1 > wert_2) \text{ OR } (wert_1 = wert_2))$.

Operator	Beispiele
BETWEEN	$wert \text{ BETWEEN } wert_1 \text{ AND } wert_2$ menge BETWEEN 1 AND 100

Liefert TRUE, wenn $wert$ in dem Intervall von $(wert_1, wert_2)$ liegt. Kurzschreibweise von: $((wert \geq wert_1) \text{ AND } (wert \leq wert_2))$.

Operator	Beispiele
NOT BETWEEN	$wert \text{ NOT BETWEEN } wert_1 \text{ AND } wert_2$ menge NOT BETWEEN 1 AND 100

Liefert TRUE, wenn BETWEEN den Wert FALSE liefert, und umgekehrt.

Operator	Beispiele
IN	

	<i>wert</i> IN (<i>wert</i> ₁ , <i>wert</i> ₂ ,...) plz IN (44879, 44877, 44878)

Liefert TRUE, wenn *wert* in der Werteliste (*wert*₁,*wert*₂,...) liegt. Kurzschreibweise von: ((*wert* = *wert*₁) OR (*wert* = *wert*₂) usw.).

Operator	Beispiele
NOT IN	<i>wert</i> NOT IN (<i>wert</i> ₁ , <i>wert</i> ₂ , ...)

Liefert TRUE, wenn IN den Wert FALSE liefert, und umgekehrt.

Operator	Beispiele
IS TRUE	<i>wert</i> IS TRUE bezahlt IS FALSE
IS FALSE	
IS UNKNOWN	

Ermöglicht den Vergleich mit booleschen Werten. Ist der boolesche Wert nicht ermittelbar, z.B. weil *wert* NULL ist, kann dies über den IS UNKNOWN erkannt werden.

Operator	Beispiele
IS NULL	<i>wert</i> IS NULL

Wenn der Inhalt einer Spalte auf NULL überprüft werden soll, können Sie nicht *wert* = NULL schreiben, sondern müssen diese Notation verwenden.

Operator	Beispiele

IS NOT NULL	<i>wert</i> IS NOT NULL

Liefert TRUE, wenn IS NULL den Wert FALSE liefert, und umgekehrt.

Operator	Beispiele
LIKE	' <i>wert</i> ₁ ' LIKE ' <i>wert</i> ₂ '

Liefert TRUE, wenn der Text *wert*₁ dem Text von *wert*₂ ähnlich ist. Die Ähnlichkeit wird mit *Wildcards* bestimmt. Die Wildcard % steht für jedes Zeichen beliebig (auch 0 mal) oft. Die Wildcard _ steht für jedes beliebige Zeichen genau einmal. Für komplexere Abfragen sollten reguläre Ausdrücke verwendet werden.

Operator	Beispiele
NOT LIKE	' <i>wert</i> ₁ ' NOT LIKE ' <i>wert</i> ₂ '

Liefert TRUE, wenn LIKE den Wert FALSE liefert, und umgekehrt.

Operator	Beispiele
MEMBER OF	<i>wert</i> MEMBER OF <i>json_array</i>

Liefert TRUE, wenn der *wert* im *json_array* vorkommt.

Operator	Beispiele
REGEXP	<i>ausdruck</i> REGEXP <i>pattern</i>
RLIKE	<i>ausdruck</i> RLIKE <i>pattern</i>

Liefert TRUE, wenn der *ausdruck* dem *pattern* im Sinne eines *regulären Ausdrucks* entspricht. Die unter MySQL verfügbare Syntax für reguläre Ausdrücke kann unter [\[MyS23\]](#) nachgelesen werden.

Operator	Beispiele
NOT REGEXP NOT RLIKE	<i>ausdruck</i> REGEXP <i>pattern</i> <i>ausdruck</i> RLIKE <i>pattern</i>

Liefert TRUE, wenn REGEXP bzw. RLIKE den Wert FALSE liefern, und umgekehrt.

Operator	Beispiele
SOUNDS LIKE	<i>str_1</i> SOUNDS LIKE <i>str_2</i>

Liefert TRUE, wenn *str_1* den gleichen Soundwert wie *str_2* hat. Dieser Operator liefert nur bei Wörtern englischer Sprache verlässliche Ergebnisse. Leider kann er nicht gut mit Zeichenkodierungen wie Unicode umgehen.

26.3.2 Logikoperatoren

Tabelle 26.11 Wahrheitstabelle der Logikoperatoren

<i>wert1</i>	<i>wert2</i>	NOT <i>wert1</i> ! <i>wert1</i>	<i>wert1</i> AND <i>wert2</i> <i>wert1</i> && <i>wert2</i>	<i>wert1</i> OR <i>wert2</i> <i>wert1</i> <i>wert2</i>	<i>wert1</i> XOR <i>wert2</i>
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN

UNKNOWN	FALSE	UNKNOWN	FALSE	UNKNOWN	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN	TRUE	UNKNOWN
FALSE	UNKNOWN	TRUE	FALSE	UNKNOWN	UNKNOWN
FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
FALSE	TRUE	TRUE	FALSE	TRUE	TRUE
TRUE	UNKNOWN	FALSE	UNKNOWN	TRUE	UNKNOWN
TRUE	FALSE	FALSE	FALSE	TRUE	TRUE
TRUE	TRUE	FALSE	TRUE	TRUE	FALSE

26.3.2.1 NOT, Negation, $\neg$

Die logische Funktion ist ein *unärer* Operator, d.h., er wird nur mit einem Operanden verwendet. Wann immer eine Bedingung logische Werte liefert, kann mit NOT der logische Ausdruck ins Gegenteil umgewandelt werden. Ausnahme ist der Wert UNKNOWN.

So können beispielsweise die ganzen Tests wie IN, LIKE, BETWEEN etc. leicht erweitert werden: NOT IN, NOT LIKE, NOT BETWEEN.

Die Wahrheitstabelle finden Sie in [Tabelle 26.11](#) (Spalte 1 und 3).



Hinweis: Im Allgemeinen wird in SQL wegen der besseren Lesbarkeit das NOT dem Ausrufezeichen ! vorgezogen.

26.3.2.2 AND, Konjunktion, $\wedge$

Mit AND wird überprüft, ob beide Teilbedingungen TRUE sind. Durch Verkettung mit weiteren AND-Teilbedingungen können Sachverhalte abgebildet werden, die mehrere zwingende Voraussetzungen erfüllen müssen.



Hinweis: Im Allgemeinen wird in SQL wegen der besseren Lesbarkeit das AND dem verdoppelten Kaufmannsund && vorgezogen.

Auf [[MyS18a](#)] wird ein Versuchsaufbau beschrieben, der nachweist, dass unter MySQL Sprunglogik implementiert ist, durch die man u.U. Rechenzeit einsparen kann.



Definition 76: Sprunglogik

Kann der Wert einer Gesamtbedingung nicht mehr durch die weitere Auswertung seiner Teilbedingungen verändert werden, wird die Auswertung abgebrochen und der aktuelle Wert für den Gesamtausdruck verwendet. Ein solches Vorgehen wird *Sprunglogik* genannt.

Beispiel: Der erste Wert der Teilbedingung einer AND-Verknüpfung ist FALSE, dann können andere Teilbedingungen den Gesamtausdruck nicht zu TRUE werden lassen.



Hinweis: Bei Sprunglogik kann es zu nicht gewollten Nebeneffekten kommen, wenn in einer nicht mehr berücksichtigten Teilbedingung relevante Operationen nicht mehr ausgeführt werden.

Die Wahrheitstabelle finden Sie in [Tabelle 26.11](#) (Spalte 1, 2 und 4).

26.3.2.3 OR, Disjunktion, $\vee$

Wie oben muss auch hier Sprunglogik (siehe [Definition 76](#)) beachtet werden. Die OR-Verknüpfung wird immer dann verwendet, wenn schon das Erfüllen einer Teilbedingung für die Gesamtbedingung ausreicht.

Die Wahrheitstabelle finden Sie in [Tabelle 26.11](#) (Spalte 1, 2 und 5).

26.3.2.4 XOR, Antivalenz, $\otimes$

Für das XOR gilt folgende Gleichung:

$$\text{wer } t_1 \otimes \text{wer } t_2 = (\text{wer } t_1 \wedge (\neg \text{wer } t_2)) \vee ((\neg \text{wer } t_1) \wedge \text{wer } t_2)$$

Umgangssprachlich kann man ein XOR mit einer *Entweder-oder*-Aussage umschreiben. Nur eine der beiden Teilbedingungen darf TRUE sein, um den Gesamtausdruck TRUE werden zu lassen.

Die Wahrheitstabelle finden Sie in [Tabelle 26.11](#) (Spalte 1, 2 und 6).

26.3.2.5 Verallgemeinerung auf mehr als zwei Operanden

Anders als bei der AND- und OR-Verknüpfung ist eine Verallgemeinerung der XOR-Verknüpfung auf mehr als zwei Operanden/Eingangswerte nicht so einfach. Bei der AND-Verknüpfung reicht *ein* FALSE und das Ergebnis ist immer FALSE. Bei der OR-Verknüpfung reicht *ein* TRUE und das Ergebnis ist immer TRUE.

Bei der XOR-Verknüpfung ist das Ergebnis dann TRUE, wenn die Anzahl der TRUE-Werte ungerade ist – also $\text{MOD } 2 = 1$. Sie können dies ja mal selbst anhand einer Wahrheitstabelle mit drei oder vier Operanden überprüfen.

26.4 Befehle

Die hier vorgestellte Referenz orientiert sich am aktuellen MySQL-Sprachstandard (Version 8.0). Es ist mir nicht gelungen, Ihnen hier eine lesbare und effiziente Gegenüberstellung der Sprachstandards zur Verfügung zu stellen. In den vorherigen Kapiteln ist aber bei vielen Befehlen der Unterschied dargestellt worden, sodass ich mich hiermit Verweisen begnüge.

Laut der Dokumentation von MariaDB (siehe [[Mar16](#)]) sind die Befehle zwischen MySQL 5.6 und MariaDB 10.0 alle kompatibel und können problemlos ausgetauscht werden; einen Abgleich von MariaDB 11.0 mit MySQL 8.0 finden Sie hier: [[Mar23a](#)].

26.4.1 Data Definition Language

Die Data Definition Language (DDL) hat die Aufgabe, die Struktur eines Servers, einer Datenbank, einer Tabelle etc. festzulegen, nicht den Inhalt.

ALTER DATABASE	Eigenschaften einer Datenbank ändern

```
ALTER {DATABASE|SCHEMA} datenbankname
[[DEFAULT] CHARACTER SET [=] zeichensatz]
[[DEFAULT] COLLATE [=] sortierung]
[DEFAULT ENCRYPTION [=] {'Y'|'N'}]
;
```

Ändert die Spezifikation einer Datenbank. Den Datenbanknamen kann man nicht ändern, dies erreicht man nur durch Löschen und Neuanlage. Wird die Verschlüsselung geändert, werden nur neu angelegte Tabellen mit dem geänderten Verfahren verschlüsselt. Vorhandene Daten werden nicht migriert. Weitere Hinweise: siehe CREATE SCHEMA in [Abschnitt 26.4.1](#) und [Abschnitt 8.1](#).

ALTER EVENT	Eigenschaften eines Events ändern

```
ALTER
[DEFINER = user]
EVENT eventname
[ON SCHEDULE termin]
[ON COMPLETION [NOT] PRESERVE]
[RENAME TO eventname_neu]
[ENABLE|DISABLE|DISABLE ON SLAVE]
[COMMENT 'kommentar']
[DO anweisungsblock]
;
```

Ändert die Spezifikation eines Ereignisses. Ebenfalls kann der Anweisungsblock verändert werden. Weitere Hinweise siehe CREATE EVENT in [Abschnitt 26.4.1.](#)

ALTER FUNCTION	Eigenschaften einer Funktion ändern

```
ALTER FUNCTION funktionsname
COMMENT 'kommentar'
| LANGUAGE SQL
| {CONTAINS SQL|NO SQL|READS SQL DATA|MODIFIES
SQL DATA}
| SQL SECURITY {DEFINER|INVOKER}
;
```

Hinweise siehe CREATE FUNCTION in [Abschnitt 26.4.1.](#)

ALTER INSTANCE	Eigenschaften der MySQL-Instanz ändern

```
ALTER INSTANCE
ROTATE INNODB MASTER KEY
| ROTATE BINLOG MASTER KEY
| RELOAD TLS [NO ROLLBACK ON ERROR]
;
```

Die Option ALTER INSTANCE ROTATE INNODB MASTER KEY löst aus, dass der Verschlüsselungsschlüssel der InnoDB-Tablespaces verändert wird.

Die Option ALTER INSTANCE ROTATE BINLOG MASTER KEY löst aus, dass der Verschlüsselungsschlüssel des Binärlogs verändert wird.

Beide Optionen verwenden einen Schlüsselring als Basis der Veränderung.

Mit `ALTER INSTANCE RELOAD TLS` werden Systemvariablen der SSL-Umgebung neu eingelesen. Mit der `NO ROLLBACK ON ERROR`-Option wird gesteuert, ob bei einem Fehler die Änderungen rückgängig gemacht werden oder nicht.

ALTER LOGFILE GROUP	Eigenschaften einer Logdateigruppe ändern

```
ALTER LOGFILE GROUP logdateigruppe
ADD UNDOFILE 'dateiname'
[INITIAL_SIZE [=] größe]
[WAIT]
ENGINE [=] engine
;
```

Hinweise siehe `CREATE LOGFILE GROUP` in [Abschnitt 26.4.1.](#)

ALTER PROCEDURE	Eigenschaften einer Prozedur ändern

```
ALTER PROCEDURE prozedurname
COMMENT 'kommentar'
| LANGUAGE SQL
| {CONTAINS SQL|NO SQL|READS SQL DATA|MODIFIES
SQL DATA}
| SQL SECURITY {DEFINER|INVOKER}
;
```

Ändert die Spezifikation einer Prozedur, aber nicht den Anweisungsblock. Dieser muss mit `CREATE OR REPLACE` verändert werden. Weitere Hinweise siehe `CREATE PROCEDURE` in [Abschnitt 26.4.1.](#)

ALTER SERVER	Eigenschaften eines Servers ändern

```
ALTER SERVER servername
OPTIONS (option [, option]*)
;
```

Hinweise siehe CREATE SERVER in [Abschnitt 26.4.1](#).

ALTER TABLE	Eigenschaften einer Tabelle ändern

```
ALTER [ONLINE|OFFLINE] [IGNORE] TABLE tabellenname
[tabellenspezifikation [, tabellenspezifikation]*] [partitionsoption]
;
ALTER [ONLINE|OFFLINE] [IGNORE] TABLE tabellenname
partitionsoption
;
tabellenspezifikation:
tabellenoption
ADD [COLUMN] spaltenname spaltendefinition [FIRST|AFTER spaltenname]
|ADD {INDEX|KEY} [indexname][indextyp]
(indexspaltenname [, indexspaltenname]* ) [indexoption]*
|ADD {FULLTEXT|SPATIAL} [INDEX|KEY] [indexname]
(indexspaltenname [, indexspaltenname]* ) [indexoption]*
|ADD [CONSTRAINT [name]] PRIMARY KEY [indextyp]
(indexspaltenname [, indexspaltenname]* ) [indexoption]*
|ADD [CONSTRAINT [name]] UNIQUE [INDEX|KEY] [indexname] [indextyp]
(spaltenname [, spaltenname]* ) [indexoption]*
|ADD FULLTEXT [INDEX|KEY] [indexname]
```

```

(indexspaltenname[, indexspaltenname]*) [index
option]*
|ADD [CONSTRAINT [name]] FOREIGN KEY [indexnam
e]
(indexspaltenname[, indexspaltenname]*) refere
nz
|DROP CHECK checkname
|ALTER CHECK checkname [NOT] ENFORCED
|ALGORITHM [=] {DEFAULT|INSTANT|INPLACE|COPY}
|ALTER [COLUMN] spaltenname {SET DEFAULT liter
al|DROP DEFAULT}
|ALTER INDEX indexname {VISIBLE|INVISIBLE}
|LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
|CHANGE [COLUMN] spaltenname_alt
spaltenname_neu spaltendefinition [FIRST|AFTER
spaltenname]
|MODIFY [COLUMN] spaltenname spaltendefinition
[FIRST|AFTER spaltenname]
|DROP [COLUMN] spaltenname
|DROP PRIMARY KEY
|DROP {INDEX|KEY} indexname
|DROP FOREIGN KEY fk_name
|{DISABLE|ENABLE} KEYS
|MAX_ROWS = anzahl
|RENAME [TO|AS] tabellenname_neu
|RENAME {COLUMN|INDEX|KEY} name_alt TO name_ne
u
|ORDER BY spaltenname [, spaltenname]*
|CONVERT TO CHARACTER SET Zeichensatz [COLLATE
sortierung]
|[DEFAULT] CHARACTER SET [=] Zeichensatz [COLL
ATE [=] sortierung]
|{IMPORT|DISCARD} TABLESPACE
|FORCE
|{WITHOUT|WITH} VALIDATION
|ADD PARTITION (partitionsdefinition)
|DROP PARTITION partitionsname
|DISCARD PARTITION {partitionsnamen|ALL} TABLE

```

```

SPACE
|IMPORT PARTITION {partitionsnamen|ALL} TABLES
PACE
|TRUNCATE PARTITION {partitionsname|ALL}
|COALESCE PARTITION nummer
|REORGANIZE PARTITION partitionsname INTO (par
titionsdefinition)
|ANALYZE PARTITION {partitionsname|ALL}
|CHECK PARTITION {partitionsname|ALL}
|OPTIMIZE PARTITION {partitionsname|ALL}
|REBUILD PARTITION {partitionsname|ALL}
|REPAIR PARTITION {partitionsname|ALL}
|PARTITION BY partitionierungsausdruck
|REMOVE PARTITIONING
|UPGRADE PARTITIONING

```

```

indexspaltenname:
{spaltenname[(länge)|(ausdruck)] [ASC|DESC]
indextyp:
USING {BTREE|HASH}
indexoption:
KEY_BLOCK_SIZE [=] wert
|indextyp
|WITH_PARSER parsername
|COMMENT 'kommentar'
tabellenoption:
siehe CREATE TABLE

```

partitionsdefinition:

siehe CREATE TABLE

Ändert fast alle Einstellungen, die man bei einer Tabelle machen kann. Weitere Hinweise siehe CREATE TABLE in [Abschnitt 26.4.1](#) und [Abschnitt 8.3](#).

ALTER TABLESPACE	Eigenschaften eines Tablespace ändern


```

ALTER [UNDO] TABLESPACE name
nur bei NDB:
{ADD|DROP} DATAFILE 'dateiname'
[INITIAL_SIZE [=] größe]
[WAIT]
bei InnoDB und NDB:
[RENAME TO name]
nur bei InnoDB:
[SET {ACTIVE|INACTIVE}]
[ENCRYPTION [=] {'Y'|'N'}]
bei InnoDB und NDB:
[ENGINE [=] engine]
;

```

Hinweise siehe CREATE TABLESPACE in [Abschnitt 26.4.1.](#)

ALTER VIEW	Eigenschaften einer Ansicht ändern

```

ALTER
[ALGORITHM = {UNDEFINED|MERGE|TEMPTABLE}]
[DEFINER = user]
[SQL SECURITY {DEFINER|INVOKER}]
VIEW name [(spaltenname[, spaltenname]*)]
AS
SELECT auswahl
[WITH [CASCADED|LOCAL] CHECK OPTION]
;

```

Ändert die Spezifikation und die Auswahl der Ansicht. Weitere Hinweise siehe CREATE VIEW in [Abschnitt 26.4.1.](#)

CREATE DATABASE	Anlegen einer Datenbank

```

CREATE {DATABASE|SCHEMA} [IF NOT EXISTS] daten
bankname

```

```

[[DEFAULT] CHARACTER SET [=] zeichensatz]
[[DEFAULT] COLLATE [=] sortierung]
[DEFAULT ENCRYPTION [=] {'Y'|'N'}]
;

```

Legt eine neue Datenbank an. Weitere Hinweise siehe [Abschnitt 5.2.2](#), zum Löschen DROP SCHEMA in [Abschnitt 26.4.1](#) und zum Ändern ALTER DATABASE in [Abschnitt 26.4.1](#).

CREATE EVENT	Anlegen eines zeitlichen Ereignisses

```

CREATE
[DEFINER = user]
EVENT [IF NOT EXISTS] name
ON SCHEDULE termin
[ON COMPLETION [NOT] PRESERVE]
[ENABLE|DISABLE|DISABLE ON SLAVE]
[COMMENT 'kommentar']
DO
anweisungsblock
;
termin:
AT zeitpunkt [+ INTERVAL intervall]*
|EVERY intervall
[STARTS zeitpunkt [+ INTERVAL intervall]*]
[ENDS zeitpunkt [+ INTERVAL intervall]*]

intervall:
{YEAR|QUARTER|MONTH|WEEK|DAY|HOUR|MINUTE|SECON
D
|YEAR_MONTH|DAY_HOUR|DAY_MINUTE
|DAY_SECOND|HOUR_MINUTE|HOUR_SECOND|MINUTE_SEC
OND}

```

Legt ein zeitgesteuertes Ereignis an. Weitere Hinweise: siehe

[Abschnitt 23.1](#), zum Löschen DROP EVENT in [Abschnitt 26.4.1](#) und zum Ändern ALTER EVENT in [Abschnitt 26.4.1](#).

CREATE FUNCTION	Anlegen einer Funktion

```
CREATE
[DEFINER = user]
FUNCTION name ([parameter][, parameter]*)
RETURNS typ
[option]* anweisungsblock
;
parameter:
[IN|OUT|INOUT] name typ
typ:
jeder gültigeMySQL Datentyp
option:
COMMENT 'kommentar'
| LANGUAGE SQL
|[NOT] DETERMINISTIC
|{CONTAINS SQL|NO SQL|READS SQL DATA|MODIFIES
SQL DATA}
|SQL SECURITY {DEFINER|INVOKER}
```

anweisungsblock:

gültige Deklarationen und Anweisungen

Hinweise siehe [Abschnitt 26.2](#), zum Löschen DROP FUNCTION in [Abschnitt 26.4.1](#) und zum Ändern ALTER FUNCTION in [Abschnitt 26.4.1](#).

CREATE INDEX	Anlegen eines Index

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX name
[index_typ]
ON tabellenname (indexspaltenname[, indexspaltenname]*)
```

```

[option]*
[algorithmus_typ]*
[lock_typ]*
;
indexspaltenname:
{spaltenname[(länge)|(ausdruck)]} [ASC|DESC]
index_typ:
USING {BTREE|HASH}
option:
KEY_BLOCK_SIZE [=] wert |index_typ
|WITH PARSEr parsername
|COMMENT 'kommentar'
|{VISIBLE|INVISIBLE}
algorithmus_typ:
ALGORITHM [=] {DEFAULT|INPLACE|COPY}
lock_typ:
LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}

```

Legt einen INDEX an. Ein Index kann nicht verändert werden. Er muss gelöscht und neu angelegt werden. Weitere Hinweise siehe [Abschnitt 6.1.2](#) und zum Löschen DROP INDEX in [Abschnitt 26.4.1](#).

CREATE LOGFILE GROUP	Anlegen einer Logdateigruppe

```

CREATE LOGFILE GROUP logdateigruppe
ADD UNDOFILE 'dateiname'
[INITIAL_SIZE [=] startgröße]
[UNDO_BUFFER_SIZE [=] puffergröße_rückgängig]
[REDO_BUFFER_SIZE [=] puffergröße_wiederholen]
[NODEGROUP [=] knotengruppe_id]
[WAIT]
[COMMENT [=] kommentar]
ENGINE [=] engine
;

```

Anlegen einer Logdateigruppe, zu denen die Log- oder Datendateien gehören. Diese Gruppe enthält die *Undo*-Datei `dateiname4`. Logdateigruppen werden im Zusammenhang mit Clustern verwendet. MySQL unterstützt nur die Engines NDB und NDBCLUSTER. Weitere Hinweise siehe zum Löschen `DROP LOGFILE GROUP` in [Abschnitt 26.4.1](#) und zum Ändern `ALTER LOGFILE GROUP` in [Abschnitt 26.4.1](#).

CREATE PROCEDURE	Anlegen einer Prozedur

```
CREATE
[DEFINER = user]
PROCEDURE name ([parameter] [, parameter]* )
[option]* anweisungsblock
;
parameter:
[IN|OUT|INOUT] name typ
typ:
jeder gültigeMySQL Datentyp (siehe Abschnitt 26.1)
option:
COMMENT 'kommentar'
| LANGUAGE SQL
| [NOT] DETERMINISTIC
| {CONTAINS SQL|NO SQL|READS SQL DATA|MODIFIES SQL DATA}
| SQL SECURITY {DEFINER|INVOKER}
anweisungsblock:
gültige Deklarationen und Anweisungen
```

Hinweise siehe [Kapitel 20](#) und zum Löschen `DROP PROCEDURE` in [Abschnitt 26.4.1](#).

CREATE SCHEMA	Anlegen einer Datenbank

Siehe `CREATE DATABASE` in [Abschnitt 26.4.1](#).

CREATE SERVER

Anlegen eines neuen Servers

```
CREATE SERVER server_name
FOREIGN DATA WRAPPER wrappername
OPTIONS (option [, option]* )
;
option:
{HOST 'hostname'
|DATABASE 'datenbankname'
|USER 'username'
|PASSWORD 'passwort'
|SOCKET 'dateiname'
|OWNER 'username'
|PORT nummer}
```

Erstellt einen Server für die Engine FEDERATED. Damit können Datenbanken anderer MySQL Server (Engines: MyISAM oder InnoDB) so eingerichtet werden, als ob diese lokal wären.

```
CREATE SERVER server_106_test
FOREIGN DATA WRAPPER mysql
OPTIONS (
USER 'root',
HOST '192.168.1.106',
DATABASE 'test'
)
;
```

Auf dieser Serverbindung können nun weitere Befehle abgesetzt werden:

```
CREATE TABLE fremdetabelle ( s1 INT
) ENGINE=FEDERATED CONNECTION='server_106_test'
;
```

... ..

Weitere Hinweise siehe zum Löschen DROP SERVER in [Abschnitt 26.4.1](#) und zum Ändern ALTER SERVER in [Abschnitt 26.4.1](#).

CREATE SPATIAL REFERENCE SYSTEM	Anlegen eines räumlichen Referenzsystems

```
CREATE OR REPLACE SPATIAL REFERENCE SYSTEM
referrenzid attribut[, attribute]*
CREATE SPATIAL REFERENCE SYSTEM [IF NOT EXISTS
]
referrenzid attribut[, attribute]*
attribut: {
NAME name
| DEFINITION definition
| ORGANIZATION name IDENTIFIED BY organisations
id
| DESCRIPTION beschreibung
}
```

referrenzid, organisationsid: 32-bit unsigned integer

CREATE TABLE	Anlegen einer Tabelle

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tabel
lename
(spalte_eigenschaft[, spalte_eigenschaft]*)
[tabellenoptionen]
[partitionsoptionen]
;
spalte_eigenschaft:
spaltenname spaltendefinition
| {INDEX|KEY} [indexname] [indextyp]
(indexspaltenname[, indexspaltenname]*) [ index
option]* | {FULLTEXT|SPATIAL} [INDEX|KEY] [inde
xname]
(indexspaltenname[, indexspaltenname]*) [ index
```

```

option]* | [CONSTRAINT [name]] PRIMARY KEY [index
typ]
(indexspaltenname[, indexspaltenname]*) [ index
option]*
|[CONSTRAINT [name]] UNIQUE [INDEX|KEY] [index
name] [indextyp] (indexspaltenname[, indexspal
tenname]*) [ indexoption]*
|[CONSTRAINT [name]] FOREIGN KEY [indexname]
(indexspaltenname[, indexspaltenname]*) refere
nz |check_definition
spaltendefinition: datentyp
[NOT NULL|NULL] [DEFAULT vorbelegung]
[AUTO_INCREMENT] [UNIQUE [KEY]] [[PRIMARY] KEY
]
[COMMENT kommentar]
[COLLATE sortierung]
[COLUMN_FORMAT {FIXED|DYNAMIC|DEFAULT}]
[STORAGE {DISK|MEMORY}] [referenz]
[check_definition] |datentyp
[COLLATE sortierung]
[GENERATED ALWAYS] AS (ausdruck)
[VIRTUAL|STORED] [NOT NULL|NULL]
[UNIQUE [KEY]] [[PRIMARY] KEY]
[COMMENT kommentar]
[referenz]
[check_definition]
datentyp:
siehe Abschnitt 26.1
indexspaltenname:
{spaltenname[(länge)|(ausdruck)] [ASC|DESC]
indextyp:
USING BTREE|HASH
indexoption:
KEY_BLOCK_SIZE [=] wert
|indextyp
|WITH PARSER parsername
|COMMENT kommentar
|{VISIBLE|INVISIBLE}

```


referenz:

REFERENCES *tabellenname* (*spaltenname*[, *spaltenname*]*)

[MATCH FULL|MATCH PARTIAL|MATCH SIMPLE]

[ON DELETE *referenzoption*]

[ON UPDATE *referenzoption*]

referenzoption:

RESTRICT|CASCADE|SET NULL|NO ACTION|SET DEFAULT

tabellenoptionen:

tabellenoption [[,] *tabellenoption*]*

tabellenoption:

ENGINE [=] *engine*

|AUTO_INCREMENT [=] *wert*

|AVG_ROW_LENGTH [=] *wert*

|[DEFAULT] CHARACTER SET [=] *zeichensatz*

|CHECKSUM [=] {0|1}

|[DEFAULT] COLLATE [=] *sortierung*

|COMMENT [=] *kommentar*

|COMPRESSION [=] {'ZLIB'|'LZ4'|'NONE'}

|CONNECTION [=] *verbindungsparameter*

|{DATA|INDEX} DIRECTORY [=] *absoluter pfad*

|DELAY_KEY_WRITE [=] {0|1}

|ENCRYPTION [=] {'Y'|'N'}

|INSERT_METHOD [=] {NO|FIRST|LAST}

|KEY_BLOCK_SIZE [=] *wert*

|MAX_ROWS [=] *wert*

|MIN_ROWS [=] *wert*

|PACK_KEYS [=] {0|1|DEFAULT}

|PASSWORD [=] *passwort*

|ROW_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}

|STATS_AUTO_RECALC [=] {DEFAULT|0|1}

|STATS_PERSISTENT [=] {DEFAULT|0|1}

|STATS_SAMPLE_PAGES [=] *wert*

|TABLESPACE *tablespacename* [STORAGE {DISK|MEMORY}]

|UNION [=] (*tabellenname*[, *tabellenname*]*)

partitionsoptionen:

PARTITION BY

```
{ [LINEAR] HASH(ausdruck)  
| [LINEAR] KEY [ALGORITHM={1|2}](spaltenliste)  
| RANGE{(ausdruck)|COLUMNS(spaltenliste)}  
| LIST{(ausdruck)|COLUMNS(spaltenliste)} }  
[PARTITIONS anzahl]  
[SUBPARTITION BY  
{[LINEAR] HASH(ausdruck)  
|[LINEAR] KEY [ALGORITHM={1|2}](spaltenliste)}  
[SUBPARTITIONS anzahl]  
(partitionsdefinition [, partitionsdefinition]  
*)]
```

partitionsdefinition:

PARTITION *partitionsname*

```
[VALUES  
{LESS THAN {(ausdruck|werteliste)|MAXVALUE}  
|  
IN (werteliste)}]  
[[STORAGE] ENGINE [=] engine]  
[COMMENT [=] kommentar]  
[DATA DIRECTORY [=] datenverzeichnis]  
[INDEX DIRECTORY [=] indexverzeichnis]  
[MAX_ROWS [=] anzahl]  
[MIN_ROWS [=] anzahl]  
[TABLESPACE [=] tablespacename]  
[(subpartitionsdefinition [, subpartitionsdefi  
nition]*)]
```

subpartitionsdefinition:

SUBPARTITION *name*

```
[[STORAGE] ENGINE [=] engine]  
[COMMENT [=] kommentar]  
[DATA DIRECTORY [=] datenverzeichnis]  
[INDEX DIRECTORY [=] indexverzeichnis]  
[MAX_ROWS [=] anzahl]  
[MIN_ROWS [=] anzahl]  
[TABLESPACE [=] tablespacename]
```

Weitere Hinweise siehe [Abschnitt 5.3](#), zum Ändern ALTER TABLE in [Abschnitt 26.4.1](#) und zum Löschen DROP TABLE in [Abschnitt 26.4.1](#).

CREATE TABLE .. .SELECT	SELECT-Ergebnis als Tabelle speichern

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tabellenname
(spalte_eigenschaft[, spalte_eigenschaft]* )
[tabellenoptionen]
[partitionsoptionen]
selectangabe
;
spalte_eigenschaft:
Siehe CREATE TABLE in Abschnitt 26.4.1
tabellenoptionen:
Siehe CREATE TABLE in Abschnitt 26.4.1
partitionsoptionen:
Siehe CREATE TABLE in Abschnitt 26.4.1
selectangabe:
[IGNORE|REPLACE] [AS] SELECT auswahl
Legt eine neue Tabelle anhand des Ergebnisses
einer Auswahl mit SELECT an.
```

CREATE TABLE .. .LIKE	Tabellenstruktur kopieren

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tabellenname_ziel {LIKE tabellenname_quelle|(LIKE tabellenname_quelle)}
;
```

Legt eine neue leere Tabelle *tabellenname_ziel* mit den gleichen Spezifikationen wie *tabellenname_quelle* an. Weitere

Hinweise siehe [Abschnitt 5.3.7](#).

CREATE TABLESPACE	Anlegen eines neuen Tablespaces

```
CREATE [UNDO] TABLESPACE tablespacename
bei InnoDB und NDB:
ADD DATAFILE 'dateiname'
nur bei InnoDB:
[FILE_BLOCK_SIZE = wert]
[ENCRYPTION [=] {'Y'|'N'}]
nur bei InnoDB:
USE LOGFILE GROUP logfilegruppe
[EXTENT_SIZE [=] erweiterungsgröße]
[INITIAL_SIZE [=] startgröße]
[AUTOEXTEND_SIZE [=] automatische_erweiterungsgröße]
[MAX_SIZE [=] maximalgröße]
[NODEGROUP [=] knotengruppe_id]
[WAIT]
[COMMENT [=] kommentar]
bei InnoDB und NDB:
ENGINE [=] engine
;
```

Anlegen eines festen Speicherplatzes für Tabellen. Normalerweise wird der Speicherplatz vom Dateisystem automatisch verwaltet. Legt man den Tablespace manuell an, erweitert der sich nicht mehr automatisch, ist aber erheblich schneller im Zugriff. Weitere Hinweise siehe zum Ändern ALTER TABLESPACE in [Abschnitt 26.4.1](#) und zum Löschen DROP TABLESPACE in [Abschnitt 26.4.1](#).

CREATE TRIGGER	Anlegen eines Triggers

CREATE

```
[DEFINER = user]
TRIGGER name {BEFORE|AFTER} {INSERT|UPDATE|DELETE}
ON tabellenname FOR EACH ROW
[{{FOLLOWS|PRECEDES} ein_anderer_triggername}]
anweisungsblock
```

Hinweise siehe [Kapitel 22](#) und zum Löschen DROP TRIGGER in [Abschnitt 26.4.1](#).

CREATE VIEW	Anlegen einer Ansicht

```
CREATE
[OR REPLACE]
[ALGORITHM = {UNDEFINED|MERGE|TEMPTABLE}]
[DEFINER = user]
[SQL SECURITY {DEFINER|INVOKER}]
VIEW name [(spaltenname[, spaltenname]*)]
AS SELECT auswahl
[WITH [CASCADED|LOCAL] CHECK OPTION]
;
```

Hinweise siehe [Kapitel 16](#), zum Ändern ALTER VIEW in [Abschnitt 26.4.1](#) und zum Löschen DROP VIEW in [Abschnitt 26.4.1](#).

DROP DATABASE	Löscht die Datenbank und alle ihre Elemente

```
DROP DATABASE [IF EXISTS] name;
```

Alle Datenobjekte wie Tabellen, Ansichten etc. werden ohne Rückfrage endgültig gelöscht. Weitere Hinweise siehe CREATE DATABASE in [Abschnitt 26.4.1](#).

DROP EVENT	Löscht ein zeitgesteuertes Ereignis

`DROP EVENT [IF EXISTS] name;`

Hinweise siehe CREATE EVENT in [Abschnitt 26.4.1.](#)

DROP FUNCTION	Löscht eine selbsterstellte Funktion

`DROP FUNCTION [IF EXISTS] name;`

Hinweise siehe CREATE FUNCTION in [Abschnitt 26.4.1.](#)

DROP INDEX	Löscht einen Index

```
DROP INDEX name ON tabellenname
[ALGORITHM [=] {DEFAULT|INPLACE|COPY}
| LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}]
;
```

Hinweise siehe CREATE INDEX in [Abschnitt 26.4.1.](#)

DROP LOGFILE GROUP	Löscht eine Logdateigruppe

`DROP LOGFILE GROUP logdateigruppe ENGINE [=] engine;`

Hinweise siehe CREATE LOGFILE GROUP in [Abschnitt 26.4.1.](#)

DROP PROCEDURE	Löscht eine Prozedur

DROP PROCEDURE [IF EXISTS] *name*;

Hinweise siehe CREATE PROCEDURE in [Abschnitt 26.4.1.](#)

DROP SCHEMA	Löscht die Datenbank und alle ihre Elemente

DROP SCHEMA [IF EXISTS] *name*;

Alle Datenobjekte wie Tabellen, Ansichten etc. werden ohne Rückfrage endgültig gelöscht. Weitere Hinweise siehe CREATE SCHEMA in [Abschnitt 26.4.1.](#)

DROP SERVER	Löscht einen FEDERATED Server

DROP SERVER [IF EXISTS] *name*;

Hinweise siehe CREATE SERVER in [Abschnitt 26.4.1.](#)

DROP SPATIAL REFERENCE SYSTEM	Löscht ein räumliches Referenzsystem

DROP SPATIAL REFERENCE SYSTEM [IF EXISTS] *referenzid*;
referenzid: 32-bit unsigned integer

Hinweise siehe CREATE SPATIAL REFERENCE SYSTEM in [Abschnitt 26.4.1.](#)

DROP TABLE	Löscht eine Tabelle

DROP [TEMPORARY] TABLE [IF EXISTS]
tabellenname [, *tabellenname*]*
[RESTRICT|CASCADE]

;

Alle Daten der Tabelle werden ohne Rückfrage endgültig gelöscht. Weitere Hinweise siehe CREATE TABLE in [Abschnitt 26.4.1.](#)

DROP TABLESPACE	Löscht einen Tablespace

```
DROP [UNDO] TABLESPACE name ENGINE [=] engine;
```

Hinweise siehe CREATE TABLESPACE in [Abschnitt 26.4.1.](#)

DROP TRIGGER	Löscht einen Trigger

```
DROP TRIGGER [IF EXISTS] [datenbankname.] name;
```

Hinweise siehe CREATE TRIGGER in [Abschnitt 26.4.1.](#)

DROP VIEW	Löscht eine Ansicht

```
DROP VIEW [IF EXISTS] name [, name]* [RESTRICT|  
CASCADE]; Hinweise siehe CREATE VIEW in Abschnitt 26.4.1.
```

RENAME TABLE	Ändert einen Tabellennamen

```
RENAME  
TABLE name_alt TO name_neu [, TABLE name_alt T  
O name_neu]*;
```

Hinweise siehe CREATE TABLE in [Abschnitt 26.4.1.](#)

TRUNCATE	Setzt eine Tabelle zurück


```
TRUNCATE [TABLE] tabellenname;
```

Versetzt die Tabelle in den Startzustand zurück. Die Daten werden gelöscht und der ggf. vorhandene AUTO_INCREMENT-Zähler wird auf 1 gesetzt. Weitere Hinweise siehe [Abschnitt 9.3.4.](#)

26.4.2 Data Manipulation Language

Mithilfe der Befehle der Data Manipulation Language (DML) werden die Tabelleninhalte ausgewertet oder verändert.

CALL	Aufruf einer Prozedur

```
CALL prozedurname([parameter[, parameter]*]);
```

Die Prozedur mit dem Namen *prozedurname* wird mit den angegebenen Parametern aufgerufen. Weitere Hinweise siehe [Kapitel 20.](#)

DELETE	Inhalte einer einer Tabelle löschen

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tabellenname [[AS] alias]  
[PARTITION (partitionsname[, partitionsname]*)]  
[WHERE bedingung]  
[ORDER BY sortierangabe]  
[LIMIT anzahl]  
;
```

Löscht *anzahl* viele Zeilen in der Reihenfolge der *sortierangabe* der Tabelle *tabellenname*, für welche die *bedingung* den Wert

TRUE ergibt. Weitere Hinweise siehe [Abschnitt 9.3.](#)

DELETE	Inhalt einer Tabellenreferenz löschen

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
tabellenname[.*] [, Tabellenname[.*]]*
FROM Tabellenreferenz
[WHERE bedingung]
;
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
FROM Tabellenname[.*] [, Tabellenname[.*]]*
USING Tabellenreferenz
[WHERE bedingung]
;
```

Hinweise siehe DELETE in [Abschnitt 26.4.2.](#)

DO	Ausdrücke auswerten

```
DO ausdrück[, ausdrück]*;
```

Mit DO wird der Ausdruck ausgeführt, ohne dass ggf. ermittelte Ergebnisse zurückgeliefert werden. Es handelt sich dabei um eine meist etwas schnellere Ausführung als bei einem SELECT. Fehler werden als Warnung ausgegeben.

INSERT INTO	Zeilen mit VALUES einfügen

```
INSERT [LOW_PRIORITY|HIGH_PRIORITY] [IGNORE]
[INTO] Tabellenname
[PARTITION (partitionsname[, partitionsname]*)]
[(spaltenname[, spaltenname]*)]
{VALUES|VALUE}
```

```
({ausdruck|DEFAULT}[ , {ausdruck|DEFAULT}*])
[ , ({ausdruck|DEFAULT}[ , {ausdruck|DEFAULT}*])
]*
[ON DUPLICATE KEY UPDATE
spaltenname=ausdruck
[ , spaltenname=ausdruck]*]
;
```

Fügt die Werte hinter VALUES als neue Zeilen in die Tabelle *tabellenname* ein. Wird dabei ein schon vorhandener Schlüssel verwendet, kann man nach ON DUPLICATE KEY UPDATE angeben, wie damit verfahren werden soll.

Diese Version eignet sich gut für das Einfügen vieler Datensätze mit einem Befehl. Weitere Hinweise siehe [Abschnitt 7.2.1.](#)

INSERT INTO	Zeilen mit SET einfügen

```
INSERT [LOW_PRIORITY|HIGH_PRIORITY] [IGNORE]
[INTO] tabellenname
[PARTITION (partitionsname[ , partitionsname]* )
]
SET spaltenname={ausdruck|DEFAULT}[ , spaltenname={ausdruck|DEFAULT}]*
[ON DUPLICATE KEY UPDATE
spaltenname=ausdruck
[ , spaltenname=ausdruck]*]
;
```

Fügt eine neue Zeile in die Tabelle *tabellenname* ein. Dabei werden den Spaltennamen hinter dem SET die Werte zugewiesen. Wird dabei ein schon vorhandener Schlüssel verwendet, kann man nach ON DUPLICATE KEY UPDATE angeben, wie damit verfahren werden soll.

Diese Version eignet sich gut für das Einfügen vieler

Diese Version eignet sich gut für das Einfügen vieler Datensätze mit einem Befehl. Weitere Hinweise siehe [Abschnitt 7.2.2.](#)

INSERT INTO	Zeilen mit SELECT einfügen

```
INSERT [LOW_PRIORITY|HIGH_PRIORITY] [IGNORE]
[INTO] tabellenname
[PARTITION (partitionsname[, partitionsname]* )
]
SELECT auswahl
[ON DUPLICATE KEY UPDATE
spaltenname=ausdruck
[, spaltenname=ausdruck]*]
;
```

Fügt das Ergebnis der *auswahl* als neue Zeilen der Tabelle *tabellenname* hinzu. Wird dabei ein schon vorhandener Schlüssel verwendet, kann man nach ON DUPLICATE KEY UPDATE angeben, wie damit verfahren werden soll.

Diese Version eignet sich gut für das Einfügen vieler Datensätze mit einem Befehl. Weitere Hinweise siehe [Abschnitt 7.3.](#)

LOAD DATA	Daten aus einer CSV-Datei einlesen

```
LOAD DATA [LOW_PRIORITY|CONCURRENT] [LOCAL] IN
FILE 'dateiname'
[REPLACE|IGNORE]
INTO TABLE tabellenname
[PARTITION (partitionsname[, partitionsname]* )
]
[CHARACTER SET zeichensatz]
[{FIELDS|COLUMNS}
```

```

[TERMINATED BY 'zeichenkette']
[[OPTIONALLY] ENCLOSED BY 'zeichen']
[ESCAPED BY 'zeichen']
LINES
[STARTING BY 'zeichenkette']
[TERMINATED BY 'zeichenkette']
IGNORE anzahl {LINES|ROWS}
[({spaltennamen|ausdruck} [{spaltennamen|ausdruck}]*)]
[SET spaltenname=ausdruck [, spaltenname=ausdruck]*]
;

```

Lädt die Daten aus einer CSV-Datei in eine Tabelle. Weitere Hinweise siehe [Abschnitt 7.1](#).

LOAD XML	Daten aus einer XML-Datei einlesen

```

LOAD XML [LOW_PRIORITY|CONCURRENT] [LOCAL] INF
ILE 'dateiname'
[REPLACE|IGNORE]
INTO TABLE [datenbankname.]tabellenname
[PARTITION (partitionsname [, partitionsname]*)]
]
[CHARACTER SET zeichensatz]
[ROWS IDENTIFIED BY '<tagname>']
[IGNORE anzahl [LINES|ROWS]]
[({spaltennamen|ausdruck} [{spaltennamen|ausdruck}]*)]
[SET spaltenname=ausdruck [, spaltenname=ausdruck]*]
;

```

Lädt Daten aus einer XML-Datei in eine Tabelle. Weitere Hinweise siehe [Abschnitt 20.5](#).

SELECT**Daten auswählen**

```
SELECT
[ALL|DISTINCT|DISTINCTROW]
[HIGH_PRIORITY]
[STRAIGHT_JOIN]
[SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
[SQL_CACHE] [SQL_CALC_FOUND_ROWS]
{*|spaltenliste|ausdruck}
[INTO |variable[, variable]*]]
[FROM {tabellenreferenzen|tabellenname}
[PARTITION (partitionsname[, partitionsname]*)
]
[WHERE bedingung]
[GROUP BY
{spaltenname|ausdruck|position} [ASC|DESC] [,
{spaltenname|ausdruck|position} [ASC|DESC]]*
[WITH ROLLUP]]
[HAVING bedingung]
[WINDOW fenstername AS (fensterdefinition) [,
fenstername AS (fensterdefinition)]*
[ORDER BY
{spaltenname|ausdruck|position} [ASC|DESC] [,
{spaltenname|ausdruck|position} [ASC|DESC]]*
[WITH ROLLUP]]
[LIMIT {[offset,] anzahl|anzahl OFFSET offset}
]
[INTO OUTFILE 'dateiname'
[CHARACTER SET Zeichensatz]
exportoptionen
INTO {DUMPFILE 'dateiname'}]]
[FOR {UPDATE|SHARE} [OF tabellenname [, tabellenname]*
{NOWAIT|SKIP LOCKED}|LOCK IN SHARE MODE]]
;
```

Wählt Zeilen aus einer Tabelle *tabellenname* oder *tabellenreferenz* aus. Eine Tabellenreferenz kann beispielsweise das Ergebnis eines JOIN sein. Die Spalten können dabei mit Konstanten und Ausdrücken aufbereitet werden. Es werden nur die Zeilen ausgewählt, für welche die *bedingung* der WHERE-Klausel den Wert TRUE ergibt. Weitere Hinweise siehe die Kapitel im Teil [IV](#).

tabellenreferenzen	Referenzen auf Tabellen erstellen

```

tabellenreferenzen:
erweiterte_tabellenreferenz[, erweiterte_tabel
lenreferenz]*
erweiterte_tabellenreferenz:
tabellenreferenz|{OJ tabellenreferenz}
tabellenreferenz:
tabellenfaktor|verknüpfung
tabellenfaktor:
tabellenname [PARTITION(partitionsnamen)] [[AS
] alias] [indexhinweise]
|unterabfrage [AS] alias [(spaltenliste)]
|(tabellenreferenzen)
verknüpfung:
tabellenreferenz [INNER|CROSS] JOIN tabellenfa
ktor [verknüpfungsbedingung]
|tabellenreferenz STRAIGHT_JOIN tabellenfaktor
[verknüpfungsbedingung]
|tabellenreferenz {LEFT|RIGHT} [OUTER] JOIN ta
bellenreferenz verknüpfungsbedingung
|tabellenreferenz NATURAL [INNER|{LEFT|RIGHT}
[OUTER]] JOIN tabellenfaktor
verknüpfungsbedingung:
ON bedingung|USING(spaltenliste)
indexhinweise:
indexhinweis [, indexhinweis]*
indexhinweis:

```

```

USE {INDEX|KEY}
[FOR {JOIN|ORDER BY|GROUP BY}] [(indexname[, i
ndexname]*)]
|{IGNORE|FORCE} {INDEX|KEY}
[FOR {JOIN|ORDER BY|GROUP BY}] (indexname[, i
ndexname]*))

```

Tabellenreferenzen sind in der Regel keine Tabellen, die mit CREATE angelegt wurden, sondern solche, die dynamisch bei der Ausführung von Anweisungen entstehen.

UNION	Ergebnisse von Abfragen vereinigen

```

SELECT auswahl
UNION [ALL|DISTINCT]
SELECT auswahl
[UNION [ALL|DISTINCT]
SELECT auswahl]*
;

```

Hinweise siehe [Abschnitt 14.1.](#)

UPDATE	Inhalte einer Tabelle ändern

```

UPDATE [LOW_PRIORITY] [IGNORE] tabellenreferen
z SET
spaltenname={ausdruck|DEFAULT}
[, spaltenname={ausdruck|DEFAULT}]*
[WHERE bedingung]
[ORDER BY sortierangabe]
[LIMIT anzahl]
;

```

Hinweise siehe [Abschnitt 9.2.](#)

UPDATE	Inhalte einer Tabellenreferenz ändern

```
UPDATE [LOW_PRIORITY] [IGNORE] tabellenreferenz
SET
spaltenname={ausdruck|DEFAULT}
[, spaltenname={ausdruck|DEFAULT}] *
[WHERE bedingung]
;
```

Hinweise siehe [Abschnitt 9.2.](#)

WITH	Abfragen mit Common Table Expressions

```
cte_ausdruck:
WITH [RECURSIVE]
cte_name [(spaltenliste)] AS (unterabfrage)
[cte_name [(spaltenliste)] AS (unterabfrage)] *
anweisung;
```

Syntaktisch sind CTE etwas schwierig zu beschreiben. Sie werden der entsprechenden SELECT-, INSERT-, UPDATE- oder DELETE-Anweisung direkt – ohne Semikolon – vorangestellt und so weiter verwendet, als wäre das Ergebnis des CTE eine Tabelle.

Hinweise siehe [Abschnitt 11.5.](#)

26.4.3 Benutzerverwaltung

ALTER USER	Benutzer ändern

```
ALTER USER [IF EXISTS]
benutzerspezifikation [, benutzerspezifikation
]*
[REQUIRE {NONE|verschlüsselung [[AND] verschlü
ssellung]*}]
[WITH ressourceoption[ ressourceoption]*]
[passwortoption|sperroption]
;
```

ALTER USER	Benutzerrolle ändern

```
ALTER USER [IF EXISTS] user DEFAULT ROLE {NONE
|ALL|benutzerrolle[, benutzerrolle]*};
```

ALTER USER	Authentifizierung des aktuellen Benutzers ändern

```
ALTER USER [IF EXISTS] USER() benutzerauthenti
fizierungsoption;
```

CREATE ROLE	Benutzerrolle anlegen

```
CREATE ROLE [IF NOT EXISTS] rollenname[, rolle
nname]*;
```

Über den frei wählbaren Rollennamen können Kataloge von Benutzerrechten einem Benutzer zugewiesen werden (siehe GRANT in [Abschnitt 26.4.3](#) und REVOKE in [Abschnitt 26.4.3](#)).

CREATE USER	Benutzer anlegen

```
CREATE USER benutzerspezifikation[, benutzersp
ezifikation]*;
```

Hinweise siehe [Abschnitt 24.2.](#)

DROP ROLE	Benutzerrolle löschen

```
DROP ROLE [IF EXISTS] rollenname[, rollenname]  
*;
```

DROP USER	Benutzer löschen

```
DROP USER [IF EXISTS] benutzername[, benutzern  
ame]*;
```

Hinweise siehe [Abschnitt 24.2.](#)

GRANT	Benutzer Rechte zuweisen

```
GRANT  
benutzerrecht [(spaltenliste)][, benutzerrecht  
[(spaltenliste)]*  
ON [objekttyp] privilegtiefe  
TO {benutzername|rollenname} [, {benutzername|  
rollenname}*  
[WITH GRANT OPTION]  
[AS benutzername  
[WITH ROLE  
DEFAULT  
|NONE  
|ALL  
|ALL EXCEPT rollenname[, rollenname]*  
|rollenname[, rollenname]*  
]  
]  
;
```

Hinweise siehe [Abschnitt 24.2.](#)

GRANT PROXY	Benutzer über Proxy Rechte zuweisen

```
GRANT PROXY ON {vorlage_benutzername|vorlage_rolle}
TO {benutzername|rolle} [, {benutzername|rolle}]* [WITH GRANT OPTION]
;
```

Weist einem Benutzernamen oder einer Benutzerrolle die Rechte der Vorlage *vorlage_** zu.

GRANT	Benutzerrolle Rechte zuweisen

```
GRANT rolle [, rolle]*
TO {benutzername|rolle} [, {benutzername|rolle}]* [WITH ADMIN OPTION]
;
```

RENAME USER	Benutzer umbenennen

```
RENAME USER
benutzername_alt TO benutzername_neu
[, benutzername_alt TO benutzername_neu]*
;
```

REVOKE	Benutzer ein Recht entziehen

```
REVOKE
benutzerrecht [(spaltenname[, spaltenname]*)]
[, benutzerrecht [(spaltenname[, spaltenname]*)]
)]*
ON [objekttyp] privilegtiefe
FROM {benutzername|rolle}[, {benutzername|rolle}]
```

```
| rolldname}] *  
;
```

Hinweise siehe [Abschnitt 24.2.](#)

REVOKE ALL	Benutzer alle Rechte entziehen

```
REVOKE ALL [PRIVILEGES] GRANT OPTION  
FROM {benutzername|rolldname}[, {benutzername  
|rolldname}] *  
;
```

Hinweise siehe [Abschnitt 24.2.](#)

REVOKE PROXY	Benutzer Rechte über einen Proxy entziehen

```
REVOKE PROXY ON {benutzername|rolldname}  
FROM {benutzername|rolldname}[, {benutzername  
|rolldname}] *  
;
```

REVOKE benutzerrolle	Benutzerrollen Rechte entziehen

```
REVOKE rolldname[, rolldname] *  
FROM {benutzername|rolldname}[, {benutzername  
|rolldname}] *  
;
```

SET PASSWORD	Ein neues Passwort vergeben

```
SET PASSWORD [FOR benutzername] = 'passwort'  
[REPLACE 'passwort']
```

[RETAIN CURRENT PASSWORD]

;

benutzerspezifikation	Festlegen der Benutzereigenschaften

```
benutzerspezifikation:
benutzername {
IDENTIFIED BY 'passwort' [REPLACE 'passwort']
[RETAIN CURRENT PASSWORD]
|IDENTIFIED WITH authentifizierungsplugin
|IDENTIFIED WITH authentifizierungsplugin BY 'passwort'
[REPLACE 'passwort'] [RETAIN CURRENT PASSWORD]
|IDENTIFIED WITH authentifizierungsplugin AS 'passwort'
|DISCARD OLD PASSWORD
}
```

Hinweise siehe [Abschnitt 24.2](#).

benutzerauthentifizierungsoption	Optionen der Funktion zur Benutzerauthentifizierung

```
benutzerauthentifizierungsoption: {
IDENTIFIED BY 'passwort' [REPLACE 'passwort']
[RETAIN CURRENT PASSWORD]
|DISCARD OLD PASSWORD
}
```

grantoption	Optionen eines Rechts

```
grantoption:
GRANT OPTION|ressourceoption
```

Hinweise siehe [Abschnitt 24.2.](#)

sperroption	Angaben zur Sperrung (lock)

sperroption:
ACCOUNT LOCK|ACCOUNT UNLOCK

objekttyp	Objekt eines Rechts

objekttyp:
TABLE|FUNCTION|PROCEDURE

Hinweise siehe [Abschnitt 24.2.](#)

passwortoption	Optionen eines Passworts

passwortoption:
PASSWORD EXPIRE [DEFAULT|NEVER|INTERVAL *n* DAY]
|PASSWORD HISTORY {DEFAULT|*n*}
|PASSWORD REUSE INTERVAL {DEFAULT|*n* DAY}
|PASSWORD REQUIRE CURRENT [DEFAULT|OPTIONAL]

ressourceoption	Optionen einer Ressource

ressourceoption:
MAX_QUERIES_PER_HOUR *anzahl*
|MAX_UPDATES_PER_HOUR *anzahl*
|MAX_CONNECTIONS_PER_HOUR *anzahl*
|MAX_USER_CONNECTIONS *anzahl*

Hinweise siehe [Abschnitt 24.2.](#)

privilegtiefe	Ebene, auf der das Recht angewendet wird

privilegtiefe:

.

*| datenbankname.**

| datenbankname.tabellenname

| tabellenname

| datenbankname.{funktionsname|procedurename}

Weitere Hinweise siehe [Abschnitt 24.2](#).

verschlüsselung	Angaben zur Verschlüsselung

verschlüsselung:

SSL

| X509

| CIPHER 'cipher'

| ISSUER 'issuer'

| SUBJECT 'subject'

¹ Ich möchte diese Aussage durch den Hinweis auf die Volltextsuche relativieren.

² In MariaDB ist JSON ab 10.2.7 als Alias für LONGTEXT eingeführt worden, um mit MySQL weiterhin einen Datenaustausch beispielsweise über mysqldump durchführen zu können.

³ Diese Tabelle würde dann nicht mehr den Normalformen entsprechen.

⁴ Es gibt also doch ein *Undo*! Finden Sie heraus, wie es funktioniert ;-).

27 Ausgewählte Quelltexte

27.1 DOUBLE versus DECIMAL



Laden Sie dieses Experiment aus dem MySQL-Skriptverzeichnis mit SOURCE perfDecimal01.sql.

Wenn man im MySQL oder MariaDB Client eine Abfrage ausführt, bekommt man immer eine Ausführungszeit angegeben. Das ist nicht die hier ermittelte. Diese Information liegt nicht auf dem Server, sondern wird vom Client als Zeitraum vom Start bis zum Ende der Ausführung selbst ermittelt.

```
2  SET max_heap_table_size = 1024*1024*25; --
   max. 25MB
3
4  DROP DATABASE IF EXISTS perfDecimal01;
5  CREATE DATABASE perfDecimal01;
6  USE perfDecimal01;
7
8  -- Tabellen für die Testdaten
9  CREATE TABLE testdaten (
10     spalte_double DOUBLE,
11     spalte_decimal DECIMAL(30,15)
```

```

12 ) ENGINE=MEMORY;
13
14 CREATE TABLE testdaten_quelle (
15     spalte_double DOUBLE,
16     spalte_decimal DECIMAL(30,15)
17 ) ENGINE=MEMORY;
18
19 -- Tabelle für die Messergebnisse
20 CREATE TABLE messung (
21     anzahl_ds BIGINT UNSIGNED,
22     sekunden_double DECIMAL(12,8),
23     sekunden_decimal DECIMAL(12,8)
24 ) ENGINE=MEMORY;

```

Damit das Experiment garantiert unter den gleichen Datenbedingungen startet, wird die dazugehörige Datenbank auf jeden Fall gelöscht und wieder neu angelegt. Für die Testdaten verwende ich zwei Tabellen: `testdaten` und `testdaten_quelle`. Zuerst werden – beispielsweise 2000 – Zufallsdaten in die Tabelle `testdaten_quelle` eingefügt. Später werden diese Daten in die Tabelle `testdaten` kopiert. Der Grund für dieses Vorgehen ist, dass das Einfügen eines Datensatzes mit `INSERT INTO ... VALUES` sehr langsam bezogen auf `INSERT INTO ... SELECT` ist.

Die Tabelle `messung` enthält pro Zeile eine Zeitmessung. Dabei wird die Dauer in Anzahl der Datensätze und die jeweilige Dauer in Sekunden gemessen.

Um die Ausführungsgeschwindigkeit zu erhöhen, werden alle Tabellen mit der Engine *MEMORY* angelegt. Dadurch, dass RAM-Zugriffe deutlich schneller als Festspeicherplattenzugriffe sind, wird die Performance erheblich gesteigert. In der ersten Zeile wird die maximale Größe einer Tabelle – hier 25 MB – in der Engine *MEMORY* festgelegt.

```

28 DELIMITER //

```

```

29 CREATE PROCEDURE testdaten_insert
30 (
31 IN iAnzahl INT -- Anzahl der einzufügenden
Testdaten
32 )
33 BEGIN
34 DECLARE vAnzahl INT DEFAULT 0; -- Schleife
nvariable
35 DECLARE vDouble DOUBLE;
36 DECLARE vDecimal DECIMAL(30,15);
37
38 WHILE vAnzahl < iAnzahl DO -- Schleifenbed
ingung
39     SELECT RAND() * FLOOR(-10000 + (RAND() *
10000)) -- Zufallswert
40     INTO vDecimal;
41     SET vDouble = vDecimal;
42     INSERT INTO testdaten_quelle -- Einfügen
43     VALUES (vDouble, vDecimal);
44     SET vAnzahl = vAnzahl + 1; -- Schleifeni
nkrement
45 END WHILE;
46 END//
47 DELIMITER ;

```

Die Prozedur testdaten_insert baut den Bestand der Tabelle testdaten_quelle auf. Die Anzahl der Zufallswerte wird als Übergabeparameter festgelegt. Passend dazu wird in [Zeile 34](#) eine Schleifenvariable deklariert. Diese startet mit 0 (DEFAULT) und wird bei jedem Schleifendurchlauf um 1 in [Zeile 44](#) erhöht. In [Zeile 38](#) werden beide Variablen in der Schleifenbedingung verwendet.

In [Zeile 39](#) werden Zufallswerte erzeugt. Der dabei verwendete Ausdruck liefert Zufallswerte im Bereich von -10000 bis fast 0. Entscheidend ist, dass Nachkommastellen unterschiedlicher

Länge erzeugt werden. Die werden dann in [Zeile 42](#) in die Tabelle eingefügt.

```
52 DELIMITER //
53 CREATE PROCEDURE perfDecimal01
54 (
55   IN iAnzahlNeueZeilen INT, -- Anzahl der neuen Datensätze
    pro Messung
56   IN iAnzahlSchleifen INT -- Anzahl der Messschleifen
57 )
58 BEGIN
59   DECLARE vAnzahlSchleifen INT DEFAULT 0; -- Schleifenvariable
60   DECLARE vSekundenDouble DECIMAL(12,8); -- Zeitdauer
61   DECLARE vSekundenDecimal DECIMAL(12,8); -- Zeitdauer
62   DECLARE vAnzahlDS BIGINT; -- Anzahl der Testdaten
63   DECLARE vTmpDouble DOUBLE; -- Hilfsvariable
64   DECLARE vTmpDec DECIMAL(30,15); -- Hilfsvariable
```

Die Prozedur `perfDecimal01` implementiert die eigentliche Messung. Die beiden Übergabeparameter bestimmen zum einen die Anzahl der Erhöhung der Testdaten pro Schleifendurchlauf. Das ist gleichbedeutend mit der Anzahl der Zufallswerte in `testdaten_quelle`. Zum anderen wird die Anzahl der Messungen festgelegt. Deshalb gibt es passend zu `iAnzahlSchleifen` die Schleifenvariable `vAnzahlSchleifen`.

Die Variablen `vSekundenDouble` und `vSekundenDecimal` nehmen die Ausführungszeit der jeweiligen Berechnung auf.

vAnzahlDS wird verwendet, um sich die Anzahl der Datensätze in der Tabelle testdaten zu merken, damit diese in die Tabelle messung eingefügt werden kann. Die beiden vTmp-Variablen sind funktionslos und werden nur gebraucht, um sinnlose Bildschirmausgaben zu verhindern.

```
66 TRUNCATE messung; -- Alte Daten entfernen
67 TRUNCATE testdaten; -- Alte Daten entfernen
68 TRUNCATE testdaten_quelle; -- Alte Daten entfernen
```

Die Tabellen werden geleert. Dadurch wird bei jedem Messstart auf 0 gesetzt.

```
70 SET PROFILING = 1; -- Profiling einschalten
71
72 CALL testdaten_insert(iAnzahlNeueZeilen);
-- Testdatenquelle aufbauen
73 WHILE vAnzahlSchleifen < iAnzahlSchleifen
DO -- Hauptschleife
74     SELECT 'Schleifendurchlauf ', vAnzahlSchleifen, ' von ', iAnzahlSchleifen;
75     INSERT INTO testdaten SELECT * FROM testdaten_quelle; -- Testdaten erweitern
76     -- START Messung DOUBLE
77     SELECT SQL_NO_CACHE AVG(spalte_double) INTO vTmpDouble FROM testdaten;
78     SELECT SUM(DURATION) INTO vSekundenDouble
79     FROM INFORMATION_SCHEMA.PROFILING
80     GROUP BY query_id
81     ORDER BY query_id DESC LIMIT 1;
82     -- ENDE Messung DOUBLE
```

Um die Ausführungszeiten ermitteln zu können muss das

Um die Ausführungszeiten ermitteln zu können, muss das Profiling für diese Session eingeschaltet ([Zeile 70](#)) und am Ende wieder ausgeschaltet ([Zeile 102](#)) werden. In [Zeile 72](#) wird die Testdatenquelle aufgebaut, indem die Prozedur `testdaten_insert` passend parametrisiert aufgerufen wird.

Die Hauptschleife ([73-100](#)) geht wie folgt vor: Erweitern des Testdatenbestands, Berechnung mit DOUBLE, Zeitmessung der Berechnung mit DOUBLE, Berechnung mit DECIMAL, Zeitmessung der Berechnung mit DECIMAL und Einfügen der Messwerte in die Tabelle `messung`. Schauen wir uns eine Messung mal genauer an.

```
85     SELECT SQL_NO_CACHE AVG(spalte_decimal)
      INTO vTmpDec FROM testdaten;
86     SELECT SUM(DURATION) INTO vSekundenDecim
      al
87     FROM INFORMATION_SCHEMA.PROFILING
88     GROUP BY query_id
89     ORDER BY query_id DESC LIMIT 1;
90 -- Ende Messung DECIMAL
91
92 -- Sichern der Ergebnisse
93     SELECT COUNT(*) FROM testdaten INTO vAnz
      ahlDS;
94     INSERT INTO messung
95     (anzahl_ds, sekunden_double, sekunden_de
      cimal)
96     VALUES
97     (vAnzahlDS, vSekundenDouble, vSekundenDe
      cimal);
98
99     SET vAnzahlSchleifen = vAnzahlSchleifen
      + 1;
100  END WHILE; -- Hauptschleifenende
```

Die Berechnung ist eine Mittelwerthildung über die gesamte

Die Berechnung ist eine Mittelwertbildung über die gesamte Testdatentabelle. Der Zusatz SQL_NO_CACHE soll verhindern, dass die Messung durch Caching verfälscht wird. Mit INTO vTempDec wird verhindert, dass der errechnete Mittelwert auf dem Bildschirm ausgegeben wird.

Und jetzt kommt das Herzstück: die Zeitmessung. Ist das Profiling eingeschaltet, wird in der Tabelle INFORMATION_SCHEMA.PROFILING für jeden Arbeitsschritt, der zur Ausführung einer Anweisung nötig ist, eine Zeitdauer (DURATION) eingefügt. Hier ein Beispiel:

```

10  SELECT query_id, state, duration
11  FROM INFORMATION_SCHEMA.PROFILING
12  WHERE query_id = (SELECT MAX(query_id)
13  FROM INFORMATION_SCHEMA.PROFILING);
14
15  +-----+-----+-----+-----+
16  | query_id | state | duration |
17  +-----+-----+-----+-----+
18  | 1261359 | Starting | 0.000051 |
19  | 1261359 | Checking permissions | 0.00000
20  | 1261359 | Opening tables | 0.000166 |
21  | 1261359 | After opening tables | 0.00000
22  | 1261359 | System lock | 0.000004 |
23  | 1261359 | Table lock | 0.000009 |
24  | 1261359 | Init | 0.000022 |
25  | 1261359 | Optimizing | 0.000011 |
26  | 1261359 | Statistics | 0.000016 |
27  | 1261359 | Preparing | 0.000015 |
28  | 1261359 | Executing | 0.000005 |
29  | 1261359 | Sending data | 0.000393 |
30  | 1261359 | End of update loop | 0.000009

```

```

31 | 1261359 | Query end | 0.000005 |
32 | 1261359 | Commit | 0.000005 |
33 | 1261359 | Closing tables | 0.000004 |
34 | 1261359 | Unlocking tables | 0.000004 |
35 | 1261359 | Closing tables | 0.000008 |
36 | 1261359 | Starting cleanup | 0.000004 |
37 | 1261359 | Freeing items | 0.000007 |
38 | 1261359 | Updating status | 0.000032 |
39 | 1261359 | Reset for next command | 0.000
005 |
40 +-----+-----+-----+-----+
-----+

```

Gruppiert man nun die Tabelle nach der `query_id` und summiert die einzelnen Zeiten auf, erhält man die Gesamtdauer. Will man nun die Dauer der letzten Anweisung, so sortiert man die Gruppenergebnisse nach `query_id` absteigend und holt sich nur den ersten Datensatz. Die Variante mit der Unterabfrage geht natürlich auch ;-)

Zum Schluss wird und das Experiment aufgerufen. Den Aufruf und das Löschen der Datenbank habe ich allerdings auskommentiert, da Sie dies lieber manuel tun sollten.

```

102 SET PROFILING = 0; -- Profiling ausschalt
en
103 END//
104 DELIMITER ;
105
106 -- CALL perfDecimal01(2000, 500);
107 -- DROP DATABASE IF EXISTS perfDecimal01;

```

27.2 Rundungsfehler



Laden Sie dieses Experiment aus dem MySQL-Skriptverzeichnis mit SOURCE rundungsfehler01.sql.

Zuerst wird die Testumgebung gebaut:

```
2  DROP DATABASE IF EXISTS rundungsfehler01;
3  CREATE DATABASE rundungsfehler01;
4  USE rundungsfehler01;
5
6  -- Tabellen für die Testdaten
7  CREATE TABLE bla (
8      floSpalte FLOAT,
9      decSpalte DECIMAL(30,15)
10 ) ENGINE=MEMORY;
```

Anschließend basteln wir uns eine Prozedur, die die MEMORY-Tabelle mit lauter 0.001 füllt.

```
14 DELIMITER //
15 CREATE PROCEDURE testdaten(IN iAnzahl INT)
16 BEGIN
17 DECLARE vAnzahl INT DEFAULT 0;
18 DECLARE floSpalte FLOAT;
19 DECLARE decSpalte DECIMAL(10,4);
20
21 SET floSpalte = 0.001;
22 SET decSpalte = 0.001;
23 WHILE vAnzahl < iAnzahl DO
24     INSERT INTO bla VALUES(floSpalte, decSpalte);
25     SET vAnzahl = 1 + vAnzahl;
26 END WHILE;
27 END//
28 DELIMITER ;
```

Zum Schluss der Aufruf der Prozedur und die Ausgabe der Summen. Natürlich hätten wir das ganze auch ohne eine Tabelle nur mit Variablen durchführen können, aber ich wollte explizit auf Tabellenspalten arbeiten.

```
31 CALL testdaten(1000); -- Erzeuge 1000 Test
daten
32 SELECT
33 SUM(floSpalte) AS 'FLOAT', SUM(decSpalte)
AS 'DECIMAL'
34 FROM bla ;
35
36 DROP DATABASE rundungsfehler01; -- Kann we
ch.
```

27.3 NULL versus NOT NULL



Laden Sie dieses Experiment aus dem MySQL-Skriptverzeichnis mit SOURCE perfNotNull01.sql.

```
2 SET max_heap_table_size = 1024*1024*1000;
-- max. 1GB
3 DROP DATABASE IF EXISTS perfNotNull01;
4 CREATE DATABASE perfNotNull01;
5 USE perfNotNull01;
6
7 -- Tabelle für die Testdaten
8 CREATE TABLE testdaten (
9     spalte_notnull VARCHAR(255) NOT NULL DEF
AULT '',
10     spalte_null VARCHAR(255) NULL
11 ) ENGINE=MEMORY;
```

```

12
13 CREATE TABLE testdaten_quelle
14 LIKE testdaten;
15
16 -- Tabelle für die Messergebnisse
17 CREATE TABLE messung (
18     anzahl_ds BIGINT UNSIGNED,
19     sekunden_notnull DECIMAL(12,8),
20     sekunden_null DECIMAL(12,8)
21 ) ENGINE=MEMORY;

```

Aufbau und Funktion dieser Zeilen entsprechen denen des Experiments zur Performancemessung von DECIMAL und DOUBLE (siehe [Abschnitt 27.1](#)).

```

25 DELIMITER //
26 CREATE PROCEDURE testdaten_insert (
27     IN iAnzahl INT -- Anzahl der einzufügenden
    Testdaten
28 )
29 BEGIN
30     DECLARE vAnzahl INT DEFAULT 0; -- Schleife
    nvariable
31     DECLARE vString VARCHAR(255);
32
33     WHILE vAnzahl < iAnzahl DO -- Schleifenbed
    ingung
34         INSERT INTO testdaten_quelle
35             VALUES('', NULL);
36     SET vAnzahl = vAnzahl + 1; -- Schleifenink
    rement
37     END WHILE;
38     END//
39 DELIMITER ;

```

Der Aufbau der Testdaten ist allerdings viel einfacher als oben. Es werden eine bestimmte Anzahl (*iAnzahl*) Datensätze

erzeugt. In der ersten Spalte wird dabei ein Leer-String und in der zweiten NULL verwendet.

```
44 DELIMITER //
45 CREATE PROCEDURE perfNotNull01
46 (
47     IN iAnzahlNeueZeilen INT, -- Anzahl der
Testdatensätze
48     IN iAnzahlSchleifen INT -- Anzahl der Me
ssungen
49 )
50 BEGIN
51 DECLARE vAnzahlSchleifen INT DEFAULT 0;
52 DECLARE vSekundenNotNull DECIMAL(12,8); --
Zeitdauer
53 DECLARE vSekundenNull DECIMAL(12,8); -- Ze
itdauer
54 DECLARE vAnzahlDS BIGINT;
55 DECLARE vTmp VARCHAR(255);
56
57 TRUNCATE messung;
58 TRUNCATE testdaten;
59 TRUNCATE testdaten_quelle;
60
61 SET PROFILING = ON; -- Profiling einschalt
en
62
63 CALL testdaten_insert(iAnzahlNeueZeilen);
-- Testdatenquelle aufbauen
64
65 WHILE vAnzahlSchleifen < iAnzahlSchleifen
DO -- Hauptschleife
66     SELECT 'Schleifendurchlauf ', vAnzahlSch
leifen, ' von ', iAnzahlSchleifen;
67     INSERT INTO testdaten -- Testdaten erwei
tern
68     SELECT * FROM testdaten_quelle;
69
```

```

70  -- START Messung NULL
71      SELECT SQL_NO_CACHE COUNT(*) -- Testquer
72  y
73      INTO vTmp
74      FROM testdaten
75      WHERE spalte_null IS NULL;
76      SELECT SUM(DURATION)
77      INTO vSekundenNull
78      FROM INFORMATION_SCHEMA.PROFILING
79      GROUP BY query_id
80      ORDER BY query_id DESC
81      LIMIT 1;
82  -- ENDE Messung NULL
83
84  -- START Messung NOT NULL
85      SELECT SQL_NO_CACHE COUNT(*) -- Testquer
86  y
87      INTO vTmp
88      FROM testdaten
89      WHERE spalte_notnull = '';
90      SELECT SUM(DURATION)
91      INTO vSekundenNotNull
92      FROM INFORMATION_SCHEMA.PROFILING
93      GROUP BY query_id
94      ORDER BY query_id DESC
95      LIMIT 1;
96  -- ENDE Messung NOT NULL
97
98  -- Messung
99  -- Sichern der Ergebnisse
100      SELECT COUNT(*)
101      INTO vAnzahlDS
102      FROM testdaten;
103      INSERT INTO messung (anzahl_ds, sekunde
n_notnull, sekunden_null)
104      VALUES (vAnzahlDS, vSekundenNotNull, vS
ekundenNull);
105      SET vAnzahlSchleifen = vAnzahlSchleifen

```

```

+ 1;
104 END WHILE; -- Hauptschleifenende
105
106 SET PROFILING = OFF; -- Profiling ausschalten
107 END//
108 DELIMITER ;

```

Fast alle Teilschritte der Messung entsprechen dem des Experiments oben (siehe [Abschnitt 27.1](#)). Lediglich die zu messenden Anweisungen sind – natürlich – andere:

```

71 SELECT SQL_NO_CACHE COUNT(*) -- Testquery
72 INTO vTmp
73 FROM testdaten
74 WHERE spalte_null IS NULL;

```

und

```

84 SELECT SQL_NO_CACHE COUNT(*) -- Testquery
85 INTO vTmp
86 FROM testdaten
87 WHERE spalte_notnull = '';

```

Ich habe diese beiden Abfragen gewählt, weil sie genau das tun, wo Zeitunterschiede festgestellt werden müssen: Ein Vergleich mit einem Leer-String (= '') und NULL (IS NULL).

Die Messdaten wurden mit folgendem Aufruf ermittelt:

```

110 CALL perfNotNull01(1000, 500);

```

Die in [Bild 5.3](#) in [Abschnitt 5.3.5](#) abgebildeten Messdaten sind auf die ersten 100.000 Ergebnisse beschränkt, damit überhaupt ein Abstand sichtbar wird.

27.4 Suchen mit und ohne Index



Laden Sie dieses Experiment aus dem MySQL-Skriptverzeichnis mit SOURCE perfIndexMitOhne01.sql.

Um die Performanceunterschiede bei der Verwendung und Nichtverwendung von Indizes darzustellen, habe ich Echtdaten – nämlich die Liste aller deutschen Bankleitzahlen¹ – verwendet. Zuerst wird die Testdatenbank angelegt und ausgewählt.

```
2  SET max_heap_table_size = 1024*1024*200; -  
- max. 200MB  
3  DROP DATABASE IF EXISTS perfIndexMitOhne01  
;  
4  CREATE DATABASE perfIndexMitOhne01  
5  CHARACTER SET utf8mb4  
6  COLLATE utf8mb4_german2_ci;  
7  
8  USE perfIndexMitOhne01;
```

Anschließend werden die Testdatentabellen angelegt. Dabei wird auf einer Spalte ein Index erzeugt (bezeichnung_mit) und auf der anderen (bezeichnung_ohne) nicht.

```
11 CREATE TABLE bank (  
12   blz CHAR(8),  
13   merkmäl CHAR(1),  
14   bezeichnung VARCHAR(255),  
15   plz CHAR(5),  
16   ort VARCHAR(255),  
17   kurz VARCHAR(255),  
18   pan CHAR(5),
```

```

19  bic VARCHAR(255),
20  sm CHAR(2),
21  ds CHAR(6),
22  kz CHAR(1),
23  blzl CHAR(1),
24  blzn CHAR(8),
25  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY
26  ) ENGINE=MEMORY;
27
28  CREATE TABLE testdaten (
29  bezeichnung_mit VARCHAR(255),
30  bezeichnung_ohne VARCHAR(255)
31  ) ENGINE=MEMORY;
32
33  CREATE INDEX idx_testdaten ON testdaten(be
zeichnung_mit);

```

Die Messergebnisse werden in einer eigenen Tabelle abgelegt und für die Auswertung wird eine View angelegt.

```

36  CREATE TABLE messung (
37    anzahl_ds BIGINT UNSIGNED,
38    sekunden_mitindex DECIMAL(12,8),
39    sekunden_ohneindex DECIMAL(12,8)
40  ) ENGINE=MEMORY;
41
42  CREATE VIEW view_messung AS
43  SELECT
44    anzahl_ds,
45    AVG(sekunden_mitindex) AS mitindex,
46    AVG(sekunden_ohneindex) AS ohneindex
47  FROM messung
48  GROUP BY anzahl_ds
49  ;

```

Jetzt werden die Bankleitzahlen in die Tabelle bank importiert. Dabei wird eine id per AUTO_INCREMENT mit erzeugt. Über diese id erfolgt später die zufällige Selektion.


```

53     INTO TABLE bank
54     FIELDS TERMINATED BY ';'
55     LINES TERMINATED BY '\n'
56     IGNORE 1 LINES
57     (blz, merkmäl, bezeichnung, plz, ort, ku
rz, pan, bic, sm, ds, kz, blzl, blzn)
58 ;

```

Die Prozedur testdaten_insert kopiert iAnzahl viele Datensätze aus der Tabelle bank in die Tabelle testdaten. Damit es nicht immer die gleichen sind, wird per Zufall eine Zahl ermittelt und diese als id verwendet. Damit es diesen Zufallswert auch wirklich in der Banktabelle gibt, wird vorher die Anzahl der Bankdatensätze ermittelt und als Obergrenze für die Zufallszahl benutzt.

```

61 CREATE PROCEDURE testdaten_insert
62 (
63 iAnzahl INT
64 )
65 BEGIN
66 DECLARE vAnzahl INT;
67 DECLARE vCount INT;
68 DECLARE vId INT UNSIGNED;
69
70 SELECT COUNT(*) INTO vCount FROM bank;
71 SET vAnzahl = 0;
72 WHILE vAnzahl < iAnzahl DO
73     SET vId = FLOOR(RAND() * vCount);
74     INSERT INTO testdaten (bezeichnung_mit,
bezeichnung_ohne)
75     SELECT bezeichnung, bezeichnung
76     FROM bank
77     WHERE id = vId;
78     SET vAnzahl = vAnzahl + 1;
79 END WHILE;
80 END//

```

81 DELIMITER ;

Die Prozedur `perfIndexMitOhne01` hat als ersten Parameter die Anzahl der Messungen und als zweiten die Größe der Erweiterung der Testdaten pro Messung. Es werden notwendige Variablen deklariert und das Profiling eingeschaltet. Über das Profiling und wie man die Ausführungszeiten ermittelt, erfahren Sie im ersten Experiment oben (siehe [Abschnitt 27.1](#)) mehr.

```
85 CREATE PROCEDURE perfIndexMitOhne01
86 (
87     IN iAnzahlNeueZeilen INT, -- Anzahl der
Testdatensätze
88     IN iAnzahlSchleifen INT -- Anzahl der Me
ssungen
89 )
90 BEGIN
91     DECLARE vAnzahlSchleifen INT DEFAULT 0;
92     DECLARE vSekundenMitIndex DECIMAL(12,8);
-- Zeitdauer
93     DECLARE vSekundenOhneIndex DECIMAL(12,8)
; -- Zeitdauer
94     DECLARE vAnzahlDS BIGINT; -- Anzahl der
Zeilen
95     DECLARE vN INT; -- Stichprobenzähler
96     DECLARE vTmp VARCHAR(255);
97
98 TRUNCATE messung;
99 TRUNCATE testdaten;
100
101 SET PROFILING = ON; -- Profiling einschal
ten
```

In der Hauptschleife wird nach einer Bildschirmausgabe zuerst der Testdatenbestand erweitert. Dann wird eine Suche auf der Spalte mit und anschließend eine auf der Spalte ohne Index

durchgeführt. Die Ausführungszeiten werden ermittelt und in die Variablen vSekundenMit und vSekundenOhne abgelegt.

Die Messwerte werden dann in die Tabelle messwert eingefügt und die Schleifenvariable inkrementiert.

```
104     SELECT 'Schleifendurchlauf ', vAnzahlSc
hleifen, ' von ', iAnzahlSchleifen;
105     CALL testdaten_insert(iAnzahlNeueZeilen
); -- Testdatenquelle aufbauen
106
107     SET vN = 0;
108     WHILE vN < 20 DO
109         -- START Messung mit Index
110         SELECT SQL_NO_CACHE COUNT(*) -- Testq
uery
111         INTO vTmp
112         FROM testdaten
113         WHERE bezeichnung_mit = 'ABC';
114         SELECT SUM(DURATION)
115         INTO vSekundenMitIndex
116         FROM INFORMATION_SCHEMA.PROFILING
117         GROUP BY query_id
118         ORDER BY query_id DESC LIMIT 1;
119         -- ENDE Messung mit Index
120
121         -- START Messung ohne Index
122         SELECT SQL_NO_CACHE COUNT(*) -- Testque
ry
123         INTO vTmp
124         FROM testdaten
125         WHERE bezeichnung_ohne = 'ABC';
126         SELECT SUM(DURATION)
127         INTO vSekundenOhneIndex
128         FROM INFORMATION_SCHEMA.PROFILING
129         GROUP BY query_id
130         ORDER BY query_id DESC LIMIT 1;
131         -- ENDE Messung ohne Index
```

```

132
133      -- Sichern der Ergebnisse
134      SELECT COUNT(*)
135      INTO vAnzahlDS
136      FROM testdaten;
137      INSERT INTO messung
138      (anzahl_ds, sekunden_mitindex, sekund
139      en_ohneindex)
140      VALUES
141      (vAnzahlDS, vSekundenMitIndex, vSekun
142      denOhneIndex);
141      SET vN = vN + 1;
142      END WHILE;
143      SET vAnzahlSchleifen = vAnzahlSchleifen
144      + 1;
144  END WHILE; -- Hauptschleifenende

```

Abschließend wird das Profiling wieder ausgeschaltet und der DELIMITER wieder zurückgesetzt.

```

147  END//
148  DELIMITER ;

```

Um die Ergebnisse der ersten Messung zu erhalten, rufen Sie die Prozedur wie folgt auf:

```

150  CALL perfIndexMitOhne01(100, 500);

```

Die zweite Messung wurde mit anderen Werten durchgeführt:

```

150  CALL perfIndexMitOhne01(2, 300);

```

Für beide Ergebnisgrafiken [Bild 6.1](#) und [Bild 6.2](#) in [Abschnitt 6.1](#) wurden die obere Grenze der X-Achse reduziert, damit die Ergebnisse leichter zu lesen sind. Die Messdaten sind natürlich bis zur Obergrenze vorhanden.

27.5 Messen der Performance der Einfügeoperation



Laden Sie dieses Experiment aus dem MySQL-Skriptverzeichnis mit SOURCE perfIndexInsert01.sql.

Zuerst wird wieder eine Testumgebung gebastelt. Die Tabelle testdaten dient als Kopierquelle des INSERT. Es werden dazu die Bankleitzahlendaten verwendet.

```
2  SET max_heap_table_size = 1024*1024*200; -
- max. 200MB
3  DROP DATABASE IF EXISTS perfIndexInsert01;
4  CREATE DATABASE perfIndexInsert01
5  CHARACTER SET utf8mb4
6  COLLATE utf8mb4_german2_ci;
7
8  USE perfIndexInsert01;
9
10 -- Tabellen für die Testdaten
11 CREATE TABLE testdaten (
12  blz CHAR(8), merkm1 CHAR(1), bezeichnung
  VARCHAR(255),
13  plz CHAR(5), ort VARCHAR(255), kurz VARCHA
  R(255),
14  pan CHAR(5), bic VARCHAR(255), sm CHAR(2),
15  ds CHAR(6), kz CHAR(1), blzl CHAR(1), blzn
  CHAR(8)
16  ) Engine=MEMORY;
17
18 -- Einlesen der Bankdaten
19 SET GLOBAL local_infile = ON;
20 LOAD DATA LOCAL INFILE 'blz_20120305.csv'
```

```
21 INTO TABLE testdaten
22 FIELDS TERMINATED BY ';'
23 LINES TERMINATED BY '\n'
24 IGNORE 1 LINES
25 ;
```

Dann die beiden Tabellen bank_ohne und bank_mit. Die eine hat keine Indizes und die andere sehr wohl. Ein Index ist ein einfacher und der andere ein aus drei Spalten zusammengesetzter.

```
27
28 CREATE TABLE bank_ohne LIKE testdaten;
29 CREATE TABLE bank_mit LIKE testdaten;
30 ALTER TABLE bank_ohne ENGINE = MyISAM;
31 ALTER TABLE bank_mit ENGINE = MyISAM;
32
33 CREATE INDEX idx_bankmit_bzeichnung ON bank_mit(bzeichnung(200)); -- !
```

Jetzt noch die Tabelle für die Messergebnisse:

```
36 -- Tabelle für die Messergebnisse
37 CREATE TABLE messung (
38   anzahl_ds BIGINT UNSIGNED,
39   sekunden_mitindex DECIMAL(12,8),
40   sekunden_ohneindex DECIMAL(12,8)
```

Wir haben zwei Prozeduren für den Test: Die erste kopiert anhängig vom Übergabeparameter iName den Inhalt der Tabelle testdaten nach bank_ohne oder bank_mit. Anschließend wird die Messung vorgenommen. Das Verfahren dazu wird oben (siehe [Abschnitt 27.1](#)) beschrieben. Der ermittelte Wert wird in den Übergabeparameter oSekunden geschrieben. Da dieser mit OUT deklariert wurde, steht der Wert auch außerhalb der Prozedur zur Verfügung.

```

45 -- Prozedur kopieren der Testdaten
46 DELIMITER //
47 CREATE PROCEDURE kopiere
48 (
49   IN iName VARCHAR(255),
50   OUT oSekunden DECIMAL(12,8) -- Ein OUT Parameter!
51 )
52 BEGIN
53   CASE iName
54     WHEN 'bank_ohne' THEN
55       INSERT INTO bank_ohne SELECT * FROM testdaten LIMIT 19000;
56     WHEN 'bank_mit' THEN
57       INSERT INTO bank_mit SELECT * FROM testdaten LIMIT 19000;
58   END CASE;
59   SELECT
60     SUM(DURATION)
61     INTO oSekunden -- Ergebnis in den OUT-Parameter
62   FROM INFORMATION_SCHEMA.PROFILING
63   GROUP BY query_id
64   ORDER BY query_id DESC LIMIT 1
65   ;
66 END//

```

Die eigentliche Testprozedur bekommt als Übergabeparameter die Anzahl der Messungen. Vor dem Start werden die entscheidenden Tabellen geleert. Innerhalb der Schleife wird die Prozedur kopiere aufgerufen; einmal mit bank_ohne und einmal mit bank_mit. Anschließend werden die gemessenen Werte in die Messtabelle eingefügt.

```

70 -- Testprozedur
71 DELIMITER //
72 CREATE PROCEDURE perfIndexInsert01

```

```

73  (
74  IN iAnzahlSchleifen INT UNSIGNED
75  )
76  BEGIN
77  DECLARE vAnzahlSchleifen INT DEFAULT 0;
78  DECLARE vSekundenMitIndex DECIMAL(12,8); -
- Zeitdauer
79  DECLARE vSekundenOhneIndex DECIMAL(12,8);
-- Zeitdauer
80  DECLARE vAnzahlDS BIGINT;
81
82  TRUNCATE messung;
83  TRUNCATE bank_mit;
84  TRUNCATE bank_ohne;
85
86  SET PROFILING = ON; -- Profiling einschalt
en
87
88  SET vAnzahlSchleifen = 0;
89  -- Hauptschleife
90  WHILE vAnzahlSchleifen < iAnzahlSchleifen
DO
91  SELECT 'Durchlauf ', vAnzahlSchleifen, ' v
on ', iAnzahlSchleifen;
92
93      -- START Messung
94      CALL kopiere('bank_ohne', vSekundenOhneI
ndex);
95      CALL kopiere('bank_mit', vSekundenMitInd
ex);
96      -- Ende Messung
97
98  -- Sichern der Messergebnisse
99  SELECT
100      COUNT(*)
101  INTO vAnzahlDS
102      FROM bank_ohne;
103      INSERT INTO messung (anzahl_ds, sekunde

```



```

n_mitindex, sekunden_ohneindex)
104     VALUES (vAnzahlDS, vSekundenMitIndex, v
SekundenOhneIndex);
105
106     SET vAnzahlSchleifen = 1 + vAnzahlSchleif
en;
107     END WHILE;
108
109     SET PROFILING = OFF; -- Profiling ausscha
lten
110
111     END//
112     DELIMITER ;

```

Bei diesem Versuch wurde die Engine MyISAM verwendet. Ich hatte eigentlich keinen besonderen Grund, diese zu wählen, fragte mich aber, ob die InnoDB die gleichen Ergebnisse liefern würden. Also: Umbau des Versuchs und Neustart! Das Ergebnis ist in [Bild 27.1](#) zu sehen. Das Einfügen in einer Tabelle ohne Index unterscheidet sich bei den beiden Engines kaum in der Ausführungsgeschwindigkeit. Anders sieht es bei der Tabelle mit Index aus. Hier ist die InnoDB deutlich schneller als MyISAM.

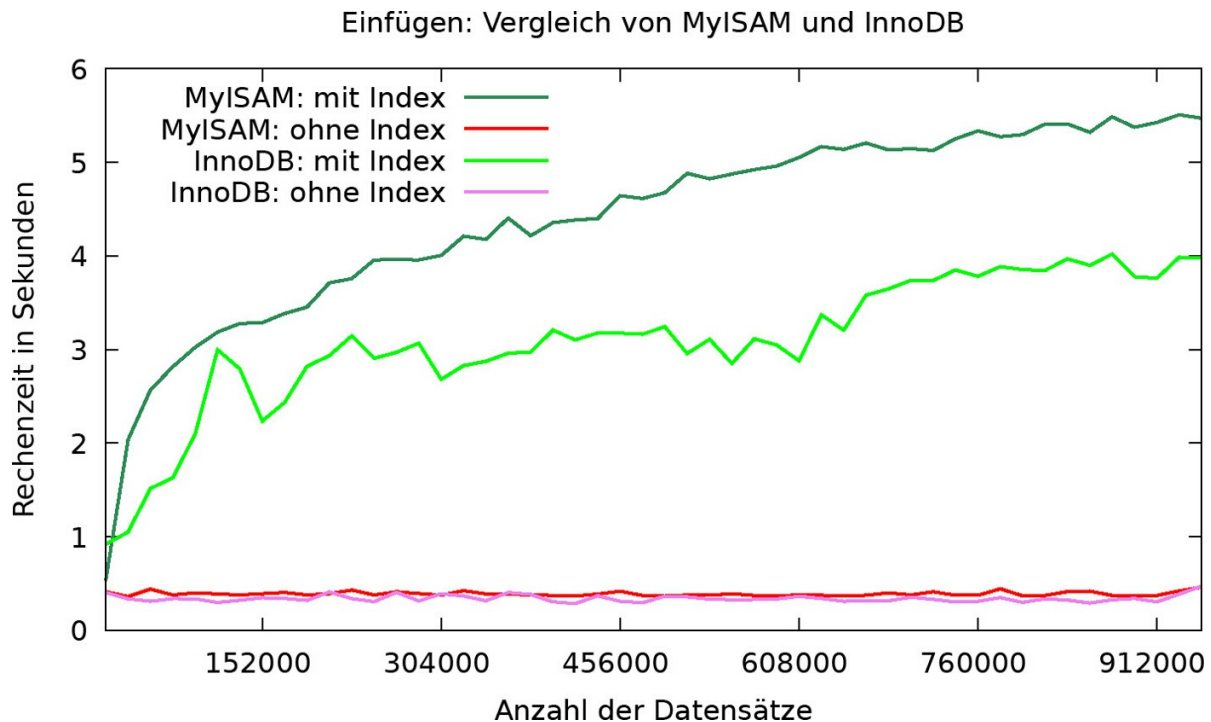


Bild 27.1 Einfügen: Vergleich von MyISAM und InnoDB



Hinweis: Warum habe ich die Bezeichnung bei der Indexbildung in [Zeile 33](#) auf 200 Zeichen begrenzt? Nun, wenn ich das nicht tue, erhalte ich bei MyISAM eine hässliche Fehlermeldung:

ERROR 1071 (42000): Specified key was too long; max key length is 1000 bytes.

Hintergrund ist die UTF8mb4-Kodierung bei MyISAM:
 $4 \times 255 > 1000$.

27.6 Messen der Indexselektivität



Laden Sie dieses Experiment aus dem MySQL-Skriptverzeichnis mit SOURCE indexSelektiv01.sql.

Eine Testumgebung muss her. Diese beinhaltet die Tabelle testdaten, in welche ich die Bankleitzahlen importiere.

```
2  DROP DATABASE IF EXISTS indexSelektiv01;
3  CREATE DATABASE indexSelektiv01
4  CHARACTER SET utf8mb4
5  COLLATE utf8mb4_german2_ci
6  ;
7
8  USE indexSelektiv01;
9
10 CREATE TABLE testdaten (
11   blz CHAR(8), merkm1 CHAR(1), bezeichnung
   VARCHAR(255),
12   plz CHAR(5), ort VARCHAR(255), kurz VARCHA
   R(255),
13   pan CHAR(5), bic VARCHAR(255), sm CHAR(2),
14   ds CHAR(6), kz CHAR(1), blzl CHAR(1), blzn
   CHAR(8)
15 ) Engine=InnoDB;
16
17 SET GLOBAL local_infile = ON;
18 LOAD DATA LOCAL INFILE 'blz_20120305.csv'
19 INTO TABLE testdaten
20 FIELDS TERMINATED BY ';'
21 LINES TERMINATED BY '\n'
22 IGNORE 1 LINES
23 ;
24 SET GLOBAL local_infile = OFF;
```

In der Tabelle messung wird die Länge des Index und seine Selektivität abgespeichert.

```

26 CREATE TABLE messung (
27   laenge INT UNSIGNED,
28   selektiv DOUBLE
29 ) Engine=MEMORY;

```

In der eigentlichen Messprozedur wird erst mal ein Haufen Variablen bereitgestellt. Die Variable `vLaengeMax` wird die längste String-Länge enthalten, die im Index möglich ist. In `vLaengeAkt` steht die aktuelle Indexlänge, die in der Schleife bei 1 beginnend bis `vLaengeMax` inkrementiert wird.

`vSelektiv` nimmt die berechnete Indexselektivität auf, damit diese später in der Tabelle `messung` abgespeichert werden kann. Verbleibt die Info, wie viele Zeilen überhaupt als Testdaten zur Verfügung stehen. Dazu ist `vAnzahl` da.

```

33 DELIMITER //
34 CREATE PROCEDURE indexSelektiv01()
35 BEGIN
36
37   DECLARE vLaengeAkt INT UNSIGNED; -- Schleifenvariable
38   DECLARE vLaengeMax INT UNSIGNED; -- Maximale Stringlänge
39   DECLARE vSelektiv DOUBLE; -- Gemessene Selektivität
40   DECLARE vAnzahl INT UNSIGNED; -- Anzahl der Testdaten

```

Jetzt können die Gesamtanzahl und die maximale String-Länge bestimmt werden.

```

42 -- Testdatenanzahl ermitteln
43 SELECT
44   COUNT(*)
45 INTO vAnzahl
46 FROM testdaten
47 ;
48 -- Maximale Stringlänge ermitteln
49 SELECT
50   MAX(LENGTH(bezeichnung))
51 INTO vLaengeMax
52 FROM

```

testdaten 53 ;

In der Hauptschleife wird die Anzahl der unterscheidbaren Inhalte der Länge vLaengeAkt in der Spalte bezeichnung gezählt. Diese muss durch die Anzahl aller Zeilen dividiert werden (siehe Formel zur Indexselektivität in [Definition 32](#)). Die Schleife selbst erhöht die Länge bei jeder Iteration um 1.

```
56 SET vLaengeAkt = 1;
57 WHILE vLaengeAkt <= vLaengeMax DO
58     -- Messung der Selektivität
59     SELECT
60         COUNT(DISTINCT LEFT(bezeichnung, vLaenge
Akt)) / vAnzahl
61     INTO vSelektiv
62     FROM testdaten
63 ;
64
65     -- Speichern der Messung
66     INSERT INTO messung(laenge, selektiv)
67     VALUES (vLaengeAkt, vSelektiv);
68     SELECT 'Laenge ', vLaengeAkt, ' von ', v
LaengeMax, ' mit ', vSelektiv;
69
70 SET vLaengeAkt = 1 + vLaengeAkt;
71 END WHILE;
72 END//
73 DELIMITER ;
```

Und der Aufruf der Testprozedur:

```
75 CALL indexSelektiv01();
```

27.7 Sortieren ohne und mit Index



Laden Sie dieses Experiment aus dem MySQL-Skriptverzeichnis mit SOURCE
perfIndexOrderBy01.sql.

Zuerst wird – wie üblich – eine Testumgebung bereitgestellt. Dazu gehören eine Datenbank, eine Tabelle ohne Index, eine Tabelle mit Index und eine Tabelle für die Messergebnisse.

```
2  DROP DATABASE IF EXISTS perfIndexOrderBy01
;
3  CREATE DATABASE perfIndexOrderBy01
4  CHARACTER SET utf8mb4
5  COLLATE utf8mb4_german2_ci
6  ;
7
8  USE perfIndexOrderBy01;
9
10 -- Tabelle ohne Index
11 CREATE TABLE name_ohne (
12 id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY
13 ,
14 nachname VARCHAR(255),
15 vorname VARCHAR(255)
16 ) Engine=InnoDB;
17
18 -- Tabelle mit Index
19 CREATE TABLE name_mit
20 LIKE name_ohne;
21
22 -- Index erstellen
23 CREATE INDEX idx_namemit_nnvn ON name_mit(
nachname, vorname);
24
25 -- Tabelle fuer die Messergebnisse
26 CREATE TABLE messung (
```

```

26     anzahl_ds BIGINT UNSIGNED,
27     sekunden_ohne DECIMAL(12,8),
28     sekunden_mit DECIMAL(12,8)
29 ) ENGINE=MEMORY ;
30
31 DROP VIEW IF EXISTS view_messung;
32 CREATE VIEW view_messung AS
33 SELECT
34     anzahl_ds, AVG(sekunden_ohne) AS ohne, A
VG(sekunden_mit) AS mit
35     FROM messung
36     GROUP BY anzahl_ds
37     ORDER BY anzahl_ds
38 ;

```

In der Prozedur datenerzeugen() werden, gesteuert durch den Übergabeparameter iAnzahl, zufällige Testdaten in die beiden Tabellen eingefügt. Die Inhalte der Datensätze sind dabei gleich, nur der Index unterscheidet die beiden Tabellen.

```

42 DELIMITER //
43 CREATE PROCEDURE datenerzeugen
44 (
45     IN iAnzahl INT UNSIGNED
46 )
47 BEGIN
48     DECLARE vAnzahl INT UNSIGNED DEFAULT 0;
49     DECLARE vNachname VARCHAR(255);
50     DECLARE vVorname VARCHAR(255);
51
52     WHILE vAnzahl < iAnzahl DO
53         SELECT MD5(RAND()) INTO vNachname;
54         SELECT MD5(RAND()) INTO vVorname;
55         INSERT INTO name_ohne (nachname, vorname
) VALUES (vNachname, vVorname);
56         INSERT INTO name_mit (nachname, vorname)
VALUES (vNachname, vVorname);
57         SET vAnzahl = 1 + vAnzahl;

```

```

58  END WHILE;
59  END//
60  DELIMITER ;

```

Die Prozedur `dauer()` ermittelt den Zeitverbrauch der zuletzt durchgeführten Anweisung (Näheres in [Abschnitt 27.1](#)). Der Zeitverbrauch wird in die OUT-Variable `oSekunden` geschrieben und steht daher der aufrufenden Prozedur zur Verfügung.

```

63  DELIMITER //
64  CREATE PROCEDURE dauer
65  (
66  OUT oSekunden DECIMAL(12,8)
67  )
68  BEGIN
69  SELECT
70      SUM(DURATION)
71      INTO oSekunden
72      FROM INFORMATION_SCHEMA.PROFILING
73      GROUP BY query_id
74      ORDER BY query_id DESC LIMIT 1
75  ;
76  END//
77  DELIMITER ;

```

Das eigentliche Experiment geht nach bekanntem Muster vor. Die Variable `vAnzahl` ist die Schleifenvariable für die Messschleife. Sie wird bei jedem Schleifendurchlauf um 1 inkrementiert und mit dem Übergabeparameter `iAnzahl` in der Schleifenbedingung verglichen. In `vAnzahlDS` wird die aktuelle Anzahl der Testdaten abgelegt. `vDauerMit` und `vDauerOhne` enthalten die Ausführungszeiten der jeweiligen Sortieroperationen. Der `vDummy` wird nur dazu gebraucht, das Ergebnis des `SELECT` irgendwo hinzuspeichern, damit es die Bildschirmausgabe nicht stört. `vN` ist der Schleifenzähler der

Stichprobengröße. Anschließend werden alle Tabellen zurückgesetzt und das Profiling eingeschaltet.

```
81 DELIMITER //
82 CREATE PROCEDURE perfIndexOrderBy01
83 (
84   IN iAnzahl INT UNSIGNED
85 )
86 BEGIN
87   DECLARE vAnzahl INT UNSIGNED DEFAULT 0;
88   DECLARE vAnzahlDS BIGINT UNSIGNED DEFAULT
89   0;
90   DECLARE vDauerOhne DECIMAL(12,8);
91   DECLARE vDauerMit DECIMAL(12,8);
92   DECLARE vDummy VARCHAR(255);
93   DECLARE vN INT UNSIGNED DEFAULT 0;
94   TRUNCATE messung; -- Alte Daten entferne
95   TRUNCATE name_ohne; -- Alte Daten entfer
96   TRUNCATE name_mit; -- Alte Daten entfer
97   SET PROFILING = 1; -- Profiling einschalte
98   n
```

In der Hauptschleife wird jeweils ein SELECT mit einem ORDER BY abgesetzt: einmal auf die Tabelle ohne und einmal auf die Tabelle mit Index. Unmittelbar nach dem SELECT wird die Zeit der Ausführung gemessen. Zuletzt werden die Messergebnisse in die Tabelle messung eingefügt und die Schleifenvariable inkrementiert.

```
101 SET vAnzahl = 1;
102 WHILE vAnzahl <= iAnzahl DO
103   -- SELECT 'Durchlauf ', vAnzahl, ' von
```

```

', iAnzahl;
104    -- Testdaten erweitern
105    CALL datenerzeugen(10);
106    SET vN = 0;
107    WHILE vN < 20 DO
108        SELECT SQL_NO_CACHE id -- Messung ohne
109            INTO vDummy
110            FROM name_ohne
111            ORDER BY nachname, vorname
112            LIMIT 1;
113        CALL dauer(vDauerOhne);
114
115        SELECT SQL_NO_CACHE id -- Messung mit
116            INTO vDummy
117            FROM name_mit
118            ORDER BY nachname, vorname
119            LIMIT 1;
120        CALL dauer(vDauerMit);
121
122        SELECT COUNT(*) -- Sichern der Messer
123        gebnisse
124        INTO vAnzahlDS
125        FROM name_ohne;
126        INSERT INTO messung(anzahl_ds, sekunden_ohne, sekunden_mit)
127        VALUES (vAnzahlDS, vDauerOhne, vDauerMit);
128    SET vN = 1 + vN;
129    END WHILE;
130    SET vAnzahl = 1 + vAnzahl;
131    END WHILE;

```

Nach der Schleife wird das Profiling wieder ausgeschaltet und der DELIMITER wieder auf das Semikolon gesetzt. Wenn Sie die Testergebnisse nachvollziehen wollen, müssen Sie CALL perfIndexOrderBy01(150); ausführen.

```
132 SET PROFILING = 0; -- Profiling ausschalt  
en  
133 END//  
134 DELIMITER ;
```

¹ Siehe [[Bun16](#)] und dort besonders die Anmerkung zur Verfügbarkeit.

28 Quelltexte

Damit Sie direkt die Quelltexte und Beispiele studieren können, ohne dass dazu ein Rechner parallel zum E-Book laufen muss, sind diese hier weitestgehend abgedruckt. Natürlich sind diese nicht selbsterklärend und nur zum Nachschlagen geeignet.

28.1 MySQL/MariaDB

28.1.1 Quelltexte zu Teil II

Listing 28.1 mysql/listing01.sql

```
1  -- Warum formatiere ich den Quelltext so
   ganz anders, als beispielsweise diese Beautyfi
   er?
2  -- * http://www.dpriver.com/pp/sqlformat.
   htm
3  -- * https://codebeautify.org/sqlformatte
   r
4  -- Ich finde die Arbeit dieser Programme
   sehr gut und auch die Ergebnisse sind ansprech
   end,
5  -- aber sie widersprechen meinem Anspruch
   an die leichte Verständlichkeit der Befehle.
```

```

6  -- Zwei Beispiele:
7  -- * Es werden keine Einrückungen eingefügt, die den inhaltlichen Zusammenhang
8  -- zwischen dem Befehl und seinen Parametern abbilden:
9  -- SELECT blz,
10 -- bankname
11 -- FROM bank
12 -- ORDER BY blz DESC
13 -- LIMIT 1;
14 --
15 -- anstelle von
16 --
17 -- SELECT
18 -- blz, bankname
19 -- FROM bank
20 -- ORDER BY blz DESC
21 -- LIMIT 1
22 -- ;
23
24 -- * Hinter dem FROM wird ein Tabellename gestellt, obwohl die Auswertung aus dem ganzen JOIN erfolgt:
25 -- SELECT k.kunde_id,
26 -- k.nachname,
27 -- k.vorname,
28 -- b.datum
29 -- FROM bestellung b
30 -- INNER JOIN kunde k USING (kunde_id)
31 -- ORDER BY k.nachname,
32 -- k.vorname;
33
34 -- anstelle von
35
36 -- SELECT
37 -- k.kunde_id, k.nachname, k.vorname, b.d
atum
38 -- FROM

```

```

39  -- bestellung b INNER JOIN kunde k USING
(kunde_id)
40  -- ORDER BY
41  -- k.nachname, k.vorname
42  -- ;
43
44  -- Ich werde daher meine Darstellung beib
halten.
45
46
47  -- Anlegen der Datenbank auf der grünen W
iese
48  SELECT 'Start listing01. sql //////////////////
////////////////////////////////////////
////////////////////////////////////////';
49  SET NAMES utf8;
50  SET AUTOCOMMIT=ON;
51  SELECT 'Löschen einer ggf. vorhandenen Da
tenbank oshop (Abschnitt 5.2.3)';
52  DROP DATABASE IF EXISTS oshop;
53
54  SELECT 'Anzeigen der verfügbaren Zeichenk
odierungen (Abschnitt 5.2.4)';
55  SHOW CHARACTER SET;
56  SHOW CHARACTER SET LIKE 'latin';
57  SHOW CHARACTER SET LIKE 'utf%';
58
59  SELECT 'Anzeigen der verfügbaren Sortieru
ngen (Abschnitt 5.2.5)';
60  SHOW COLLATION;
61  SHOW COLLATION LIKE 'latin1%';
62  SHOW COLLATION LIKE 'utf8mb4%';
63
64  SELECT 'Anlegen der Datenbank mit Zeichen
satz und Sortierung (Abschnitt 5.2.1)';
65  CREATE DATABASE oshop
66  DEFAULT CHARACTER SET utf8mb4
67  COLLATE utf8mb4_german2_ci
68  ;

```

```

69
70 SELECT 'Anzeigen aller Datenbanken (Abschnitt 5.2.5)';
71 SHOW DATABASES;
72 SHOW CREATE DATABASE oshop;
73 SELECT 'Ende listing01. sql //////////////////////////////////
////////////////////////////////////////';

```

Listing 28.2 mysql/listing02.sql

```

1  -- Ausführen der vorherigen Befehle
2  SOURCE listing01.sql
3
4  SELECT 'Start listing02. sql //////////////////////////////////
////////////////////////////////////////';
5
6  -- Erspare mir ein wenig Tipparbeit
7  USE oshop; -- Abschnitt 5.3.2
8
9  -- adresse
10 SELECT 'Tabelle adresse anlegen (Abschnitt 5.3.5)';
11 CREATE TABLE adresse (
12     adresse_id INT UNSIGNED AUTO_INCREMENT,
13     strasse VARCHAR(255) NOT NULL DEFAULT '',
14     hnr VARCHAR(255) NOT NULL DEFAULT '',
15     lkz CHAR(2) NOT NULL DEFAULT '',
16     plz CHAR(9) NOT NULL DEFAULT '',
17     ort VARCHAR(255) NOT NULL DEFAULT '',
18     deleted TINYINT UNSIGNED NOT NULL DEFAULT
19     0,
20     PRIMARY KEY(adresse_id)
21 ) ENGINE=InnoDB;
22
23 -- kunde
24 SELECT 'Tabelle kunde mit Fremdschlüssel
anlegen (Abschnitt 5.3.6)';
25 CREATE TABLE kunde (

```

```

25  kunde_id INT UNSIGNED AUTO_INCREMENT,
26  nachname VARCHAR(255) NOT NULL DEFAULT ''
',
27  vorname VARCHAR(255) NOT NULL DEFAULT '',
28  rechnung_adresse_id INT UNSIGNED,
29  liefer_adresse_id INT UNSIGNED,
30  bezahlart INT UNSIGNED NOT NULL DEFAULT 0
',
31  art ENUM('unb', 'prv', 'gsch') NOT NULL D
EFAULT 'unb',
32  deleted TINYINT UNSIGNED NOT NULL DEFAULT
0,
33  PRIMARY KEY(kunde_id),
34  FOREIGN KEY(rechnung_adresse_id)
35  REFERENCES adresse(adresse_id)
36  ON UPDATE CASCADE
37  ON DELETE RESTRICT,
38  FOREIGN KEY(liefer_adresse_id)
39  REFERENCES adresse(adresse_id)
40  ON UPDATE CASCADE
41  ON DELETE SET NULL
42  ) ENGINE=InnoDB;
43
44  -- bank
45  SELECT 'Tabelle bank anlegen (Abschnitt 5
.3.2)';
46  CREATE TABLE bank (
47  bank_id CHAR(12),
48  bankname VARCHAR(255) NOT NULL DEFAULT ''
',
49  lkz CHAR(2) NOT NULL DEFAULT 'DE',
50  deleted TINYINT UNSIGNED NOT NULL DEFAULT
0,
51  PRIMARY KEY(bank_id)
52  ) ENGINE=InnoDB;
53
54  -- bankverbindung
55  SELECT 'Tabelle bankverbindung mit Fremds

```



```

chlüssel anlegen (Abschnitt 5.3.2);
56 CREATE TABLE bankverbindung (
57 kunde_id INT UNSIGNED,
58 bankverbindung_nr INT UNSIGNED,
59 bank_id CHAR(12) NOT NULL,
60 kontonummer CHAR(25) NOT NULL DEFAULT '',
61 iban CHAR(34) NOT NULL DEFAULT '',
62 deleted TINYINT UNSIGNED NOT NULL DEFAULT
0,
63 PRIMARY KEY(kunde_id, bankverbindung_nr),
64 FOREIGN KEY(kunde_id)
65 REFERENCES kunde(kunde_id)
66 ON UPDATE RESTRICT
67 ON DELETE RESTRICT,
68 FOREIGN KEY(bank_id)
69 REFERENCES bank(bank_id)
70 ON UPDATE RESTRICT
71 ON DELETE RESTRICT
72 ) ENGINE=InnoDB;
73
74
75 -- artikel
76 SELECT 'Tabelle artikel anlegen';
77 CREATE TABLE artikel (
78 artikel_id INT UNSIGNED AUTO_INCREMENT,
79 bezeichnung VARCHAR(255) NOT NULL DEFAULT
'',
80 einzelpreis DECIMAL(14,6) NOT NULL DEFAULT
0.0,
81 waehrung CHAR(3) DEFAULT 'EUR',
82 deleted TINYINT UNSIGNED NOT NULL DEFAULT
0,
83 PRIMARY KEY(artikel_id)
84 ) ENGINE=InnoDB;
85
86
87 -- warengruppe
88 SELECT 'Tabelle warengruppe anlegen';

```

```

89 CREATE TABLE warengruppe (
90 warengruppe_id INT UNSIGNED AUTO_INCREMENT,
91 bezeichnung VARCHAR(255) NOT NULL DEFAULT
  '',
92 deleted TINYINT UNSIGNED NOT NULL DEFAULT
  0,
93 PRIMARY KEY(warengruppe_id)
94 ) ENGINE=InnoDB;
95
96
97 -- lieferant
98 SELECT 'Tabelle lieferant mit Fremdschlüssel
anlegen';
99 CREATE TABLE lieferant (
100 lieferant_id INT UNSIGNED AUTO_INCREMENT,
101 firmenname VARCHAR(255) NOT NULL DEFAULT
  '',
102 adresse_id INT UNSIGNED NOT NULL DEFAULT
  0,
103 deleted TINYINT UNSIGNED NOT NULL DEFAULT
  0,
104 PRIMARY KEY(lieferant_id),
105 FOREIGN KEY(adresse_id)
106 REFERENCES adresse(adresse_id)
107 ON UPDATE RESTRICT
108 ON DELETE RESTRICT
109 ) ENGINE=InnoDB;
110
111
112 -- Hilfstabelle artikel zu warengruppe
113 SELECT 'Hilfstabelle artikel_nm_warengruppe
anlegen';
114 CREATE TABLE artikel_nm_warengruppe (
115 warengruppe_id INT UNSIGNED,
116 artikel_id INT UNSIGNED,
117 PRIMARY KEY(warengruppe_id, artikel_id),
118 FOREIGN KEY(warengruppe_id)

```

```

119     REFERENCES warengruppe(warengruppe_id)
120     ON UPDATE RESTRICT
121     ON DELETE RESTRICT,
122 FOREIGN KEY(artikel_id)
123     REFERENCES artikel(artikel_id)
124     ON UPDATE RESTRICT
125     ON DELETE RESTRICT
126 ) ENGINE=InnoDB;
127
128
129 -- Hilfstabelle artikel zu lieferant
130 SELECT 'Hilfstabelle artikel_nm_lieferant
    anlegen';
131 CREATE TABLE artikel_nm_lieferant (
132     lieferant_id INT UNSIGNED,
133     artikel_id INT UNSIGNED,
134     PRIMARY KEY(lieferant_id, artikel_id),
135     FOREIGN KEY(lieferant_id)
136         REFERENCES lieferant(lieferant_id)
137         ON UPDATE RESTRICT
138         ON DELETE RESTRICT,
139     FOREIGN KEY(artikel_id)
140         REFERENCES artikel(artikel_id)
141         ON UPDATE RESTRICT
142         ON DELETE RESTRICT
143 ) ENGINE=InnoDB;
144
145
146 -- bestellung
147 SELECT 'Tabelle bestellung mit Fremdschlü
    ssel anlegen';
148 CREATE TABLE bestellung (
149     bestellung_id INT UNSIGNED AUTO_INCREMENT
150     ,
151     kunde_id INT UNSIGNED NOT NULL DEFAULT 0,
152     adresse_id INT UNSIGNED NOT NULL DEFAULT
    0,
153     datum DATETIME NOT NULL DEFAULT '1000-01-

```

```

01 00:00:00',
153 status ENUM('offen', 'versendet', 'angeko
mmen', 'retour', 'bezahlt') NOT NULL DEFAULT '
offen',
154 deleted TINYINT UNSIGNED NOT NULL DEFAULT
0,
155 PRIMARY KEY(bestellung_id),
156 FOREIGN KEY(kunde_id)
157 REFERENCES kunde(kunde_id)
158     ON UPDATE RESTRICT
159     ON DELETE RESTRICT,
160 FOREIGN KEY(adresse_id)
161 REFERENCES adresse(adresse_id)
162     ON UPDATE RESTRICT
163     ON DELETE RESTRICT
164 ) ENGINE=InnoDB;
165
166 -- position der bestellung
167 SELECT 'Tabelle bestellung_position mit F
remdschlüssel anlegen';
168 CREATE TABLE bestellung_position (
169 bestellung_id INT UNSIGNED,
170 position_nr INT UNSIGNED,
171 artikel_id INT UNSIGNED NOT NULL DEFAULT
0,
172 menge DECIMAL(14,6) NOT NULL DEFAULT 0.0,
173 deleted TINYINT UNSIGNED NOT NULL DEFAULT
0,
174 PRIMARY KEY(bestellung_id, position_nr),
175 FOREIGN KEY(bestellung_id)
176     REFERENCES bestellung(bestellung_id)
177     ON UPDATE RESTRICT
178     ON DELETE RESTRICT,
179 FOREIGN KEY(artikel_id)
180     REFERENCES artikel(artikel_id)
181     ON UPDATE RESTRICT
182     ON DELETE RESTRICT
183 ) ENGINE=InnoDB;

```

```

184
185
186 -- rechnung
187 SELECT 'Tabelle rechnung mit Fremdschlüssel anlegen';
188 CREATE TABLE rechnung (
189 rechnung_id INT UNSIGNED AUTO_INCREMENT,
190 kunde_id INT UNSIGNED NOT NULL DEFAULT 0,
191 bestellung_id INT UNSIGNED NOT NULL DEFAULT 0,
192 adresse_id INT UNSIGNED NOT NULL DEFAULT 0,
193 bezahlart ENUM('lastschrift', 'rechnung', 'kredit', 'bar') NOT NULL DEFAULT 'lastschrift',
194 datum DATETIME NOT NULL DEFAULT '1000-01-01 00:00:00',
195 status ENUM('offen', 'gestellt', 'mahnung 1', 'inkasso', 'storno', 'beglichen') NOT NULL DEFAULT 'offen',
196 rabatt DECIMAL(6,2) NOT NULL DEFAULT 0.0,
197 skonto DECIMAL(6,2) NOT NULL DEFAULT 0.0,
198 deleted TINYINT UNSIGNED NOT NULL DEFAULT 0,
199 PRIMARY KEY(rechnung_id),
200 FOREIGN KEY(kunde_id)
201 REFERENCES kunde(kunde_id)
202 ON UPDATE RESTRICT
203 ON DELETE RESTRICT,
204 FOREIGN KEY(adresse_id)
205 REFERENCES adresse(adresse_id)
206 ON UPDATE RESTRICT
207 ON DELETE RESTRICT
208 ) ENGINE=InnoDB;
209
210
211 -- position der rechnung
212 SELECT 'Tabelle rechnung_position mit Fre

```

```

mdschlüssel anlegen';
213 CREATE TABLE rechnung_position (
214     rechnung_id INT UNSIGNED,
215     position_nr INT UNSIGNED,
216     artikel_id INT UNSIGNED NOT NULL DEFAULT
0,
217     steuer DECIMAL(14,6) NOT NULL DEFAULT 19.
0,
218     menge DECIMAL(14,6) NOT NULL DEFAULT 0.0,
219     deleted TINYINT UNSIGNED NOT NULL DEFAULT
0,
220     PRIMARY KEY(rechnung_id, position_nr),
221     FOREIGN KEY(rechnung_id)
222         REFERENCES rechnung(rechnung_id)
223         ON UPDATE RESTRICT
224         ON DELETE RESTRICT,
225     FOREIGN KEY(artikel_id)
226         REFERENCES artikel(artikel_id)
227         ON UPDATE RESTRICT
228         ON DELETE RESTRICT
229 ) ENGINE=InnoDB;
230
231 SELECT 'Neue Tabelle mit Struktur einer a
lten anlegen (Abschnitt 5.3.7)';
232 CREATE TABLE wurstbrot LIKE adresse;
233
234 SELECT 'Neue Tabelle sofort wieder lösche
n, da nicht mehr gebraucht';
235 DROP TABLE wurstbrot;
236
237 SELECT 'Temporäre Tabelle anlegen (Abschn
itt 5.3.7)';
238 CREATE TEMPORARY TABLE tmp_kunde LIKE kun
de;
239
240 SHOW TABLES;
241 DROP TABLE tmp_kunde;
242 SELECT 'Ende listing02. sql //////////////////////////////////

```

```
/////////////////////////////////////////';
```

Listing 28.3 mysql/listing03.sql

```
1  -- Ausführen der vorherigen Befehle
2  SOURCE listing02.sql
3
4  SELECT 'Start listing03. sql //////////////////////////////////
/////////////////////////////////////////';
5
6  -- Abschnitt 6.1.1
7  SHOW INDEX FROM kunde\G
8
9  -- Anlegen der Indizes nach Liste auf Abschnitt 6.1.2
10 SELECT 'CREATE INDEX idx_kunde_nachname_v
orname';
11 CREATE INDEX idx_kunde_nachname_vorname
12 ON kunde (nachname, vorname)
13 ;
14
15 SELECT 'CREATE INDEX idx_bank_bankname';
16 CREATE INDEX idx_bank_bankname
17 ON bank (bankname)
18 ;
19
20 SELECT 'CREATE INDEX idx_bankverbindung_b
ankid_kontonummer';
21 CREATE INDEX idx_bankverbindung_bankid_ko
ntonummer
22 ON bankverbindung (bank_id, kontonummer)
23 ;
24
25 SELECT 'CREATE UNIQUE INDEX idx_bankverbi
ndung_iban';
26 CREATE UNIQUE INDEX idx_bankverbindung_ib
an
27 ON bankverbindung (iban)
```

```

28  ;
29
30  SELECT 'CREATE INDEX idx_artikel_bezeichnung
ung';
31  CREATE INDEX idx_artikel_bezeichnung
32  ON artikel (bezeichnung)
33  ;
34
35  SELECT 'CREATE INDEX idx_warengruppe_bezei
chnung';
36  CREATE INDEX idx_warengruppe_bezeichnung
37  ON warengruppe (bezeichnung)
38  ;
39
40  SELECT 'CREATE INDEX idx_lieferant_firmen
name';
41  CREATE INDEX idx_lieferant_firmenname
42  ON lieferant (firmenname)
43  ;
44
45  SELECT 'CREATE INDEX idx_bestellung_kunde
id_datum';
46  CREATE INDEX idx_bestellung_kundeid_datum
47  ON bestellung (kunde_id, datum)
48  ;
49
50
51  SELECT 'CREATE UNIQUE INDEX idx_adresse_d
ublette (Abschnitt 6.2.2)';
52  CREATE UNIQUE INDEX idx_adresse_dublette
53  ON adresse (strasse(100), hnr(100), plz);
54
55  -- ENABLE / DISABLE
56  -- SELECT 'DISABLE/ENABLE KEYS: Erwarte z
wei Warnungen!';
57  -- ALTER TABLE bankverbindung DISABLE KEY
S; SHOW WARNINGS\G
58  -- ALTER TABLE bankverbindung ENABLE KEYS

```



```

; SHOW WARNINGS\G
59
60  SELECT 'Ende listing03. sql //////////////////////////////////
////////////////////////////////////////';

```

Listing 28.4 mysql/listing04.sql

```

1  -- Ausführen der vorherigen Befehle
2  SOURCE listing03.sql
3
4  SELECT 'Start listing04. sql //////////////////////////////////
////////////////////////////////////////';
5  -- Abschnitt 7.1.2
6  -- Alternativ: SET @@local_infile = 1;
7  SET GLOBAL local_infile = ON;
8
9  SELECT 'Importieren von artikel01.csv (Ab
schnitt 7.1.2)';
10  LOAD DATA LOCAL INFILE 'artikel01.csv'
11  INTO TABLE oshop.artikel
12  CHARACTER SET utf8mb4
13  FIELDS
14  TERMINATED BY ';'
15  LINES
16  TERMINATED BY '\n'
17  ;
18
19  SELECT 'Löschen der schlecht importierten
Daten (Abschnitt 7.1.2)';
20  DELETE FROM oshop.artikel;
21
22  SET GLOBAL local_infile = ON;
23  SELECT 'Importieren von artikel01.csv (Ab
schnitt 7.1.2)';
24  LOAD DATA LOCAL INFILE 'artikel01.csv'
25  INTO TABLE oshop.artikel
26  CHARACTER SET utf8mb4
27  FIELDS

```

```

28     TERMINATED BY ';'
29     LINES
30     TERMINATED BY '\n'
31     IGNORE 1 LINES
32     (artikel_id, bezeichnung, einzelpreis,
waehrung)
33     SET deleted=0
34 ;
35
36 -- Start REPLACE oder IGNORE
37 SELECT 'Beispiel artikel02.csv mit IGNORE
(Abschnitt 7.1.3)';
38 LOAD DATA LOCAL INFILE 'artikel02.csv' IG
NORE
39 CHARACTER SET utf8mb4
40 INTO TABLE oshop.artikel
41 FIELDS
42 TERMINATED BY ';'
43 LINES
44 TERMINATED BY '\n'
45 IGNORE 1 LINES
46 (artikel_id, bezeichnung, einzelpreis,
waehrung)
47 SET deleted=0
48 ;
49 SELECT * FROM artikel;
50
51 SELECT 'Beispiel artikel02.csv mit IGNORE
(Abschnitt 7.1.3)';
52 DELETE FROM artikel;
53
54 LOAD DATA LOCAL INFILE 'artikel01.csv'
55 INTO TABLE oshop.artikel
56 CHARACTER SET utf8mb4
57 FIELDS
58 TERMINATED BY ';'
59 LINES
60 TERMINATED BY '\n'

```

```

61     IGNORE 1 LINES
62     (artikel_id, bezeichnung, einzelpreis,
waehrung)
63     SET deleted=0
64 ;
65
66 LOAD DATA LOCAL INFILE 'artikel02.csv' RE
PLACE
67 INTO TABLE oshop.artikel
68     CHARACTER SET utf8mb4
69     FIELDS
70     TERMINATED BY ';'
71     LINES
72     TERMINATED BY '\n'
73     IGNORE 1 LINES
74     (artikel_id, bezeichnung, einzelpreis,
waehrung)
75     SET deleted=0
76 ;
77 SET GLOBAL local_infile = OFF;
78
79 SELECT * FROM artikel;
80
81 -- Ende REPLACE oder IGNORE
82
83 -- Start INSERT
84
85 SELECT 'Anlegen mit INSERT INTO ... VALUE
S (Abschnitt 7.2.1)';
86 SELECT 'Warengruppen anlegen';
87 INSERT INTO warengruppe (warengruppe_id,
bezeichnung)
88 VALUES
89     (1, 'Bürobedarf')
90     ,(2, 'Pflanzen')
91     ,(3, 'Gartenbedarf')
92     ,(4, 'Werkzeug')
93 ;

```

```
94  SELECT * FROM warengruppe;
95
96  SELECT 'Anlegen mit INSERT INTO ... SET (  
Abschnitt 7.2.2)';
97  SELECT 'Warengruppen verbinden';
98  INSERT INTO artikel_nm_warengruppe
99  SET
100     artikel_id=3001
101     ,warengruppe_id=1
102  ;
103  INSERT INTO artikel_nm_warengruppe
104  SET
105     artikel_id=3005
106     ,warengruppe_id=1
107  ;
108  INSERT INTO artikel_nm_warengruppe
109  SET
110     artikel_id=3006
111     ,warengruppe_id=1
112  ;
113  INSERT INTO artikel_nm_warengruppe
114  SET
115     artikel_id=3007
116     ,warengruppe_id=1
117  ;
118  INSERT INTO artikel_nm_warengruppe
119  SET
120     artikel_id=3010
121     ,warengruppe_id=1
122  ;
123  INSERT INTO artikel_nm_warengruppe
124  SET
125     artikel_id=7856
126     ,warengruppe_id=2
127  ;
128  INSERT INTO artikel_nm_warengruppe
129  SET
130     artikel_id=7856
```

```
131     ,warengruppe_id=3
132 ;
133 INSERT INTO artikel_nm_warengruppe
134 SET
135     artikel_id=7863
136     ,warengruppe_id=2
137 ;
138 INSERT INTO artikel_nm_warengruppe
139 SET
140     artikel_id=7863
141     ,warengruppe_id=3
142 ;
143 INSERT INTO artikel_nm_warengruppe
144 SET
145     artikel_id=9010
146     ,warengruppe_id=3
147 ;
148 INSERT INTO artikel_nm_warengruppe
149 SET
150     artikel_id=9010
151     ,warengruppe_id=4
152 ;
153 INSERT INTO artikel_nm_warengruppe
154 SET
155     artikel_id=9015
156     ,warengruppe_id=3
157 ;
158 INSERT INTO artikel_nm_warengruppe
159 SET
160     artikel_id=9015
161     ,warengruppe_id=4
162 ;
163 -- Ende INSERT
164
165 -- Start Kundendaten
166 SELECT 'Testdaten INSERT adresse';
167
168 -- adresse
```

```

169 INSERT INTO adresse (adresse_id, strasse,
    hnr, lkz, plz, ort)
170 VALUES
171     (1, 'Beutelhaldenweg', '5', 'AL', '6767
172     6', 'Hobbingen')
173     ,(2, 'Beutelhaldenweg', '1', 'AL', '676
174     76', 'Hobbingen')
175     ,(3, 'Auf der Feste', '1', 'GO', '54786
176     ', 'Minas Tirith')
177     ,(4, 'Letztes Haus', '4', 'ER', '87567'
178     , 'Bruchtal')
179     ,(5, 'Baradur', '1', 'MO', '62519', 'Lu
180     gburz')
181     ,(10, 'Hochstrasse', '4a', 'DE', '44879'
182     , 'Bochum')
183     ,(11, 'Industriegebiet', '8', 'DE', '448
184     78', 'Bochum')
185 ;
186
187 SELECT 'Testdaten INSERT kunde';
188 -- kunde
189 INSERT INTO kunde (kunde_id, rechnung_adr
190     esse_id, nachname, vorname, art)
191 VALUES
192     (1, 1, 'Gamdschie', 'Samweis', 'prv')
193     ,(2, 2, 'Beutlin', 'Frodo', 'prv')
194     ,(3, 2, 'Beutlin', 'Bilbo', 'prv')
195     ,(4, 3, 'Telcontar', 'Elessar', 'prv')
196     ,(5, 4, 'Earendilionn', 'Elrond', 'gsch'
197     )
198 ;
199
200 SELECT 'Lieferadresse bei zwei Adressen a
201     endern';
202 UPDATE kunde SET liefer_adresse_id = 2 WH
203     ERE kunde_id = 1;
204 UPDATE kunde SET liefer_adresse_id = 4 WH
205     ERE kunde_id = 3;

```

```

194  -- Ende Kundendaten
195
196  -- Start Aufbau von 2 Bestellungen
197  SELECT 'Aufbau von 2 Bestellungen: Bestel
ldaten';
198  INSERT INTO bestellung (kunde_id, adresse
_id, datum)
199  VALUES
200  (1, 1, '2012-03-24 17:41:00')
201  ,(2, 2, '2012-03-23 16:11:00')
202  ;
203
204  SELECT 'Aufbau von 2 Bestellungen: Bestel
lpositionen';
205  INSERT INTO bestellung_position (bestellu
ng_id, position_nr, artikel_id, menge)
206  VALUES
207  (1, 1, 7856, 30)
208  ,(1, 2, 7863, 50)
209  ,(1, 3, 9015, 1)
210  ,(2, 1, 7856, 10)
211  ,(2, 2, 9010, 5)
212  ;
213  -- Ende Aufbau Bestellung
214
215  -- SELECT 'Drei Fehlermeldung wegen Const
raints (Abschnitt 7.2.3)!';
216  SELECT bestellung_id, kunde_id, adresse_i
d, datum FROM bestellung;
217  -- INSERT INTO bestellung (datum) VALUES
(NOW());
218  -- INSERT INTO bestellung (adresse_id, da
tum) VALUES (10, NOW());
219  -- INSERT INTO adresse (adresse_id, stras
se) VALUES (10, 'Oberer Weg');
220
221  -- Bildttabelle
222  SELECT 'Bildtabelle anlegen (Abschnitt 7.

```

```

2.4)';
223 CREATE TABLE bild (
224     bild_id INT UNSIGNED AUTO_INCREMENT PRI
MARY KEY,
225     bild BLOB,
226     dateiname VARCHAR(255),
227     dateityp VARCHAR(255),
228     dateigroesse BIGINT UNSIGNED,
229     artikel_id INT UNSIGNED REFERENCES arti
kel(artikel_id)
230 );
231
232 SELECT 'Bild importieren (Abschnitt 7.2.5
)';
233 INSERT INTO bild
234 SET
235     bild = LOAD_FILE('/var/lib/mysql/upload
s/spaten01.png'),
236     dateiname = 'spaten01.png',
237     dateityp = 'png',
238     dateigroesse = 7168,
239     artikel_id = 9015
240 ;
241
242 INSERT INTO bild
243 SET
244     bild = LOAD_FILE('/var/lib/mysql/interc
hange/spaten02.png'),
245     dateiname = 'spaten02.png',
246     dateityp = 'png',
247     dateigroesse = 6910,
248     artikel_id = 9015
249 ;
250
251 INSERT INTO bild
252 SET
253     bild = LOAD_FILE('/var/lib/mysql/interc
hange/spaten03.png'),

```



```

254     dateiname = 'spaten03.png',
255     dateityp = 'png',
256     dateigroesse = 7095,
257     artikel_id = 9015
258 ;
259
260 SELECT 'Daten kopieren (Abschnitt 7.3)';
261 DROP TABLE IF EXISTS kunde_beutlin;
262 CREATE TEMPORARY TABLE kunde_beutlin LIKE
    kunde;
263 INSERT INTO kunde_beutlin
264 SELECT *
265 FROM kunde
266 WHERE nachname='Beutlin';
267 SELECT * FROM kunde_beutlin;
268
269 SELECT 'Daten kopieren (Abschnitt 7.3)';
270 DROP TABLE IF EXISTS kunde_namen;
271 CREATE TEMPORARY TABLE kunde_namen LIKE k
unde;
272 INSERT INTO kunde_namen (vorname, nachnam
e)
273 SELECT
274 vorname, nachname
275 FROM kunde;
276 SELECT * FROM kunde_namen;
277
278
279 -- Start DROP (auskommentiert)
280 -- SELECT * FROM warengruppe INTO OUTFILE
    'bla.txt';
281 -- DROP DATABASE oshop;
282 -- SELECT 'Löschen Sie nun MySQL-Datenver
    zeichnis oshop/bla.txt!'
283 -- Ende DROP
284
285 SELECT 'Ende listing04. sql //////////////////////////////////
    //////////////////////////////////';

```

28.1.2 Quelltexte zu Teil III

Listing 28.5 mysql/listing05.sql

```
1  -- Ausführen der vorherigen Befehle
2  SOURCE listing04.sql
3
4  SELECT 'Start listing05. sql //////////////////////////////////
////////////////////////////////////////';
5
6  SELECT 'Aktuelle Datenbankeinstellung (Abschnitt 8.1)';
7  SHOW CREATE DATABASE oshop\G
8
9  SELECT 'DATABASE löschen (Abschnitt 8.2)'
;
10 CREATE DATABASE wurstbrot;
11 SHOW DATABASES;
12 DROP DATABASE wurstbrot;
13 SHOW DATABASES;
14
15
16 -- Start Tabellen umbenennen
17 SELECT 'Tabellen umbenennen (Abschnitt 8.3.1)';
18 DROP TABLE IF EXISTS wurstbrot;
19 SHOW TABLES;
20 ALTER TABLE adresse RENAME TO wurstbrot;
21 SHOW TABLES;
22 SHOW CREATE TABLE kunde\G
23 ALTER TABLE wurstbrot RENAME TO adresse;
24 -- Ende Tabellen umbenennen
25
26
27 -- Start Spalten verändern
28 SELECT 'Spalte hinzufügen (Abschnitt 8.3.2)';
29 DESCRIBE kunde;
```

```

30 ALTER TABLE kunde
31 ADD firmenname VARCHAR(255) NOT NULL DEFA
ULT '' AFTER vorname
32 ;
33 DESCRIBE kunde;
34
35 SELECT 'Spalte Datentyp ändern (Abschnitt
8.3.3)';
36 ALTER TABLE kunde
37 MODIFY
38 bezahlart ENUM('rechnung', 'bankeinzug',
'nachname') DEFAULT 'rechnung'
39 ;
40 DESCRIBE kunde;
41
42 SELECT 'Spaltenlänge ändern (Abschnitt 8.
3.4)';
43 DROP TABLE IF EXISTS bla ;
44 CREATE TEMPORARY TABLE bla LIKE artikel;
45 INSERT INTO bla SELECT * FROM artikel;
46 -- ALTER TABLE bla MODIFY bezeichnung CHA
R(10);
47 SELECT MAX(CHAR_LENGTH(bezeichnung)) FROM
artikel;
48
49 SELECT 'Zeichen nach Zahl ändern (Abschni
tt 8.3.6)';
50 DROP TABLE IF EXISTS wurstbrot ;
51 CREATE TEMPORARY TABLE wurstbrot LIKE adr
esse;
52 INSERT INTO wurstbrot SELECT * FROM adres
se;
53 SELECT * FROM wurstbrot;
54 ALTER TABLE wurstbrot MODIFY plz INT;
55 SELECT * FROM wurstbrot;
56
57 -- ALTER TABLE wurstbrot MODIFY plz TINYI
NT;

```

```

58  -- ALTER TABLE wurstbrot MODIFY hnr INT;
59  -- SELECT * FROM wurstbrot;
60
61  SELECT 'Datum- und Zeitspalten ändern (Abschnitt 8.3.8)';
62  DROP TABLE IF EXISTS wurstbrot ;
63  CREATE TEMPORARY TABLE wurstbrot (
64      a DATETIME,
65      b TIME,
66      c YEAR
67  );
68
69  INSERT INTO wurstbrot (a, b, c)
70  VALUES ('2012-03-24 14:57:00', '14:57:00'
, '2012')
71  ;
72
73  ALTER TABLE wurstbrot
74  MODIFY a BIGINT,
75  MODIFY b BIGINT,
76  MODIFY c BIGINT
77  ;
78  SELECT * FROM wurstbrot;
79
80  ALTER TABLE wurstbrot
81  MODIFY a DATETIME,
82  MODIFY b TIME,
83  MODIFY c YEAR
84  ;
85  SELECT * FROM wurstbrot;
86
87  ALTER TABLE wurstbrot
88  MODIFY a VARCHAR(255),
89  MODIFY b VARCHAR(255),
90  MODIFY c VARCHAR(255)
91  ;
92  SELECT * FROM wurstbrot;
93

```

```

94  ALTER TABLE wurstbrot
95  MODIFY a DATETIME,
96  MODIFY b TIME,
97  MODIFY c YEAR
98  ;
99  SELECT * FROM wurstbrot;
100
101  SELECT 'Spalten löschen (Abschnitt 8.3.9)
';
102  DROP TABLE IF EXISTS wurstbrot;
103  CREATE TEMPORARY TABLE wurstbrot AS SELEC
T * FROM adresse;
104  DESCRIBE wurstbrot;
105  ALTER TABLE wurstbrot
106  DROP deleted,
107  DROP strasse,
108  DROP ort
109  ;
110  DESCRIBE wurstbrot;
111
112  -- Ende Spalten verändern
113
114  -- Start Tabellen löschen
115  SELECT 'Tabelle löschen (Abschnitt 8.4.1)
';
116  SHOW TABLES;
117  -- DROP TABLE
118  -- rechnung_position
119  -- ,rechnung
120  -- ,bestellung_position
121  -- ,bestellung
122  -- ;
123  SHOW TABLES;
124  -- Ende Tabellen löschen
125  SELECT 'Ende listing05. sql //////////////////////////////////
////////////////////////////////////////';

```

Listing 28.6 mysql/listing06.sql

```

1  -- Ausführen der vorherigen Befehle
2  SOURCE listing05.sql
3
4  SELECT 'Start listing06. sql //////////////////////////////////
////////////////////////////////////////';
5
6  SELECT 'WHERE mit TRUE / FALSE (Abschnitt
9.1.1)';
7  SELECT * FROM artikel WHERE TRUE;
8  SELECT * FROM artikel WHERE FALSE;
9
10
11 -- Start Groß-/Kleinschreibung
12 SELECT 'Groß-/Kleinschreibung (Abschnitt
9.1.2)';
13 DROP TABLE IF EXISTS wurstbrot;
14 CREATE TEMPORARY TABLE wurstbrot (
15   name VARCHAR(255)
16 );
17
18 INSERT INTO wurstbrot
19 VALUES
20 ('Jaqueline')
21 ,('Kevin')
22 ,('kevin')
23 ,('jaqueline')
24 ;
25
26 SELECT * FROM wurstbrot WHERE name = 'kev
in';
27
28 ALTER TABLE wurstbrot
29 MODIFY name VARCHAR(255) BINARY
30 ;
31
32 SELECT * FROM wurstbrot WHERE name = 'kev
in';
33 -- Ende Groß- Kleinschreibung

```

```

34
35  -- Start Einfache Wertzuweisung
36  SELECT 'Einfache Wertzuweisung (Abschnitt 9.2.1)';
37  SELECT * FROM bestellung_position;
38  UPDATE bestellung_position
39  SET menge = 2
40  WHERE bestellung_id = 1 AND position_nr =
41  3
42  ;
43  SELECT * FROM bestellung_position;
44  -- Ende Einfache Wertzuweisung
45
46  -- Start Wertberechnung
47  SELECT 'Wertberechnung (Abschnitt 9.2.2)'
48  ;
49  SELECT * FROM artikel;
50  UPDATE artikel SET einzelpreis = einzelpr
eis + einzelpreis / 100.0;
51  SELECT * FROM artikel;
52  UPDATE artikel SET einzelpreis = ROUND(ei
nzelpreis, 2);
53  SELECT * FROM artikel;
54  -- Ende Wertberechnung
55
56  -- Start Gebastelte Zeichenketten
57  SELECT 'Gebastelte Zeichenketten: Anrede
(Abschnitt 9.2.3)';
58  ALTER TABLE kunde ADD anrede VARCHAR(255)
AFTER nachname;
59  UPDATE kunde
60  SET anrede = CONCAT('Sehr geehrte/r Frau/
Herr ', nachname)
61  WHERE nachname <> ''
62  ;
63  SELECT * FROM kunde;
64  -- Ende Gebastelte Zeichenketten
65

```

```

64
65  -- Start DELETE
66  SELECT 'Löschen mit DELETE (Abschnitt 9.3
)';
67  SELECT * FROM bestellung_position;
68  DELETE FROM bestellung_position WHERE bes
tellung_id = 1;
69  SELECT * FROM bestellung_position;
70
71  -- Start Wiederherstellen der Daten
72  INSERT INTO bestellung_position (
73  bestellung_id, position_nr, artikel_id, m
enge)
74  VALUES
75      (1, 1, 7856, 30)
76      ,(1, 2, 7863, 50)
77      ,(1, 3, 9015, 1)
78  ;
79  -- Ende Wiederherstellen der Daten
80
81  -- SELECT 'Löschversuch mit Constraint.';
82  -- SELECT 'Erwarte eine Fehlermeldung! (Abschnitt 9.3.1)';
83  -- DELETE FROM kunde WHERE kunde_id = 1;
84
85
86  SELECT 'Test AUTO_INCREMENT (Abschnitt 9.3.2)';
87  SELECT kunde_id, nachname FROM kunde;
88  DELETE FROM kunde WHERE kunde_id IN (3,5)
;
89  INSERT INTO kunde (nachname, vorname) VAL
UES ('Eichenschild', 'Thorin');
90  SELECT kunde_id, nachname FROM kunde;
91  -- Start Wiederherstellen der Daten
92  INSERT INTO kunde (kunde_id, rechnung_adr
esse_id, nachname, vorname, art)
93  VALUES

```



```

94      (3, 2, 'Beutlin', 'Bilbo', 'prv')
95      ,(5, 4, 'Earendilionn','Elrond', 'gsch'
)
96  ;
97  -- Ende Wiederherstellen der Daten
98
99  SELECT 'Tabelle leeren (Abschnitt 9.3.4)'
;
100  -- DELETE FROM bestellung_position;
101  -- TRUNCATE bestellung_position;
102  -- Ende DELETE
103  SELECT 'Ende listing06. sql //////////////////////////////////
////////////////////////////////////////';

```

28.1.3 Quelltexte zu Teil IV

Listing 28.7 mysql/listing07.sql

```

1  -- Ausführen der vorherigen Befehle
2  SOURCE listing06.sql
3
4  SELECT 'Start listing07. sql //////////////////////////////////
////////////////////////////////////////';
5
6  SELECT 'Ausdrücke (Abschnitt 10.1.1)';
7
8  SELECT 5, 'Beginn der Auswertung';
9  SELECT -5, 9 + 4, 9 - 4, 9 * 4, 9 / 4, 9
DIV 4, 9 % 4, 9 MOD 4;
10 SELECT 3 * 4 DIV 3, (3 * 4) DIV 3, 3 * (4
DIV 3);
11 SELECT 9 / 0, 9 * NULL, SQRT(-5);
12
13 SELECT 'Zufallszahlen (Abschnitt 10.1.3)'
;
14 SELECT RAND(1);

```

```

15  SELECT RAND(1);
16  SELECT RAND(1) FROM artikel;
17  -- FLOOR(a + RAND() * (b - a))
18
19  SELECT 'Variablen (Abschnitt 10.1.4)';
20
21  SELECT POWER(2,8) INTO @x;
22  SELECT @x - 1;
23
24  -- SELECT 'Ungültiger Versuch: erwarte ei
ne Fehlermeldung (Abschnitt 10.1.4)';
25  -- SELECT einzelpreis INTO @y FROM artike
l;
26
27  SET @artnr=3010;
28  SELECT * FROM artikel WHERE artikel_id=@a
rtnr;
29
30
31  SELECT 'Spalten- und Zeilenwahl (Abschnit
t 10.2)';
32  SELECT * FROM artikel;
33  SELECT
34  artikel_id, bezeichnung, einzelpreis, wae
hrung, deleted
35  FROM artikel
36  ;
37
38  SELECT
39  artikel_id + 10000 AS Unsinn, deleted AS
Löschkennzeichen
40  FROM artikel
41  ;
42
43  SELECT
44  artikel_id, bezeichnung
45  FROM artikel
46  WHERE einzelpreis BETWEEN 10 AND 100

```

```
47 ;
48
49 SELECT 'Sortierung (Abschnitt 10.3)';
50 SELECT
51 artikel_id, bezeichnung, einzelpreis
52 FROM artikel
53 ORDER BY einzelpreis ASC
54 ;
55
56 SELECT
57 nachname, vorname
58 FROM kunde
59 ORDER BY nachname, vorname
60 ;
61
62 SELECT
63 nachname, vorname
64 FROM kunde
65 ORDER BY nachname DESC, vorname ASC
66 ;
67
68 DROP TABLE IF EXISTS wurstbrot;
69 SET NAMES latin1;
70 CREATE TEMPORARY TABLE wurstbrot (name VARCHAR(255) CHARACTER SET 'latin1' COLLATE 'latin1_bin');
71 INSERT INTO wurstbrot
72 VALUES
73     ('winfried')
74     ,('achim')
75     ,('olga')
76     ,('zechine')
77     ,('ägidius')
78 ;
79 SELECT * FROM wurstbrot ORDER BY name;
80
81 DROP TABLE IF EXISTS wurstbrot;
82 SET NAMES utf8mb4;
```

```
83 CREATE TEMPORARY TABLE wurstbrot (name VA
RCHAR(255) CHARACTER SET 'utf8mb4' COLLATE 'ut
f8mb4_german2_ci');
84 INSERT INTO wurstbrot
85 VALUES
86     ('winfried')
87     ,('achim')
88     ,('olga')
89     ,('zechine')
90     ,('ägidius')
91 ;
92
93 SELECT * FROM wurstbrot ORDER BY name;
94
95 DROP TABLE IF EXISTS wurstbrot;
96 CREATE TEMPORARY TABLE wurstbrot (name VA
RCHAR(255));
97 INSERT INTO wurstbrot
98 VALUES
99     ('winfried')
100     ,('achim')
101     ,('olga')
102     ,('Olga')
103     ,('zechine')
104 ;
105 SELECT * FROM wurstbrot ORDER BY name;
106 SELECT * FROM wurstbrot ORDER BY name DES
C;
107
108 DROP TABLE IF EXISTS wurstbrot;
109 CREATE TEMPORARY TABLE wurstbrot (name VA
RCHAR(255) BINARY);
110 INSERT INTO wurstbrot
111 VALUES
112     ('winfried')
113     ,('achim')
114     ,('olga')
115     ,('Olga')
```

```
116  ,('zechine')
117  ;
118  SELECT * FROM wurstbrot ORDER BY name;
119  SELECT * FROM wurstbrot ORDER BY name DESC;
120
121
122  SELECT
123  bestellung_id, datum
124  FROM bestellung
125  ORDER BY datum;
126
127  SELECT
128  bestellung_id, TIME(datum) Uhrzeit
129  FROM bestellung
130  ORDER BY uhrzeit;
131
132  EXPLAIN SELECT * FROM kunde ORDER BY nach
name, vorname\G
133
134  SELECT 'Mehrfachausgaben unterbinden (Abschnitt 10.4)';
135
136  SELECT ort FROM adresse ORDER BY ort;
137  SELECT DISTINCT ort FROM adresse ORDER BY
ort;
138  SELECT DISTINCT nachname FROM kunde ORDER
BY nachname;
139
140  -- Start Bankimport
141  SELECT 'Bankdaten importieren (Abschnitt 10.4.1)';
142  DROP TABLE IF EXISTS bank_import;
143  CREATE TEMPORARY TABLE bank_import (
144  bankleitzahl CHAR(8),
145  bezeichnung VARCHAR(255)
146  );
147
```

```

148 SET GLOBAL local_infile = ON;
149 LOAD DATA LOCAL INFILE 'blz_20120305.csv'
    -- Hier ggf. anpassen!
150 INTO TABLE bank_import
151 CHARACTER SET utf8mb4
152 FIELDS
153 TERMINATED BY ';'
154 LINES
155 TERMINATED BY '\n'
156 IGNORE 1 LINES
157 (bankleitzahl, @dummy, bezeichnung)
158 ;
159 SET GLOBAL local_infile = OFF;
160 -- SELECT DISTINCT * FROM bank_import;
161 -- SELECT DISTINCT bankleitzahl FROM bank
_import;
162 -- SELECT DISTINCT bankleitzahl, bezeichn
ung FROM bank_import;
163
164 ALTER TABLE bankverbindung
165 DROP FOREIGN KEY bankverbindung_ibfk_1,
166 DROP FOREIGN KEY bankverbindung_ibfk_2,
167 DROP INDEX idx_bankverbindung_bankid_kont
onummer,
168 DROP PRIMARY KEY
169 ;
170 ALTER TABLE bank
171 DROP PRIMARY KEY
172 ;
173
174 ALTER TABLE bank
175 MODIFY bank_id INT UNSIGNED AUTO_INCREMEN
T PRIMARY KEY,
176 ADD blz CHAR(12) NOT NULL DEFAULT '' AFTE
R bank_id
177 ;
178
179 ALTER TABLE bankverbindung

```

```
180     MODIFY bank_id INT UNSIGNED,
181     ADD PRIMARY KEY(kunde_id, bankverbindun
g_nr),
182     ADD INDEX idx_bankverbindung_bankid_kon
tonummer (bank_id, kontonummer),
183     ADD FOREIGN KEY (kunde_id)
184     REFERENCES kunde(kunde_id)
185     ON UPDATE RESTRICT
186     ON DELETE RESTRICT,
187     ADD FOREIGN KEY (bank_id)
188     REFERENCES bank(bank_id)
189     ON UPDATE RESTRICT
190     ON DELETE RESTRICT
191 ;
192
193 INSERT INTO bank (blz, bankname)
194 SELECT DISTINCT bankleitzahl, bezeichnung
FROM bank_import
195 ;
196
197 CREATE INDEX idx_bank_blzbankname
198 ON bank (blz, bankname)
199 ;
200
201 -- Beginn Für die vorhandenen Kunden eine
Bankverbindung bauen
202
203 INSERT INTO bankverbindung (kunde_id, ban
kverbindung_nr, bank_id, kontonummer, iban )
204     VALUES
205     (1, 1, 2, '1111111111', '10010010111111
1111'),
206     (1, 2, 2, '1111111112', '10010010111111
1112'),
207     (2, 1, 35, '2222222221', '1006019822222
22221'),
208     (3, 1, 35, '3333333331', '1006019833333
33331'),
```

```
209      (4, 1, 90, '4444444441', '1207000044444
44441'),
210      (5, 1, 90, '5555555551', '1207000055555
55551')
211  ;
212  -- Ende Für die vorhandenen Kunden eine B
ankverbindung bauen
213
214
215  -- Ende Bankimport
216
217  SELECT 'Ausschneiden mit LIMIT (Abschnitt
10.5)';
218  -- SELECT * FROM bank;
219  SELECT blz, bankname FROM bank ORDER BY b
lz LIMIT 3;
220  SELECT
221  blz, bankname
222  FROM bank
223  ORDER BY blz DESC
224  LIMIT 1
225  ;
226  SELECT
227  blz
228  FROM bank
229  ORDER BY blz DESC
230  LIMIT 1
231  INTO @blzmin
232  ;
233  SELECT
234  bankname
235  FROM bank
236  WHERE blz = @blzmin
237  ;
238  SELECT blz, bankname FROM bank ORDER BY b
lz LIMIT 2 OFFSET 1;
239  SELECT blz, bankname FROM bank ORDER BY b
lz LIMIT 1;
```



```

240 SELECT blz, bankname FROM bank ORDER BY b
    lz LIMIT 1 OFFSET 0;
241
242 SELECT 'CSV-Datei exportieren (Abschnitt
10.6.1)';
243
244 SET GLOBAL local_infile = ON;
245 SELECT
246 blz Bankleitzahl, bankname Bankname
247 INTO OUTFILE 'bank.csv'
248     FIELDS
249     TERMINATED BY ';'
250     ENCLOSED BY '"'
251     LINES
252     TERMINATED BY '\r\n'
253     FROM
254     bank
255     ORDER BY
256     Bankleitzahl, Bankname;
257 SET GLOBAL local_infile = OFF;
258 SELECT 'Ende listing07. sql //////////////////////////////////
////////////////////////////////';

```

Listing 28.8 mysql/listing08.sql

```

1  -- Ausführen der vorherigen Befehle
2  SOURCE listing07.sql
3
4  SELECT 'Start listing08. sql //////////////////////////////////
////////////////////////////////';
5
6  -- Start INNER JOIN
7  SELECT 'CROSS JOIN (Abschnitt 11.1)';
8  SELECT
9  kunde_id, nachname, vorname, rechnung_adr
    esse_id, adresse_id
10 FROM
11 kunde, adresse

```

```

12  ;
13
14  SELECT 'CROSS JOIN mit WHERE (Abschnitt 1
1.1)';
15  SELECT
16  kunde_id, nachname, vorname, rechnung_adr
esse_id, adresse_id
17      FROM
18      kunde, adresse
19      WHERE
20      rechnung_adresse_id = adresse_id
21  ;
22
23  SELECT 'INNER JOIN (Abschnitt 11.2)';
24  SELECT
25  kunde_id, nachname, vorname, rechnung_adr
esse_id, adresse_id
26  FROM
27  kunde INNER JOIN adresse
28  ON rechnung_adresse_id = adresse_id
29  ;
30
31  SELECT
32  kunde_id, kontonummer, blz, bankname
33  FROM
34      bankverbindung INNER JOIN bank
35      ON bankverbindung.bank_id = bank.bank_i
d
36      ORDER BY
37      kunde_id, kontonummer
38  ;
39  SELECT
40  nachname, vorname, kontonummer
41  FROM
42      bankverbindung INNER JOIN kunde ON bank
verbindung.kunde_id = kunde.kunde_id
43      ORDER BY
44      nachname, vorname, kontonummer

```

```

45  ;
46
47  SELECT 'Abkürzung mit USING (Abschnitt 11.2.2)';
48  SELECT
49  kunde_id, kontonummer, blz, bankname
50  FROM
51  bankverbindung INNER JOIN bank USING(bank
_id)
52  ;
53
54  SELECT 'Abkürzung mit NATURAL JOIN (Abschnitt 11.2.2)';
55  SELECT
56  kunde_id, kontonummer, blz, bankname
57  FROM
58  bankverbindung NATURAL JOIN bank
59  ;
60  -- Ende INNER JOIN
61
62  -- Start Datenquelle
63  SELECT 'JOIN als Datenquelle (Abschnitt 11.2.3)';
64  SELECT
65  k.kunde_id, k.nachname, k.vorname, a.strasse, a.hnr, a.plz, a.ort
66  FROM
67  kunde k INNER JOIN adresse a
68  ON k.rechnung_adresse_id = a.adresse_id
69  ;
70  DROP TABLE IF EXISTS tmp_kadresse;
71  CREATE TEMPORARY TABLE tmp_kadresse
72  SELECT
73  k.kunde_id, k.nachname, k.vorname, a.strasse, a.hnr, a.plz, a.ort
74  FROM
75  kunde k INNER JOIN adresse a
76  ON k.rechnung_adresse_id = a.adresse_id

```

```

77  ;
78  SELECT
79  t.kunde_id, t.nachname, t.ort, b.bestellu
ng_id, DATE(b.datum)
80  FROM
81  bestellung b INNER JOIN tmp_kadresse t US
ING(kunde_id)
82  ;
83  DROP TABLE IF EXISTS tmp_kbank;
84  CREATE TEMPORARY TABLE tmp_kbank
85  SELECT
86  bv.kunde_id, bv.bankverbindung_nr, bv.kon
tonummer,
87  bv.iban, ba.blz, ba.bankname
88  FROM
89  bankverbindung bv INNER JOIN bank ba USIN
G(bank_id)
90  ;
91  DROP TABLE IF EXISTS tmp_kbankeinzug;
92  CREATE TEMPORARY TABLE tmp_kbankeinzug
93  SELECT
94  ka.*, kb.bankverbindung_nr, kb.kontonumme
r,
95  kb.iban, kb.blz, kb.bankname
96  FROM
97  tmp_kadresse ka INNER JOIN tmp_kbank kb U
SING(kunde_id);
98
99  SELECT kunde_id, nachname, ort, bankname
FROM tmp_kbankeinzug ;
100 -- Ende Datenquelle
101
102 -- Start Keine Schlüssel
103 SELECT 'Ohne Schlüssel (Abschnitt 11.2.4)
';
104 DESCRIBE tmp_kadresse;
105 DESCRIBE tmp_kbank;
106 SELECT

```

```

107 k.kunde_id, k.vorname, a.strasse, a.hnr,
a.plz, a.ort
108 FROM
109 kunde k INNER JOIN adresse a
110 ON k.rechnung_adresse_id <= a.adresse_id
111 ;
112 -- Ende Kein Schlüssel
113
114 -- Start mehr als zwei Tabellen
115 SELECT 'Mit mehr als 2 Tabellen (Abschnitt 10)';
116 SELECT
117 a.bezeichnung, nm.warengruppe_id
118 FROM
119 artikel_nm_warengruppe nm INNER JOIN artikel a USING(artikel_id)
120 ;
121 SELECT
122 w.bezeichnung, nm.artikel_id
123 FROM
124 artikel_nm_warengruppe nm INNER JOIN warengruppe w USING(warengruppe_id)
125 ;
126 SELECT
127 w.bezeichnung Warengruppe, a.bezeichnung Artikel
128 FROM
129 artikel_nm_warengruppe nm INNER JOIN warengruppe w USING(warengruppe_id)
130 INNER JOIN artikel a USING(artikel_id)
131 ;
132 -- Ende mehr als zwei Tabellen
133
134 -- Start Aufbau Lieferanten
135 SELECT 'Aufbau der Lieferanten';
136 INSERT INTO lieferant (firmenname, adresse_id)
137 VALUES

```

```

138      ('Gartenbedarf AllesGrün', 10)
139      ,('Office International', 11)
140      ,('Bürohengst GmbH', 11)
141  ;
142
143  INSERT INTO artikel_nm_lieferant (lieferan
nt_id, artikel_id)
144  VALUES
145      (1, 7856)
146      ,(1, 7863)
147      ,(1, 9010)
148      ,(1, 9015)
149      ,(3, 3001)
150      ,(3, 3005)
151      ,(3, 3006)
152      ,(3, 3007)
153      ,(3, 3010)
154  ;
155  -- Ende Aufbau Lieferanten
156
157  -- Start OUTER JOIN
158  SELECT 'OUTER JOIN (Abschnitt 11.4)';
159  SELECT
160  k.kunde_id, k.nachname, k.vorname, b.datu
m
161  FROM
162  bestellung b INNER JOIN kunde k USING(kun
de_id)
163  ORDER BY
164  k.nachname, k.vorname
165  ;
166  SELECT
167  k.kunde_id, k.nachname, k.vorname, b.datu
m
168  FROM
169  bestellung b RIGHT OUTER JOIN kunde k USI
NG(kunde_id)
170  ORDER BY

```

```

171 k.nachname, k.vorname
172 ;
173 SELECT
174 k.kunde_id, k.nachname, k.vorname, b.datu
m
175 FROM
176 kunde k LEFT OUTER JOIN bestellung b USIN
G(kunde_id)
177 ORDER BY
178 k.nachname, k.vorname
179 ;
180 SELECT
181 l.lieferant_id, l.firmenname, a.artikel_i
d, a.bezeichnung
182 FROM
183 artikel_nm_lieferant nm INNER JOIN artike
l a USING(artikel_id)
184 INNER JOIN lieferant l USING(lieferant_id
)
185 ORDER BY firmenname
186 ;
187 SELECT
188 l.lieferant_id, l.firmenname, a.artikel_i
d, a.bezeichnung
189 FROM
190 artikel_nm_lieferant nm INNER JOIN artike
l a USING(artikel_id)
191 RIGHT JOIN lieferant l USING(lieferant_id
)
192 ORDER BY firmenname
193 ;
194 SELECT
195 l.firmenname
196 FROM
197     artikel_nm_lieferant nm INNER JOIN arti
kel a USING(artikel_id)
198     RIGHT JOIN lieferant l USING(lieferant_
id)

```

```
199 WHERE artikel_id IS NULL
200 ORDER BY firmenname
201 ;
202 SELECT k.kunde_id, k.nachname, k.vorname
203 FROM
204 bestellung b RIGHT JOIN kunde k USING(kun
de_id)
205 WHERE b.bestellung_id IS NULL
206 ;
207 -- Ende OUTER JOIN
208
209 -- Start Aufbau Forum
210 SELECT 'SELF JOIN (Abschnitt 11.5)' Hinwe
is;
211 SELECT 'CREATE account';
212 CREATE TABLE account (
213 kunde_id INT UNSIGNED NOT NULL DEFAULT 0,
214 name VARCHAR(255) NOT NULL DEFAULT '',
215 passwort CHAR(34) NOT NULL DEFAULT '',
216 status ENUM('aktiv','gesperrt') NOT NULL
DEFAULT 'aktiv',
217 kommentar TEXT ,
218 admin BOOL NOT NULL DEFAULT FALSE,
219 deleted TINYINT UNSIGNED NOT NULL DEFAULT
0,
220 PRIMARY KEY(kunde_id),
221 FOREIGN KEY (kunde_id)
222 REFERENCES kunde(kunde_id)
223 ON UPDATE RESTRICT
224 ON DELETE RESTRICT
225 ) ENGINE=InnoDB;
226
227 SELECT 'CREATE beitrag';
228 CREATE TABLE beitrag (
229 beitrag_id INT UNSIGNED NOT NULL AUTO_INC
REMENT,
230 account_id INT UNSIGNED NOT NULL,
231 bezug_beitrag_id INT UNSIGNED NOT NULL DE
```



```

FAULT 1,
232  zeitstempel DATETIME NOT NULL DEFAULT '10
00-01-01 00:00:00',
233  nachricht TEXT ,
234  deleted TINYINT UNSIGNED NOT NULL DEFAULT
0,
235  PRIMARY KEY(beitrag_id),
236  FOREIGN KEY (account_id)
237      REFERENCES account(kunde_id)
238      ON UPDATE RESTRICT
239      ON DELETE RESTRICT,
240  FOREIGN KEY (bezug_beitrag_id)
241      REFERENCES beitrags(beitrag_id)
242      ON UPDATE RESTRICT
243      ON DELETE RESTRICT
244  ) ENGINE=InnoDB;
245
246  SELECT 'account daten';
247  INSERT INTO account (kunde_id, name, pass
wort, admin)
248  VALUES
249      (1, 'admin', md5('rosi'), TRUE)
250      ,(2, 'frodo', md5('elbereth'), FALSE)
251      ,(3, 'bilbo', md5('blitzerschlagen'), F
ALSE)
252      ,(5, 'elle', md5('feanor'), FALSE)
253  ;
254
255
256  SELECT 'beitrag daten';
257  INSERT INTO beitrags (beitrag_id, account_
id, bezug_beitrag_id, zeitstempel, nachricht)
258  VALUES
259      (1, 1, 1, '1000-01-01 00:00:00', '')
260      ,(2, 2, 1, '2011-05-01 14:13:00', 'Der
Lieferservice ist super.')
261      ,(3, 3, 2, '2011-05-02 11:45:00', 'Das
finde ich auch.')

```

```

262      ,(4, 5, 2, '2011-05-01 17:01:00', 'Aber
    ein wenig langsam.')
```

```

263      ,(5, 2, 4, '2011-05-01 17:15:00', 'Find
    e ich nicht.')
```

```

264      ,(6, 5, 1, '2011-06-12 09:07:00', 'Ange
    bot könnte besser sein.')
```

```

265  ;
266  -- Ende Aufbau Forum
267
268  -- Start SELF JOIN
269  SELECT 'SELF JOIN (Abschnitt 11.5)';
270  SELECT
271  kunde_id, name, status, admin
272  FROM
273  account
274  ;
275  SELECT
276  beitrags_id, account_id, bezug_bei_trags_id,
    LEFT(nachricht, 20)
277  FROM
278  beitrags
279  ;
280  -- SELECT 'Erwarte eine Fehlermeldung (Ab
    schnitt 11.5)';
281  -- SELECT nachricht
282  -- FROM
283  -- beitrags INNER JOIN beitrags ON bezug_bei
    trags_id = beitrags_id
284  -- WHERE beitrags_id = 2
285  -- ;
286  SELECT ant.nachricht 'Antwort'
287  FROM
288  beitrags ant INNER JOIN beitrags orig ON an
    t.bezug_bei_trags_id = orig.beitrags_id
289  WHERE orig.beitrags_id = 2
290  ;
291
292  SELECT 'CTE (Abschnitt 11.5)';
```

```

293 WITH ant_auf AS
294 (
295 SELECT * FROM beitrags WHERE bezug_beitrags
_id = 2
296 )
297 SELECT beitrags_id, bezug_beitrags_id, nach
richt FROM ant_auf;
298
299 SELECT 'CTE rekursiv (Abschnitt 11.5)';
300 WITH RECURSIVE ant_auf AS
301 (
302 -- nicht rekursiver Teil
303 SELECT * FROM beitrags WHERE bezug_beitrags
_id = 2
304 UNION ALL
305 -- rekursiver Teil
306 SELECT ant.*
307 FROM
308 beitrags ant INNER JOIN ant_auf orig
309 ON ant.bezug_beitrags_id = orig.beitrags_
id
310 )
311 SELECT beitrags_id, bezug_beitrags_id, nach
richt FROM ant_auf;
312 -- Ende SELF JOIN
313
314 -- Start Redundant
315 SELECT 'Beschleunigen mit Redundanzen (Ab
schnitt 11.6)';
316 ALTER TABLE
317 kunde
318 ADD r_strasse VARCHAR(255),
319 ADD r_ort VARCHAR(255),
320 ADD r_aktuell BOOL NOT NULL DEFAULT TRUE,
321 ADD l_strasse VARCHAR(255),
322 ADD l_ort VARCHAR(255),
323 ADD l_aktuell BOOL NOT NULL DEFAULT TRUE
324 ;

```

```

325 UPDATE kunde INNER JOIN adresse ON rechnu
ng_adresse_id = adresse_id
326 SET
327 r_strasse = CONCAT(strasse, ' ', hnr),
328 r_ort = CONCAT(lkz, '-', plz, ' ', ort),
329 r_aktuell = TRUE
330 ;
331 SELECT
332 kunde_id, r_strasse, r_ort, r_aktuell
333 FROM kunde
334 ;
335 -- Ende Redundant
336 SELECT 'Ende listing08. sql //////////////////////////////////
////////////////////////////////////////';

```

Listing 28.9 mysql/listing09.sql

```

1  -- Ausführen der vorherigen Befehle
2  SOURCE listing08.sql
3
4  SELECT 'Start listing09. sql //////////////////////////////////
////////////////////////////////////////';
5
6  SELECT 'Aggregatfunktionen (Abschnitt 12.1)';
7  SELECT AVG(einzelpreis) FROM artikel;
8  SELECT COUNT(*) FROM kunde;
9  SELECT COUNT(liefer_adresse_id) FROM kund
e;
10 SELECT COUNT(DISTINCT(rechnung_adresse_id
)) FROM kunde;
11 SELECT MAX(einzelpreis), MIN(einzelpreis)
FROM artikel;
12 SELECT SUM(menge) FROM bestellung_positio
n;
13 SELECT
14 bp.menge, a.einzelpreis
15 FROM

```

```

16  bestellung_position bp INNER JOIN artikel
a USING(artikel_id)
17  ;
18  SELECT
19  bp.menge * a.einzelpreis 'Positionswert'
20  FROM
21  bestellung_position bp INNER JOIN artikel
a USING(artikel_id)
22  ;
23
24  SELECT
25  SUM(bp.menge * a.einzelpreis)
26  FROM
27  bestellung_position bp INNER JOIN artikel
a USING(artikel_id)
28  ;
29
30  -- Start Aufbau Lagerverwaltung
31  SELECT 'Aufbau Lagerbestand (Abschnitt 12
.1)';
32  SELECT 'CREATE lagerbestand';
33  DROP TABLE IF EXISTS lagerbestand;
34  CREATE TABLE lagerbestand (
35  artikel_id INT UNSIGNED NOT NULL DEFAULT
0,
36  menge_mindest DECIMAL(14,6) NOT NULL DEFA
ULT 0.0,
37  menge_aktuell DECIMAL(14,6) NOT NULL DEFA
ULT 0.0,
38  deleted TINYINT UNSIGNED NOT NULL DEFAULT
0,
39  PRIMARY KEY(artikel_id),
40  FOREIGN KEY (artikel_id)
41  REFERENCES artikel(artikel_id)
42  ON UPDATE RESTRICT
43  ON DELETE RESTRICT
44  ) ENGINE=InnoDB;
45

```

```

46 SELECT 'Lagerbestand festlegen';
47 INSERT INTO lagerbestand (artikel_id, men
ge_aktuell, menge_mindest)
48 VALUES
49     (7856, 500, 100)
50     ,(7863, 400, 100)
51     ,(9010, 30, 10)
52     ,(9015, 25, 10)
53     ,(3001, 5000, 1000)
54     ,(3005, 250, 200)
55     ,(3006, 250, 200)
56     ,(3007, 149, 200)
57     ,(3010, 3, 100)
58 ;
59 -- Ende Aufbau Lagerverwaltung
60
61
62 SELECT 'GROUP BY (Abschnitt 12.2)';
63 -- Start Group by
64 SELECT
65 COUNT(*)
66 FROM
67 bestellung_position
68 ;
69 SELECT
70 bestellung_id Bestellnummer, COUNT(*) 'An
zahl der Positionen'
71 FROM
72 bestellung_position
73 GROUP BY
74 bestellung_id
75 ;
76 SELECT
77 a.bezeichnung, bp.menge
78 FROM
79 bestellung_position bp INNER JOIN artikel
a USING(artikel_id)
80 ;

```

```

81  SELECT
82  a.bezeichnung 'Artikelname', SUM(bp.menge
) 'Anzahl bestellter Artikel'
83  FROM
84  bestellung_position bp INNER JOIN artikel
a USING(artikel_id)
85  GROUP BY
86  artikel_id
87  ;
88  SELECT
89  a.bezeichnung 'Artikelname', SUM(bp.menge
) 'Anzahl bestellter Artikel'
90  FROM
91  bestellung_position bp RIGHT JOIN artikel
a USING(artikel_id)
92  GROUP BY
93  artikel_id
94  ;
95  SELECT
96  a.bezeichnung 'Artikelname',
97  CASE
98  WHEN SUM(bp.menge) IS NULL THEN 0
99  ELSE SUM(bp.menge)
100 END AS 'Anzahl bestellter Artikel'
101 FROM
102 bestellung_position bp RIGHT JOIN artikel
a USING(artikel_id)
103 GROUP BY
104 artikel_id
105 ;
106 -- SELECT 'Eine Fehlermeldung! (Abschnitt
12.3)';
107 -- SELECT
108 -- a.bezeichnung 'Artikelname', SUM(bp.me
nge) 'Anzahl bestellter Artikel'
109 -- FROM
110 -- bestellung_position bp INNER JOIN arti
kel a USING(artikel_id)

```

```

111  -- WHERE
112  -- SUM(bp.menge) > 10
113  -- GROUP BY
114  -- artikel_id
115  -- ;
116
117  SELECT 'Gruppenergebnisse mit WITH ROLLUP
    summieren (Abschnitt 12.2)';
118  SELECT
119  artikel_id,
120  CASE
121  WHEN SUM(bp.menge) IS NULL THEN 0
122  ELSE SUM(bp.menge)
123  END AS 'Anzahl bestellter Artikel'
124  FROM
125  bestellung_position bp RIGHT JOIN artikel
    a USING(artikel_id)
126  GROUP BY
127  artikel_id WITH ROLLUP
128  ;
129
130
131  SELECT 'Gruppenergebnisse mit HAVING filt
    ern (Abschnitt 12.3)';
132  SELECT
133  a.bezeichnung 'Artikelname', SUM(bp.menge
    ) 'Anzahl bestellter Artikel'
134  FROM
135  bestellung_position bp INNER JOIN artikel
    a USING(artikel_id)
136  GROUP BY
137  artikel_id
138  HAVING
139  SUM(bp.menge) > 10
140  ;
141  SELECT
142  a.bezeichnung 'Artikelname', SUM(bp.menge
    ) 'Anzahl bestellter Artikel'

```



```

143 FROM
144 bestellung_position bp INNER JOIN artikel
   a USING(artikel_id)
145 WHERE
146 a.bezeichnung LIKE 'Silber%'
147 GROUP BY
148 artikel_id
149 HAVING
150 SUM(bp.menge) > 10
151 ;
152
153 SELECT 'Fragen zu Gruppen (Abschnitt 12.4
.1)';
154 SELECT
155 SUBSTRING(blz, 1, 1) 'Clearinggebiet', CO
UNT(*) 'Anzahl'
156 FROM
157 bank
158 GROUP BY
159 'Clearinggebiet'
160 ORDER BY 'Anzahl' DESC
161 ;
162 -- Start Bestellungen erweitern
163 SELECT 'Bestellungen erweitern';
164 INSERT INTO bestellung (bestellung_id, ku
nde_id, adresse_id, datum)
165 VALUES
166     (3, 1, 1, '2011-01-15 16:43:00')
167     ,(4, 1, 1, '2011-01-16 09:15:00')
168     ,(5, 1, 1, '2011-01-16 09:16:00')
169     ,(6, 2, 2, '2012-04-01 13:11:00')
170 ;
171
172 INSERT INTO bestellung_position (
173 bestellung_id, position_nr, artikel_id, m
enge)
174 VALUES
175     (3, 1, 7856, 10)

```

```

176      ,(3, 2, 7863, 10)
177      ,(4, 1, 3006, 1)
178      ,(4, 2, 3010, 4)
179      ,(5, 1, 3001, 100)
180      ,(5, 2, 3010, 5)
181      ,(5, 3, 3006, 1)
182      ,(5, 4, 3005, 4)
183  ;
184  -- Ende Bestellungen erweitern
185  SELECT
186  YEAR(datum) 'Jahr', MONTH(datum) 'Monat',
    COUNT(*) 'Anzahl'
187  FROM bestellung
188  GROUP BY
189  YEAR(datum), MONTH(datum)
190  ORDER BY
191  YEAR(datum) DESC, MONTH(datum) DESC
192  ;
193  EXPLAIN
194  SELECT
195  SUBSTRING(blz, 1, 1) 'Clearinggebiet', CO
UNT(*) 'Anzahl'
196  FROM
197  bank
198  GROUP BY
199  'Clearinggebiet'
200  ORDER BY
201  'Anzahl' DESC\G
202  EXPLAIN
203  SELECT
204  CONCAT(lkz, SUBSTRING(blz, 6, 3)) 'Sinn
los', COUNT(*) 'Anzahl'
205  FROM
206  bank
207  GROUP BY
208  'Sinnlos'
209  ORDER BY
210  'Sinnlos'\G

```

```

211
212  -- Ende Group by
213  SELECT 'Ende listing09. sql //////////////////////////////////
////////////////////////////////////////';

```

Listing 28.10 mysql/listing10.sql

```

1  -- Ausführen der vorherigen Befehle
2  SOURCE listing09.sql
3
4  SELECT 'Start listing10. sql //////////////////////////////////
////////////////////////////////////////';
5
6  SELECT 'Start Aufbau Rechnungswesen';
7  USE oshop;
8
9
10 INSERT INTO rechnung
11 (kunde_id, bestellung_id, adresse_id, dat
um)
12 SELECT kunde_id, bestellung_id, adresse_i
d, DATE_ADD(datum, INTERVAL 1 DAY)
13 FROM bestellung;
14
15
16 INSERT INTO rechnung (rechnung_id, kunde_
id, bestellung_id, adresse_id, datum, bezahlar
t, status)
17 VALUES
18 (7, 3, 0, 2, '2012-04-06 12:11:00', 'kr
edit', 'gestellt')
19 ,(8, 3, 0, 2, '2012-04-07 13:12:00', 'k
redit', 'mahnung1')
20 ,(9, 5, 0, 4, '2012-04-08 14:13:00', 'r
echnung', 'storno')
21 ,(10, 5, 0, 4, '2012-04-09 15:14:00', '
rechnung', 'beglichen')
22 ,(11, 5, 0, 4, '2012-04-10 16:15:00', '

```

```

rechnung', 'beglichen')
23  ;
24  UPDATE rechnung SET status = 'inkasso' WH
ERE rechnung_id = 2;
25  INSERT INTO rechnung_position (rechnung_i
d, position_nr, artikel_id, menge)
26  SELECT bestellung_id, position_nr, artike
l_id, menge FROM bestellung_position;
27
28  INSERT INTO rechnung_position
29  (rechnung_id, position_nr, artikel_id, me
nge)
30  VALUES
31      (7, 1, 3001, 5)
32      ,(7, 2, 3005, 1)
33      ,(8, 1, 3006, 7)
34      ,(8, 2, 3007, 1)
35      ,(10, 1, 3010, 15)
36      ,(10, 2, 3001, 5)
37      ,(11, 1, 3005, 20)
38  ;
39
40  UPDATE rechnung SET rabatt = 7 WHERE kund
e_id = 1;
41  UPDATE rechnung SET skonto = 3 WHERE kund
e_id = 5;
42
43  SELECT 'Ende Aufbau Rechnungswesen';
44
45  SELECT 'Problem und Lösung (Abschnitt 13.1)';
46  SELECT
47  bp.bestellung_id, SUM(bp.menge * a.einzel
preis) Bestellwert
48  FROM
49      bestellung_position bp INNER JOIN artik
el a USING(artikel_id)
50      GROUP BY

```

```

51  bp.bestellung_id
52  ;
53
54  SELECT
55  k.nachname, k.vorname, SUM(bp.menge * a.e
inzelpreis) Bestellwert
56  FROM
57  bestellung_position bp INNER JOIN artikel
a USING(artikel_id)
58  INNER JOIN bestellung b USING(bestellung_
id)
59  INNER JOIN kunde k USING(kunde_id)
60  GROUP BY
61  bp.bestellung_id
62  ;
63
64  DROP TABLE IF EXISTS tmp_bestellwert;
65  CREATE TEMPORARY TABLE tmp_bestellwert
66  SELECT
67  k.nachname, k.vorname, SUM(bp.menge * a.e
inzelpreis) bestellwert
68  FROM
69  bestellung_position bp INNER JOIN artikel
a USING(artikel_id)
70  INNER JOIN bestellung b USING(bestellung_
id)
71  INNER JOIN kunde k USING(kunde_id)
72  GROUP BY
73  bp.bestellung_id
74  ;
75
76  SELECT * FROM tmp_bestellwert;
77  SELECT
78  nachname, vorname, SUM(bestellwert) Umsat
z
79  FROM tmp_bestellwert
80  GROUP BY nachname, vorname
81  ;

```

```

82
83  SELECT
84  bw.nachname, bw.vorname, SUM(bw.bestellwe
rt) Umsatz
85      FROM
86      (
87      SELECT
88          k.nachname, k.vorname, SUM(bp.menge *
a.einzelpreis) bestellwert
89          FROM
90          bestellung_position bp INNER JOIN art
ikel a USING(artikel_id)
91          INNER JOIN bestellung b USING(bestell
ung_id)
92          INNER JOIN kunde k USING(kunde_id)
93          GROUP BY
94          bp.bestellung_id
95      ) AS bw
96      GROUP BY nachname, vorname
97      ;
98
99  -- Start nicht korrelierende Unterabfrage
100  SELECT 'Nicht korrelierende Unterabfrage
(Abschnitt 13.2.1.1)';
101  SELECT MAX(blz) FROM bank;
102  SELECT bankname
103  FROM bank
104  WHERE blz = '37010050'
105  ;
106  SELECT bankname
107  FROM bank
108  WHERE blz =
109  (
110  SELECT MAX(blz)
111  FROM bank
112  )
113  ;
114

```

```

115  SELECT
116  AVG(einzelpreis) AS durchschnittspreis
117  FROM artikel
118  ;
119
120  SELECT *
121  FROM artikel
122  WHERE
123  einzelpreis > 30
124  ;
125
126  SELECT *
127  FROM artikel
128  WHERE
129  einzelpreis >
130  (
131      SELECT
132      AVG(einzelpreis) AS durchschnittspreis
133      FROM artikel
134  )
135  ;
136
137  SELECT
138  bp.bestellung_id, SUM(bp.menge * a.einzel
preis) bestellwert
139  FROM
140  bestellung_position bp INNER JOIN artikel
a USING(artikel_id)
141  GROUP BY
142  bp.bestellung_id
143  ;
144
145  SELECT AVG(bw.bestellwert)
146  FROM (
147      SELECT
148      bp.bestellung_id, SUM(bp.menge * a.ei
nzelpreis) bestellwert

```

```

149      FROM
150      bestellung_position bp INNER JOIN art
151      ikel a USING(artikel_id)
152      GROUP BY
153      bp.bestellung_id
154      ) AS bw
155      ;
156  SELECT
157      bp1.bestellung_id, SUM(bp1.menge * a1.e
158      einzelpreis) bestellwert1
159      FROM
160      bestellung_position bp1 INNER JOIN arti
161      kel a1 USING(artikel_id)
162      GROUP BY
163      bp1.bestellung_id
164      HAVING bestellwert1 > 100
165      ;
166  SELECT
167      bp1.bestellung_id, SUM(bp1.menge * a1.ein
168      zelpreis) bwert1
169      FROM
170      bestellung_position bp1 INNER JOIN artike
171      l a1 USING(artikel_id)
172      GROUP BY
173      bp1.bestellung_id
174      HAVING
175      bwert1 >
176      (
177      SELECT AVG(bw.bwert2)
178      FROM
179      (
180      SELECT

```



```
artikel a2 USING(artikel_id)
181         GROUP BY
182         bp2.bestellung_id
183     ) AS bw
184 )
185 ;
186
187 SELECT bestellung_id FROM bestellung;
188 SELECT rechnung_id
189 FROM rechnung
190 WHERE bestellung_id IN
191 (
192     SELECT bestellung_id FROM bestellung
193 )
194 ;
195
196 SELECT a.einzelpreis
197 FROM
198     artikel_nm_warengruppe INNER JOIN artikel
199     a USING(artikel_id)
200     INNER JOIN warengruppe w USING(warengrupp
201     e_id)
202 WHERE
203     w.bezeichnung = 'Gartenbedarf'
204 ;
205 SELECT
206     a.bezeichnung, a.einzelpreis
207 FROM
208     artikel a
209 WHERE
210     a.einzelpreis > ALL(SELECT 10)
211 ;
212 SELECT
213     a.bezeichnung, a.einzelpreis
214 FROM
215     artikel a
216 WHERE
```

```

216 a.einzelpreis > ALL
217 (
218     SELECT a.einzelpreis
219     FROM
220     artikel_nm_warengruppe INNER JOIN artikel
221     el a USING(artikel_id)
222     INNER JOIN warengruppe w USING(warengru
223     ppe_id)
224     WHERE
225     w.bezeichnung = 'Gartenbedarf'
226 )
227 ;
228
229 SELECT
230     rechnung_id, kunde_id, SUM(einzelpreis*
231     menge) AS umsatz
232     FROM
233     rechnung_position INNER JOIN artikel US
234     ING(artikel_id)
235     INNER JOIN rechnung USING(rechnung_id)
236 GROUP BY
237     rechnung_id
238 ;
239
240 SELECT SUM(umsatz) AS umsatzsumme
241 FROM
242 (
243     SELECT
244     rechnung_id, kunde_id, SUM(einzelpreis*
245     menge) AS umsatz
246     FROM
247     rechnung_position INNER JOIN artikel US
248     ING(artikel_id)
249     INNER JOIN rechnung USING(rechnung_id)
250 GROUP BY
251     rechnung_id
252 ) AS ksum
253 GROUP BY

```

```
248  kunde_id
249  ;
250
251  SELECT SUM(umsatz) AS umsatzsumme
252  FROM
253      (
254          SELECT
255              rechnung_id, kunde_id, SUM(einzelpreis*
menge) AS umsatz
256          FROM
257              rechnung_position INNER JOIN artikel US
ING(artikel_id)
258          INNER JOIN rechnung USING(rechnung_id)
259          INNER JOIN kunde USING(kunde_id)
260          WHERE nachname = 'beutlin'
261          GROUP BY
262              rechnung_id
263      ) AS ksum
264  GROUP BY
265      kunde_id
266  ;
267
268  SELECT kunde_id, SUM(umsatz) AS umsatzsum
me
269  FROM
270      (
271          SELECT
272              rechnung_id, kunde_id, SUM(einzelpreis*
menge) AS umsatz
273          FROM
274              rechnung_position INNER JOIN artikel US
ING(artikel_id)
275          INNER JOIN rechnung USING(rechnung_id)
276          GROUP BY
277              rechnung_id
278      ) AS ksum
279  GROUP BY
280      kunde_id
```

```

281  HAVING umsatzsumme > ALL (SELECT 100)
282  ;
283
284  SELECT kunde_id, kunde.nachname, kunde.vo
rname, SUM(umsatz) AS umsatzsumme
285  FROM
286  (
287  SELECT
288      rechnung_id, kunde_id, SUM(einzelpreis*
menge) AS umsatz
289      FROM
290      rechnung_position INNER JOIN artikel US
ING(artikel_id)
291      INNER JOIN rechnung USING(rechnung_id)
292      GROUP BY
293      rechnung_id
294      ) AS ksum
295      INNER JOIN kunde USING(kunde_id)
296      GROUP BY
297      kunde_id
298      HAVING umsatzsumme > ALL
299      (
300      SELECT SUM(umsatz) AS umsatzsumme
301  FROM
302      (
303      SELECT
304      rechnung_id, kunde_id, SUM(einzelprei
s*menge) AS umsatz
305      FROM
306      rechnung_position INNER JOIN artikel
USING(artikel_id)
307      INNER JOIN rechnung USING(rechnung_id
)
308      INNER JOIN kunde USING(kunde_id)
309      WHERE nachname = 'beutlin'
310      GROUP BY
311      rechnung_id
312      ) AS ksum

```

```
313     GROUP BY
314     kunde_id
315 )
316 ;
317
318 SELECT a.einzelpreis
319 FROM
320 artikel_nm_warengruppe INNER JOIN artikel
   a USING(artikel_id)
321 INNER JOIN warengruppe w USING(warengrupp
   e_id)
322 WHERE
323 w.bezeichnung = 'Bürobedarf'
324 ;
325
326 SELECT
327 a.bezeichnung, a.einzelpreis
328 FROM
329 artikel a
330 WHERE
331 a.einzelpreis > ANY(SELECT 10)
332 ;
333 SELECT
334 a.bezeichnung, a.einzelpreis
335 FROM
336 artikel a
337 WHERE
338 a.einzelpreis > ANY
339 (
340     SELECT a.einzelpreis
341     FROM
342     artikel_nm_warengruppe INNER JOIN artik
   el a USING(artikel_id)
343     INNER JOIN warengruppe w USING(warengru
   ppe_id)
344     WHERE
345     w.bezeichnung = 'Bürobedarf'
346 )
```

```

347 ;
348
349 SELECT
350 a.bezeichnung, a.einzelpreis
351 FROM
352 artikel a
353 WHERE
354 a.artikel_id NOT IN
355 (
356     SELECT
357     artikel_id
358     FROM
359         artikel_nm_warengruppe nm INNER JOIN
warengruppe w USING(warengruppe_id)
360     WHERE w.bezeichnung = 'Bürobedarf'
361 )
362 AND
363 a.einzelpreis > ANY
364 (
365     SELECT a.einzelpreis
366     FROM
367         artikel_nm_warengruppe nm INNER JOIN
artikel a USING(artikel_id)
368     INNER JOIN warengruppe w USING(wareng
ruppe_id)
369     WHERE
370     w.bezeichnung = 'Bürobedarf'
371 )
372 ;
373
374 SELECT DISTINCT k.nachname, k.vorname
375 FROM
376 rechnung r INNER JOIN kunde k USING(kunde
_id)
377 WHERE
378     YEAR(r.datum) = '2011' AND (k.nachname,
k.vorname) IN
379 (

```

```

380      SELECT
381      k.nachname, k.vorname
382      FROM
383      rechnung r INNER JOIN kunde k USING(k
unde_id)
384      WHERE
385      YEAR(r.datum) = '2012'
386  )
387  ;
388  -- Ende nicht korrelierende Unterabfrage
389
390  -- Start korrelierende Unterabfrage
391  SELECT 'Korrelierende Unterabfrage (Abschnitt 13.3.1)';
392  SELECT rechnung_id
393      FROM rechnung r
394      WHERE
395      (
396      SELECT
397          COUNT(position_nr)
398          FROM rechnung_position rp
399          WHERE
400          rp.rechnung_id = r.rechnung_id
401      ) >= 3
402  ;
403  SELECT rechnung_id
404      FROM rechnung r
405      WHERE
406      EXISTS
407      (
408      SELECT
409          bestellung_id
410          FROM bestellung b
411          WHERE b.bestellung_id = r.bestellung_
id
412      )
413  ;
414  -- Ende korrelierende Unterabfrage

```

```
415
416 -- Start CTE
417 SELECT 'CTE vs SUBSELECT (Abschnitt 13.5)
';
418 WITH
419 rsum AS (
420 SELECT
421 rechnung_id, kunde_id, nachname, vorname,
  SUM(einzelpreis*menge) AS umsatz_r
422 FROM
423 rechnung_position INNER JOIN artikel USING(artikel_id)
424 INNER JOIN rechnung USING(rechnung_id)
425 INNER JOIN kunde USING(kunde_id)
426 GROUP BY rechnung_id
427 ),
428 ksum AS (
429 SELECT
430 kunde_id, nachname, vorname, SUM(umsatz_r
) AS umsatz_k
431 FROM rsum
432 GROUP BY kunde_id
433 ),
434 bsum AS (
435 SELECT umsatz_k
436 FROM ksum
437 WHERE nachname='Beutlin'
438 )
439 SELECT kunde_id, nachname, vorname, umsatz_k
440 FROM ksum
441 WHERE umsatz_k > ALL (SELECT * FROM bsum)
442 ;
443 -- Ende CTE
444
445 -- Start Fallstudie
446 SELECT 'Fallstudie Datenimport (Abschnitt 13.5)';
```



```
447 DROP TABLE IF EXISTS tmp_import;
448 CREATE TEMPORARY TABLE tmp_import
449 (
450     nachname VARCHAR(255),
451     vorname VARCHAR(255),
452     strasse VARCHAR(255),
453     hnr VARCHAR(255),
454     lkz VARCHAR(255),
455     plz VARCHAR(255),
456     ort VARCHAR(255),
457     ktnr VARCHAR(255),
458     blz VARCHAR(255)
459 );
460
461 SET GLOBAL local_infile = ON;
462 LOAD DATA LOCAL INFILE 'kunden01.csv'
463 INTO TABLE tmp_import
464 FIELDS
465 TERMINATED BY ';'
466 LINES
467 TERMINATED BY '\n'
468 IGNORE 1 LINES
469 (nachname, vorname, strasse, hnr, lkz, pl
470 z, ort, ktnr, blz)
471 ;
472 SET GLOBAL local_infile = OFF;
473
474 INSERT INTO kunde (nachname, vorname)
475 SELECT DISTINCT nachname, vorname
476     FROM tmp_import tmp
477     WHERE
478         (tmp.vorname, tmp.nachname) NOT IN
479         (
480             SELECT vorname, nachname FROM kunde
481         )
482 ;
483 INSERT INTO adresse (strasse, hnr, lkz, p
```

```

lz, ort)
484 SELECT DISTINCT strasse, hnr, lkz, plz, o
rt
485 FROM tmp_import tmp
486 WHERE
487     (tmp.strasse, tmp.hnr, tmp.lkz, tmp.p
lz, tmp.ort) NOT IN
488     (
489     SELECT strasse, hnr, lkz, plz, ort
490     FROM adresse
491     )
492 ;
493
494 ALTER TABLE tmp_import ADD kunde_id INT U
NSIGNED,
495 ADD adresse_id INT UNSIGNED,
496 ADD bank_id CHAR(12)
497 ;
498
499 UPDATE tmp_import t
500 SET
501     t.kunde_id =
502     (
503     SELECT kunde_id
504     FROM kunde k
505     WHERE t.vorname = k.vorname AND t.nachn
ame = k.nachname
506     ),
507     t.adresse_id =
508     (
509     SELECT adresse_id
510     FROM adresse a
511     WHERE
512     t.strasse = a.strasse
513     AND t.hnr = a.hnr
514     AND t.plz = a.plz
515     AND t.ort = a.ort
516     ),

```

```
517  t.bank_id =
518      (
519          SELECT bank_id FROM bank WHERE t.blz =
bank.blz LIMIT 1
520      )
521  ;
522
523  INSERT INTO
524  bankverbindung (kunde_id, bankverbindung_
nr, bank_id, kontonummer, iban)
525  SELECT DISTINCT kunde_id, 1, bank_id, ktn
r, CONCAT(blz, ktnr)
526      FROM tmp_import tmp
527      WHERE
528      (tmp.kunde_id, 1, tmp.bank_id, tmp.ktnr
) NOT IN
529      (
530          SELECT kunde_id, 1, bank_id, kontonumme
r
531          FROM bankverbindung
532      )
533  ;
534
535  ALTER TABLE tmp_import
536  ADD bankverbindung_nr INT UNSIGNED NOT NU
LL DEFAULT 1
537  ;
538
539  UPDATE kunde k SET
540  k.rechnung_adresse_id =
541      (
542          SELECT adresse_id FROM tmp_import tmp W
HERE tmp.kunde_id = k.kunde_id
543      )
544  WHERE k.rechnung_adresse_id IS NULL
545  ;
546
547  SELECT kunde_id, nachname, vorname, stras
```

```

se, ort
548 FROM
549 kunde LEFT JOIN adresse ON kunde.rechnung
_adresse_id = adresse_id
550 ;
551 -- Ende Fallstudie
552
553
554 SELECT 'Ende listing10. sql //////////////////////////////////
////////////////////////////////////////';

```

Listing 28.11 mysql/listing11.sql

```

1  -- Ausführen der vorherigen Befehle
2  SOURCE listing10.sql
3
4  SELECT 'Start listing11. sql //////////////////////////////////
////////////////////////////////////////';
5
6  SELECT 'UNION (Abschnitt 14.1)';
7  SELECT
8  strasse, hnr, lkz, plz, ort
9  FROM
10 kunde INNER JOIN adresse ON rechnung_adre
sse_id = adresse_id
11 ;
12 SELECT
13 strasse, hnr, lkz, plz, ort
14 FROM
15 lieferant INNER JOIN adresse USING(adress
e_id)
16 ;
17 SELECT
18     strasse, hnr, lkz, plz, ort
19     FROM
20     kunde INNER JOIN adresse ON rechnung_ad
resse_id = adresse_id
21     UNION

```

```

22  SELECT
23      strasse, hnr, lkz, plz, ort
24  FROM
25      lieferant INNER JOIN adresse USING(adre
sse_id)
26  ;
27  SELECT
28      strasse, hnr, lkz, plz, ort
29  FROM
30      kunde INNER JOIN adresse ON rechnung_adre
sse_id = adresse_id
31  UNION ALL
32  SELECT
33      strasse, hnr, lkz, plz, ort
34  FROM
35      lieferant INNER JOIN adresse USING(address
e_id)
36  ;
37
38  SELECT 'INTERSECT (Abschnitt 14.2.1)';
39  SELECT 'Schnittmenge mit Unterabfrage (Ab
schnitt 14.2.2)';
40  SELECT DISTINCT kunde_id FROM rechnung WH
ERE YEAR(datum) = '2011';
41  SELECT DISTINCT kunde_id
42  FROM rechnung
43  WHERE YEAR(datum) = '2012' AND kunde_id I
N (1, 2, 3)
44  ;
45  SELECT DISTINCT kunde_id
46      FROM rechnung
47      WHERE
48          YEAR(datum) = '2012' AND kunde_id IN
49          (
50          SELECT DISTINCT kunde_id FROM rechnun
g WHERE YEAR(datum) = '2011'
51      )
52  ;

```

```

53
54 SELECT 'EXCEPT (Abschnitt 14.3.1)';
55
56 SELECT 'Differenz mit Unterabfrage (Abschnitt 14.3.1)';
57 SELECT
58 strasse, hnr, lkz, plz, ort
59 FROM
60 adresse
61 ;
62 SELECT
63 strasse, hnr, lkz, plz, ort
64 FROM
65     kunde INNER JOIN adresse
66     ON liefer_adresse_id = adresse_id
67 ;
68 SELECT DISTINCT kunde_id FROM rechnung WH
ERE YEAR(datum) < 2012;
69 SELECT DISTINCT kunde_id
70 FROM rechnung
71 WHERE
72 YEAR(datum) = 2012 AND kunde_id NOT IN (1
, 2, 3)
73 ;
74 SELECT DISTINCT kunde_id
75     FROM rechnung
76     WHERE
77     YEAR(datum) = '2012' AND kunde_id NOT I
N
78     (
79     SELECT DISTINCT kunde_id FROM rechnung
WHERE YEAR(datum) < 2012
80     )
81 ;
82 SELECT 'Ende listing11. sql //////////////////////////////////
////////////////////////////////';

```

Listing 28.12 mysql/listing12.sql

```

1  -- Ausführen der vorherigen Befehle
2  SOURCE listing11.sql
3
4  SELECT 'Start listing12. sql //////////////////////////////////
////////////////////////////////////////';
5
6  SELECT kunde_id, nachname, vorname, art F
ROM kunde;
7  SELECT 'Start Aufbau einer temporären art
-Tabelle (Abschnitt 15.1)';
8
9  DROP TABLE IF EXISTS tmp_art;
10 CREATE TEMPORARY TABLE tmp_art (
11  art VARCHAR(255),
12  k_art VARCHAR(255)
13 )
14 ;
15
16 INSERT INTO tmp_art
17 VALUES
18     ('prv', 'Privatkunde')
19     ,('gsch', 'Geschäftskunde')
20     ,('unb', 'Unbekannt')
21 ;
22
23 SELECT
24  kunde_id, nachname, vorname, k_art
25  FROM
26  kunde INNER JOIN tmp_art USING(art)
27  ;
28 SELECT 'Ende Aufbau einer temporaeren art
-Tabelle';
29
30 SELECT 'Lösung mit UNIONs (Abschnitt 15.1
)' ;
31 SELECT kunde_id, nachname, vorname, k_art
32  FROM
33  kunde INNER JOIN

```

```

34      (
35      SELECT 'prv' AS art, 'Privatkunde' AS k
_art
36      UNION
37      SELECT 'gsch' AS art, 'Geschäftskunde'
AS k_art
38      UNION
39      SELECT 'unb' AS art, 'Unbekannt' AS k_art
40      ) t
41      USING(art)
42      ;
43
44      SELECT 'Nö (Abschnitt 15.1)';
45      SELECT kunde_id, nachname, vorname,
46      CASE art
47          WHEN 'prv' THEN 'Privatkunde'
48          WHEN 'gsch' THEN 'Geschäftskunde'
49          ELSE 'Unbekannt'
50          END AS k_art
51      FROM kunde
52      ;
53
54      -- Start Einfacher CASE
55      SELECT 'Einfacher CASE (Abschnitt 15.2)';
56      SELECT artikel_id, bezeichnung,
57          CASE waehrung
58              WHEN 'EUR' THEN LPAD(CONCAT(FORMAT(einz
elpreis, 2, 'de_DE'), ' e'), 7, ' ')
59              WHEN 'USD' THEN LPAD(CONCAT(FORMAT(einz
elpreis, 2, 'de_DE'), ' $'), 7, ' ')
60              ELSE '?????'
61          END AS preis
62      FROM artikel
63      ;
64      UPDATE artikel SET waehrung = 'XYZ' WHERE
artikel_id = 3001;
65      SELECT artikel_id, bezeichnung,
66      CASE waehrung

```



```

67  WHEN 'EUR' THEN LPAD(CONCAT(FORMAT(einzel
preis, 2, 'de_DE'),' e'), 7, ' ')
68  WHEN 'USD' THEN LPAD(CONCAT(FORMAT(einzel
preis, 2, 'de_DE'),' $'), 7, ' ')
69  END AS preis
70  FROM artikel
71  ;
72  UPDATE artikel SET waehrung = 'EUR' WHERE
artikel_id = 3001;
73  -- Ende Einfacher CASE
74
75  -- Start Searched CASE
76  SELECT 'Searched CASE (Abschnitt 15.3)';
77  SELECT beitrags_id,
78  CASE
79  WHEN bezug_bei trags_id > 1 THEN 'Antwort'
80  WHEN bezug_bei trags_id <= 1 THEN 'Keine An
twort'
81  ELSE '???????'
82  END AS Typ
83  FROM beitrags
84  ORDER BY beitrags_id
85  ;
86  SELECT beitrags_id,
87  CASE
88  WHEN bezug_bei trags_id > 1
89  THEN CONCAT(nachricht, ' Antwort auf ',
bezug_bei trags_id, ': >',
90  (
91  SELECT nachricht
92  FROM beitrags b2
93  WHERE b2.bei trags_id = b1.bezug_bei trags_
id
94  )
95  ) -- Ende CONCAT
96  WHEN bezug_bei trags_id <= 1 THEN nachricht
97  ELSE '???????'
98  END AS Inhalt

```

```

99  FROM beitrags b1
100 WHERE beitrags_id > 1
101 ;
102 SELECT beitrags_id,
103 CASE
104 WHEN bezug_beitrags_id >= 0 THEN 'Antwort'
105 WHEN bezug_beitrags_id > 1 THEN 'Keine Ant
wort'
106 ELSE '???????'
107 END AS Typ
108 FROM beitrags
109 ORDER BY beitrags_id
110 ;
111 -- Ende Searched CASE
112
113 -- Start Fallbeispiele
114 SELECT 'Fallbeispiel Lagerbestand (Abschn
itt 15.4.1)';
115 SELECT * FROM lagerbestand;
116 SELECT a.bezeichnung,
117 CASE
118 WHEN l.menge_aktuell <= l.menge_mindest
119 THEN 'Artikel nachbestellen'
120 ELSE 'Bestand ausreichend'
121 END AS lagerstand
122 FROM lagerbestand l INNER JOIN artikel a
USING(artikel_id)
123 ORDER BY l.menge_aktuell / l.menge_mindest
124 ;
125
126 SELECT 'Fallbeispiel Kundengruppen (Absch
nitt 15.4.2)';
127 SELECT DISTINCT k.kunde_id, k.nachname, k
.vorname,
128 CASE
129 WHEN 3 <
130 (

```

```

131     SELECT COUNT(rechnung_id) FROM rechnung
132     r
133     WHERE r.kunde_id = k.kunde_id
134     ) THEN 'Prämiumkunde'
135     WHEN 2 <
136     (
137     SELECT COUNT(rechnung_id) FROM rechnung
138     r
139     WHERE r.kunde_id = k.kunde_id
140     ) THEN 'Guter Kunde'
141     ELSE 'Kleinkunde'
142     END kundenart
143     FROM
144     rechnung r RIGHT OUTER JOIN kunde k USING
145     (kunde_id)
146     ;
147     SELECT k.kunde_id, k.nachname, k.vorname,
148     (
149     SELECT SUM(a.einzelpreis * rp.menge)
150     FROM rechnung r INNER JOIN rechnung_pos
151     ition rp
152     USING (rechnung_id)
153     INNER JOIN artikel a
154     USING (artikel_id)
155     WHERE r.kunde_id = k.kunde_id
156     ) rechnungswert,
157     CASE
158     WHEN
159     kunde_id NOT IN (SELECT kunde_id FROM r
160     echnung)
161     THEN 'Kleinkunde'
162     WHEN
163     (
164     SELECT SUM(a.einzelpreis * rp.menge) re
165     chnungswert
166     FROM rechnung r INNER JOIN rechnung_pos
167     ition rp
168     USING (rechnung_id)

```

```

162             INNER JOIN artikel a
163             USING (artikel_id)
164     WHERE r.kunde_id = k.kunde_id
165     ) BETWEEN 0 AND 300 THEN 'Kleinkunde'
166     WHEN
167     (
168     SELECT SUM(a.einzelpreis * rp.menge) re
rechnungswert
169     FROM rechnung r INNER JOIN rechnung_pos
ition rp
170             USING (rechnung_id)
171             INNER JOIN artikel a
172             USING (artikel_id)
173     WHERE r.kunde_id = k.kunde_id
174     ) BETWEEN 300 AND 1000 THEN 'Guter Kund
e'
175     ELSE 'Prämiumkunde'
176     END AS kundenart
177     FROM kunde k
178     ORDER BY
179     (
180     SELECT SUM(a.einzelpreis * rp.menge) rech
nungswert
181     FROM rechnung r INNER JOIN rechnung_posit
ion rp
182             USING (rechnung_id)
183             INNER JOIN artikel a
184             USING (artikel_id)
185     WHERE r.kunde_id = k.kunde_id
186     ) DESC
187     ;
188     DROP TABLE IF EXISTS tmp_umsatz;
189     CREATE TEMPORARY TABLE tmp_umsatz
190     SELECT kunde_id, SUM(a.einzelpreis * rp.m
enge) rechnungswert
191     FROM rechnung r INNER JOIN rechnung_p
osition rp USING (rechnung_id)
192     INNER JOIN artikel a USING (artikel_id)

```

```

193 GROUP BY kunde_id
194 ;
195 SELECT k.kunde_id, k.nachname, k.vorname,
    tmp.rechnungswert,
196     CASE
197     WHEN
198     tmp.kunde_id IS NULL THEN 'Kleinkunde'
199     WHEN
200     rechnungswert BETWEEN 0 AND 300 THEN 'K
leinkunde'
201     WHEN
202     rechnungswert BETWEEN 300 AND 1000 THEN
    'Guter Kunde'
203     ELSE 'Premiumkunde'
204 END AS kundenart
205 FROM kunde k LEFT JOIN tmp_umsatz tmp USI
NG(kunde_id)
206 ORDER BY rechnungswert DESC
207 ;
208
209 WITH tmp_umsatz (kunde_id, rechnungswert)
210 AS (
211 SELECT kunde_id, SUM(a.einzelpreis * rp.m
enge) rechnungswert
212     FROM rechnung r INNER JOIN rechnung_pos
ition rp USING (rechnung_id)
213     INNER JOIN artikel a USING (artikel_id)
214 GROUP BY kunde_id
215 )
216 SELECT k.kunde_id, k.nachname, k.vorname,
    tmp.rechnungswert,
217     CASE
218     WHEN
219     tmp.kunde_id IS NULL THEN 'Kleinkunde'
220     WHEN
221     rechnungswert BETWEEN 0 AND 300 THEN 'K
leinkunde'
222     WHEN

```

```

223     rechnungswert BETWEEN 300 AND 1000 THEN
224     'Guter Kunde'
225 END AS kundenart
226 FROM kunde k LEFT JOIN tmp_umsatz tmp USI
227 NG(kunde_id)
228 ORDER BY rechnungswert DESC
229 ;
230 SELECT 'Fallbeispiel Lieferanten (Abschnitt 15.4.3)';
231 SELECT l.firmenname,
232 CASE
233     WHEN
234     l.lieferant_id IN
235     (
236     SELECT lieferant_id FROM artikel_nm_lie
237     ferant
238     ) THEN 'aktiv'
239 ELSE 'inaktiv'
240 END AS status
241 FROM lieferant l
242 ORDER BY firmenname
243 ;
244 SELECT a.bezeichnung,
245 CASE
246     WHEN l.menge_aktuell <= l.menge_minde
247     st THEN 'Artikel sofort nachbestellen'
248     WHEN l.menge_aktuell <= l.menge_minde
249     st + 0.3 * l.menge_mindest THEN 'Artikel bald
250     nachbestellen'
251     ELSE 'Bestand ausreichend' END AS lag
252 erstand FROM lagerbestand l INNER JOIN artikel
253 a USING(artikel_id)
254 ORDER BY l.menge_aktuell / l.menge_mi
255 ndest
256 ;
257 -- Ende Fallbeispiele

```

```
251 SELECT 'Ende listing12. sql //////////////////////////////////  
////////////////////////////////////////';
```

Listing 28.13 mysql/listing13.sql

```
1  -- Ausführen der vorherigen Befehle  
2  SOURCE listing12.sql  
3  
4  SELECT 'Start listing13. sql //////////////////////////////////  
////////////////////////////////////////';  
5  
6  SELECT 'Meine erste Ansicht (Abschnitt 16  
.1.1)';  
7  CREATE VIEW  
8  view_kundenrechnungsadresse (id, nachname  
, vorname, strasse, ort)  
9  AS  
10 SELECT  
11     kunde_id, nachname, vorname,  
12     CONCAT_WS(' ', strasse, hnr),  
13     CONCAT_WS(' ', lkz, plz, ort)  
14 FROM  
15     kunde INNER JOIN adresse  
16     ON rechnung_adresse_id = adresse_id  
17 WHERE kunde.deleted = 0  
18 ;  
19 SELECT * FROM view_kundenrechnungsadresse  
;  
20 SHOW TABLES;  
21  
22 SHOW FULL TABLES  
23 IN oshop  
24 WHERE TABLE_TYPE = 'VIEW'  
25 ;  
26  
27 SELECT 'VIEW-Verarbeitung (Abschnitt 16.1  
.2)';  
28 CREATE VIEW view_artikel_aktiv
```

```

29  AS
30      SELECT *
31      FROM artikel
32      WHERE deleted = 0
33  ;
34  SELECT FLOOR(artikel_id / 1000) gruppe, A
VG(einzelpreis)
35      FROM view_artikel_aktiv
36      GROUP BY gruppe
37  ;
38  EXPLAIN SELECT FLOOR(artikel_id / 1000) g
ruppe, AVG(einzelpreis)
39  FROM view_artikel_aktiv
40  GROUP BY gruppe
41  \G
42
43  SELECT artikel_id, bezeichnung FROM
44  view_artikel_aktiv
45  WHERE
46  artikel_id BETWEEN 7000 AND 9000
47  ;
48  EXPLAIN SELECT artikel_id, bezeichnung FR
OM
49  view_artikel_aktiv
50  WHERE
51  artikel_id BETWEEN 7000 AND 9000
52  \G
53
54  CREATE ALGORITHM=TEMPTABLE VIEW view_arti
kel_aktiv_tmp
55  AS
56      SELECT * FROM artikel
57      WHERE deleted = 0
58  ;
59  SELECT artikel_id, bezeichnung
60  FROM view_artikel_aktiv_tmp
61  WHERE artikel_id BETWEEN 7000 AND 8000
62  ;

```



```
63 SELECT artikel_id, bezeichnung
64 FROM view_artikel_aktiv_tmp
65 WHERE artikel_id BETWEEN 3000 AND 4000
66 ;
67 EXPLAIN SELECT artikel_id, bezeichnung
68 FROM view_artikel_aktiv_tmp
69 WHERE artikel_id BETWEEN 3000 AND 4000
70 \G
71
72 -- Start DROP VIEW
73 SELECT 'Ansicht löschen (Abschnitt 16.1.3
)';
74 SHOW TABLES;
75 DROP VIEW view_artikel_aktiv_tmp;
76 SHOW TABLES;
77
78 DROP DATABASE IF EXISTS tmp1;
79 CREATE DATABASE tmp1;
80 USE tmp1;
81 CREATE TABLE a (i INT);
82 CREATE TABLE b (i INT);
83 CREATE VIEW ab
84 AS
85 SELECT * FROM a INNER JOIN b USING(i)
86 ;
87 SHOW TABLES;
88 DROP TABLE a;
89 SHOW TABLES;
90 -- SELECT 'Erwarte eine Fehlermeldung';
91 -- SELECT * FROM ab;
92 CHECK TABLE ab\G
93 DROP DATABASE tmp1;
94
95 -- Ende DROP VIEW
96 USE oshop;
97 SELECT 'Anwendungsgebiet: Vereinfachung (Abschnitt 16.2)';
98 CREATE OR REPLACE
```

```

99  ALGORITHM = TEMPTABLE
100 VIEW view_artikel_aktiv
101     AS
102     SELECT * FROM artikel
103     WHERE deleted = 0
104     ;
105 CREATE OR REPLACE VIEW view_artikel_verfü
gbar
106 AS
107     SELECT artikel_id, bezeichnung
108     FROM artikel INNER JOIN lagerbestand US
ING(artikel_id)
109     WHERE menge_aktuell >= menge_mindest
110     ORDER BY artikel_id
111     ;
112 SELECT * FROM view_artikel_verfügbar;
113 CREATE OR REPLACE
114 ALGORITHM = TEMPTABLE
115 VIEW view_lagerbestand_aktiv
116     AS
117     SELECT * FROM lagerbestand
118     WHERE deleted = 0
119     ;
120 CREATE OR REPLACE VIEW view_lagerbestand_
artikelbezeichnung
121 AS
122 SELECT
123     l.artikel_id, l.menge_mindest, l.menge_
aktuell, a.bezeichnung
124     FROM
125     view_lagerbestand_aktiv l INNER JOIN
view_artikel_aktiv a
126     USING(artikel_id)
127     ;
128 SELECT * FROM view_lagerbestand_artikelbe
zeichnung;
129
130 -- SELECT 'Grenzen der VIEW. Erwarte 3 Fe

```

```

hlermeldungen (Abschnitt 16.4)';
131  -- CREATE OR REPLACE VIEW v
132  -- AS SELECT bezeichnung FROM (SELECT * F
FROM artikel) a;
133  -- SET @b = 3000;
134  -- CREATE OR REPLACE VIEW v
135  -- AS SELECT * FROM artikel WHERE artikel
_id > @b;
136  DROP TABLE IF EXISTS tmp;
137  CREATE TEMPORARY TABLE tmp AS SELECT * FR
OM artikel;
138  -- CREATE OR REPLACE VIEW view_tmp AS SEL
ECT artikel_id FROM tmp;
139  DROP TABLE IF EXISTS view_tmp;
140  CREATE OR REPLACE VIEW view_tmp
141  AS SELECT nachname FROM kunde ORDER BY na
chname DESC;
142  SELECT * FROM view_tmp;
143  SELECT * FROM view_tmp ORDER BY nachname;
144  CREATE OR REPLACE VIEW view_tmp_eins
145  AS
146  SELECT nachname FROM kunde
147  ;
148  CREATE OR REPLACE VIEW view_tmp_zwei
149  AS
150      SELECT rechnung_id, COUNT(*) anzahl
151      FROM rechnung_position
152      GROUP BY rechnung_id
153  ;
154  UPDATE view_tmp_eins
155      SET nachname = 'Streicher'
156      WHERE nachname = 'Telcontar'
157  ;
158
159  -- SELECT 'Grenzen der VIEW. Erwarte Fehl
ermeldungen';
160  -- UPDATE view_tmp_zwei
161  -- SET anzahl = 2

```

```

162 -- WHERE anzahl > 2
163 -- ;
164 -- Rückbau
165 UPDATE view_tmp_eins
166 SET nachname = 'Telcontar'
167 WHERE nachname = 'Streicher'
168 ;
169 -- UPDATE view_tmp_zwei SET rechnung_id =
170   1 WHERE rechnung_id > 1;
170 INSERT INTO view_tmp_eins VALUES ('Wurst'
), ('Brot');
171 SELECT * FROM view_tmp_eins;
172 DELETE FROM view_tmp_eins WHERE nachname
IN ('Wurst', 'Brot');
173
174 SELECT 'Ende listing13. sql //////////////////////////////////
////////////////////////////////////////';

```

Listing 28.14 mysql/listing13NoSQL.js

```

1  // Globale Variablen (Abschnitt 17.3)
2  var connect = { host: 'localhost'
3                  , port: 33060
4                  , user: 'root'
5                  , password: 'weinschlauchX10'
6  };
7  var mysqlx = require('mysqlx');
8
9  //////////////////////////////////////////
10 // Skript 1: Verbinden, anlegen und schli
11 eßen (Abschnitt 17.3)
12 function warenkorbCollectionAnlegen(mySes
13 sion) {
14     var db0shop = mySession.getSchema('osho
15 p');
16     db0shop.dropCollection('warenkörbe');
17     return db0shop.createCollection('warenk

```

```

örbe');
15  }
16
17  var session = mysqlx.getSession(connect);
18  warenkorbCollectionAnlegen(session);
19  session.getSchema('oshop').getCollections
();
20  session.close();
21
22
23  ////////////////////////////////////////////
24  // Skript 2: Leere Warenkörbe verteilen (
Abschnitt 17.3.1)
25  function warenkorbLeeren(mySession) {
26      var db0shop = mySession.getSchema('osho
p');
27      var collWarenkörbe = db0shop.getCollect
ion('warenkörbe');
28
29      collWarenkörbe.remove("_id>0").execute(
);
30
31      // Für alle Kunden einen leeren Warenko
rb anlegen.
32      var tabKunde = db0shop.getTable("kunde"
);
33      var alleKunden = tabKunde.select("kunde
_id").execute();
34
35      var zeilen = alleKunden.fetchAll();
36      for (index in zeilen) {
37          var zeile = zeilen[index];
38          collWarenkörbe.add({
39              _id: zeile.kunde_id,
40              warenkorb: {
41                  kunde_id: zeile.kunde_id,
42                  positionen: new Array(),
43                  warenwerte: new Array()

```

```

44         }
45     }).execute();
46 }
47 }
48
49 var session = mysqlx.getSession(connect);
50 warenkorbLeeren(session);
51 session.getSchema('oshop').getCollection(
'warenkörbe').find();
52 mySession.close();
53
54
55 ////////////////////////////////////////////////////
56 // Skript 3: Kunde 2 legt drei neue Waren
korbbposition an (Abschnitt 17.3.2)
57 function warenkorbPositionEinfügen(mySess
ion, myKundeId, myArtikelId, myMenge) {
58     var dbOshop = mySession.getSchema('osho
p');
59     var collWarenkörbe = dbOshop.getCollect
ion('warenkörbe');
60     var sql = "SELECT";
61     sql += " artikel_id, bezeichnung, ein
zelpreis, waehrung";
62     sql += " FROM oshop.artikel";
63     sql += " WHERE artikel_id=?";
64     var artikel = mySession.sql(sql).bind(m
yArtikelId).execute().fetchOne();
65     var objArtikel = {
66         artikel_id: artikel.artikel_id,
67         artikelname: artikel.bezeichnung,
68         preis: artikel.einzelpreis
69     };
70     var objPosition = {
71         artikel: objArtikel,
72         menge: myMenge,
73         gesamtpreis: myMenge * artikel.einzel
preis,

```

```

74     währung: artikel.waehrung
75 };
76     collWarenkörbe.modify('_id=:param').arrayInsert('warenkorb.positionen[0]', {
77         position: objPosition
78     }).bind('param', myKundeId).execute()
79 ;
80
81 var session = mysqlx.getSession(connect);
82 warenkorbPositionEinfügen(session, 2, 3007, 10);
83 warenkorbPositionEinfügen(session, 2, 9010, 5);
84 warenkorbPositionEinfügen(session, 2, 3010, 2);
85 session.getSchema('oshop').getCollection('warenkörbe').find("_id=2");
86 session.close();
87
88
89 ///////////////////////////////////////////////////////////////////
90 // Skript 4: Aktualisierung von Gesamtpreis und Warenwert pro Währung ( Abschnitt 17.3.2)
91 function aktualisiereGesamtpreisWarenwert(myWarenkorb) {
92     var ergebnis = Object();
93     var posindizes = Object.keys(myWarenkorb.warenkorb.positionen);
94     posindizes.forEach(function(posindex) {
95         // Positionen durchwandern
96         var myPos = myWarenkorb.warenkorb.positionen[posindex].position;
97         // gesamtpreis der Position berechnen
98         myPos.gesamtpreis = myPos.artikel.preis * myPos.menge;
99         if (ergebnis.hasOwnProperty(myPos.wäh

```

```

rung)) {
    99         ergebnis[myPos.währung] += myPos.gesam
mtpreis;
100     } else {
101         ergebnis[myPos.währung] = myPos.gesam
tpreis;
102     }
103 });
104     var warenwert = new Array();
105     for (index in Object.keys(ergebnis)) {
106         var währung = Object.keys(ergebnis)[i
ndex];
107         var wert = ergebnis[währung];
108         warenwert.push({ [währung]: wert });
109     }
110
111     myWarenkorb.warenkorb.warenwerte = ware
nwert;
112     return myWarenkorb;
113 }
114
115
116     ////////////////////////////////////////////
117     // Skript 3a: insert mit Aktualisierung (
Abschnitt 17.3.2)
118     function warenkorbPositionEinfügen(mySess
ion, myKundeId, myArtikelId, myMenge) {
119         var db0shop = mySession.getSchema('osho
p');
120         var collWarenkörbe = db0shop.getCollect
ion('warenkörbe');
121         var sql = "SELECT";
122         sql += " artikel_id, bezeichnung, einze
lpreis, waehrung";
123         sql += " FROM oshop.artikel";
124         sql += " WHERE artikel_id=?";
125         var artikel = mySession.sql(sql).bind(m
yArtikelId).execute().fetchOne();

```



```

126     var objArtikel = {
127         artikel_id: artikel.artikel_id,
128         artikelname: artikel.bezeichnung,
129         preis: artikel.einzelpreis
130     };
131     var objPosition = {
132         artikel: objArtikel,
133         menge: myMenge,
134         gesamtpreis: myMenge * artikel.einzel
preis,
135         währung: artikel.waehrung
136     };
137     collWarenkörbe.modify('_id=:param').arr
ayInsert('warenkorb.positionen[0]', {
138         position: objPosition
139     }).bind('param', myKundeId).execute();
140     var wk = collWarenkörbe.find("_id=:para
m").bind('param', myKundeId).execute(). fetchO
ne();
141     wk = aktualisiereGesamtpreisWarenwert(w
k);
142     collWarenkörbe.modify('_id=:param').set
('$', wk).bind('param', myKundeId).execute ();
143 }
144
145 var session = mysqlx.getSession(connect);
146 warenkorbPositionEinfügen(session, 2, 300
7, 10);
147 warenkorbPositionEinfügen(session, 2, 901
0, 5);
148 warenkorbPositionEinfügen(session, 2, 301
0, 2);
149 session.getSchema('oshop').getCollection(
'warenkörbe').find("_id=2");
150 session.close();
151
152
153

```

```

154
155  ////////////////////////////////////////////
156  // Skript 5: Auswertung Warenkorb mit War
engruppe (Abschnitt 17.3.3)
157  function printWarenkorbMitWarengruppen(my
Session, myKundeId) {
158      var dbOshop = mySession.getSchema('osho
p');
159      var collWarenkörbe = dbOshop.getCollect
ion('warenkörbe');
160
161      var select = "SELECT";
162      select += " artikel.artikel_id,";
163      select += " GROUP_CONCAT(warengruppe.be
zeichnung) 'warengruppen'";
164      select += " FROM";
165      select += " oshop.artikel INNER JOIN os
hop.artikel_nm_warengruppe";
166      select += " USING(artikel_id)";
167      select += " INNER JOIN";
168      select += " oshop.warengruppe";
169      select += " USING(warengruppe_id)";
170      select += " GROUP BY artikel.artikel_id
";
171      var alleArtikel = mySession.sql(select)
.execute().fetchAll();
172      var arrArtikel = new Array();
173
174      for (index in alleArtikel) {
175          var zeile = alleArtikel[index];
176          arrArtikel.push({
177              artikel_id: zeile["artikel_id"],
178              warengruppen: zeile["warengruppen"]
179          });
180      }
181
182      var wk = collWarenkörbe.find("_id=:para
m").bind('param', myKundeId).execute(). fetch0

```

```

ne());
183     var n = 1;
184     var items = Object.keys(wk.warenkorb.po
sitionen);
185     items.forEach(function(item) {
186         var myPos = wk.warenkorb.positionen[i
tem].position;
187         var wg = arrArtikel.find(function(ele
ment) {
188             return element.artikel_id == myPos.ar
tikel.artikel_id
189         });
190         print('Warenkorbposition ' + n++ + ':
');
191         print(' ' + myPos.artikel.artikel_id)
;
192         print(', ' + myPos.artikel.artikelnam
e);
193         print(', "' + wg.warengruppen + '"');
194         print(', ' + myPos.artikel.preis);
195         print(', ' + myPos.menge);
196         print(', ' + myPos.gesamtpreis);
197         print(', ' + myPos.währung + "\n");
198     });
199 }
200 var session = mysqlx.getSession(connect);
201 printWarenkorbMitWarengruppen(session, 2)
;
202 session.close();
203
204
205 ///////////////////////////////////////////////////
206 // Skript 6: Auswertung Warenkorb 2 Anzahl
l der Bestellungen pro Warengruppe (
Abschnitt 17.3.3)
207 function erweiternUmWarengruppen(mySessio
n, myKundeId) {
208     var dbOshop = mySession.getSchema('osho

```

```

p');
209     var collWarenkörbe = dbOshop.getCollect
ion('warenkörbe');
210     var wk = collWarenkörbe.find("_id=:para
m").bind('param', myKundeId).execute(). fetchO
ne();
211
212     var items = Object.keys(wk.warenkorb.po
sitionen);
213     items.forEach(function(item) {
214         var myPos = wk.warenkorb.positionen[i
tem].position;
215
216         var select = "SELECT";
217         select += " warengruppe_id, bezeichnu
ng";
218         select += " FROM";
219         select += " oshop.artikel_nm_warengru
ppe";
220         select += " INNER JOIN";
221         select += " oshop.warengruppe";
222         select += " USING(warengruppe_id)";
223         select += " WHERE artikel_id = ?";
224
225         var wgs = mySession.sql(select).bind(
myPos.artikel.artikel_id).execute(). fetchAll(
);
226         arr = new Array();
227         for (index in wgs) {
228             var zeile = wgs[index];
229             arr.push({
230                 warengruppe_id: zeile.warengruppe_i
d,
231                 bezeichnung: zeile.bezeichnung
232             });
233         }
234         myPos.artikel.warengruppen = arr;
235     });

```

```

236     collWarenkörbe.modify('_id=:param').set
('$$', wk).bind('param', myKundeId).execute ();
237
238     return wk;
239 }
240
241 var session = mysqlx.getSession(connect);
242 erweiternUmWarengruppen(session, 2);
243 session.close();
244
245
246 ////////////////////////////////////////////////////
247 // Skript 7: Auswertung Anzahl der Bestel
lungen pro Warengruppe (Abschnitt 17.3.3)
248 function getAnzahlWarengruppe(mySession,
myKundeId) {
249     var wk = erweiternUmWarengruppen(mySessio
n, myKundeId);
250
251     var ergebnis = Object();
252     var posindizes = Object.keys(wk.warenko
rb.positionen);
253     posindizes.forEach(function(posindex) {
254         var myPos = wk.warenkorb.positionen[p
osindex].position;
255         var wgs = myPos.artikel.warengruppen;
256         var wgindizes = Object.keys(wgs);
257         wgindizes.forEach(function(wgindex) {
258             if (ergebnis.hasOwnProperty(wgs[wgi
ndex].bezeichnung)) {
259                 ergebnis[wgs[wgindex].bezeichnung]+
+;
260             } else {
261                 ergebnis[wgs[wgindex].bezeichnung]
= 1;
262             }
263         })
264     });

```

```

265
266 return ergebnis;
267 }
268
269 var session = mysqlx.getSession(connect);
270 print(getAnzahlBestellungenProWarengruppe
(session, 2));
271 session.close();
272
273
274 ////////////////////////////////////////////
275 // Skript 8: Ändern von Mengenangaben (Abschnitt 17.3.4)
276 function mengenÄndern(mySession, myKundeId, myArtikelId, myMenge) {
277     var db0shop = mySession.getSchema('osho
p');
278     var collWarenkörbe = db0shop.getCollect
ion('warenkörbe');
279     var wk = collWarenkörbe.find("_id=:para
m").bind('param', myKundeId).execute(). fetchO
ne();
280
281     for (index in Object.keys(wk.warenkorb.
positionen)) {
282         if (wk.warenkorb.positionen[index].po
sition.artikel.artikel_id == myArtikelId) {
283             wk.warenkorb.positionen[index].positi
on.menge = myMenge;
284         }
285     }
286     wk = aktualisiereGesamtpreisWarenwert(w
k);
287     collWarenkörbe.modify('_id=:param').set
('$', wk).bind('param', myKundeId).execute ();
288
289     return wk;
290 }

```

```

291
292  var session = mysqlx.getSession(connect);
293  mengenÄndern(session, 2, 3010, 0);
294  mengenÄndern(session, 2, 3007, 1);
295  mengenÄndern(session, 2, 9010, 2);
296  session.close();
297
298
299  ////////////////////////////////////////////
300  // Skript 9: Löschen von Positionen (Abschnitt 17.3.4)
301  function warenkorbPositionLöschen(mySession, myKundeId, myArtikelId) {
302      var db0shop = mySession.getSchema('oshop');
303      var collWarenkörbe = db0shop.getCollection('warenkörbe');
304      var wk = collWarenkörbe.find("_id=:param").bind('param', myKundeId).execute().fetchOne();
305
306      for (index in Object.keys(wk.warenkorb.positionen)) {
307          if (wk.warenkorb.positionen[index].position.artikel.artikel_id == myArtikelId) {
308              var str = 'warenkorb.positionen[' + index + ']';
309              collWarenkörbe.modify('_id=:param').arrayDelete(str).bind('param', myKundeId).execute();
310              wk = collWarenkörbe.find("_id=:param").bind('param', myKundeId).execute().fetchOne();
311              wk = aktualisiereGesamtpreisWarenwert(wk);
312          }
313      }
314

```

```

315     return wk;
316 }
317
318 var session = mysqlx.getSession(connect);
319 warenkorbPositionLöschen(session, 2, 3007
);
320 session.close();

```

Listing 28.15 mysql/listing13NoSQL.sql

```

1  -- Ausführen der vorherigen Befehle
2  -- SOURCE listing12.sql
3
4  SELECT 'Start listingNoSQL.sql //////////
//////////';
5  SELECT 'Hinzufügen der JSON-Spalte in kun
de (Abschnitt 17.4)';
6
7  ALTER TABLE kunde
8  ADD warenkorb JSON AFTER art
9  ;
10
11 UPDATE kunde
12 SET warenkorb = JSON_OBJECT(
13     'warenkorb',
14     JSON_OBJECT(
15         'kunde_id', kunde_id,
16         'positionen', JSON_ARRAY(),
17         'warenwerte', JSON_ARRAY()
18     )
19 )
20 ;
21 SELECT kunde_id, warenkorb FROM kunde;
22
23
24 SELECT 'Hinzufügen von Warenkorbpositione
n (Abschnitt 17.4.1)';
25 SELECT 'Kunde 2 10mal 3007';

```



```

26  SELECT 2 INTO @kunde_id;
27  SELECT 3007 INTO @artikel_id;
28  SELECT 10 INTO @menge;
29
30  SELECT warenkorb FROM kunde WHERE kunde_i
d = @kunde_id INTO @wk;
31  SELECT JSON_EXTRACT(@wk, "$.warenkorb.pos
itionen");
32
33  SELECT
34      JSON_OBJECT(
35          'position', JSON_OBJECT(
36              'artikel', JSON_OBJECT(
37                  'artikel_id', CAST(artikel_id
AS CHAR),
38                  'bezeichnung', bezeichnung,
39                  'einzelpreis', einzelpreis
40              ),
41              'menge', @menge,
42              'gesamtpreis', @menge * einze
lpreis,
43              'währung', waehrung
44          )
45      )
46  FROM artikel
47  WHERE artikel_id = @artikel_id
48  INTO @position
49  ;
50
51  UPDATE kunde
52  SET warenkorb = JSON_ARRAY_APPEND(
53      @wk,
54      "$.warenkorb.positionen",
55      CAST(@position AS JSON)
56  )
57  WHERE kunde_id = @kunde_id
58  ;
59

```

```

60  SELECT JSON_PRETTY(warenkorb) FROM kunde
WHERE kunde_id = @kunde_id\G
61
62  SELECT 'Kunde 2 2mal 3010';
63  SELECT 2 INTO @kunde_id;
64  SELECT 3010 INTO @artikel_id;
65  SELECT 2 INTO @menge;
66
67  SELECT warenkorb FROM kunde WHERE kunde_i
d = @kunde_id INTO @wk;
68  SELECT JSON_EXTRACT(@wk, "$.warenkorb.pos
itionen");
69
70  SELECT
71      JSON_OBJECT(
72          'position', JSON_OBJECT(
73              'artikel', JSON_OBJECT(
74                  'artikel_id', CAST(artikel_id
AS CHAR),
75                  'bezeichnung', bezeichnung,
76                  'einzelpreis', einzelpreis
77              ),
78              'menge', @menge,
79              'gesamtpreis', @menge * einze
lpreis,
80              'währung', waehrung
81          )
82      )
83  FROM artikel
84  WHERE artikel_id = @artikel_id
85  INTO @position
86  ;
87
88  UPDATE kunde
89  SET warenkorb = JSON_ARRAY_APPEND(
90      @wk,
91      "$.warenkorb.positionen",
92      CAST(@position AS JSON)

```

```

93         )
94 WHERE kunde_id = @kunde_id
95 ;
96
97 SELECT warenkorb FROM kunde WHERE kunde_i
d = @kunde_id\G
98
99 SELECT 'Kunde 2 5mal 9010';
100 SELECT 2 INTO @kunde_id;
101 SELECT 9010 INTO @artikel_id;
102 SELECT 5 INTO @menge;
103
104 SELECT warenkorb FROM kunde WHERE kunde_i
d = @kunde_id INTO @wk;
105 SELECT JSON_EXTRACT(@wk, "$.warenkorb.pos
itionen");
106
107 SELECT
108     JSON_OBJECT(
109         'position', JSON_OBJECT(
110             'artikel', JSON_OBJECT(
111                 'artikel_id', CAST(artikel_id
AS CHAR),
112                 'bezeichnung', bezeichnung,
113                 'einzelpreis', einzelpreis
114             ),
115             'menge', @menge,
116             'gesamtpreis', @menge * einze
lpreis,
117             'währung', waehrung
118         )
119     )
120 FROM artikel
121 WHERE artikel_id = @artikel_id
122 INTO @position
123 ;
124
125 UPDATE kunde

```

```

126 SET warenkorb = JSON_ARRAY_APPEND(
127     @wk,
128     "$.warenkorb.positionen",
129     CAST(@position AS JSON)
130 )
131 WHERE kunde_id = @kunde_id
132 ;
133
134 SELECT JSON_PRETTY(warenkorb) FROM kunde
WHERE kunde_id = @kunde_id\G
135
136
137 SELECT 'Auswerten von Warenkorbpositionen
';
138
139 SELECT 2 INTO @kunde_id;
140 SELECT warenkorb FROM kunde WHERE kunde_i
d = @kunde_id INTO @wk;
141
142 SELECT @kunde_id AS 'kunde_id', tt.*
143 FROM
144     JSON_TABLE(
145     @wk,
146     '$.warenkorb.positionen[*]',
147     COLUMNS (
148     posnr FOR ORDINALITY,
149     artikel_id INT PATH '$.position.artik
el.artikel_id',
150     einzelpreis DECIMAL(12,2) PATH '$.pos
ition.artikel.einzelpreis',
151     menge INT PATH '$.position.menge',
152     währung CHAR(3) PATH '$.position.währ
ung',
153     )
154     ) AS tt
155 ;
156
157 SELECT bezeichnung AS 'warengruppe', COUN

```

```

T(*) AS 'anzahl'
158     FROM
159     JSON_TABLE(
160     @wk,
161     '$.warenkorb.positionen[*]',
162     COLUMNS (
163     posnr FOR ORDINALITY,
164     artikel_id INT PATH '$.position.artik
el.artikel_id',
165     einzelpreis DECIMAL(12,2) PATH '$.pos
ition.artikel.einzelpreis',
166     menge INT PATH '$.position.menge',
167     währung CHAR(3) PATH '$.position.währ
ung',
168     )
169     ) AS wk
170     INNER JOIN artikel_nm_warengruppe USING
(artikel_id)
171     INNER JOIN warengruppe USING(warengrupp
e_id)
172     GROUP BY bezeichnung
173 ;
174
175 WITH wk AS (
176 SELECT * FROM JSON_TABLE(
177     @wk,
178     '$.warenkorb.positionen[*]',
179     COLUMNS (
180     posnr FOR ORDINALITY,
181     artikel_id INT PATH '$.position.artik
el.artikel_id',
182     einzelpreis DECIMAL(12,2) PATH '$.pos
ition.artikel.einzelpreis',
183     menge INT PATH '$.position.menge',
184     währung CHAR(3) PATH '$.position.währ
ung',
185     )
186     ) AS t

```

```

187 )
188 SELECT bezeichnung AS 'warengruppe', COUNT(*) AS 'anzahl'
189 FROM
190     wk INNER JOIN artikel_nm_warengruppe USING(artikel_id)
191     INNER JOIN warengruppe USING(warengruppe_id)
192 GROUP BY bezeichnung
193 ;
194
195
196 SELECT 'Ändern von Warenkorbpositionen (Abschnitt 17.4.3)';
197
198 SELECT 'Kunde 2 1mal 3007';
199 SELECT 2 INTO @kunde_id;
200 SELECT 3007 INTO @artikel_id;
201 SELECT 1 INTO @menge;
202
203 SELECT warenkorb FROM kunde WHERE kunde_id = @kunde_id INTO @wk;
204 SELECT JSON_SEARCH(@wk, 'one', @artikel_id) INTO @path;
205
206 SELECT CONCAT(
207     SUBSTRING(
208         @path,
209         2,
210         22+LOCATE('.', LEFT(@path, 13))
211     ),
212     '.position.menge'
213 )
214 INTO @path_to_pos;
215
216 UPDATE kunde
217 SET warenkorb = JSON_REPLACE(@wk, @path_to_pos

```

```

o_pos, @menge)
218 WHERE kunde_id = @kunde_id
219 ;
220 SELECT JSON_PRETTY(warenkorb) FROM kunde
WHERE kunde_id = @kunde_id\G
221
222 SELECT 'Kunde 2 0mal 3010';
223 SELECT 2 INTO @kunde_id;
224 SELECT 3010 INTO @artikel_id;
225 SELECT 0 INTO @menge;
226
227 SELECT warankorb FROM kunde WHERE kunde_i
d = @kunde_id INTO @wk;
228 SELECT JSON_SEARCH(@wk, 'one', @artikel_i
d) INTO @path;
229 SELECT CONCAT(
230     SUBSTRING(
231         @path,
232         2,
233         22+LOCATE('.', LEFT(@path, 13
))
234     ),
235     '.position.menge'
236 )
237 INTO @path_to_pos;
238
239 SELECT JSON_REPLACE(@wk, @path_to_pos, @m
enge);
240 UPDATE kunde
241 SET warankorb = JSON_REPLACE(@wk, @path_t
o_pos, @menge)
242 WHERE kunde_id = @kunde_id
243 ;
244 SELECT JSON_PRETTY(warenkorb) FROM kunde
WHERE kunde_id = @kunde_id\G
245
246 SELECT 'Kunde 2 10mal 9010';
247 SELECT 2 INTO @kunde_id;

```

```

248 SELECT 9010 INTO @artikel_id;
249 SELECT 10 INTO @menge;
250
251 SELECT warenkorb FROM kunde WHERE kunde_i
d = @kunde_id INTO @wk;
252 SELECT JSON_SEARCH(@wk, 'one', @artikel_i
d) INTO @path;
253 SELECT CONCAT(
254     SUBSTRING(
255         @path,
256         2,
257         22+LOCATE('.', LEFT(@path, 13
))
258     ),
259     '.position.menge'
260 )
261 INTO @path_to_pos;
262
263 SELECT JSON_REPLACE(@wk, @path_to_pos, @m
enge);
264 UPDATE kunde
265 SET warenkorb = JSON_REPLACE(@wk, @path_t
o_pos, @menge)
266 WHERE kunde_id = @kunde_id
267 ;
268 SELECT JSON_PRETTY(warenkorb) FROM kunde
WHERE kunde_id = @kunde_id\G
269
270
271 SELECT 'Kunde 2 1mal 3007 mit Aktualisier
ungen (Abschnitt 17.4.4)';
272 SELECT 2 INTO @kunde_id;
273 SELECT 3007 INTO @artikel_id;
274 SELECT 1 INTO @menge;
275
276 SELECT warenkorb FROM kunde WHERE kunde_i
d = @kunde_id INTO @wk;
277 SELECT JSON_SEARCH(@wk, 'one', @artikel_i

```



```

d) INTO @path;
278 SELECT CONCAT(SUBSTRING(@path, 2, 22+LOCATE('.', LEFT(@path, 13))), '.position.menge ')
279 INTO @path_to_pos;
280 SELECT CONCAT(SUBSTRING(@path, 2, 22+LOCATE('.', LEFT(@path, 13))), '.position.gesamtpreis')
281 INTO @path_to_gesamtpreis;
282 SELECT '$.warenkorb.warenwerte' INTO @path_to_warenwerte;
283
284 CREATE TEMPORARY TABLE wk
285 SELECT * FROM JSON_TABLE(
286     @wk,
287     '$.warenkorb.positionen[*]',
288     COLUMNS (
289         posnr FOR ORDINALITY,
290         artikel_id INT PATH '$.position.artikel.artikel_id',
291         einzelpreis DECIMAL(12,2) PATH '$.position.artikel.einzelpreis',
292         menge INT PATH '$.position.menge',
293         währung CHAR(3) PATH '$.position.währung',
294     )
295 ) AS t
296 ;
297
298 SELECT einzelpreis*menge
299 FROM wk
300 WHERE artikel_id = @artikel_id
301 INTO @gesamtpreis
302 ;
303
304 SELECT JSON_ARRAYAGG(warenwert)
305 FROM
306     (SELECT JSON_OBJECT(währung, SUM(einzelpreis*menge)) warenwert

```

```

307         FROM wk
308         GROUP BY währung) AS a
309 INTO @warenwerte
310 ;
311
312 UPDATE kunde
313 SET warenkorb = JSON_REPLACE(@wk,
314                             @path_to_pos, @menge,
315                             @path_to_gesamtpreis, @gesamt
preis,
316                             @path_to_warenwerte, CAST(@wa
renwerte AS JSON))
317 WHERE kunde_id = @kunde_id
318 ;
319 SELECT JSON_PRETTY(warenkorb) FROM kunde
WHERE kunde_id = @kunde_id\G
320
321 SELECT 'Löschen von Warenkorbpositionen (Abschnitt 17.4.5)';
322
323 SELECT 'Kunde 2 Position mit Artikel 9010
';
324 SELECT 2 INTO @kunde_id;
325 SELECT 9010 INTO @artikel_id;
326
327 SELECT warenkorb FROM kunde WHERE kunde_i
d = @kunde_id INTO @wk;
328 SELECT JSON_SEARCH(@wk, 'one', @artikel_i
d) INTO @path;
329
330 SELECT SUBSTRING(@path, 2, 22+LOCATE('.',
LEFT(@path, 13)))
331 INTO @path_to_pos;
332 UPDATE kunde
333 SET warenkorb = JSON_REMOVE(@wk, @path_to
_pos)
334 WHERE kunde_id = @kunde_id
335 ;

```

```

336 SELECT JSON_PRETTY(warenkorb) FROM kunde
WHERE kunde_id = @kunde_id\G
337
338 SELECT 'Kunde 2 Warenkorb leeren';
339 SELECT 2 INTO @kunde_id;
340
341 SELECT warnekorb FROM kunde WHERE kunde_i
d = @kunde_id INTO @wk;
342
343 UPDATE kunde
344 SET warnekorb = JSON_REPLACE(@wk,
345                               '$.warenkorb.positionen', JSO
N_ARRAY(),
346                               '$.warenkorb.warenwerte', JSO
N_ARRAY())
347 WHERE kunde_id = @kunde_id
348 ;
349 SELECT JSON_PRETTY(warenkorb) FROM kunde
WHERE kunde_id = @kunde_id\G
350 SELECT 'Ende listingNoSQL.sql //////////////////////////////////
////////////////////////////////////////';

```

28.1.4 Quelltexte zu Teil V

Listing 28.16 mysql/listing14.sql

```

1  -- Ausführen der vorherigen Befehle
2  SOURCE listing13.sql
3
4  SELECT 'Start listing14. sql //////////////////////////////////
////////////////////////////////////////';
5  SELECT 'Lagerbestand (Abschnitt 18)';
6  UPDATE lagerbestand
7  SET menge_aktuell = menge_aktuell - 5
8  WHERE artikel_id = 9015;
9

```

```

10 UPDATE lagerbestand
11 SET menge_aktuell = menge_aktuell - 3
12 WHERE artikel_id = 9015;
13
14 SELECT * FROM lagerbestand WHERE artikel_
id = 9015;
15
16 -- Start Lock
17 SELECT 'LOCK (Abschnitt 18)';
18 LOCK TABLES artikel WRITE;
19 SELECT COUNT(*) FROM artikel;
20 -- SELECT 'Erwarte eine Fehlermeldung (Ab
schnitt 16)';
21 -- SELECT COUNT(*) FROM warengruppe;
22 UNLOCK TABLES;
23 SELECT COUNT(*) FROM warengruppe;
24 -- Ende Lock
25 SELECT 'Ende listing14. sql //////////////////////////////////
//////////////////////////////////';

```

Listing 28.17 mysql/listing15.sql

```

1 -- Ausführen der vorherigen Befehle
2 SOURCE listing14.sql
3
4 SELECT 'Start listing15. sql //////////////////////////////////
//////////////////////////////////';
5 SELECT 'Das Problem (Abschnitt 19.1)';
6 INSERT INTO artikel
7 (bezeichnung, einzelpreis, waehrung)
8 VALUES
9 ('Säge', 17.85, 'EUR')
10 ;
11 SET @id = LAST_INSERT_ID();
12 INSERT INTO artikel_nm_warengruppe
13 VALUES
14 (3, @id),
15 (4, @id)

```

```
16 ;
17 UPDATE lagerbestand
18     SET menge_aktuell = menge_aktuell - 1
19     WHERE artikel_id = 9015
20 ;
21 UPDATE bestellung SET status = 'versendet
' WHERE bestellung_id = 1;
22
23 SELECT 'Isolationsebenen (Abschnitt 19.3)
';
24 SHOW VARIABLES LIKE 'AUTOCOMMIT';
25 SET AUTOCOMMIT = OFF;
26 SHOW VARIABLES LIKE 'AUTOCOMMIT';
27
28
29 SELECT 'READ UNCOMMITTED (Abschnitt 19.3.
1)';
30 -- Session 1
31 SET AUTOCOMMIT = 0;
32
33
34 SELECT artikel_id, menge_aktuell
35 FROM lagerbestand
36 WHERE artikel_id = 9015
37 ;
38 UPDATE lagerbestand
39 SET menge_aktuell = menge_aktuell - 1
40 WHERE artikel_id = 9015
41 ;
42
43 SELECT artikel_id, menge_aktuell
44 FROM lagerbestand
45 WHERE artikel_id = 9015
46 ;
47 ROLLBACK;
48 SELECT artikel_id, menge_aktuell
49 FROM lagerbestand
50 WHERE artikel_id = 9015
```

```

51 ;
52 -- Session 2
53 SET AUTOCOMMIT = 0;
54 START TRANSACTION;
55
56 SET SESSION TRANSACTION ISOLATION LEVEL
57 READ UNCOMMITTED;
58
59 SELECT artikel_id, menge_aktuell
60 FROM lagerbestand
61 WHERE artikel_id = 9015
62 ;
63 SELECT artikel_id, menge_aktuell
64 FROM lagerbestand
65 WHERE artikel_id = 9015
66 ;
67 SELECT artikel_id, menge_aktuell
68 FROM lagerbestand
69 WHERE artikel_id = 9015
70 ;
71 -- Ende read uncommitted
72
73 SELECT 'READ COMMITTED (Abschnitt 19.3.2)
';
74 -- Session 1
75 SET AUTOCOMMIT = 0;
76 START TRANSACTION; SELECT artikel_id, meng
e_aktuell
77 FROM lagerbestand
78 WHERE artikel_id = 9015
79 ;
80 UPDATE lagerbestand
81 SET menge_aktuell = menge_aktuell - 1
82 WHERE artikel_id = 9015
83 ;
84 SELECT artikel_id, menge_aktuell
85 FROM lagerbestand
86 WHERE artikel_id = 9015

```

```

87  ;
88  COMMIT;
89  SELECT artikel_id, menge_aktuell
90  FROM lagerbestand
91  WHERE artikel_id = 9015;
92  -- Session 2
93  SET AUTOCOMMIT = 0;
94  SET SESSION TRANSACTION ISOLATION LEVEL R
EAD COMMITTED;
95  START TRANSACTION;
96  SELECT artikel_id, menge_aktuell
97  FROM lagerbestand
98  WHERE artikel_id = 9015
99  ;
100 SELECT artikel_id, menge_aktuell
101 FROM lagerbestand
102 WHERE artikel_id = 9015
103 ;
104 SELECT artikel_id, menge_aktuell
105 FROM lagerbestand
106 WHERE artikel_id = 9015
107 ;
108 -- Ende read committed
109
110 SELECT 'REPEATABLE READ (Abschnitt 19.3.3
)';
111 -- Session 1
112 SET AUTOCOMMIT = 0;
113 SELECT artikel_id, menge_aktuell
114 FROM lagerbestand
115 WHERE artikel_id = 9015
116 ;
117 UPDATE lagerbestand
118 SET menge_aktuell = menge_aktuell - 1
119 WHERE artikel_id = 9015
120 ;
121 COMMIT;
122 SELECT artikel_id, menge_aktuell

```

```
123 FROM lagerbestand
124 WHERE artikel_id = 9015
125 ;
126 -- Session 2
127 SET AUTOCOMMIT = 0;
128 SET SESSION TRANSACTION ISOLATION LEVEL R
REPEATABLE READ;
129 START TRANSACTION;
130 SELECT artikel_id, menge_aktuell
131 FROM lagerbestand
132 WHERE artikel_id = 9015
133 ;
134 SELECT artikel_id, menge_aktuell
135 FROM lagerbestand
136 WHERE artikel_id = 9015
137 ;
138 COMMIT;
139 SELECT artikel_id, menge_aktuell
140 FROM lagerbestand
141 WHERE artikel_id = 9015
142 ;
143 -- Ende repeatable read
144
145 SELECT 'SERIALIZABLE (Abschnitt 19.3.4)';
146 -- Session 1
147 SET AUTOCOMMIT = 0;
148 SELECT COUNT(*) FROM artikel;
149 INSERT INTO artikel VALUES (1, 'X', 0.0,
'EUR', 0);
150 COMMIT;
151
152 -- Session 2
153 SET AUTOCOMMIT = 0;
154 SET SESSION TRANSACTION ISOLATION LEVEL S
ERIALIZABLE;
155 START TRANSACTION;
156 SELECT COUNT(*) FROM artikel;
157 DELETE FROM artikel WHERE artikel_id = 1;
```



```

158 DELETE FROM artikel WHERE artikel_id = 1;
159 SELECT COUNT(*) FROM artikel;
160 -- Ende Serializable
161 SET SESSION TRANSACTION ISOLATION LEVEL R
EPEATABLE READ;
162 SELECT 'Ende listing15. sql //////////////////////////////////
////////////////////////////////////////';

```

Listing 28.18 mysql/listing16.sql

```

1  -- Ausführen der vorherigen Befehle
2  SOURCE listing15.sql
3
4  SELECT 'Start listing16. sql //////////////////////////////////
////////////////////////////////////////';
5
6  -- Start Prozedur insert_artikel Version
1
7  SELECT 'Prozedur insert_artikel Version 1
';
8  DELIMITER //
9  CREATE PROCEDURE insert_artikel
10 (
11 IN iArtikelname VARCHAR(255),
12 IN iWarengruppe VARCHAR(255)
13 )
14
15 BEGIN
16
17 DECLARE v_artikel_id INT DEFAULT 0;
18 DECLARE v_warengruppe_id INT DEFAULT 0;
19
20
21     INSERT INTO artikel
22     (bezeichnung)
23     VALUES (iArtikelname);
24
25     SET v_artikel_id = LAST_INSERT_ID(); -- V

```

```

variable setzen
26
27     SELECT warengruppe_id
28     INTO v_warengruppe_id -- Oder so
29     FROM warengruppe
30     WHERE bezeichnung = iWarengruppe;
31
32     INSERT INTO artikel_nm_warengruppe -- E
intrag Hilfstabelle
33     (warengruppe_id, artikel_id)
34     VALUES
35     (v_warengruppe_id, v_artikel_id);
36
37 END// -- Anderer DELIMITER
38 DELIMITER ;
39 -- Start Prozedur insert_artikel Version
1
40 CALL insert_artikel ('Schlauch', 'Gartenb
edarf');
41 SELECT artikel_id, bezeichnung FROM artik
el ORDER BY artikel_id;
42 SELECT warengruppe_id, bezeichnung FROM w
arengruppe;
43 SELECT * FROM artikel_nm_warengruppe;
44 SHOW PROCEDURE STATUS\G
45
46 -- SELECT 'Verzweigung. Erwarte eine Fehl
ermeldung';
47 -- CALL insert_artikel ('Fleischwurst', '
Lebensmittel');
48
49 -- Start Prozedur insert_artikel Version
2
50 DROP PROCEDURE IF EXISTS insert_artikel;
51
52 DELIMITER //
53 CREATE PROCEDURE insert_artikel
54 (

```

```

55  IN iArtikelname VARCHAR(255),
56  IN iWarengruppe VARCHAR(255)
57  )
58
59  BEGIN
60
61      DECLARE v_artikel_id INT DEFAULT -1;
62      DECLARE v_warengruppe_id INT DEFAULT -1
;
63  DECLARE v_count_zuordnung INT DEFAULT 0;
64
65  -- Teste, ob Warengruppe vorhanden ist
66      SELECT warengruppe_id
67      INTO v_warengruppe_id
68      FROM warengruppe
69      WHERE bezeichnung = iWarengruppe;
70
71  -- Wenn nein, füge eine ein
72      IF v_warengruppe_id < 0 THEN -- Einfach
e Verzweigung
73          INSERT INTO warengruppe (bezeichnung) V
ALUES (iWarengruppe);
74          SET v_warengruppe_id = LAST_INSERT_ID()
;
75      END IF;
76
77  -- Teste, ob artikel vorhanden ist
78      SELECT artikel_id
79      INTO v_artikel_id
80      FROM artikel
81      WHERE bezeichnung = iArtikelname;
82
83  -- Wenn nein, teste auf füge ein
84      IF v_artikel_id < 0 THEN
85          INSERT INTO artikel (bezeichnung) VALUE
S (iArtikelname);
86          SET v_artikel_id = LAST_INSERT_ID();
87      END IF;

```

```

88
89  -- Test, ob Zuordnung schon vorhanden
90      SELECT COUNT(*)
91      INTO v_count_zuordnung
92      FROM artikel_nm_warengruppe
93      WHERE artikel_id = v_artikel_id AND war
engruppe_id = v_warengruppe_id;
94
95  -- Wenn nein, füge eine hinzu
96  IF v_count_zuordnung <= 0 THEN
97      INSERT INTO artikel_nm_warengruppe
98      (warengruppe_id, artikel_id)
99      VALUES
100      (v_warengruppe_id, v_artikel_id);
101  ELSE
102      SELECT 'Zuordnung schon vorhanden'; --
Hinweis
103  END IF;
104  END//
105  DELIMITER ;
106  -- Ende Prozedur insert_artikel Version 2
107  CALL insert_artikel ('Fleischwurst', 'Leb
ensmittel');
108  CALL insert_artikel ('Fleischwurst', 'Leb
ensmittel');
109
110  SELECT 'Mehrfachverzweigung';
111  -- Start Prozedur Umsatzreport
112  DELIMITER //
113  CREATE PROCEDURE umsatzreport
114  (
115  IN iZeitraum INT
116  )
117  BEGIN
118  DROP TABLE IF EXISTS umsatzzahlen;
119
120  CASE iZeitraum
121      WHEN 1 THEN -- Total

```

```

122      CREATE TABLE umsatzzahlen
123      SELECT 'Total', SUM(einzelpreis * men
124      ge) 'Umsatz'
125      FROM
126      rechnung_position INNER JOIN artikel
127      USING (artikel_id)
128      INNER JOIN rechnung USING (rechnung_i
129      d)
130      ;
131      WHEN 2 THEN -- Pro Jahr
132      CREATE TABLE umsatzzahlen
133      SELECT YEAR(datum) 'Jahr', SUM(einzel
134      preis * menge) 'Umsatz'
135      FROM
136      rechnung_position INNER JOIN artike
137      l USING (artikel_id)
138      INNER JOIN rechnung USING (rechnung
139      _id)
140      GROUP BY 'Jahr'
141      ORDER BY 'Jahr' DESC
142      ;
143      WHEN 3 THEN -- Pro Monat
144      CREATE TABLE umsatzzahlen
145      SELECT YEAR(datum) 'Jahr', MONTH(datu
146      m) 'Monat', SUM(einzelpreis * menge) ' Umsatz'
147      FROM
148      rechnung_position INNER JOIN artikel
149      USING (artikel_id)
150      INNER JOIN rechnung USING (rechnung_i
151      d)
152      GROUP BY 'Jahr', 'Monat'
153      ORDER BY 'Jahr' DESC, 'Monat' DESC
154      ;
155      ELSE -- So was sollte es immer geben
156      CREATE TABLE umsatzzahlen
157      SELECT 'Unbekannte Berichtsart', iZeitr
158      aum;
159      END CASE;

```

```

150 END//
151 DELIMITER ;
152 -- Ende Prozedur Umsatzreport
153 CALL umsatzreport(1); SELECT * FROM umsatz
zahlen;
154 CALL umsatzreport(2); SELECT * FROM umsatz
zahlen;
155 CALL umsatzreport(3); SELECT * FROM umsatz
zahlen;
156 CALL umsatzreport(0); SELECT * FROM umsatz
zahlen;
157
158 -- Start Prozedur b
159 DELIMITER //
160 CREATE PROCEDURE b()
161 BEGIN
162 DECLARE vZeitraum INT ;
163 SET vZeitraum = 2;
164 CASE
165     WHEN vZeitraum = 2 THEN SELECT 'A';
166     WHEN vZeitraum > 0 THEN SELECT 'B';
167     WHEN vZeitraum < 2 THEN SELECT 'C';
168 END CASE;
169 END//
170 DELIMITER ;
171
172 CALL b();
173 -- Ende Prozedur b
174
175 SELECT 'Schleifen';
176 -- Start Prozedur warte_auf_ueberlauf
177 DELIMITER //
178 CREATE PROCEDURE warte_auf_ueberlauf()
179 BEGIN
180 DECLARE vInt TINYINT;
181
182 SET vInt = 0;
183 LOOP -- Schleifenstart

```

```

184 SELECT vInt;
185 SET vInt = vInt + 1;
186 END LOOP; -- Schleifenende
187 END;
188 //
189 DELIMITER ;
190
191 -- CALL warte_auf_ueberlauf();
192 -- Ende Prozedur warte_auf_ueberlauf
193
194 -- Start Prozedur min_max
195 DELIMITER //
196 CREATE PROCEDURE min_max()
197 BEGIN
198 DECLARE vInt TINYINT;
199 DECLARE vMin INT;
200 DECLARE vMax INT;
201
202 SET vInt = 0;
203     maxsuche: LOOP -- Jetzt mit Label
204         SET vMax = vInt;
205         SET vInt = vInt + 1;
206         IF vMax = vInt THEN
207             LEAVE maxsuche; -- Verlasse Label
208         END IF;
209     END LOOP maxsuche;
210
211     SET vInt = 0;
212     minsuche: LOOP
213         SET vMin = vInt;
214         SET vInt = vInt - 1;
215         IF vMin = vInt THEN
216             LEAVE minsuche;
217         END IF;
218     END LOOP minsuche;
219
220 SELECT 'MIN = ', vMin, 'MAX = ', vMax;
221 END;

```

```

222 //
223 DELIMITER ;
224 CALL min_max();
225 -- Ende Prozedur min_max
226
227 -- Start Prozedur Sieb
228 DELIMITER //
229 CREATE PROCEDURE sieb
230 (
231   iGrenze BIGINT UNSIGNED
232 )
233 BEGIN
234   DECLARE vZahl BIGINT UNSIGNED DEFAULT 2;
235   -- Start mit 2
236   DECLARE vPosition BIGINT UNSIGNED DEFAULT
237     2;
238   -- Aufbau der Tabelle
239   DROP TABLE IF EXISTS zahlenstrahl;
240   CREATE TABLE zahlenstrahl
241   (
242     position BIGINT UNSIGNED,
243     zahl BIGINT UNSIGNED,
244     PRIMARY KEY(position)
245   );
246   WHILE vZahl <= iGrenze DO
247     INSERT INTO zahlenstrahl VALUES (vZahl, v
248     Zahl);
249     SET vZahl = vZahl + 1;
250   END WHILE;
251   -- Sieb
252   WHILE vPosition < CEILING(SQRT(iGrenze))
253     DO
254     SELECT zahl INTO vZahl FROM zahlenstrah
255     l WHERE position = vPosition;
256     IF vZahl > 0 THEN

```



```

255     UPDATE zahlenstrahl SET zahl = 0 WHERE
(position > vPosition) AND (zahl % vZahl = 0);
256     END IF;
257     SET vPosition = vPosition + 1;
258     END WHILE;
259
260
261 -- Bereinigen der Tabelle
262 DELETE FROM zahlenstrahl WHERE zahl = 0;
263 END//
264 DELIMITER ;
265 CALL sieb(20);
266 SELECT zahl FROM zahlenstrahl ORDER BY za
hl;
267 -- Ende Prozedur Sieb
268
269 -- Start Prozedur kopiere_blz
270 DROP DATABASE IF EXISTS tmpSchleife;
271 CREATE DATABASE tmpSchleife CHARACTER SET
utf8mb4;
272
273 USE tmpSchleife;
274
275 CREATE TABLE bank_quelle
276 (
277     blz CHAR(8),
278     merkmal CHAR(1),
279     bezeichnung VARCHAR(255),
280     plz CHAR(5),
281     ort VARCHAR(255),
282     kurz VARCHAR(255),
283     pan CHAR(5),
284     bic VARCHAR(255),
285     sm CHAR(2),
286     ds CHAR(6),
287     kz CHAR(1),
288     blzl CHAR(1),
289     blzn CHAR(8)

```

```

290 )
291 ;
292
293 SET GLOBAL local_infile = ON;
294 LOAD DATA LOCAL INFILE 'blz_20120305.csv'
295 INTO TABLE bank_quelle
296 FIELDS TERMINATED BY ';'
297 LINES TERMINATED BY '\n'
298 ;
299 SET GLOBAL local_infile = OFF;
300 CREATE TABLE bank_ziel
301 (
302 blz CHAR(8),
303 bezeichnung VARCHAR(255),
304 adresse TEXT
305 )
306 ;
307
308 DELIMITER //
309 CREATE PROCEDURE kopiere_blz
310 (
311 iAnzahl INT UNSIGNED -- Anzahl der Zeilen
   pro Kopiervorgang
312 )
313 BEGIN
314 DECLARE vOffset INT UNSIGNED; -- Aktuelle
   r Startpunkt
315 DECLARE vCount INT UNSIGNED; -- Anzahl de
   r zu kopierenden Zeilen
316
317 SELECT COUNT(*) FROM bank_quelle INTO vCo
   unt;
318
319 SET vOffset = 0;
320 WHILE vOffset < vCount DO -- Start der Sc
   hleife
321     SELECT 'vOffset ', vOffset; -- Nur zum
   gucken

```

```

322     INSERT INTO bank_ziel (blz, bezeichnung
, adresse)
323     SELECT blz, bezeichnung, CONCAT(plz,
' ', ort)
324     FROM bank_quelle
325     LIMIT vOffset, iAnzahl;
326     SET vOffset = vOffset + iAnzahl; -- Neu
er Startwert
327 END WHILE; -- Ende der Schleife
328 END//
329 DELIMITER ;
330
331 CALL kopiere_blz(1000);
332 USE oshop;
333 -- Ende Prozedur kopiere_blz
334
335 -- Start Prozedur ggt
336 DELIMITER //
337 CREATE PROCEDURE ggt
338 (
339     iZahl1 INT UNSIGNED,
340     iZahl2 INT UNSIGNED
341 )
342 BEGIN
343     SET @m = iZahl1; -- Deklaration durch
344     SET @n = iZahl2; -- eine Zuweisung
345
346     REPEAT -- Schleifenstart
347         SET @t = @m % @n;
348         SET @m = @n;
349         SET @n = @t;
350     UNTIL @n = 0 END REPEAT; -- Schleifenende
351
352     SELECT CONCAT('GGT(',iZahl1,', ', ',iZahl2,'
) = ') 'GgT()', @m;
353 END//
354 DELIMITER ;
355

```

```

356 CALL ggt(20, 15);
357 -- Ende Prozedur ggt
358
359 SELECT 'Transaktion innerhalb einer Proze
dur';
360 -- Start Prozedur insert_artikel Version
3
361 DROP PROCEDURE insert_artikel;
362
363 DELIMITER //
364 CREATE PROCEDURE insert_artikel
365 (
366 IN iArtikelname VARCHAR(255),
367 IN iWarengruppe VARCHAR(255)
368 )
369
370 BEGIN
371 DECLARE v_artikel_id INT DEFAULT -1;
372 DECLARE v_warengruppe_id INT DEFAULT -1;
373 DECLARE v_count_zuordnung INT DEFAULT 0;
374
375 DECLARE EXIT HANDLER FOR SQLEXCEPTION --
Handle fuer Fehler
376 BEGIN
377 SELECT 'Wegen Fehler ROLLBACK';
378 ROLLBACK;
379 END;
380
381 START TRANSACTION;
382
383 -- Teste, ob Warengruppe vorhanden ist
384 SELECT warengruppe_id
385 FROM warengruppe
386 WHERE bezeichnung = iWarengruppe
387 INTO v_warengruppe_id;
388
389 -- Wenn nein, fuege eine ein
390 IF v_warengruppe_id < 0 THEN -- Einfache

```

Verzweigung

```
391  INSERT INTO warengruppe (bezeichnung) VAL
UES (iWarengruppe);
392  SET v_warengruppe_id = LAST_INSERT_ID();
393  END IF;
394
395  -- Teste, ob artikel vorhanden ist
396  SELECT artikel_id
397      FROM artikel
398      WHERE bezeichnung = iArtikelname
399      INTO v_artikel_id;
400
401  -- Wenn nein, teste auf fuege ein
402      IF v_artikel_id < 0 THEN
403          INSERT INTO artikel (bezeichnung) VALUE
S (iArtikelname);
404          SET v_artikel_id = LAST_INSERT_ID();
405      END IF;
406
407  -- Test, ob Zuordnung schon vorhanden
408  SELECT COUNT(*)
409      FROM artikel_nm_warengruppe
410      WHERE artikel_id = v_artikel_id AND war
engruppe_id = v_warengruppe_id
411      INTO v_count_zuordnung;
412
413  -- Wenn nein, fuege eine hinzu
414  IF v_count_zuordnung <= 0 THEN
415  INSERT INTO artikel_nm_warengruppe
416  (warengruppe_id, artikel_id)
417  VALUES
418  (v_warengruppe_id, v_artikel_id);
419  ELSE
420  SELECT 'Zuordnung schon vorhanden'; -- Hi
nweis
421  END IF;
422
423  COMMIT;
```

```

424
425 END//
426 DELIMITER ;
427 -- Ende Prozedur insert_artikel Version 3
428
429 SELECT 'Cursor';
430 -- Start import mit CURSOR
431 SOURCE importMitCursor01.sql;
432 SELECT bestellung_id, kunde_id, adresse_i
d, datum, status FROM tmp_bestellung;
433 SELECT bestellung_id, position_nr, artike
l_id, menge, position_id FROM tmp_position ;
434 CALL importBestellung();
435 SELECT * FROM bestellung;
436 SELECT * from bestellung_position;
437 -- Ende import mit CURSOR
438
439
440 SELECT 'Ende listing16. sql //////////////////////////////////
////////////////////////////////////////';

```

Listing 28.19 mysql/listing17.sql

```

1  -- Ausführen der vorherigen Befehle
2  SOURCE listing16.sql
3
4  SELECT 'Start listing17. sql //////////////////////////////////
////////////////////////////////////////';
5
6  SELECT 'Funktionen (\abschnittref{lst17_0
01})';
7  DROP FUNCTION IF EXISTS preis_anpassen;
8  DELIMITER //
9  CREATE FUNCTION preis_anpassen
10 (
11  iPreisOriginal DECIMAL(14,6), -- Original
preis
12  iProzentsatz DECIMAL(7,3) -- Anpassung in

```

```

Prozent (-100-100)
13 )
14 RETURNS DECIMAL(14,6) DETERMINISTIC
15 BEGIN
16 DECLARE vPreisNeu DECIMAL(14,6); -- Neuer
Preis
17
18 IF iPreisOriginal IS NULL THEN -- Test au
f NULL
19 RETURN NULL;
20 END IF;
21 IF iProzentsatz IS NULL THEN -- Test auf
NULL
22 RETURN NULL;
23 END IF;
24 IF iProzentsatz NOT BETWEEN -100 AND 100
THEN -- Wertebereich Prozent
25 RETURN NULL;
26 END IF;
27 SET vPreisNeu = iPreisOriginal + ((iPreis
Original * iProzentsatz) / 100.0);
28 SET vPreisNeu = ROUND(vPreisNeu, 2);
29 RETURN vPreisNeu;
30 END //
31 DELIMITER ;
32
33 SELECT 'Ende listing17. sql //////////////////
////////////////////////////////////////';

```

Listing 28.20 mysql/listing18.sql

```

1 -- Ausfuehren der vorherigen Befehle
2 SOURCE listing17.sql
3
4 SELECT 'Start listing18. sql //////////////////
////////////////////////////////////////';
5
6 SELECT 'INSERT Trigger (Abschnitt 22.2)';

```

```

7  DROP TRIGGER IF EXISTS tri_bestellung_aft
er_insert;
8  DELIMITER //
9  CREATE TRIGGER tri_bestellung_after_inser
t -- ein schöner Name
10     AFTER INSERT -- wann?
11     ON bestellung -- bei wem?
12     FOR EACH ROW
13     BEGIN
14         INSERT INTO rechnung -- was?
15         SET
16             kunde_id = NEW.kunde_id,
17             bestellung_id = NEW.bestellung_id,
18             adresse_id = NEW.adresse_id,
19             datum = NEW.datum;
20     END//
21     DELIMITER ;
22     INSERT INTO bestellung (kunde_id, adres
se_id, datum) VALUES (1, 1, NOW());
23     SELECT bestellung_id, kunde_id, adresse
_id, datum FROM bestellung;
24     SELECT rechnung_id, kunde_id, bestellun
g_id, adresse_id, datum FROM rechnung\G
25
26     SELECT 'UPDATE Trigger (Abschnitt 22.3)';
27     DROP TRIGGER IF EXISTS tri_bestellung_aft
er_update;
28     DELIMITER //
29     CREATE TRIGGER tri_bestellung_after_updat
e
30     AFTER UPDATE
31     ON bestellung
32     FOR EACH ROW
33     BEGIN
34         DECLARE vBestellungId INT;
35
36         SET vBestellungId = OLD.bestellung_id
; -- Damit wir es wiederfinden

```



```

37
38     IF OLD.kunde_id != NEW.kunde_id THEN --
Nur, wenn nötig
39         UPDATE rechnung
40         SET kunde_id = NEW.kunde_id
41         WHERE bestellung_id = vBestellungId;
42     END IF;
43
44     IF OLD.adresse_id != NEW.adresse_id THE
N
45         UPDATE rechnung
46         SET adresse_id = NEW.adresse_id
47         WHERE bestellung_id = vBestellungId;
48     END IF;
49
50     IF OLD.datum != NEW.datum THEN
51         UPDATE rechnung
52         SET datum = NEW.datum
53         WHERE bestellung_id = vBestellungId;
54     END IF;
55
56 END//
57 DELIMITER ;
58 UPDATE bestellung SET adresse_id = 2 WHERE
E kunde_id = 1;
59 SELECT bestellung_id, kunde_id, adresse_i
d, datum
60 FROM bestellung
61 WHERE kunde_id=1;
62 SELECT rechnung_id, kunde_id, bestellung_
id, adresse_id
63 FROM rechnung
64 WHERE kunde_id = 1;
65
66 SELECT 'DELETE Trigger (Abschnitt 22.4)';
67 DROP TRIGGER IF EXISTS tri_bestellung_bef
ore_delete;
68 DELIMITER //

```

```

69 CREATE TRIGGER tri_bestellung_before_delete
70 BEFORE DELETE
71 ON bestellung
72 FOR EACH ROW
73 BEGIN
74     DECLARE vBestellungId INT;
75     DECLARE vRechnungId INT;
76
77     SET vBestellungId = OLD.bestellung_id
; -- Damit wir es wiederfinden
78     SELECT rechnung_id
79     FROM rechnung
80     WHERE bestellung_id = vBestellungId
81     INTO vRechnungId;
82
83     DELETE FROM rechnung_position WHERE rechnung_id = vRechnungId;
84     DELETE FROM rechnung WHERE rechnung_id = vRechnungId;
85     DELETE FROM bestellung_position WHERE bestellung_id = vBestellungId;
86 END//
87 DELIMITER ;
88 SELECT
89     (
90         SELECT COUNT(*) FROM bestellung_position WHERE bestellung_id = 5
91     ) AS AnzPosBestellung,
92     (
93         SELECT COUNT(*)
94         FROM
95             rechnung_position INNER JOIN rechnung
96             USING (rechnung_id)
97             WHERE bestellung_id = 5
98         ) AS AnzPosRechnung
99 ;
100 DELETE FROM bestellung WHERE bestellung_id

```

```

d = 5;
100 SELECT
101     (
102         SELECT COUNT(*) FROM bestellung WHERE b
estellung_id = 5
103     ) AS AnzBestellung,
104     (
105         SELECT COUNT(*) FROM bestellung_positio
n WHERE bestellung_id = 5
106     ) AS AnzPosBestellung,
107     (
108         SELECT COUNT(*) FROM rechnung WHERE bes
tellung_id = 5
109     ) AS AnzRechnung,
110     (
111         SELECT COUNT(*)
112         FROM
113             rechnung_position INNER JOIN rechnung
        USING (rechnung_id)
114         WHERE bestellung_id = 5
115     ) AS AnzPosRechnung
116 ;
117 SELECT 'Ende listing18. sql //////////////////////////////////
////////////////////////////////////////';

```

Listing 28.21 mysql/listing19.sql

```

1  -- Ausfuehren der vorherigen Befehle
2  SOURCE listing18.sql
3
4  SELECT 'Start listing19. sql //////////////////////////////////
////////////////////////////////////////';
5
6  SELECT 'EVENT anlegen (\abschnittref{lst1
9_001})';
7
8  DROP EVENT IF EXISTS event_umsatz_woche;
9  CREATE EVENT event_umsatz_woche

```

```

10  ON SCHEDULE EVERY 1 WEEK
11  DO
12      CALL umsatzreport(1);
13
14  SHOW VARIABLES LIKE 'event%';
15  SET GLOBAL event_scheduler = ON;
16  ALTER EVENT event_umsatz_woche DISABLE;
17  DROP EVENT IF EXISTS event_umsatz_woche;
-- Wegen Vorsicht!
18
19  DROP TABLE IF EXISTS a;
20  CREATE TABLE a
21  (
22      zeit DATETIME
23  );
24  DROP EVENT IF EXISTS event_zeit_sekunde;
25  CREATE EVENT event_zeit_sekunde
26  ON SCHEDULE EVERY 2 SECOND
27  DO
28      INSERT INTO a VALUES (NOW());
29
30  -- Warte 20 Sekunden
31  SELECT * FROM a;
32
33
34  SELECT 'EVENT löschen (\abschnittref{lst1
9_006})';
35  DROP EVENT IF EXISTS event_zeit_sekunde;
36  SET GLOBAL event_scheduler = OFF;
37  DROP TABLE IF EXISTS a;
38  SELECT 'Ende listing19. sql //////////////////////////////////
////////////////////////////////////////';

```

28.1.5 Quelltexte zu Teil VI

Listing 28.22 mysql/listing20.sql

```

1  -- Ausfuehren der vorherigen Befehle
2  SOURCE listing19.sql
3
4  SELECT 'Start listing20. sql //////////////////////////////////
////////////////////////////////////////';
5  SELECT 'Ggf. Kommentare vor DROP USER wie
der löschen';
6  -- DROP USER
7  -- 'oshop_root'@'%'
8  -- , 'oshop_root'@'localhost'
9  -- , 'oshop_user'@'%'
10 -- , 'oshop_user'@'localhost'
11 -- ;
12
13 CREATE USER
14 'oshop_root'@'%'
15 IDENTIFIED BY 'root'
16 , 'oshop_user'@'%'
17 IDENTIFIED BY 'user'
18 ;
19 CREATE USER
20 'oshop_root'@'localhost'
21 IDENTIFIED BY 'root'
22 , 'oshop_user'@'localhost'
23 IDENTIFIED BY 'user'
24 ;
25
26 GRANT ALL PRIVILEGES
27 ON oshop.*
28 TO 'oshop_root'@'%', 'oshop_root'@'localh
ost',
29 WITH GRANT OPTION
30 ;
31
32 GRANT INSERT, UPDATE, DELETE, SELECT
33 ON TABLE oshop.*
34 TO 'oshop_user'@'%', 'oshop_user'@'localh
ost',

```

```

35  ;
36
37  REVOKE DELETE
38  ON TABLE oshop.*
39  FROM 'oshop_user'@'%', 'oshop_user'@'loca
lhost'
40  ;
41  SELECT 'Ende listing20. sql //////////////////////////////////
////////////////////////////////////////';

```

Listing 28.23 mysql/listing21.sql

```

1  DROP DATABASE IF EXISTS injection;
2  CREATE DATABASE IF NOT EXISTS injection
3  DEFAULT CHARACTER SET utf8mb4
4  COLLATE utf8mb4_german2_ci
5  ;
6  USE injection;
7
8  CREATE TABLE artikel (
9      artikel_id INT UNSIGNED AUTO_INCREMENT,
10     bezeichnung VARCHAR(255),
11     einzelpreis DECIMAL(14,6),
12     waehrung CHAR(3) DEFAULT 'EUR',
13     deleted TINYINT UNSIGNED NOT NULL DEFAU
LT '0',
14     PRIMARY KEY (artikel_id)
15 );
16
17 INSERT INTO artikel (artikel_id, bezeichn
ung, einzelpreis, waehrung, deleted)
18 VALUES
19     (3001, 'Papier (100)', 2.320000, 'EUR',
0),
20     (3005, 'Tinte (gold)', 56.260000, 'EUR'
, 0),
21     (3006, 'Tinte (rot)', 6.260000, 'EUR',
0),

```

```

22      (3007, 'Tinte (blau)', 4.170000, 'EUR',
23      0),
24      (3010, 'Feder', 5.050000, 'EUR', 0),
25      (7856, 'Silberzwiebel', 0.520000, 'EUR'
26      , 0),
27      (7863, 'Tulpenzwiebel', 3.420000, 'EUR'
28      , 0),
29      (9010, 'Schaufel', 15.100000, 'USD', 0)
30  ,
31      (9015, 'Spaten', 20.100000, 'EUR', 0),
32      (9016, 'Schlauch', 0.000000, 'EUR', 0)
33  ;
34
35  CREATE TABLE account (
36      kunde_id INT UNSIGNED,
37      name VARCHAR(255) ,
38      passwort CHAR(34),
39      status ENUM('aktiv','gesperrt') NOT NUL
40  L DEFAULT 'aktiv',
41      kommentar TEXT,
42      admin BOOL NOT NULL DEFAULT '0',
43      deleted TINYINT unsigned NOT NULL DEFAU
44  LT '0',
45      PRIMARY KEY (kunde_id)
46  );
47
48  INSERT INTO account (kunde_id, name, pass
49  wort, status, kommentar, admin, deleted)
50  VALUES
51      (1, 'admin', '45c6f0923e6a87454183b56c0
52  935d253', 'aktiv', NULL, TRUE, 0),
53      (2, 'frodo', '92cba3dc4cd1db0628fadeebf
54  aa16ac8', 'aktiv', NULL, FALSE, 0),
55      (3, 'bilbo', 'dd509890e7bcece80b1a9c20e
56  c4affb5', 'aktiv', NULL, FALSE, 0),
57      (5, 'elle', 'ee74f41a33032548bf6d315d7d
58  b3f1ee', 'aktiv', NULL, FALSE, 0)
59  ;

```

Listing 28.24 mysql/index01.php

```
1  <?php
2  if (ISSET($_GET['search']) && $_GET['search'] == '1') { // Schaltfläche gedrückt?
3      $server = "localhost";
4      $user = "root";
5      $password = "weinschlauchX20!";
6      $dbname = "injection";
7
8      $conn = new mysqli($server, $user, $password, $dbname); // Verbindung aufbauen
9      if ($conn->connect_error) { // Fehler aufgetreten?
10         die("Connection failed: ".$conn->connect_error); // Fehlerausgabe
11     }
12
13     if (ISSET($_GET['artikel_id'])) { // Variable vorhanden?
14         $artikel_id = $_GET['artikel_id'];
15         $sql = "SELECT * FROM artikel WHERE artikel_id=$artikel_id"; // SQL-Bau
16         echo "<p>$sql</p>"; // NUR FÜR UNS!
17     }
18     if (!$result = $conn->query($sql)) { // Query -> Server
19         die("Execution failed: ".$conn->error);
20     }
21
22     if ($result->num_rows > 0) {
23         echo "<table border=\"1\">";
24         echo "<tr>";
25         echo "<th>Artikelnr</th>";
26         echo "<th>Bezeichnung</th>";
27         echo "<th>Einzelpreis</th>";
28         echo "<th>Währung</th><th>LKZ</th>";
```



```

29         echo "</tr>";
30         while($row = $result->fetch_assoc())
{
31             echo "<tr>";
32             echo "<td>".$row['artikel_id']."</t
d>";
33             echo "<td>".$row['bezeichnung']."</
td>";
34             echo "<td>".$row['einzelpreis']."</
td>";
35             echo "<td>".$row['waehrung']."</td>
";
36             echo "<td>".$row['deleted']."</td>"
;
37             echo "</tr>";
38         }
39         echo "</table>";
40     }
41     echo "<p>".$result->num_rows." results<
/p>";
42     $conn->close(); // Verbindung schließen
43 }
44 else
45     ?>
46 <!DOCTYPE HTML>
47 <html>
48 <head>
49 <title>Beispiel 1: Artikelsuche</title>
50 <meta name="author" content="Ralf Adams">
51 </head>
52 <body text="#000000" bgcolor="#FFFFFF">
53 <form action="http://localhost//buch//ind
ex01.php">
54     <label for="artikel_id">Artikelnummer:
</label>
55     <input type="text" id="artikel_id" name
="artikel_id"> <!-- Eingabefeld -->
56     <button name="search" value="1">Suchen<

```

```

/button> <!-- Submitbutton -->
57 </form>
58 </body>
59 </html>

```

Listing 28.25 mysql/index02.php

```

1  <?php
2      if (ISSET($_GET['search']) && $_GET['se
arch'] == '1') {
3          $server = "localhost";
4          $user = "root";
5          $password = "weinschlauchX20!";
6          $dbname = "injection";
7
8          $conn = new mysqli($server, $user, $pas
sword, $dbname);
9          if ($conn->connect_error) {
10             die("Connection failed: ".$conn->conn
ect_error);
11         }
12
13         if (ISSET($_GET['artikel_id'])) {
14             $artikel_id = $_GET['artikel_id'];
15             $sql = "SELECT * FROM artikel WHERE a
rtikel_id=$artikel_id";
16             echo "<p>$sql</p>";
17         }
18         if (!$conn->multi_query($sql)) { // Mul
ti-Query -> Server
19             die("Execution failed: ".$conn->error
);
20         }
21         $result = $conn->store_result();
22
23
24         if ($result->num_rows > 0) {
25             echo "<table border=\"1\">";

```

```

26     echo "<tr>";
27     echo "<th>Artikelnr</th>";
28     echo "<th>Bezeichnung</th>";
29     echo "<th>Einzelpreis</th>";
30     echo "<th>Währung</th><th>LKZ</th>";
31     echo "</tr>";
32     while($row = $result->fetch_assoc())
{
    33         echo "<tr>";
    34         echo "<td>".$row['artikel_id']."</t
d>";
    35         echo "<td>".$row['bezeichnung']."</
td>";
    36         echo "<td>".$row['einzelpreis']."</
td>";
    37         echo "<td>".$row['waehrung']."</td>
";
    38         echo "<td>".$row['deleted']."</td>"
;
    39         echo "</tr>";
    40     }
    41     echo "</table>";
    42 }
    43     echo "<p>".$result->num_rows." results<
/p>";
    44     $conn->close(); // Verbindung schließen
    45 }
    46 else
    47 ?>
    48 <!DOCTYPE HTML>
    49 <html>
    50 <head>
    51 <title>Beispiel 2: Artikelsuche</title>
    52 <meta name="author" content="Ralf Adams">
    53 </head>
    54 <body text="#000000" bgcolor="#FFFFFF">
    55 <form action="http://localhost//buch//ind
ex02.php">

```

```

56     <label for="artikel_id">Artikelnummer:
</label>
57     <input type="text" id="artikel_id" name
="artikel_id"> <!-- Eingabefeld -->
58     <button name="search" value="1">Suchen<
/button> <!-- Submitbutton -->
59 </form>
60 </body>
61 </html>

```

28.2 PostgreSQL

28.2.1 Quelltexte zu Teil II

Listing 28.26 postgresql/listing01.sql

```

1  -- Warum formatiere ich den Quelltext so
ganz anders, als beispielsweise diese Beautyfi
er?
2  -- * http://www.dpriver.com/pp/sqlformat.
htm
3  -- * https://codebeautify.org/sqlformatte
r
4  -- Ich finde die Arbeit dieser Programme
sehr gut und auch die Ergebnisse sind ansprech
end,
5  -- aber sie widersprechen meinem Anspruch
an die leichte Verständlichkeit der Befehle.
6  -- Zwei Beispiele:
7  -- * Es werden keine Einrückungen eing
efügt, die den inhaltlichen Zusammenhang
8  -- zwischen dem Befehl und seinen Paramet
ern abbilden:
9  -- SELECT blz,

```

```

10  -- bankname
11  -- FROM bank
12  -- ORDER BY blz DESC
13  -- LIMIT 1;
14  --
15  -- anstelle von
16  --
17  -- SELECT
18  -- blz, bankname
19  -- FROM bank
20  -- ORDER BY blz DESC
21  -- LIMIT 1
22  -- ;
23
24  -- * Hinter dem FROM wird ein Tabellennam
e gestellt, obwohl die Auswertung aus dem ganz
en JOIN erfolgt:
25  -- SELECT k.kunde_id,
26  -- k.nachname,
27  -- k.vorname,
28  -- b.datum
29  -- FROM bestellung b
30  -- INNER JOIN kunde k USING (kunde_id)
31  -- ORDER BY k.nachname,
32  -- k.vorname;
33
34  -- anstelle von
35
36  -- SELECT
37  -- k.kunde_id, k.nachname, k.vorname, b.d
atum
38  -- FROM
39  -- bestellung b INNER JOIN kunde k USING
(kunde_id)
40  -- ORDER BY
41  -- k.nachname, k.vorname
42  -- ;
43

```

```

44  -- Ich werde daher meine Darstellung beib
    erhalten.
45
46
47  -- Anlegen der Datenbank auf der grünen W
    iese
48  \echo 'Start listing01. sql //////////////////////////////////
    //////////////////////////////////'
49  SELECT 'Loeschen einer ggf. vorhandenen D
    atenbank oshop (Abschnitt 5.2.1)' Hinweis;
50  \connect postgres
51
52  DROP DATABASE IF EXISTS oshop;
53
54  -- In PostgreSQL gibt es keine Möglichkei
    t, die Zeichensätze und Sortierungen dynamisch
    zu ermitteln
55
56  SELECT 'Anlegen der Datenbank mit Zeichen
    satz und Sortierung (Abschnitt 5.2.1)' Hinweis
    ;
57  CREATE DATABASE oshop
58  ENCODING 'UTF8'
59  LC_COLLATE='en_US.utf8'
60  LC_CTYPE='en_US.utf8'
61  ;
62
63  SELECT 'Anzeigen aller Datenbanken (Absch
    nitt 5.2.5)';
64  \l
65
66  \echo 'Ende listing01. sql //////////////////////////////////
    //////////////////////////////////'

```

Listing 28.27 postgresql/listing02.sql

```

1  -- Ausführen der vorherigen Befehle
2  SET client_encoding = 'UTF8';

```

```

3  SET standard_conforming_strings = on;
4  \i listing01.sql
5
6  \echo 'Start listing02. sql //////////////////////////////////
////////////////////////////////////////'
7
8  -- Erspare mir ein wenig Tipparbeit (Abschnitt 5.3.2)
9  \connect oshop
10
11  -- adresse
12  SELECT 'Tabelle adresse anlegen (Abschnitt 5.3.5)' Hinweis;
13  CREATE TABLE adresse (
14  adresse_id SERIAL,
15  strasse VARCHAR(255) NOT NULL DEFAULT '',
16  hnr VARCHAR(255) NOT NULL DEFAULT '',
17  lkz CHAR(2) NOT NULL DEFAULT '',
18  plz CHAR(9) NOT NULL DEFAULT '',
19  ort VARCHAR(255) NOT NULL DEFAULT '',
20  deleted SMALLINT NOT NULL DEFAULT 0,
21  PRIMARY KEY(adresse_id)
22  );
23
24  -- kunde
25  SELECT 'Tabelle kunde mit Fremdschlüssel
anlegen (Abschnitt 5.3.6)' Hinweis;
26  CREATE TYPE kundenart AS ENUM('unb', 'prv
', 'gsch');
27  CREATE TABLE kunde (
28  kunde_id SERIAL,
29  nachname VARCHAR(255) NOT NULL DEFAULT ''
,
30  vorname VARCHAR(255) NOT NULL DEFAULT '',
31  rechnung_adresse_id INT,
32  liefer_adresse_id INT,
33  bezahlart INT NOT NULL DEFAULT 0,
34  art kundenart NOT NULL DEFAULT 'unb',

```

```

35  deleted SMALLINT NOT NULL DEFAULT 0,
36  PRIMARY KEY(kunde_id),
37      FOREIGN KEY(rechnung_adresse_id)
38      REFERENCES adresse(adresse_id)
39      ON UPDATE CASCADE
40      ON DELETE RESTRICT,
41  FOREIGN KEY(liefer_adresse_id)
42      REFERENCES adresse(adresse_id)
43      ON UPDATE CASCADE
44      ON DELETE SET NULL
45  );
46
47  -- bank
48  SELECT 'Tabelle bank anlegen (Abschnitt 5
.3.2)' Hinweis;
49  CREATE TABLE bank (
50  bank_id CHAR(12),
51  bankname VARCHAR(255) NOT NULL DEFAULT ''
52  ,
53  lkz CHAR(2) NOT NULL DEFAULT 'DE',
54  deleted SMALLINT NOT NULL DEFAULT 0,
55  PRIMARY KEY(bank_id)
56  );
57  -- bankverbindung
58  SELECT 'Tabelle bankverbindung mit Fremds
chlüssel anlegen (Abschnitt 5.3.2)' Hinweis;
59  CREATE TABLE bankverbindung (
60  kunde_id INT,
61  bankverbindung_nr INT,
62  bank_id CHAR(12) NOT NULL,
63  kontonummer CHAR(25) NOT NULL DEFAULT '',
64  iban CHAR(34) NOT NULL DEFAULT '',
65  deleted SMALLINT NOT NULL DEFAULT 0,
66  PRIMARY KEY(kunde_id,bankverbindung_nr),
67      FOREIGN KEY(kunde_id)
68      REFERENCES kunde(kunde_id)
69      ON UPDATE RESTRICT

```



```

70     ON DELETE RESTRICT,
71     FOREIGN KEY(bank_id)
72     REFERENCES bank(bank_id)
73     ON UPDATE RESTRICT
74     ON DELETE RESTRICT
75 );
76
77
78 -- artikel
79 SELECT 'Tabelle artikel anlegen' Hinweis;
80 CREATE TABLE artikel (
81     artikel_id SERIAL,
82     bezeichnung VARCHAR(255) NOT NULL DEFAU
LT '',
83     einzelpreis DECIMAL(14,6) NOT NULL DEFA
ULT 0.0,
84     waehrung CHAR(3) DEFAULT 'EUR',
85     deleted SMALLINT NOT NULL DEFAULT 0,
86     PRIMARY KEY(artikel_id)
87 );
88
89
90 -- warengruppe
91 SELECT 'Tabelle warengruppe anlegen' Hinw
eis;
92 CREATE TABLE warengruppe (
93     warengruppe_id SERIAL,
94     bezeichnung VARCHAR(255) NOT NULL DEFAULT
'',
95     deleted SMALLINT NOT NULL DEFAULT 0,
96     PRIMARY KEY(warengruppe_id)
97 );
98
99
100 -- lieferant
101 SELECT 'Tabelle lieferant mit Fremdschlüs
sel anlegen' Hinweis;
102 CREATE TABLE lieferant (

```

```

103  lieferant_id SERIAL,
104  firmenname VARCHAR(255) NOT NULL DEFAULT
'',
105  adresse_id INT NOT NULL DEFAULT 0,
106  deleted SMALLINT NOT NULL DEFAULT 0,
107  PRIMARY KEY(lieferant_id),
108  FOREIGN KEY(adresse_id)
109      REFERENCES adresse(adresse_id)
110      ON UPDATE RESTRICT
111      ON DELETE RESTRICT
112  );
113
114
115  -- Hilfstabelle artikel zu warengruppe
116  SELECT 'Hilfstabelle artikel_nm_warengrup
pe anlegen' Hinweis;
117  CREATE TABLE artikel_nm_warengruppe (
118  warengruppe_id INT,
119  artikel_id INT,
120  PRIMARY KEY(warengruppe_id, artikel_id),
121  FOREIGN KEY(warengruppe_id)
122      REFERENCES warengruppe(warengruppe_id)
123      ON UPDATE RESTRICT
124      ON DELETE RESTRICT,
125  FOREIGN KEY(artikel_id)
126      REFERENCES artikel(artikel_id)
127      ON UPDATE RESTRICT
128      ON DELETE RESTRICT
129  );
130
131
132  -- Hilfstabelle artikel zu lieferant
133  SELECT 'Hilfstabelle artikel_nm_lieferant
anlegen' Hinweis;
134  CREATE TABLE artikel_nm_lieferant (
135      lieferant_id INT,
136      artikel_id INT,
137      PRIMARY KEY(lieferant_id, artikel_id),

```

```

138     FOREIGN KEY(lieferant_id)
139         REFERENCES lieferant(lieferant_id)
140         ON UPDATE RESTRICT
141         ON DELETE RESTRICT,
142     FOREIGN KEY(artikel_id)
143         REFERENCES artikel(artikel_id)
144         ON UPDATE RESTRICT
145         ON DELETE RESTRICT
146 );
147
148
149 -- bestellung
150 SELECT 'Tabelle bestellung mit Fremdschlüssel anlegen ' Hinweis;
151 CREATE TYPE bestellstatus AS ENUM('offen', 'versendet', 'angekommen', 'retour', 'bezahlt');
152 CREATE TABLE bestellung (
153     bestellung_id SERIAL,
154     kunde_id INT NOT NULL DEFAULT 0,
155     adresse_id INT NOT NULL DEFAULT 0,
156     datum TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
157     status bestellstatus NOT NULL DEFAULT 'offen',
158     deleted SMALLINT NOT NULL DEFAULT 0,
159     PRIMARY KEY(bestellung_id),
160     FOREIGN KEY(kunde_id)
161         REFERENCES kunde(kunde_id)
162         ON UPDATE RESTRICT
163         ON DELETE RESTRICT,
164     FOREIGN KEY(adresse_id)
165         REFERENCES adresse(adresse_id)
166         ON UPDATE RESTRICT
167         ON DELETE RESTRICT
168 );
169
170 -- position der bestellung

```

```

171 SELECT 'Tabelle bestellung_position mit F
remdschlüssel anlegen' Hinweis;
172 CREATE TABLE bestellung_position (
173     bestellung_id INT,
174     position_nr INT,
175     artikel_id INT NOT NULL DEFAULT 0,
176     menge DECIMAL(14,6) NOT NULL DEFAULT 0.
0,
177     deleted SMALLINT NOT NULL DEFAULT 0,
178     PRIMARY KEY(bestellung_id, position_nr)
,
179     FOREIGN KEY(bestellung_id)
180     REFERENCES bestellung(bestellung_id)
181     ON UPDATE RESTRICT
182     ON DELETE RESTRICT,
183     FOREIGN KEY(artikel_id)
184     REFERENCES artikel(artikel_id)
185     ON UPDATE RESTRICT
186     ON DELETE RESTRICT
187 );
188
189
190 -- rechnung
191 SELECT 'Tabelle rechnung mit Fremdschlüss
el anlegen' Hinweis;
192 CREATE TYPE bezahlarten AS ENUM('lastschr
ift', 'rechnung', 'kredit', 'bar');
193 CREATE TYPE rechnungstatus AS ENUM('offen
', 'gestellt', 'mahnung1', 'inkasso', ' storno
', 'beglichen');
194 CREATE TABLE rechnung (
195     rechnung_id SERIAL,
196     kunde_id INT NOT NULL DEFAULT 0,
197     bestellung_id INT NOT NULL DEFAULT 0,
198     adresse_id INT NOT NULL DEFAULT 0,
199     bezahlart bezahlarten NOT NULL DEFAULT
'lastschrift',
200     datum TIMESTAMP NOT NULL DEFAULT CURREN

```

```

T_TIMESTAMP,
201     status rechnungstatus NOT NULL DEFAULT
'offen',
202     rabatt DECIMAL(6,2) NOT NULL DEFAULT 0.
0,
203     skonto DECIMAL(6,2) NOT NULL DEFAULT 0.
0,
204     deleted SMALLINT NOT NULL DEFAULT 0,
205     PRIMARY KEY(rechnung_id),
206     FOREIGN KEY(kunde_id)
207         REFERENCES kunde(kunde_id)
208         ON UPDATE RESTRICT
209         ON DELETE RESTRICT,
210     FOREIGN KEY(adresse_id)
211         REFERENCES adresse(adresse_id)
212         ON UPDATE RESTRICT
213         ON DELETE RESTRICT
214 );
215
216
217 -- position der rechnung
218 SELECT 'Tabelle rechnung_position mit Fre
mdschlüssel anlegen' Hinweis;
219 CREATE TABLE rechnung_position (
220     rechnung_id INT,
221     position_nr INT,
222     artikel_id INT NOT NULL DEFAULT 0,
223     steuer DECIMAL(14,6) NOT NULL DEFAULT 1
9.0,
224     menge DECIMAL(14,6) NOT NULL DEFAULT 0.
0,
225     deleted SMALLINT NOT NULL DEFAULT 0,
226     PRIMARY KEY(rechnung_id, position_nr),
227     FOREIGN KEY(rechnung_id)
228         REFERENCES rechnung(rechnung_id)
229         ON UPDATE RESTRICT
230         ON DELETE RESTRICT,
231     FOREIGN KEY(artikel_id)

```

```

232     REFERENCES artikel(artikel_id)
233     ON UPDATE RESTRICT
234     ON DELETE RESTRICT
235 );
236
237 SELECT 'Neue Tabelle mit Struktur einer a
lten anlegen (Abschnitt 5.3.7)' Hinweis;
238 CREATE TABLE wurstbrot (LIKE adresse);
239
240 SELECT 'Neue Tabelle sofort wieder lösche
n, da nicht mehr gebraucht' Hinweis;
241 DROP TABLE wurstbrot;
242
243 -- Zeige mir das Ergebnis
244 \dt
245 \echo 'Ende listing02. sql //////////////////////////////////
/////////////////////////////////'

```

Listing 28.28 postgresql/listing03.sql

```

1  -- Ausführen der vorherigen Befehle
2  \i listing02.sql
3
4  \echo 'Start listing03. sql //////////////////////////////////
/////////////////////////////////'
5
6  -- Abschnitt 6.1.1
7  \d kunde
8
9  -- Anlegen der Indizes nach Liste auf Abschnitt 6.1.2
10 SELECT 'CREATE INDEX idx_kunde_nachname_v
orname' Hinweis;
11 CREATE INDEX idx_kunde_nachname_vorname
12 ON kunde (nachname, vorname)
13 ;
14
15 SELECT 'CREATE INDEX idx_bank_bankname' H

```

```
inweis;
16  CREATE INDEX idx_bank_bankname
17  ON bank (bankname)
18  ;
19
20  SELECT 'CREATE INDEX idx_bankverbindung_b
ankid_kontonummer' Hinweis;
21  CREATE INDEX idx_bankverbindung_bankid_ko
ntonummer
22      ON bankverbindung (bank_id, kontonummer
)
23  ;
24
25  SELECT 'CREATE UNIQUE INDEX idx_bankverbi
ndung_iban' Hinweis;
26  CREATE UNIQUE INDEX idx_bankverbindung_ib
an
27  ON bankverbindung (iban)
28  ;
29
30  SELECT 'CREATE INDEX idx_artikel_bezeichn
ung' Hinweis;
31  CREATE INDEX idx_artikel_bezeichnung
32      ON artikel (bezeichnung)
33  ;
34
35  SELECT 'CREATE INDEX idx_warengruppe_beze
ichnung' Hinweis;
36  CREATE INDEX idx_warengruppe_bezeichnung
37      ON warengruppe (bezeichnung)
38  ;
39
40  SELECT 'CREATE INDEX idx_lieferant_firmen
name' Hinweis;
41  CREATE INDEX idx_lieferant_firmenname
42      ON lieferant (firmenname)
43  ;
44
```

```

45  SELECT 'CREATE INDEX idx_bestellung_kunde
id_datum' Hinweis;
46  CREATE INDEX idx_bestellung_kundeid_datum
47      ON bestellung (kunde_id, datum)
48  ;
49
50
51  SELECT 'CREATE UNIQUE INDEX idx_adresse_d
ublette (Abschnitt 6.2.2)' Hinweis;
52  CREATE UNIQUE INDEX idx_adresse_dublette
53  ON adresse (substring(strasse from 1 for
100), substring(hnr from 1 for 100), plz );
54
55  -- Start Selektivität
56  SELECT 'Indexselektivität (Abschnitt 6.2.
3)' Hinweis;
57  SELECT 'CREATE TEMPORARY TABLE testdaten_
bank' Hinweis;
58  DROP TABLE IF EXISTS testdaten_bank;
59  CREATE TABLE testdaten_bank
60  (
61  blz CHAR(8),
62  merkmal CHAR(1),
63  bezeichnung VARCHAR(255),
64  plz CHAR(5),
65  ort VARCHAR(255),
66  kurz VARCHAR(255),
67  pan CHAR(5),
68  bic VARCHAR(255),
69  sm CHAR(2),
70  ds CHAR(6),
71  kz CHAR(1),
72  blzl CHAR(1),
73  blzn CHAR(8)
74  );
75
76  SELECT 'COPY' Hinweis;
77  -- Hier Dateiname und Pfad einfuegen

```



```

78
79 \copy testdaten_bank FROM './blz_20120305
.csv' WITH DELIMITER ';' CSV HEADER
80 SELECT 'CREATE INDEX' Hinweis;
81 CREATE INDEX idx_testdaten_bank_bezeichnu
ng
82     ON testdaten_bank (substring(bezeichnun
g from 1 for 30))
83 ;
84 -- Ende Selektivitaet
85
86 \echo 'Ende listing03. sql //////////////////////////////////
////////////////////////////////////////';

```

Listing 28.29 postgresql/listing04.sql

```

1  -- Ausführen der vorherigen Befehle
2  \i listing03.sql
3
4  \echo 'Start listing04. sql //////////////////////////////////
////////////////////////////////////////'
5
6  SELECT 'Importieren von artikel01.csv (Abschnitt 7.1.3)' Hinweis;
7  \copy artikel(artikel_id, bezeichnung, ei
nzelpreis, waehrung) FROM './artikel02.csv ' W
ITH DELIMITER ';' CSV HEADER
8
9  SELECT * FROM artikel;
10
11 -- Start INSERT
12 SELECT 'Anlegen mit INSERT INTO ... VALUE
S (Abschnitt 7.2.1)' Hinweis;
13 SELECT 'Warengruppen anlegen' Hinweis;
14 INSERT INTO warengruppe (warengruppe_id,
bezeichnung)
15 VALUES
16     (1, 'Bürobedarf')

```

```

17      ,(2, 'Pflanzen')
18      ,(3, 'Gartenbedarf')
19      ,(4, 'Werkzeug')
20  ;
21  SELECT * FROM warengruppe;
22
23  SELECT 'Anlegen mit INSERT INTO ... SET (
Abschnitt 7.2.2) geht in PostgreSQL nicht' Hin
weis;
24  SELECT 'Warengruppen verbinden' Hinweis;
25  INSERT INTO artikel_nm_warengruppe (artik
el_id, warengruppe_id)
26  VALUES
27      (3001,1),(3005,1),(3006,1),(3007,1),(30
10,1), (7856,2),(7856,3),(7863,2),(7863,3) ,(9
010,3),(9010,4),(9015,3),(9015,4)
28  ;
29  -- Ende INSERT
30
31  -- Start Kundendaten
32  SELECT 'Testdaten INSERT adresse' Hinweis
;
33
34  -- adresse
35  INSERT INTO adresse (adresse_id, strasse,
hnr, lkz, plz, ort)
36  VALUES
37      (1, 'Beutelhaldenweg', '5', 'AL', '6767
6', 'Hobbingen')
38      ,(2, 'Beutelhaldenweg', '1', 'AL', '676
76', 'Hobbingen')
39      ,(3, 'Auf der Feste', '1', 'GO', '54786
', 'Minas Tirith')
40      ,(4, 'Letztes Haus', '4', 'ER', '87567'
, 'Bruchtal')
41      ,(5, 'Baradur', '1', 'MO', '62519', 'Lu
gburz')
42      ,(10, 'Hochstrasse', '4a', 'DE', '44879'

```

```

, 'Bochum')
43      ,(11, 'Industriegebiet','8', 'DE', '448
78', 'Bochum')
44      ;
45
46      SELECT 'Testdaten INSERT kunde' Hinweis;
47      -- kunde
48      INSERT INTO kunde (kunde_id, rechnung_adr
esse_id, nachname, vorname, art)
49      VALUES
50      (1, 1, 'Gamdschie', 'Samweis', 'prv')
51      ,(2, 2, 'Beutlin', 'Frodo', 'prv')
52      ,(3, 2, 'Beutlin', 'Bilbo', 'prv')
53      ,(4, 3, 'Telcontar', 'Elessar', 'prv')
54      ,(5, 4, 'Earendilionn','Elrond', 'gsch'
)
55      ;
56
57      SELECT 'Lieferadresse bei zwei Adressen a
endern' Hinweis;
58      UPDATE kunde SET liefer_adresse_id = 2 WH
ERE kunde_id = 1;
59      UPDATE kunde SET liefer_adresse_id = 4 WH
ERE kunde_id = 3;
60      -- Ende Kundendaten
61
62      -- Start Aufbau von 2 Bestellungen
63      SELECT 'Aufbau von 2 Bestellungen: Bestel
ldaten' Hinweis;
64      INSERT INTO bestellung (kunde_id, adresse
_id, datum)
65      VALUES
66      (1, 1, '2012-03-24 17:41:00')
67      ,(2, 2, '2012-03-23 16:11:00')
68      ;
69
70      SELECT 'Aufbau von 2 Bestellungen: Bestel
lpositionen' Hinweis;

```

```

71 INSERT INTO bestellung_position (bestellu
ng_id, position_nr, artikel_id, menge)
72 VALUES
73     (1, 1, 7856, 30)
74     ,(1, 2, 7863, 50)
75     ,(1, 3, 9015, 1)
76     ,(2, 1, 7856, 10)
77     ,(2, 2, 9010, 5)
78 ;
79 -- Ende Aufbau Bestellung
80
81 -- SELECT 'Drei Fehlermeldung wegen Const
raints (Abschnitt 7.2.3)!';
82 -- SELECT bestellung_id, kunde_id, adress
e_id, datum FROM bestellung;
83 -- INSERT INTO bestellung (datum) VALUES
(NOW());
84 -- INSERT INTO bestellung (adresse_id, da
tum) VALUES (10, NOW());
85 -- INSERT INTO adresse (adresse_id, stras
se) VALUES (10, 'Oberer Weg');
86
87 -- Bildttabelle
88 SELECT 'Bildttabelle anlegen (Abschnitt 7.
2.4)' Hinweis;
89 CREATE TABLE bild
90 (
91     bild_id SERIAL PRIMARY KEY,
92     bild BYTEA,
93     dateiname VARCHAR(255),
94     dateityp VARCHAR(255),
95     dateigroesse BIGINT,
96     artikel_id INT REFERENCES artikel(art
ikel_id)
97 );
98
99 SELECT 'Daten kopieren (Abschnitt 7.3)' H
inweis;

```

[illegible]

28.2.2 Quelltexte zu Teil III

Listing 28.30 postgresql/listing05.sql

```
1  -- Ausführen der vorherigen Befehle
2  \i listing04.sql
3
4  \echo 'Start listing05. sql //////////////////////////////////
////////////////////////////////////////'
5
6  SELECT 'Aktuelle Schemaeinstellung (Abschnitt 8.1)' Hinweis;
7  \l+
8
9  SELECT 'DATABASE loeschen (Abschnitt 8.2)
' Hinweis;
10 CREATE DATABASE wurstbrot;
11 \list+
12 DROP DATABASE wurstbrot;
13 \list+
14
15
16 -- Start Tabellen umbenennen
17 SELECT 'Tabellen umbenennen (Abschnitt 8.3.1)' Hinweis;
18 DROP TABLE IF EXISTS wurstbrot;
19 \dt
20 ALTER TABLE adresse RENAME TO wurstbrot;
21 \dt
22 ALTER TABLE wurstbrot RENAME TO adresse;
23 -- Ende Tabellen umbenennen
24
25
26
27 -- Start Spalten verändern
28 SELECT 'Spalte hinzufügen (Abschnitt 8.3.
```

```

2)' Hinweis;
29 \d+ kunde
30 ALTER TABLE kunde
31 ADD firmenname VARCHAR(255) NOT NULL DEFA
ULT ',
32 ;
33 \d+ kunde
34
35 SELECT 'Spalte Datentyp aendern (Abschnitt 8.3.3)' Hinweis;
36 -- Konvertierung geht in PostgreSQL nur u
nter Zuhilfenahme einer Funktion.
37 -- Aus diesem Grunde verzichte ich hier a
uf die Darstellung der entsprechenden
38 -- Konvertierungseffekte, da in den selbs
terstellten Funktionen alle Fehlersituationen
39 -- abgefangen werden können.
40 -- Hier ein nicht erklärtes, aber hoffent
lich selbsterklärendes Beispiel.
41
42 CREATE OR REPLACE FUNCTION IntToBezahlart
en(wert INT)
43 RETURNS bezahlarten AS
44 $BODY$
45     SELECT CASE wert
46         WHEN 1 THEN 'lastschrift'::bezahlarte
n
47         WHEN 2 THEN 'rechnung'::bezahlarten
48         WHEN 3 THEN 'kredit'::bezahlarten
49         WHEN 4 THEN 'bar'::bezahlarten
50         ELSE 'rechnung'::bezahlarten
51     END;
52 $BODY$
53 IMMUTABLE STRICT LANGUAGE SQL
54 ;
55 ALTER TABLE kunde
56 ALTER COLUMN bezahlart
57 DROP DEFAULT

```

```

58 ;
59 ALTER TABLE kunde
60 ALTER COLUMN bezahlart
61 SET DATA TYPE bezahlarten USING IntToBeza
harten(bezahlart)
62 ;
63 ALTER TABLE kunde
64 ALTER COLUMN bezahlart
65 SET NOT NULL
66 ;
67 ALTER TABLE kunde
68 ALTER COLUMN bezahlart
69 SET DEFAULT 'rechnung'::bezahlarten
70 ;
71 DROP FUNCTION IF EXISTS IntToBezahlarten(
int);
72 \d+ kunde
73
74 SELECT 'Spaltenlänge ändern (Abschnitt 8.
3.4)' Hinweis;
75 DROP TABLE IF EXISTS bla ;
76 CREATE TEMPORARY TABLE bla (LIKE artikel)
;
77 INSERT INTO bla SELECT * FROM artikel;
78 -- Hinweis: siehe oben.
79 CREATE OR REPLACE FUNCTION Cut10(wert VAR
CHAR(255))
80 RETURNS CHAR(10) AS
81 $BODY$
82 SELECT LEFT(wert, 10);
83 $BODY$
84 IMMUTABLE STRICT LANGUAGE SQL
85 ;
86 ALTER TABLE bla ALTER COLUMN bezeichnung
SET DATA TYPE CHAR(10) USING Cut10( bezeichnun
g);
87 SELECT * FROM bla;
88 -- SHOW WARNINGS;

```



```

5
6 SELECT 'WHERE mit TRUE / FALSE (Abschnitt 9.1.1)' Hinweis;
7 SELECT * FROM artikel WHERE TRUE;
8 SELECT * FROM artikel WHERE FALSE;
9
10
11 -- Start Groß-/Kleinschreibung
12 SELECT 'Groß/Kleinschreibung (Abschnitt 9.1.2)' Hinweis;
13 DROP TABLE IF EXISTS wurstbrot;
14 CREATE TEMPORARY TABLE wurstbrot (
15     name VARCHAR(255)
16 );
17
18 INSERT INTO wurstbrot
19 VALUES
20     ('Jaqueline'),
21     ,('Kevin'),
22     ,('kevin'),
23     ,('jaqueline')
24 ;
25 SELECT * FROM wurstbrot WHERE LOWER(name)
26 = 'kevin';
27
28 SELECT * FROM wurstbrot WHERE name = 'Kevin';
29
30 -- Ende Groß- Kleinschreibung
31
32 -- Start Einfache Wertzuweisung
33 SELECT 'Einfache Wertzuweisung (Abschnitt 9.2.1)' Hinweis;
34 SELECT * FROM bestellung_position;
35 UPDATE bestellung_position
36 SET menge = 2
37 WHERE bestellung_id = 1 AND position_nr = 3
38 ;

```

```

37  SELECT * FROM bestellung_position;
38  -- Ende Einfache Wertzuweisung
39
40  -- Start Wertberechnung
41  SELECT 'Wertberechnung (Abschnitt 9.2.2)'
Hinweis;
42  SELECT * FROM artikel;
43  UPDATE artikel SET einzelpreis = einzelpr
eis + einzelpreis / 100.0;
44  SELECT * FROM artikel;
45  UPDATE artikel SET einzelpreis = ROUND(ei
nzelpreis, 2);
46  SELECT * FROM artikel;
47  -- Ende Wertberechnung
48
49  -- Start Gebastelte Zeichenketten
50  SELECT 'Gebastelte Zeichenketten: Anrede
(Abschnitt 9.2.3)' Hinweis;
51  ALTER TABLE kunde ADD anrede VARCHAR(255)
;
52  UPDATE kunde
53  SET anrede = 'Sehr geehrte/r Frau/Herr '
|| nachname
54  WHERE nachname <> ''
55  ;
56  SELECT * FROM kunde;
57  -- Ende Gebastelte Zeichenketten
58
59
60  -- Start DELETE
61  SELECT 'Löschen mit DELETE (Abschnitt 9.3
)' Hinweis;
62  SELECT * FROM bestellung_position;
63  DELETE FROM bestellung_position WHERE bes
tellung_id = 1;
64  SELECT * FROM bestellung_position;
65
66  -- Start Wiederherstellen der Daten

```

```

67 INSERT INTO bestellung_position (bestellu
ng_id, position_nr, artikel_id, menge)
68 VALUES
69     (1, 1, 7856, 30)
70     ,(1, 2, 7863, 50)
71     ,(1, 3, 9015, 1)
72 ;
73 -- Ende Wiederherstellen der Daten
74
75 -- SELECT 'Löschversuch mit Constraint. E
rwarde eine Fehlermeldung! (
Abschnitt 9.3.1)';
76 -- DELETE FROM kunde WHERE kunde_id = 1;
77
78
79 SELECT 'Test AUTO_INCREMENT (Abschnitt 9.
3.2)' Hinweis;
80 SELECT kunde_id, nachname FROM kunde;
81 DELETE FROM kunde WHERE kunde_id IN (3,5)
;
82 -- Liefert in PostgreSQL eine Fehlermeldu
ng, PK-Wert ermittelt werden muss
83 INSERT INTO kunde (kunde_id, nachname, vo
rname) VALUES (6, 'Eichenschild', 'Thorin ');
84 SELECT kunde_id, nachname FROM kunde;
85 -- Start Wiederherstellen der Daten
86 INSERT INTO kunde (kunde_id, rechnung_adr
esse_id, nachname, vorname, art)
87 VALUES
88     (3, 2, 'Beutlin', 'Bilbo', 'prv')
89     ,(5, 4, 'Earendilionn','Elrond', 'gsch'
)
90 ;
91 -- Ende Wiederherstellen der Daten
92
93 SELECT 'Tabelle leeren (Abschnitt 9.3.4)'
Hinweis;
94 -- DELETE FROM bestellung_position;

```

```

95  -- TRUNCATE bestellung_position;
96  -- Ende DELETE
97  \echo 'Ende listing06. sql //////////////////////////////////
////////////////////////////////////'
```

28.2.3 Quelltexte zu Teil IV

Listing 28.32 postgresql/listing07.sql

```

1  -- Ausführen der vorherigen Befehle
2  \i listing06.sql
3
4  \echo 'Start listing07. sql //////////////////////////////////
////////////////////////////////////'
```

5

```

6  SELECT 'Ausdrücke (Abschnitt 10.1.1)' Hin
weis;
7
8  SELECT 5, 'Beginn der Auswertung';
9  SELECT -5, 9 + 4, 9 - 4, 9 * 4, 9.0 / 4.0
, DIV(9,4), 9 % 4, MOD(9,4);
10 SELECT 3 * 4 + 3, (3 * 4) + 3, 3 * (4 + 3
); -- Beispiel auf PostgreSQL angepasst, da DI
V dort ein Funktionsaufruf und kein Operator i
st.
11 -- SELECT 9 / 0, 9 * NULL, SQRT(-5) ; --
Erzeugt in PostgreSQL eine Fehlermeldung.
12
13 SELECT 'Zufallszahlen (Abschnitt 10.1.3)'
Hinweis;
14 SELECT SETSEED(0.5);
15 SELECT RANDOM();
16 SELECT SETSEED(0.5);
17 SELECT RANDOM();
18 SELECT SETSEED(0.5);
19 SELECT RANDOM() FROM artikel;
```

```

20
21 SELECT 'Variablen (Abschnitt 10.1.4)';
22 DO $$ -- Zum Deklarieren einer Variablen
muss man in einem Anweisungsblock sein. Daher
hier der anonyme Anweisungsblock.
23 DECLARE x double precision := 0;
24 BEGIN
25 SELECT POWER(2,8) INTO x;
26 PERFORM x - 1; -- Ergebnis nicht in der K
onsole sichtbar.
27 END $$;
28
29 -- SELECT 'Ungültiger Versuch: erwarte ei
ne Fehlermeldung (Abschnitt 10.1.4)'
30 -- DO $$
31 -- DECLARE artnr INT := 3010;
32 -- BEGIN
33 -- PERFORM * FROM artikel WHERE artikel_i
d = artnr; -- Ergebnis nicht in der Konsole si
chtbar.
34 -- END $$;
35
36
37 SELECT 'Spalten- und Zeilenwahl (Abschnit
t 10.2)' Hinweis;
38 SELECT * FROM artikel;
39 SELECT
40 artikel_id, bezeichnung, einzelpreis, wae
hrung, deleted
41 FROM artikel
42 ;
43
44 SELECT
45 artikel_id + 10000 AS Unsinn, deleted AS
Löschkennzeichen
46 FROM artikel
47 ;
48

```

```

49  SELECT
50  artikel_id, bezeichnung
51  FROM artikel
52  WHERE einzelpreis BETWEEN 10 AND 100
53  ;
54
55  SELECT 'Sortierung (Abschnitt 10.3)' Hinw
eis;
56  SELECT
57      artikel_id, bezeichnung, einzelpreis
58      FROM artikel
59      ORDER BY einzelpreis ASC
60  ;
61
62  SELECT
63      nachname, vorname
64      FROM kunde
65      ORDER BY nachname, vorname
66  ;
67
68  SELECT
69      nachname, vorname
70      FROM kunde
71      ORDER BY nachname DESC, vorname ASC
72  ;
73
74  -- CP850 wird von PostgreSQL nicht unters
tützt. Beispiel ist daher nicht verwendbar .
75  -- DROP TABLE IF EXISTS wurstbrot;
76  -- SET NAMES 'cp850';
77  -- CREATE TEMPORARY TABLE wurstbrot ( nam
e VARCHAR(255) CHARACTER SET 'cp850');
78  -- INSERT INTO wurstbrot
79  -- VALUES
80  -- ('winfried')
81  -- ,('achim')
82  -- ,('olga')
83  -- ,('zechine')

```

```

84  -- ,('ägidius')
85  -- ;
86  -- SELECT * FROM wurstbrot ORDER BY name;
87  --
88  -- DROP TABLE IF EXISTS wurstbrot;
89  -- SET NAMES 'utf8';
90  -- CREATE TEMPORARY TABLE wurstbrot ( name
e VARCHAR(255) CHARACTER SET 'utf8');
91  -- INSERT INTO wurstbrot
92  -- VALUES
93  -- ('winfried')
94  -- ,('achim')
95  -- ,('olga')
96  -- ,('zechine')
97  -- ,('ägidius')
98  -- ;
99  -- SELECT * FROM wurstbrot ORDER BY name;
100 --
101 DROP TABLE IF EXISTS wurstbrot;
102 CREATE TEMPORARY TABLE wurstbrot (name VA
RCHAR(255));
103 INSERT INTO wurstbrot
104 VALUES
105     ('winfried')
106     ,('achim')
107     ,('olga')
108     ,('Olga')
109     ,('zechine')
110 ;
111 SELECT * FROM wurstbrot ORDER BY name;
112 SELECT * FROM wurstbrot ORDER BY name DES
C;
113
114 -- In PostgreSQL wird immer zwischen Groß
- und Kleinschreibung unterschieden.
115 -- DROP TABLE IF EXISTS wurstbrot;
116 -- CREATE TEMPORARY TABLE wurstbrot (name
VARCHAR(255) BINARY);

```



```
117 -- INSERT INTO wurstbrot
118 -- VALUES
119 -- ('winfried')
120 -- ,('achim')
121 -- ,('olga')
122 -- ,('Olga')
123 -- ,('zechine')
124 -- ;
125 -- SELECT * FROM wurstbrot ORDER BY name;
126 -- SELECT * FROM wurstbrot ORDER BY name
DESC;
127
128 EXPLAIN SELECT * FROM kunde ORDER BY nach
name, vorname;
129
130 SELECT
131 bestellung_id, datum
132 FROM bestellung
133 ORDER BY datum;
134
135 EXPLAIN SELECT * FROM kunde ORDER BY nach
name, vorname;
136
137 -- In PostgreSQL gibt es keinen Datentyp
TIME. Geht nur über Hilfsfunktionen in Text um
zuwandeln.
138 -- SELECT
139 -- bestellung_id, TIME(datum) Uhrzeit
140 -- FROM bestellung
141 -- ORDER BY uhrzeit;
142
143 SELECT 'Mehrfachausgaben unterbinden (Abschnitt 10.4)' Hinweis;
144
145 SELECT ort FROM adresse ORDER BY ort;
146 SELECT DISTINCT ort FROM adresse ORDER BY
ort;
147 SELECT DISTINCT nachname FROM kunde ORDER
```

```

    BY nachname;
148
149 -- Das Fallbeispiel Bankimport müsste in
PostgreSQL ganz anders als
150 -- in MySQL oder MariaDB gelöst werden.
151 -- Daher hier nur der INSERT, damit die B
ankdaten vorliegen.
152 \d+ bank
153 ALTER TABLE bank ADD blz CHAR(12) NOT NUL
L DEFAULT '';
154 \i blz_20120305.sql
155
156 CREATE INDEX idx_bank_blzbankname
157 ON bank (blz, bankname)
158 ;
159
160 -- Beginn Für die vorhandenen Kunden eine
Bankverbindung bauen
161
162 INSERT INTO bankverbindung (kunde_id, ban
kverbindung_nr, bank_id, kontonummer, iban )
163     VALUES
164     (1, 1, 2, '1111111111', '10010010111111
1111'),
165     (1, 2, 2, '1111111112', '10010010111111
1112'),
166     (2, 1, 35, '2222222221', '1006019822222
22221'),
167     (3, 1, 35, '3333333331', '100601983333333
331'),
168     (4, 1, 90, '4444444441', '120700004444444
441'),
169     (5, 1, 90, '5555555551', '120700005555555
551')
170 ;
171 -- Ende Für die vorhandenen Kunden eine B
ankverbindung bauen
172

```

```

173  -- Ende Bankimport
174
175  SELECT 'Ausschneiden mit LIMIT (Abschnitt
176      10.5)' Hinweis;
177  SELECT blz, bankname FROM bank ORDER BY b
178  lz LIMIT 3;
179  SELECT
180  blz, bankname
181  FROM bank
182  ORDER BY blz DESC
183  LIMIT 1
184  ;
185  -- Hier per Unterabfrage, da es mit Varia
186  blen in PostgreSQL nicht so einfach geht.
187  SELECT
188  bankname
189  FROM bank
190  WHERE blz = (
191  SELECT
192  blz
193  FROM bank
194  ORDER BY blz DESC
195  LIMIT 1 )
196  ;
197  SELECT blz, bankname FROM bank ORDER BY b
198  lz LIMIT 2 OFFSET 1;
199  SELECT blz, bankname FROM bank ORDER BY b
200  lz LIMIT 1;
201  SELECT blz, bankname FROM bank ORDER BY b
202  lz LIMIT 1 OFFSET 0;
203  \echo 'Ende listing07. sql //////////////////////////////////
204  //////////////////////////////////'

```

Listing 28.33 postgresql/listing08.sql

```

1  -- Ausführen der vorherigen Befehle

```

```

2  \i listing07.sql
3
4  \echo 'Start listing08. sql //////////////////////////////////
/////////////////////////////////////'
5
6  -- Start INNER JOIN
7  SELECT 'CROSS JOIN (Abschnitt 11.1)' Hinweis;
8  SELECT
9  kunde_id, nachname, vorname, rechnung_adr
esse_id, adresse_id
10 FROM
11 kunde, adresse
12 ;
13
14 SELECT 'CROSS JOIN mit WHERE (Abschnitt 1
1.1)' Hinweis;
15 SELECT
16 kunde_id, nachname, vorname, rechnung_adr
esse_id, adresse_id
17 FROM
18 kunde, adresse
19 WHERE
20 rechnung_adresse_id = adresse_id
21 ;
22
23 SELECT 'INNER JOIN (Abschnitt 11.2)' Hinweis;
24 SELECT
25 kunde_id, nachname, vorname, rechnung_adr
esse_id, adresse_id
26 FROM
27 kunde INNER JOIN adresse
28 ON rechnung_adresse_id = adresse_id
29 ;
30
31 SELECT
32 kunde_id, kontonummer, blz, bankname

```

```

33 FROM
34 bankverbindung INNER JOIN bank
35 ON bankverbindung.bank_id = bank.bank_id
36 ;
37 SELECT
38 nachname, vorname, kontonummer
39 FROM
40 bankverbindung INNER JOIN kunde ON bankve
rbindung.kunde_id = kunde.kunde_id
41 ;
42
43 SELECT 'Abkürzung mit USING (Abschnitt 11
.2.2)' Hinweis;
44 SELECT
45 kunde_id, kontonummer, blz, bankname
46 FROM
47 bankverbindung INNER JOIN bank USING (ban
k_id)
48 ;
49
50 SELECT 'Abkürzung mit NATURAL JOIN (Absch
nitt 11.2.2)' Hinweis;
51 SELECT
52 kunde_id, kontonummer, blz, bankname
53 FROM
54 bankverbindung NATURAL JOIN bank
55 ;
56 -- Ende INNER JOIN
57
58 -- Start Datenquelle
59 SELECT 'JOIN als Datenquelle (Abschnitt 1
1.2.3)' Hinweis;
60 SELECT
61 k.kunde_id, k.nachname, k.vorname, a.stra
sse, a.hnr, a.plz, a.ort
62 FROM
63 kunde k INNER JOIN adresse a
64 ON k.rechnung_adresse_id = a.adresse_id

```

```

65  ;
66  DROP TABLE IF EXISTS tmp_kadresse;
67  CREATE TEMPORARY TABLE tmp_kadresse AS
68  SELECT
69  k.kunde_id, k.nachname, k.vorname, a.stra
sse, a.hnr, a.plz, a.ort
70  FROM
71  kunde k INNER JOIN adresse a
72  ON k.rechnung_adresse_id = a.adresse_id
73  ;
74  SELECT
75  t.kunde_id, t.nachname, t.ort, b.bestellu
ng_id, DATE(b.datum)
76  FROM
77  bestellung b INNER JOIN tmp_kadresse t US
ING (kunde_id)
78  ;
79  DROP TABLE IF EXISTS tmp_kbank;
80  CREATE TEMPORARY TABLE tmp_kbank AS
81  SELECT
82  bv.kunde_id, bv.bankverbindung_nr, bv.kon
tonummer,
83  bv.iban, ba.blz, ba.bankname
84  FROM
85  bankverbindung bv INNER JOIN bank ba USIN
G (bank_id)
86  ;
87  DROP TABLE IF EXISTS tmp_kbankeinzug;
88  CREATE TEMPORARY TABLE tmp_kbankeinzug AS
89  SELECT
90  ka.*, kb.bankverbindung_nr, kb.kontonumme
r,
91  kb.iban, kb.blz, kb.bankname
92  FROM
93  tmp_kadresse ka INNER JOIN tmp_kbank kb U
SING(kunde_id);
94
95  SELECT kunde_id, nachname, ort, bankname

```

```

FROM tmp_kbankeinzug ;
96  -- Ende Datenquelle
97
98  -- Start Keine Schlüssel
99  SELECT 'Ohne Schlüssel (Abschnitt 11.2.4)
' Hinweis;
100  \d+ tmp_kadresse;
101  \d+ tmp_kbank;
102  SELECT
103  k.kunde_id, k.vorname, a.strasse, a.hnr,
a.plz, a.ort
104  FROM
105  kunde k INNER JOIN adresse a
106  ON k.rechnung_adresse_id <= a.adresse_id
107  ;
108  -- Ende Kein Schlüssel
109
110  -- Start mehr als zwei Tabellen
111  SELECT 'Mit mehr als 2 Tabellen (Abschnitt
10)' Hinweis;
112  SELECT
113  a.bezeichnung, nm.warengruppe_id
114  FROM
115  artikel_nm_warengruppe nm NATURAL JOIN ar
tikel a
116  ;
117  SELECT
118  w.bezeichnung, nm.artikel_id
119  FROM
120  artikel_nm_warengruppe nm NATURAL JOIN wa
rengruppe w
121  ;
122  SELECT
123  w.bezeichnung, nm.artikel_id
124  FROM
125  (artikel_nm_warengruppe nm INNER JOIN war
engruppe w USING(warengruppe_id))
126  ;

```

```
127  SELECT
128  w.bezeichnung, a.bezeichnung
129  FROM
130  (artikel_nm_warengruppe nm INNER JOIN warengruppe w USING (warengruppe_id))
131  INNER JOIN artikel a USING(artikel_id)
132  ;
133  SELECT
134  w.bezeichnung Warengruppe, a.bezeichnung
Artikel
135  FROM
136  artikel_nm_warengruppe nm INNER JOIN warengruppe w USING (warengruppe_id)
137  INNER JOIN artikel a USING(artikel_id)
138  ;
139  -- Ende mehr als zwei Tabellen
140
141  -- Start Aufbau Lieferanten
142  SELECT 'Aufbau der Lieferanten' Hinweis;
143  INSERT INTO lieferant (firmenname, adresse_id)
144  VALUES
145  ('Gartenbedarf AllesGrün', 10),
146  ('Office International', 11),
147  ('Bürohengst GmbH', 11)
148  ;
149
150  INSERT INTO artikel_nm_lieferant (lieferant_id, artikel_id)
151  VALUES
152  (1, 7856)
153  ,(1, 7863)
154  ,(1, 9010)
155  ,(1, 9015)
156  ,(3, 3001)
157  ,(3, 3005)
158  ,(3, 3006)
159  ,(3, 3007)
```



```

160  ,(3, 3010)
161  ;
162  -- Ende Aufbau Lieferanten
163
164  -- Start OUTER JOIN
165  SELECT 'OUTER JOIN (Abschnitt 11.4)' Hinw
eis;
166  SELECT
167  k.kunde_id, k.nachname, k.vorname, b.datu
m
168  FROM
169  bestellung b INNER JOIN kunde k USING (ku
nde_id)
170  ORDER BY
171  k.nachname, k.vorname
172  ;
173  SELECT
174  k.kunde_id, k.nachname, k.vorname, b.datu
m
175  FROM
176  bestellung b RIGHT OUTER JOIN kunde k USI
NG (kunde_id)
177  ORDER BY
178  k.nachname, k.vorname
179  ;
180  SELECT
181  k.kunde_id, k.nachname, k.vorname, b.datu
m
182  FROM
183  kunde k LEFT OUTER JOIN bestellung b USIN
G (kunde_id)
184  ORDER BY
185  k.nachname, k.vorname
186  ;
187  SELECT
188  l.lieferant_id, l.firmenname, a.artikel_i
d, a.bezeichnung
189  FROM

```

```
190 artikel_nm_lieferant nm INNER JOIN artike
l a USING(artikel_id)
191 INNER JOIN lieferant l USING(lieferant_id
)
192 ORDER BY firmenname
193 ;
194 SELECT
195 l.lieferant_id, l.firmenname, a.artikel_i
d, a.bezeichnung
196 FROM
197 artikel_nm_lieferant nm INNER JOIN artike
l a USING(artikel_id)
198 RIGHT JOIN lieferant l USING(lieferant_id
)
199 ORDER BY firmenname
200 ;
201 SELECT
202 l.firmenname
203 FROM
204 artikel_nm_lieferant nm INNER JOIN artike
l a USING(artikel_id)
205 RIGHT JOIN lieferant l USING(lieferant_id
)
206 WHERE artikel_id IS NULL
207 ORDER BY firmenname
208 ;
209 SELECT k.kunde_id, k.nachname, k.vorname
210 FROM
211 bestellung b RIGHT JOIN kunde k USING(kun
de_id)
212 WHERE b.bestellung_id IS NULL
213 ;
214 SELECT k.kunde_id, k.nachname, k.vorname,
b.bestellung_id, b.datum
215 FROM
216 bestellung b FULL OUTER JOIN kunde k USIN
G(kunde_id)
217 ;
```

```

218  -- Ende OUTER JOIN
219
220  -- Start Aufbau Forum
221  SELECT 'SELF JOIN (Abschnitt 11.5)' Hinweis;
222  SELECT 'CREATE account' Hinweis;
223  CREATE TYPE accountart AS ENUM('aktiv','gesperrt');
224  CREATE TABLE account (
225    kunde_id INT,
226    name VARCHAR(255) NOT NULL DEFAULT '',
227    password CHAR(34) NOT NULL DEFAULT '',
228    status accountart NOT NULL DEFAULT 'aktiv',
229    kommentar TEXT ,
230    admin BOOL NOT NULL DEFAULT FALSE,
231    deleted SMALLINT NOT NULL DEFAULT 0,
232    PRIMARY KEY(kunde_id),
233    FOREIGN KEY (kunde_id)
234    REFERENCES kunde(kunde_id)
235    ON UPDATE RESTRICT
236    ON DELETE RESTRICT
237  ) ;
238
239  SELECT 'CREATE beitrags' Hinweis;
240  CREATE TABLE beitrags (
241    beitrags_id SERIAL,
242    account_id INT NOT NULL,
243    bezug_beitrags_id INT NOT NULL DEFAULT 1,
244    zeitstempel TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
245    nachricht TEXT ,
246    deleted SMALLINT NOT NULL DEFAULT 0,
247    PRIMARY KEY(beitrags_id),
248    FOREIGN KEY (account_id)
249    REFERENCES account(kunde_id)
250    ON UPDATE RESTRICT
251    ON DELETE RESTRICT,

```

```

252 FOREIGN KEY (bezug_beitrag_id)
253 REFERENCES beitrag(beitrag_id)
254 ON UPDATE RESTRICT
255 ON DELETE RESTRICT
256 ) ;
257
258 SELECT 'account daten' Hinweis;
259 INSERT INTO account (kunde_id, name, pass
wort, admin)
260 VALUES
261 (1, 'admin', md5('rosi'), TRUE)
262 , (2, 'frodo', md5('elbereth'), FALSE)
263 , (3, 'bilbo', md5('blitzerschlagen'), FA
LSE)
264 , (5, 'elle', md5('feanor'), FALSE)
265 ;
266
267
268 SELECT 'beitrag daten' Hinweis;
269 INSERT INTO beitrag (beitrag_id, account_
id, bezug_beitrag_id, nachricht)
270 VALUES
271 (1, 1, 1, '')
272 ;
273 INSERT INTO beitrag (beitrag_id, account_
id, bezug_beitrag_id, zeitstempel, nachricht)
274 VALUES
275 (2, 2, 1, '2011-05-01 14:13:00', 'Der Lie
ferservice ist super.')
276 , (3, 3, 2, '2011-05-02 11:45:00', 'Das f
inde ich auch.')
277 , (4, 5, 2, '2011-05-01 17:01:00', 'Aber
ein wenig langsam.')
278 , (5, 2, 4, '2011-05-01 17:15:00', 'Finde
ich nicht.')
279 , (6, 5, 1, '2011-06-12 09:07:00', 'Angeb
ot könnte besser sein.')
280 ;

```

```

281  -- Ende Aufbau Forum
282
283  -- Start SELF JOIN
284  SELECT 'SELF JOIN (Abschnitt 11.5)' Hinweis;
285  SELECT
286  kunde_id, name, status, admin
287  FROM
288  account
289  ;
290  SELECT
291  beitrags_id, account_id, bezug_beitrags_id,
    LEFT(nachricht, 20)
292  FROM
293  beitrags
294  ;
295  -- SELECT 'Erwarte eine Fehlermeldung (Abschnitt 11.5)';
296  -- SELECT nachricht
297  -- FROM
298  -- beitrags INNER JOIN beitrags ON bezug_beitrags_id = beitrags_id
299  -- WHERE beitrags_id = 2
300  -- ;
301  SELECT ant.nachricht Antwort
302  FROM
303  beitrags ant INNER JOIN beitrags orig ON ant.bezug_beitrags_id = orig.beitrags_id
304  WHERE orig.beitrags_id = 2
305  ;
306
307  SELECT 'CTE rekursiv (Abschnitt 11.5)';
308  WITH RECURSIVE ant_auf AS
309  (
310  -- nicht rekursiver Teil
311  SELECT * FROM beitrags WHERE bezug_beitrags_id = 2
312  UNION ALL

```

```

313  -- rekursiver Teil
314  SELECT ant.*
315  FROM
316  beitrags ant INNER JOIN ant_auf orig
317  ON ant.bezugsbeitrags_id = orig.beitrags_id
318  )
319  SELECT beitrags_id, bezugsbeitrags_id, nach
richt FROM ant_auf;
320
321  -- Ende SELF JOIN
322
323  -- Start Redundant
324  SELECT 'Beschleunigen mit Redundanzen (Abschnitt 11.6)' Hinweis;
325  ALTER TABLE
326  kunde
327  ADD r_strasse VARCHAR(255),
328  ADD r_ort VARCHAR(255),
329  ADD r_aktuell BOOL NOT NULL DEFAULT TRUE,
330  ADD l_strasse VARCHAR(255),
331  ADD l_ort VARCHAR(255),
332  ADD l_aktuell BOOL NOT NULL DEFAULT TRUE
333  ;
334
335  -- In PostgreSQL kann kein Update auf ein
er abgeleiteten Tabelle durchgeführt werden.
336  -- UPDATE kunde INNER JOIN adresse ON rec
hnungs_adresse_id = adresse_id
337  -- SET
338  -- r_strasse = CONCAT(strasse, ' ', hnr),
339  -- r_ort = CONCAT(lkz, '-', plz, ' ', ort
),
340  -- r_aktuell = TRUE
341  -- ;
342  SELECT
343  kunde_id, r_strasse, r_ort, r_aktuell
344  FROM kunde
345  ;

```

```

346  -- Ende Redundant
347  \echo 'Ende listing08. sql //////////////////////////////////
/////////////////////////////////'

```

Listing 28.34 postgresql/listing09.sql

```

1  -- Ausführen der vorherigen Befehle
2  \i listing08.sql
3
4  \echo 'Start listing09. sql //////////////////////////////////
/////////////////////////////////'
5
6  SELECT 'Aggregatfunktionen (Abschnitt 12.1)' Hinweis;
7  SELECT AVG(einzelpreis) FROM artikel;
8  SELECT COUNT(*) FROM kunde;
9  SELECT COUNT(liefer_adresse_id) FROM kunde;
10 SELECT COUNT(DISTINCT(rechnung_adresse_id)) FROM kunde;
11 SELECT MAX(einzelpreis), MIN(einzelpreis) FROM artikel;
12 SELECT SUM(menge) FROM bestellung_position;
13
14 SELECT
15 bp.menge, a.einzelpreis
16 FROM
17 bestellung_position bp INNER JOIN artikel
18 a USING(artikel_id)
19 ;
20 SELECT
21 bp.menge * a.einzelpreis Positionswert
22 FROM
23 bestellung_position AS bp INNER JOIN artikel AS a USING(artikel_id)
24 ;
25 SELECT

```

```

25  SUM(bp.menge * a.einzelpreis)
26  FROM
27  bestellung_position AS bp INNER JOIN arti
kel AS a USING(artikel_id)
28  ;
29
30  -- Start Aufbau Lagerverwaltung
31  SELECT 'Aufbau Lagerbestand (Abschnitt 12
.1)' Hinweis;
32  SELECT 'CREATE lagerbestand' Hinweis;
33  DROP TABLE IF EXISTS lagerbestand;
34  CREATE TABLE lagerbestand (
35  artikel_id INT,
36  menge_mindest DECIMAL(14,6) NOT NULL DEFA
ULT 0.0,
37  menge_aktuell DECIMAL(14,6) NOT NULL DEFA
ULT 0.0,
38  deleted SMALLINT NOT NULL DEFAULT 0,
39  PRIMARY KEY(artikel_id),
40  FOREIGN KEY (artikel_id)
41  REFERENCES artikel(artikel_id)
42  ON UPDATE RESTRICT
43  ON DELETE RESTRICT
44  );
45
46  SELECT 'Lagerbestand festlegen' Hinweis;
47  INSERT INTO lagerbestand (artikel_id, men
ge_aktuell, menge_mindest)
48  VALUES
49  (7856, 500, 100)
50  ,(7863, 400, 100)
51  ,(9010, 30, 10)
52  ,(9015, 25, 10)
53  ,(3001, 5000, 1000)
54  ,(3005, 250, 200)
55  ,(3006, 250, 200)
56  ,(3007, 149, 200)
57  ,(3010, 3, 100)

```



```

58 ;
59 -- Ende Aufbau Lagerverwaltung
60
61
62 SELECT 'GROUP BY (Abschnitt 12.2)' Hinweis;
63 -- Start Group by
64 SELECT
65 COUNT(*)
66 FROM
67 bestellung_position
68 ;
69 SELECT
70 bestellung_id Bestellnummer, COUNT(*) Anzahl_der_Positionen
71 FROM
72 bestellung_position
73 GROUP BY
74 bestellung_id
75 ;
76 SELECT
77 a.bezeichnung, bp.menge
78 FROM
79 bestellung_position bp INNER JOIN artikel
80 a USING (artikel_id)
81 ;
82 SELECT
83 a.bezeichnung Artikelname, SUM(bp.menge) Anzahl_bestellter_Artikel
84 FROM
85 bestellung_position bp INNER JOIN artikel
86 a USING (artikel_id)
87 GROUP BY
88 a.bezeichnung
89 ;
90 SELECT
91 a.bezeichnung Artikelname, SUM(bp.menge)

```

```

Anzahl_bestellter_Artikel
91  FROM
92  bestellung_position bp RIGHT JOIN artikel
93  a USING (artikel_id)
94  GROUP BY
95  a.bezeichnung
96  ;
97  SELECT
98  a.bezeichnung Artikelname,
99  CASE
100  WHEN SUM(bp.menge) IS NULL THEN 0
101  ELSE SUM(bp.menge)
102  END AS Anzahl_bestellter_Artikel
103  FROM
104  bestellung_position bp RIGHT JOIN artikel
105  a USING (artikel_id)
106  GROUP BY
107  a.bezeichnung
108  ;
109  -- SELECT 'Eine Fehlermeldung! (Abschnitt
110  12.3)';
111  -- SELECT
112  -- a.bezeichnung 'Artikelname', SUM(bp.me
113  nge) 'Anzahl bestellter Artikel'
114  -- FROM
115  -- bestellung_position bp INNER JOIN arti
116  kel a USING (artikel_id)
117  -- WHERE
118  -- SUM(bp.menge) > 10
119  -- GROUP BY
120  -- a.bezeichnung
121  -- ;
122
123  SELECT 'Gruppenergebnisse mit ROLLUP summ
124  ieren (Abschnitt 12.2)';
125  SELECT
126  artikel_id, SUM(bp.menge) Anzahl_bestellt
127  er_Artikel

```

```
121 FROM
122 bestellung_position bp RIGHT JOIN artikel
   a USING (artikel_id)
123 GROUP BY
124 ROLLUP (artikel_id)
125 ;
126
127 SELECT 'Gruppenergebnisse mit HAVING filt
ern (Abschnitt 12.3)' Hinweis;
128 SELECT
129 a.bezeichnung Artikelname, SUM(bp.menge)
Anzahl_bestellter_Artikel
130 FROM
131 bestellung_position bp INNER JOIN artikel
   a USING (artikel_id)
132 GROUP BY
133 a.bezeichnung
134 HAVING
135 SUM(bp.menge) > 10
136 ;
137 SELECT
138 a.bezeichnung Artikelname, SUM(bp.menge)
Anzahl_bestellter_Artikel
139 FROM
140 bestellung_position bp INNER JOIN artikel
   a USING (artikel_id)
141 WHERE
142 a.bezeichnung LIKE 'Silber%'
143 GROUP BY
144 a.bezeichnung
145 HAVING
146 SUM(bp.menge) > 10
147 ;
148
149 SELECT 'Fragen zu Gruppen (Abschnitt 12.4
.1)' Hinweis;
150 SELECT
151 SUBSTRING(blz, 1, 1) Clearinggebiet, COUN
```

```

T(*) Anzahl
152 FROM
153 bank
154 GROUP BY
155 Clearinggebiet
156 ORDER BY Anzahl DESC
157 ;
158 -- Start Bestellungen erweitern
159 SELECT 'Bestellungen erweitern' Hinweis;
160 INSERT INTO bestellung (bestellung_id, kunde_id, adresse_id, datum)
161 VALUES
162 (3, 1, 1, '2011-01-15 16:43:00')
163 ,(4, 1, 1, '2011-01-16 09:15:00')
164 ,(5, 1, 1, '2011-01-16 09:16:00')
165 ,(6, 2, 2, '2012-04-01 13:11:00')
166 ;
167
168 INSERT INTO bestellung_position (bestellung_id, position_nr, artikel_id, menge)
169 VALUES
170 (3, 1, 7856, 10)
171 ,(3, 2, 7863, 10)
172 ,(4, 1, 3006, 1)
173 ,(4, 2, 3010, 4)
174 ,(5, 1, 3001, 100)
175 ,(5, 2, 3010, 5)
176 ,(5, 3, 3006, 1)
177 ,(5, 4, 3005, 4)
178 ;
179 -- Ende Bestellungen erweitern
180 SELECT
181 EXTRACT(ISOYEAR FROM datum) || '/' || EXTRACT(MONTH FROM datum) Monat, COUNT(*) Anzahl
182 FROM bestellung
183 GROUP BY
184 EXTRACT(ISOYEAR FROM datum), EXTRACT(MONTH FROM datum)

```

```

185 ORDER BY
186 EXTRACT(ISOYEAR FROM datum) DESC, EXTRACT
(MONTH FROM datum) DESC
187 ;
188
189 EXPLAIN
190 SELECT
191 SUBSTRING(blz, 1, 1) Clearinggebiet, COUN
T(*) Anzahl
192 FROM
193 bank
194 GROUP BY
195 Clearinggebiet
196 ORDER BY
197 Anzahl DESC;
198
199 EXPLAIN
200 SELECT
201 lkz || SUBSTRING(blz, 6, 3) Sinnlos, COUN
T(*) Anzahl
202 FROM
203 bank
204 GROUP BY
205 Sinnlos
206 ORDER BY
207 Sinnlos;
208 -- Ende Group by
209 \echo 'Ende listing09. sql //////////////////////////////////
/////////////////////////////////'

```

Listing 28.35 postgresql/listing10.sql

```

1 -- Ausführen der vorherigen Befehle
2 \i listing09.sql
3
4 \echo 'Start listing10. sql //////////////////////////////////
/////////////////////////////////'
5

```

```

6  SELECT 'Start Aufbau Rechnungswesen' Hinweis;
7
8  INSERT INTO rechnung
9  (kunde_id, bestellung_id, adresse_id, datum)
10 SELECT kunde_id, bestellung_id, adresse_id,
11 datum + INTERVAL '1 days' FROM bestellung;
12 INSERT INTO rechnung (rechnung_id, kunde_id,
13 bestellung_id, adresse_id, datum, bezahler, status)
14 VALUES
15 (7, 3, 0, 2, '2012-04-06 12:11:00', 'kredit', 'gestellt')
16 ,(8, 3, 0, 2, '2012-04-07 13:12:00', 'kredit', 'mahnung1')
17 ,(9, 5, 0, 4, '2012-04-08 14:13:00', 'rechnung', 'storno')
18 ,(10, 5, 0, 4, '2012-04-09 15:14:00', 'rechnung', 'beglichen')
19 ,(11, 5, 0, 4, '2012-04-10 16:15:00', 'rechnung', 'beglichen')
20 ;
21 UPDATE rechnung SET status = 'inkasso' WHERE rechnung_id = 2;
22 INSERT INTO rechnung_position (rechnung_id, position_nr, artikel_id, menge)
23 SELECT bestellung_id, position_nr, artikel_id, menge FROM bestellung_position;
24 INSERT INTO rechnung_position
25 (rechnung_id, position_nr, artikel_id, menge)
26 VALUES
27 (7, 1, 3001, 5)
28 ,(7, 2, 3005, 1)
29 ,(8, 1, 3006, 7)

```

```

30      ,(8, 2, 3007, 1)
31      ,(10, 1, 3010, 15)
32      ,(10, 2, 3001, 5)
33      ,(11, 1, 3005, 20)
34  ;
35
36  UPDATE rechnung SET rabatt = 7 WHERE kund
e_id = 1;
37  UPDATE rechnung SET skonto = 3 WHERE kund
e_id = 5;
38
39  SELECT 'Ende Aufbau Rechnungswesen' Hinwe
is;
40
41  SELECT 'Problem und Lösung (Abschnitt 13.
1)' Hinweis;
42  SELECT
43  bp.bestellung_id, SUM(bp.menge * a.einzel
preis) Bestellwert
44  FROM
45  bestellung_position bp INNER JOIN artikel
a USING(artikel_id)
46  GROUP BY
47  bp.bestellung_id
48  ;
49  SELECT
50  k.nachname, k.vorname, SUM(bp.menge * a.e
inzelpreis) Bestellwert
51  FROM
52  bestellung_position bp INNER JOIN artikel
a USING(artikel_id)
53  INNER JOIN bestellung b USING(bestellung_
id)
54  INNER JOIN kunde k USING(kunde_id)
55  GROUP BY
56  bp.bestellung_id, k.nachname, k.vorname
57  ;
58  DROP TABLE IF EXISTS tmp_bestellwert;

```

```

59 CREATE TEMPORARY TABLE tmp_bestellwert
60 AS
61 SELECT
62 bp.bestellung_id, k.nachname, k.vorname,
SUM(bp.menge * a.einzelpreis) bestellwert
63 FROM
64 bestellung_position bp INNER JOIN artikel
a USING(artikel_id)
65 INNER JOIN bestellung b USING(bestellung_
id)
66 INNER JOIN kunde k USING(kunde_id)
67 GROUP BY
68 bp.bestellung_id, k.nachname, k.vorname
69 ;
70
71 SELECT * FROM tmp_bestellwert;
72 SELECT
73 nachname, vorname, SUM(bestellwert) Umsat
z
74 FROM tmp_bestellwert
75 GROUP BY nachname, vorname
76 ;
77 SELECT
78 bw.nachname, bw.vorname, SUM(bw.bestellwe
rt) Umsatz
79 FROM
80 (
81 SELECT
82 bp.bestellung_id, k.nachname, k.vorname,
SUM(bp.menge * a.einzelpreis) bestellwert
83 FROM
84 bestellung_position bp INNER JOIN artikel
a USING(artikel_id)
85 INNER JOIN bestellung b USING(bestellung_
id)
86 INNER JOIN kunde k USING(kunde_id)
87 GROUP BY
88 bp.bestellung_id, k.nachname, k.vorname

```



```
89  ) AS bw
90  GROUP BY nachname, vorname
91  ;
92  -- Start nicht korrelierende Unterabfrage
93  SELECT 'Nicht korrelierende Unterabfrage
(Abschnitt 13.2.1.1)' Hinweis;
94  SELECT MAX(blz) FROM bank;
95  SELECT bankname
96  FROM bank
97  WHERE blz = '37010050'
98  ;
99  SELECT bankname
100 FROM bank
101 WHERE blz =
102 (
103 SELECT MAX(blz)
104 FROM bank
105 )
106 ;
107 SELECT
108 AVG(einzelpreis) AS durchschnittspreis
109 FROM artikel
110 ;
111 SELECT *
112 FROM artikel
113 WHERE
114 einzelpreis > 30
115 ;
116 SELECT *
117 FROM artikel
118 WHERE
119 einzelpreis >
120 (
121 SELECT
122 AVG(einzelpreis) AS durchschnittspreis
123 FROM artikel
124 )
125 ;
```

```

126  SELECT
127  bp.bestellung_id, SUM(bp.menge * a.einzel
preis) bestellwert
128  FROM
129  bestellung_position bp INNER JOIN artikel
a USING(artikel_id)
130  GROUP BY
131  bp.bestellung_id
132  ;
133  SELECT AVG(bw.bestellwert)
134  FROM (
135  SELECT
136  bp.bestellung_id, SUM(bp.menge * a.einzel
preis) bestellwert
137  FROM
138  bestellung_position bp INNER JOIN artikel
a USING(artikel_id)
139  GROUP BY
140  bp.bestellung_id
141  ) AS bw
142  ;
143  SELECT
144  bp1.bestellung_id, SUM(bp1.menge * a1.ein
zelpreis) bestellwert1
145  FROM
146  bestellung_position bp1 INNER JOIN artike
l a1 USING(artikel_id)
147  GROUP BY
148  bp1.bestellung_id
149  HAVING SUM(bp1.menge * a1.einzelpreis) >
100
150  ;
151  SELECT
152  bp1.bestellung_id, SUM(bp1.menge * a1.ein
zelpreis) bwert1
153  FROM
154  bestellung_position bp1 INNER JOIN artike
l a1 USING(artikel_id)

```

```
155  GROUP BY
156  bp1.bestellung_id
157  HAVING
158  SUM(bp1.menge * a1.einzelpreis) >
159  (
160  SELECT AVG(bw.bwert2)
161  FROM
162  (
163  SELECT
164  bp2.bestellung_id, SUM(bp2.menge * a2.einzelpreis) bwert2
165  FROM
166  bestellung_position bp2 INNER JOIN artikel
167  l a2
168  USING(artikel_id)
169  GROUP BY
170  bp2.bestellung_id
171  ) AS bw
172  ;
173  SELECT bestellung_id FROM bestellung;
174  SELECT rechnung_id
175  FROM rechnung
176  WHERE bestellung_id IN
177  (
178  SELECT bestellung_id FROM bestellung
179  )
180  ;
181  SELECT a.einzelpreis
182  FROM
183  artikel_nm_warengruppe INNER JOIN artikel
184  a USING(artikel_id)
185  INNER JOIN warengruppe w USING(warengruppe_id)
186  WHERE
187  w.bezeichnung = 'Gartenbedarf'
188  ;
189  SELECT
```

```
189  a.bezeichnung, a.einzelpreis
190  FROM
191  artikel a
192  WHERE
193  a.einzelpreis > ALL(SELECT 10)
194  ;
195  SELECT
196  a.bezeichnung, a.einzelpreis
197  FROM
198  artikel a
199  WHERE
200  a.einzelpreis > ALL
201  (
202  SELECT a.einzelpreis
203  FROM
204  artikel_nm_warengruppe INNER JOIN artikel
    a USING(artikel_id)
205  INNER JOIN warengruppe w USING(warengrupp
    e_id)
206  WHERE
207  w.bezeichnung = 'Gartenbedarf'
208  )
209  ;
210  SELECT
211  rechnung_id, kunde_id, SUM(einzelpreis*me
    nge) AS umsatz
212  FROM
213  rechnung_position INNER JOIN artikel USIN
    G(artikel_id)
214  INNER JOIN rechnung USING(rechnung_id)
215  GROUP BY
216  rechnung_id, kunde_id
217  ;
218  SELECT SUM(umsatz) AS umsatzsumme
219  FROM
220  (
221  SELECT
222  rechnung_id, kunde_id, SUM(einzelpreis*me
```

```

nge) AS umsatz
223 FROM
224 rechnung_position INNER JOIN artikel USING
G(artikel_id)
225 INNER JOIN rechnung USING(rechnung_id)
226 GROUP BY
227 rechnung_id, kunde_id
228 ) AS ksum
229 GROUP BY
230 kunde_id
231 ;
232 SELECT SUM(umsatz) AS umsatzsumme
233 FROM
234 (
235 SELECT
236 rechnung_id, kunde_id, SUM(einzelpreis*me
nge) AS umsatz
237 FROM
238 rechnung_position INNER JOIN artikel USING
G(artikel_id)
239 INNER JOIN rechnung USING(rechnung_id)
240 INNER JOIN kunde USING(kunde_id)
241 WHERE nachname = 'beutlin'
242 GROUP BY
243 rechnung_id, kunde_id
244 ) AS ksum
245 GROUP BY
246 kunde_id
247 ;
248 SELECT kunde_id, SUM(umsatz) AS umsatzsum
me
249 FROM
250 (
251 SELECT
252 rechnung_id, kunde_id, SUM(einzelpreis*me
nge) AS umsatz
253 FROM
254 rechnung_position INNER JOIN artikel USING

```

```

G(artikel_id)
255  INNER JOIN rechnung USING(rechnung_id)
256  GROUP BY
257  rechnung_id, kunde_id
258  ) AS ksum
259  GROUP BY
260  kunde_id
261  HAVING SUM(umsatz) > ALL (SELECT 100)
262  ;
263  SELECT kunde_id, kunde.nachname, kunde.vo
rname, SUM(umsatz) AS umsatzsumme
264  FROM
265  (
266  SELECT
267  rechnung_id, kunde_id, SUM(einzelpreis*me
nge) AS umsatz
268  FROM
269  rechnung_position INNER JOIN artikel USIN
G(artikel_id)
270  INNER JOIN rechnung USING(rechnung_id)
271  GROUP BY
272  rechnung_id, kunde_id
273  ) AS ksum
274  INNER JOIN kunde USING (kunde_id)
275  GROUP BY
276  kunde_id, kunde.nachname, kunde.vorname
277  HAVING SUM(umsatz) > ALL
278  (
279  SELECT SUM(umsatz) AS umsatzsumme
280  FROM
281  (
282  SELECT
283  rechnung_id, kunde_id, SUM(einzelpreis*me
nge) AS umsatz
284  FROM
285  rechnung_position INNER JOIN artikel USIN
G(artikel_id)
286  INNER JOIN rechnung USING(rechnung_id)

```

```
287 INNER JOIN kunde USING(kunde_id)
288 WHERE nachname = 'beutlin'
289 GROUP BY
290 rechnung_id, kunde_id
291 ) AS ksum
292 GROUP BY
293 kunde_id, kunde.nachname, kunde.vorname
294 )
295 ;
296 SELECT a.einzelpreis
297 FROM
298 artikel_nm_warengruppe INNER JOIN artikel
   a USING(artikel_id)
299 INNER JOIN warengruppe w USING(warengrupp
   e_id)
300 WHERE
301 w.bezeichnung = 'Bürobedarf'
302 ;
303 SELECT
304 a.bezeichnung, a.einzelpreis
305 FROM
306 artikel a
307 WHERE
308 a.einzelpreis > ANY(SELECT 10)
309 ;
310 SELECT
311 a.bezeichnung, a.einzelpreis
312 FROM
313 artikel a
314 WHERE
315 a.einzelpreis > ANY
316 (
317 SELECT a.einzelpreis
318 FROM
319 artikel_nm_warengruppe INNER JOIN artikel
   a USING(artikel_id)
320 INNER JOIN warengruppe w USING(warengrupp
   e_id)
```

```
321 WHERE
322 w.bezeichnung = 'Bürobedarf'
323 )
324 ;
325 SELECT
326 a.bezeichnung, a.einzelpreis
327 FROM
328 artikel a
329 WHERE
330 a.artikel_id NOT IN
331 (
332 SELECT
333 artikel_id
334 FROM
335 artikel_nm_warengruppe nm INNER JOIN ware
336 ngruppe w USING(warengruppe_id)
337 )
338 AND
339 a.einzelpreis > ANY
340 (
341 SELECT a.einzelpreis
342 FROM
343 artikel_nm_warengruppe nm INNER JOIN arti
344 kel a USING(artikel_id)
345 INNER JOIN warengruppe w USING(warengrupp
346 e_id)
347 )
348 ;
349 SELECT DISTINCT k.nachname, k.vorname
350 FROM
351 rechnung r INNER JOIN kunde k USING(kunde
352 _id)
353 WHERE
354 EXTRACT(ISOYEAR FROM r.datum) = '2011' AN
355 D (k.nachname, k.vorname) IN
```



```
354  (
355  SELECT
356  k.nachname, k.vorname
357  FROM
358  rechnung r INNER JOIN kunde k USING(kunde
_id)
359  WHERE
360  EXTRACT(ISOYEAR FROM r.datum) = '2012'
361  )
362  ;
363  -- Ende nicht korrelierende Unterabfrage
364
365  -- Start korrelierende Unterabfrage
366  SELECT 'Korrelierende Unterabfrage (Abschnitt 13.3.1)' Hinweis;
367  SELECT rechnung_id
368  FROM rechnung r
369  WHERE
370  (
371  SELECT
372  COUNT(position_nr)
373  FROM rechnung_position rp
374  WHERE
375  rp.rechnung_id = r.rechnung_id
376  ) >= 3
377  ;
378  SELECT rechnung_id
379  FROM rechnung r
380  WHERE
381  EXISTS
382  (
383  SELECT
384  bestellung_id
385  FROM bestellung b
386  WHERE b.bestellung_id = r.bestellung_id
387  )
388  ;
389  -- Ende korrelierende Unterabfrage
```

```

390
391  -- Start CTE
392  SELECT 'CTE vs SUBSELECT (Abschnitt 13.5)
';
393  WITH
394  rsum AS (
395  SELECT
396  rechnung_id, kunde_id, nachname, vorname,
    SUM(einzelpreis*menge) AS umsatz_r
397  FROM
398  rechnung_position INNER JOIN artikel USING(artikel_id)
399  INNER JOIN rechnung USING(rechnung_id)
400  INNER JOIN kunde USING(kunde_id)
401  GROUP BY rechnung_id, kunde_id, nachname,
    vorname
402  ),
403  ksum AS (
404  SELECT
405  kunde_id, nachname, vorname, SUM(umsatz_r
) AS umsatz_k
406  FROM rsum
407  GROUP BY kunde_id, nachname, vorname
408  ),
409  bsum AS (
410  SELECT umsatz_k
411  FROM ksum
412  WHERE nachname='Beutlin'
413  )
414  SELECT kunde_id, nachname, vorname, umsatz_k
415  FROM ksum
416  WHERE umsatz_k > ALL (SELECT * FROM bsum)
417  ;
418  -- Ende CTE
419
420  -- Start Fallstudie
421  SELECT 'Fallstudie Datenimport (Abschnitt

```

```

13.5)' Hinweis;
422 DROP TABLE IF EXISTS tmp_import;
423 CREATE TEMPORARY TABLE tmp_import
424 (
425     nachname VARCHAR(255),
426     vorname VARCHAR(255),
427     strasse VARCHAR(255),
428     hnr VARCHAR(255),
429     lkz VARCHAR(255),
430     plz VARCHAR(255),
431     ort VARCHAR(255),
432     ktnr VARCHAR(255),
433     blz VARCHAR(255)
434 );
435
436 \copy tmp_import FROM './kunden01.csv' WI
TH DELIMITER ';' CSV HEADER
437 SELECT SETVAL('kunde_kunde_id_seq', 6); -
- PK Wert manuel anpassen
438 INSERT INTO kunde (nachname, vorname)
439 SELECT DISTINCT nachname, vorname
440 FROM tmp_import tmp
441 WHERE
442 (tmp.vorname, tmp.nachname) NOT IN
443 (
444     SELECT vorname, nachname FROM kunde
445 )
446 ;
447 SELECT * FROM tmp_import;
448 SELECT SETVAL('adresse_adresse_id_seq', 1
00); -- PK Wert manuel anpassen
449 INSERT INTO adresse (strasse, hnr, lkz, p
lz, ort)
450 SELECT DISTINCT strasse, hnr, lkz, plz, o
rt
451 FROM tmp_import tmp
452 WHERE
453 (tmp.strasse, tmp.hnr, tmp.lkz, tmp.plz,

```

```
tmp.ort) NOT IN
454  (
455  SELECT strasse, hnr, lkz, plz, ort
456  FROM adresse
457  )
458  ;
459  ALTER TABLE tmp_import ADD kunde_id INT ,
460  ADD adresse_id INT ,
461  ADD bank_id CHAR(12)
462  ;
463  UPDATE tmp_import SET
464  kunde_id =
465  (
466  SELECT kunde_id
467  FROM kunde k
468  WHERE tmp_import.vorname = k.vorname AND
tmp_import.nachname = k.nachname
469  ),
470  adresse_id =
471  (
472  SELECT adresse_id
473  FROM adresse a
474  WHERE
475  tmp_import.strasse = a.strasse
476  AND tmp_import.hnr = a.hnr
477  AND tmp_import.plz = a.plz
478  AND tmp_import.ort = a.ort
479  ),
480  bank_id =
481  (
482  SELECT bank_id FROM bank WHERE tmp_import
.blz = bank.blz LIMIT 1
483  )
484  ;
485
486  INSERT INTO
487  bankverbindung (kunde_id, bankverbindung_
nr, bank_id, kontonummer, iban)
```

```

488 SELECT DISTINCT kunde_id, 1, bank_id, ktn
    r, blz || ktnr
489 FROM tmp_import tmp
490 WHERE
491 (tmp.kunde_id, 1, tmp.bank_id, tmp.ktnr)
NOT IN
492 (
493 SELECT kunde_id, 1, bank_id, kontonummer
494 FROM bankverbindung
495 )
496 ;
497 ALTER TABLE tmp_import
498 ADD bankverbindung_nr INT NOT NULL DEFAULT 1
499 ;
500 UPDATE kunde k SET
501 rechnung_adresse_id =
502 (
503 SELECT adresse_id FROM tmp_import tmp WHERE
tmp.kunde_id = k.kunde_id
504 )
505 WHERE k.rechnung_adresse_id IS NULL
506 ;
507 SELECT kunde_id, nachname, vorname, strasse, ort
508 FROM
509 kunde LEFT JOIN adresse ON kunde.rechnung
_adresse_id = adresse_id
510 ;
511 -- Ende Fallstudie
512
513
514 \echo 'Ende listing10. sql //////////////////////////////////
////////////////////////////////////'

```

Listing 28.36 postgresql/listing11.sql

```

1  -- Ausführen der vorherigen Befehle

```

```

2  \include listing10.sql
3
4  \echo 'Start listing11. sql //////////////////////////////////
/////////////////////////////////////'
5
6  SELECT 'UNION (Abschnitt 14.1)' Hinweis;
7  SELECT
8  strasse, hnr, lkz, plz, ort
9  FROM
10 kunde INNER JOIN adresse ON rechnung_adre
sse_id = adresse_id
11 ;
12 SELECT
13 strasse, hnr, lkz, plz, ort
14 FROM
15 lieferant INNER JOIN adresse USING(address
e_id)
16 ;
17 SELECT
18 strasse, hnr, lkz, plz, ort
19 FROM
20 kunde INNER JOIN adresse ON rechnung_adre
sse_id = adresse_id
21 UNION
22 SELECT
23 strasse, hnr, lkz, plz, ort
24 FROM
25 lieferant INNER JOIN adresse USING(address
e_id)
26 ;
27 SELECT
28 strasse, hnr, lkz, plz, ort
29 FROM
30 kunde INNER JOIN adresse ON rechnung_adre
sse_id = adresse_id
31 UNION ALL
32 SELECT
33 strasse, hnr, lkz, plz, ort

```

```

34 FROM
35 lieferant INNER JOIN adresse USING(adress
e_id)
36 ;
37
38 SELECT 'INTERSECT (Abschnitt 14.2.1)' Hin
weis;
39
40 SELECT kunde_id FROM rechnung WHERE EXTRA
CT(ISOYEAR FROM datum) = '2011'
41 INTERSECT
42 SELECT kunde_id FROM rechnung WHERE EXTRA
CT(ISOYEAR FROM datum) = '2012';
43
44 SELECT 'Schnittmenge mit Unterabfrage (Ab
schnitt 14.2.2)' Hinweis;
45 SELECT kunde_id FROM rechnung WHERE EXTRA
CT(ISOYEAR FROM datum) = '2011';
46 SELECT DISTINCT kunde_id
47 FROM rechnung
48 WHERE EXTRACT(ISOYEAR FROM datum) = '2012
' AND kunde_id IN (1, 2, 3)
49 ;
50 SELECT DISTINCT kunde_id
51 FROM rechnung
52 WHERE
53 EXTRACT(ISOYEAR FROM datum) = '2012' AND
kunde_id IN
54 (
55 SELECT DISTINCT kunde_id FROM rechnung WH
ERE EXTRACT(ISOYEAR FROM datum) = '2011'
56 )
57 ;
58
59 SELECT 'EXCEPT (Abschnitt 14.3.1)' Hinwei
s;
60 SELECT strasse, hnr, lkz, plz, ort FROM a
dresse

```

```

61  EXCEPT
62  SELECT strasse, hnr, lkz, plz, ort
63  FROM kunde INNER JOIN adresse ON liefer_a
adresse_id = adresse_id
64  ;
65
66  SELECT 'Differenz mit Unterabfrage (Absch
nitt 14.3.1)' Hinweis;
67  SELECT
68  strasse, hnr, lkz, plz, ort
69  FROM
70  adresse
71  ;
72  SELECT
73  strasse, hnr, lkz, plz, ort
74  FROM
75  kunde INNER JOIN adresse
76  ON liefer_adresse_id = adresse_id
77  ;
78  SELECT kunde_id
79  FROM rechnung
80  WHERE EXTRACT(ISOYEAR FROM datum) <> '201
1' AND EXTRACT(ISOYEAR FROM datum) = '2012'
81  ;
82  SELECT kunde_id FROM rechnung WHERE EXTRA
CT(ISOYEAR FROM datum) = '2011';
83  SELECT DISTINCT kunde_id
84  FROM rechnung
85  WHERE
86  EXTRACT(ISOYEAR FROM datum) = '2012' AND
kunde_id NOT IN (1, 2, 3)
87  ;
88  SELECT DISTINCT kunde_id
89  FROM rechnung
90  WHERE
91  EXTRACT(ISOYEAR FROM datum) = '2012' AND
kunde_id NOT IN
92  (

```



```

93  SELECT kunde_id FROM rechnung WHERE EXTRA
CT(ISOYEAR FROM datum) = '2011'
94  )
95  ;
96  \echo 'Ende listing11. sql //////////////////////////////////
/////////////////////////////////'

```

Listing 28.37 postgresql/listing12.sql

```

1  -- Ausführen der vorherigen Befehle
2  \include listing11.sql
3
4  \echo 'Start listing12. sql //////////////////////////////////
/////////////////////////////////'
5
6  SELECT kunde_id, nachname, vorname, art F
ROM kunde;
7  SELECT 'Start Aufbau einer temporaeren ar
t-Tabelle (Abschnitt 15.1)' Hinweis;
8
9  DROP TABLE IF EXISTS tmp_art;
10 CREATE TEMPORARY TABLE tmp_art
11 (
12  art kundenart,
13  k_art VARCHAR(255)
14 )
15 ;
16
17 INSERT INTO tmp_art
18 VALUES
19 ('prv', 'Privatkunde')
20 ,('gsch', 'Geschäftskunde')
21 ,('unb', 'Unbekannt')
22 ;
23
24 SELECT kunde_id, nachname, vorname, k_art
25 FROM
26 kunde INNER JOIN tmp_art USING(art)

```

```

27 ;
28 SELECT 'Ende Aufbau einer temporaeren art
-Tabelle' Hinweis;
29
30 SELECT 'Lösung mit UNIONs (Abschnitt 15.1
)' Hinweis;
31 SELECT kunde_id, nachname, vorname, k_art
32 FROM
33 kunde INNER JOIN
34 (
35 SELECT 'prv' AS art, 'Privatkunde' AS k_a
rt
36 UNION
37 SELECT 'gsch' AS art, 'Geschäftskunde' AS
k_art
38 UNION
39 SELECT 'unb' AS art, 'Unbekannt' AS k_art
40 ) t
41 ON kunde.art::text = t.art
42 ;
43
44 SELECT 'Nö (Abschnitt 15.1)' Hinweis;
45 SELECT kunde_id, nachname, vorname,
46 CASE art
47 WHEN 'prv' THEN 'Privatkunde'
48 WHEN 'gsch' THEN 'Geschäftskunde'
49 ELSE 'Unbekannt'
50 END AS k_art
51 FROM kunde
52 ;
53
54 -- Start Einfacher CASE
55 SELECT 'Einfacher CASE (Abschnitt 15.2)'
Hinweis;
56 SELECT artikel_id, bezeichnung,
57 CASE waehrung
58 WHEN 'EUR' THEN ROUND(einzelpreis, 2) ||
, e'

```

```

59  WHEN 'USD' THEN ROUND(einzelpreis, 2) ||
, $,
60  ELSE '?????'
61  END AS preis
62  FROM artikel
63  ;
64  UPDATE artikel SET waehrung = 'XYZ' WHERE
artikel_id = 3001;
65  SELECT artikel_id, bezeichnung,
66  CASE waehrung
67  WHEN 'EUR' THEN ROUND(einzelpreis, 2) ||
, e,
68  WHEN 'USD' THEN ROUND(einzelpreis, 2) ||
, $,
69  END AS preis
70  FROM artikel
71  ;
72  UPDATE artikel SET waehrung = 'EUR' WHERE
artikel_id = 3001;
73  -- Ende Einfacher CASE
74
75  -- Start Searched CASE
76  SELECT 'Searched CASE (Abschnitt 15.3)' H
inweis;
77  SELECT beitrags_id,
78  CASE
79  WHEN bezug_bei trags_id > 1 THEN 'Antwort'
80  WHEN bezug_bei trags_id <= 1 THEN 'Keine An
twort'
81  ELSE '???????'
82  END AS Typ
83  FROM beitrags
84  ORDER BY beitrags_id
85  ;
86  SELECT beitrags_id,
87  CASE
88  WHEN bezug_bei trags_id > 1
89  THEN nachricht || ' Antwort auf ' || bezu

```

```

g_beitrag_id || ': >' ||
90  (
91  SELECT nachricht
92  FROM beitrag b2
93  WHERE b2.beitrag_id = b1.bezug_beitrag_id
94  )
95  WHEN bezug_beitrag_id <= 1 THEN nachricht
96  ELSE '??????'
97  END AS Inhalt
98  FROM beitrag b1
99  WHERE beitrag_id > 1
100 ;
101 SELECT beitrag_id,
102 CASE
103 WHEN bezug_beitrag_id >= 0 THEN 'Antwort'
104 WHEN bezug_beitrag_id > 1 THEN 'Keine Ant
wort'
105 ELSE '??????'
106 END AS Typ
107 FROM beitrag
108 ORDER BY beitrag_id
109 ;
110 -- Ende Searched CASE
111
112 -- Start Fallbeispiele
113 SELECT 'Fallbeispiel Lagerbestand (Abschn
itt 15.4.1)' Hinweis;
114 SELECT * FROM lagerbestand;
115 SELECT a.bezeichnung,
116 CASE
117 WHEN l.menge_aktuell <= l.menge_mindest
118 THEN 'Artikel nachbestellen'
119 ELSE 'Bestand ausreichend'
120 END AS lagerstand
121 FROM lagerbestand l INNER JOIN artikel a
USING(artikel_id)
122 ORDER BY l.menge_aktuell / l.menge_mindest
t

```

```

123  ;
124
125  SELECT 'Fallbeispiel Kundengruppen (Abschnitt 15.4.2)' Hinweis;
126  SELECT DISTINCT k.kunde_id, k.nachname, k
.vorname,
127  CASE
128  WHEN 3 <
129  (
130  SELECT COUNT(rechnung_id) FROM rechnung r
131  WHERE r.kunde_id = k.kunde_id
132  ) THEN 'Prämiumkunde'
133  WHEN 2 <
134  (
135  SELECT COUNT(rechnung_id) FROM rechnung r
136  WHERE r.kunde_id = k.kunde_id
137  ) THEN 'Guter Kunde'
138  ELSE 'Kleinkunde'
139  END kundenart
140  FROM
141  rechnung r RIGHT OUTER JOIN kunde k USING
(kunde_id)
142  ;
143  SELECT k.kunde_id, k.nachname, k.vorname,
144  (
145  SELECT SUM(a.einzelpreis * rp.menge)
146  FROM rechnung r INNER JOIN rechnung_posit
ion rp
147  USING (rechnung_id)
148  INNER JOIN artikel a
149  USING (artikel_id)
150  WHERE r.kunde_id = k.kunde_id
151  ) rechnungswert,
152  CASE
153  WHEN
154  kunde_id NOT IN (SELECT kunde_id FROM rec
hnung)
155  THEN 'Kleinkunde'

```

```
156 WHEN
157 (
158 SELECT SUM(a.einzelpreis * rp.menge) rech
nungswert
159 FROM rechnung r INNER JOIN rechnung_posit
ion rp
160 USING (rechnung_id)
161 INNER JOIN artikel a
162 USING (artikel_id)
163 WHERE r.kunde_id = k.kunde_id
164 ) BETWEEN 0 AND 300 THEN 'Kleinkunde'
165 WHEN
166 (
167 SELECT SUM(a.einzelpreis * rp.menge) rech
nungswert
168 FROM rechnung r INNER JOIN rechnung_posit
ion rp
169 USING (rechnung_id)
170 INNER JOIN artikel a
171 USING (artikel_id)
172 WHERE r.kunde_id = k.kunde_id
173 ) BETWEEN 300 AND 1000 THEN 'Guter Kunde'
174 ELSE 'Prämiumkunde'
175 END AS kundenart
176 FROM kunde k
177 ORDER BY
178 (
179 SELECT SUM(a.einzelpreis * rp.menge) rech
nungswert
180 FROM rechnung r INNER JOIN rechnung_posit
ion rp
181 USING (rechnung_id)
182 INNER JOIN artikel a
183 USING (artikel_id)
184 WHERE r.kunde_id = k.kunde_id
185 ) DESC
186 ;
187 DROP TABLE IF EXISTS tmp_umsatz;
```

```
188 CREATE TEMPORARY TABLE tmp_umsatz AS
189 SELECT kunde_id, SUM(a.einzelpreis * rp.m
enge) rechnungswert
190 FROM rechnung r INNER JOIN rechnung_posit
ion rp USING (rechnung_id)
191 INNER JOIN artikel a USING (artikel_id)
192 GROUP BY kunde_id
193 ;
194 SELECT k.kunde_id, k.nachname, k.vorname,
tmp.rechnungswert,
195 CASE
196 WHEN
197 tmp.kunde_id IS NULL THEN 'Kleinkunde'
198 WHEN
199 rechnungswert BETWEEN 0 AND 300 THEN 'Kle
inkunde'
200 WHEN
201 rechnungswert BETWEEN 300 AND 1000 THEN '
Guter Kunde'
202 ELSE 'Premiumkunde'
203 END AS kundenart
204 FROM kunde k LEFT JOIN tmp_umsatz tmp USI
NG(kunde_id)
205 ORDER BY rechnungswert DESC
206 ;
207
208 WITH tmp_umsatz (kunde_id, rechnungswert)
209 AS (
210 SELECT kunde_id, SUM(a.einzelpreis * rp.m
enge) rechnungswert
211 FROM rechnung r INNER JOIN rechnung_posit
ion rp USING (rechnung_id)
212 INNER JOIN artikel a USING (artikel_id)
213 GROUP BY kunde_id
214 )
215 SELECT k.kunde_id, k.nachname, k.vorname,
tmp.rechnungswert,
216 CASE
```

```

217 WHEN
218 tmp.kunde_id IS NULL THEN 'Kleinkunde'
219 WHEN
220 rechnungswert BETWEEN 0 AND 300 THEN 'Kle
inkunde'
221 WHEN
222 rechnungswert BETWEEN 300 AND 1000 THEN '
Guter Kunde'
223 ELSE 'Premiumkunde'
224 END AS kundenart
225 FROM kunde k LEFT JOIN tmp_umsatz tmp USI
NG(kunde_id)
226 ORDER BY rechnungswert DESC
227 ;
228
229 SELECT 'Fallbeispiel Lieferanten (Abschni
tt 15.4.3)' Hinweis;
230 SELECT l.firmenname,
231 CASE
232 WHEN
233 l.lieferant_id IN
234 (
235 SELECT lieferant_id FROM artikel_nm_liefe
rant
236 ) THEN 'aktiv'
237 ELSE 'inaktiv'
238 END AS status
239 FROM lieferant l
240 ORDER BY firmenname
241 ;
242
243 SELECT a.bezeichnung,
244 CASE
245 WHEN l.menge_aktuell <= l.menge_mindest T
HEN 'Artikel sofort nachbestellen'
246 WHEN l.menge_aktuell <= l.menge_mindest +
0.3 * l.menge_mindest THEN 'Artikel bald nach
bestellen'

```



```

247 ELSE 'Bestand ausreichend' END AS lagerst
and FROM lagerbestand l INNER JOIN artikel a U
SING(artikel_id)
248 ORDER BY l.menge_aktuell / l.menge_mindes
t
249 ;
250 -- Ende Fallbeispiele
251 \echo 'Ende listing12. sql //////////////////////////////////
////////////////////////////////'

```

Listing 28.38 postgresql/listing13.sql

```

1  -- Ausführen der vorherigen Befehle
2  -- \include listing12.sql
3
4  \echo 'Start listing13. sql //////////////////////////////////
////////////////////////////////'
5
6  SELECT 'Meine erste Ansicht (Abschnitt 16
.1.1)' Hinweis;
7  CREATE VIEW
8  view_kundenrechnungsadresse (id, nachname
, vorname, strasse, ort)
9  AS
10 SELECT
11 kunde_id, nachname, vorname,
12 CONCAT_WS(' ', strasse, hnr),
13 CONCAT_WS(' ', lkz, plz, ort)
14 FROM
15 kunde INNER JOIN adresse
16 ON rechnung_adresse_id = adresse_id
17 WHERE kunde.deleted = 0
18 ;
19 SELECT * FROM view_kundenrechnungsadresse
;
20 \dv
21
22 SELECT 'VIEW-Verarbeitung (Abschnitt 16.1

```

```

.2)' Hinweis;
23 CREATE VIEW view_artikel_aktiv
24 AS
25 SELECT * FROM artikel
26 WHERE deleted = 0;
27 ;
28 SELECT FLOOR(artikel_id / 1000) gruppe, A
VG(einzelpreis)
29 FROM view_artikel_aktiv
30 GROUP BY gruppe
31 ;
32 EXPLAIN SELECT FLOOR(artikel_id / 1000) g
ruppe, AVG(einzelpreis)
33 FROM view_artikel_aktiv
34 GROUP BY gruppe
35 ;
36 SELECT artikel_id, bezeichnung FROM
37 view_artikel_aktiv
38 WHERE
39 artikel_id BETWEEN 7000 AND 9000
40 ;
41 EXPLAIN SELECT artikel_id, bezeichnung FR
OM
42 view_artikel_aktiv
43 WHERE
44 artikel_id BETWEEN 7000 AND 9000
45 ;
46 CREATE VIEW view_artikel_aktiv_tmp
47 AS
48 SELECT * FROM artikel
49 WHERE deleted = 0
50 ;
51 SELECT artikel_id, bezeichnung
52 FROM view_artikel_aktiv_tmp
53 WHERE artikel_id BETWEEN 7000 AND 8000
54 ;
55 SELECT artikel_id, bezeichnung
56 FROM view_artikel_aktiv_tmp

```

```

57 WHERE artikel_id BETWEEN 3000 AND 4000
58 ;
59 EXPLAIN SELECT artikel_id, bezeichnung
60 FROM view_artikel_aktiv_tmp
61 WHERE artikel_id BETWEEN 3000 AND 4000
62 ;
63
64 -- Start DROP VIEW
65 SELECT 'Ansicht löschen (Abschnitt 16.1.3
) ' Hinweis;
66 \dv
67 DROP VIEW view_artikel_aktiv_tmp;
68 \dv
69
70 DROP DATABASE IF EXISTS tmp1;
71 CREATE DATABASE tmp1;
72 \connect tmp1
73 CREATE TABLE a (i INT);
74 CREATE TABLE b (i INT);
75 CREATE VIEW ab
76 AS
77 SELECT * FROM a INNER JOIN b USING(i)
78 ;
79 \dt
80 \dv
81 -- DROP TABLE a;
82 \dv
83 DROP TABLE a CASCADE;
84 \dv
85 \connect oshop
86 DROP DATABASE tmp1;
87
88 -- Ende DROP VIEW
89
90 SELECT 'Anwendungsgebiet: Vereinfachung (Abschnitt 16.2) ' Hinweis;
91 CREATE OR REPLACE
92 VIEW view_artikel_aktiv

```

```

93  AS
94  SELECT * FROM artikel
95  WHERE deleted = 0
96  ;
97  CREATE OR REPLACE VIEW view_artikel_verfu
egbar
98  AS
99  SELECT artikel_id, bezeichnung
100 FROM artikel INNER JOIN lagerbestand USIN
G(artikel_id)
101 WHERE menge_aktuell >= menge_mindest
102 ORDER BY artikel_id
103 ;
104 SELECT * FROM view_artikel_verfuegbar;
105 CREATE OR REPLACE
106 VIEW view_lagerbestand_aktiv
107 AS
108 SELECT * FROM lagerbestand
109 WHERE deleted = 0
110 ;
111 CREATE OR REPLACE VIEW view_lagerbestand_
artikelbezeichnung
112 AS
113 SELECT l.*, a.bezeichnung
114 FROM
115 view_lagerbestand_aktiv l INNER JOIN view
_artikel_aktiv a
116 USING(artikel_id)
117 ;
118 SELECT * FROM view_lagerbestand_artikelbe
zeichnung;
119
120 -- SELECT 'Grenzen der VIEW. Erwarte Fehl
ermeldungen (Abschnitt 16.4)';
121 DROP TABLE IF EXISTS tmp;
122 CREATE TEMPORARY TABLE tmp AS SELECT * FR
OM artikel;
123 CREATE OR REPLACE VIEW view_tmp AS SELECT

```

```

    artikel_id FROM tmp;
124 DROP VIEW IF EXISTS view_tmp;
125 CREATE OR REPLACE VIEW view_tmp AS SELECT
    nachname FROM kunde ORDER BY nachname DESC;
126 SELECT * FROM view_tmp;
127 SELECT * FROM view_tmp ORDER BY nachname;
128 CREATE OR REPLACE VIEW view_tmp_eins
129 AS
130 SELECT nachname FROM kunde
131 ;
132 CREATE OR REPLACE VIEW view_tmp_zwei
133 AS
134 SELECT rechnung_id, COUNT(*) anzahl
135 FROM rechnung_position
136 GROUP BY rechnung_id
137 ;
138 UPDATE view_tmp_eins
139 SET nachname = 'Streicher'
140 WHERE nachname = 'Telcontar'
141 ;
142 -- SELECT 'Grenzen der VIEW. Erwarte Fehl
ermeldungen';
143 -- UPDATE view_tmp_zwei
144 -- SET anzahl = 2
145 -- WHERE anzahl > 2
146 -- ;
147 -- Rückbau
148 UPDATE view_tmp_eins
149 SET nachname = 'Telcontar'
150 WHERE nachname = 'Streicher'
151 ;
152 -- UPDATE view_tmp_zwei SET rechnung_id =
    1 WHERE rechnung_id > 1;
153 INSERT INTO view_tmp_eins VALUES ('Wurst'
), ('Brot');
154 SELECT * FROM view_tmp_eins;
155 DELETE FROM view_tmp_eins WHERE nachname
IN ('Wurst', 'Brot');

```

```

156
157  \echo 'Ende listing13. sql //////////////////////////////////
////////////////////////////////'

```

28.2.4 Quelltexte zu Teil V

Listing 28.39 postgresql/listing14.sql

```

1  -- Ausführen der vorherigen Befehle
2  \include listing13.sql
3  \echo 'Start listing14. sql //////////////////////////////////
////////////////////////////////'
4  SELECT 'Lagerbestand (Abschnitt 18)' Hinweis;
5  UPDATE lagerbestand
6  SET menge_aktuell = menge_aktuell - 5
7  WHERE artikel_id = 9015;
8
9  UPDATE lagerbestand
10 SET menge_aktuell = menge_aktuell - 3
11 WHERE artikel_id = 9015;
12
13 SELECT * FROM lagerbestand WHERE artikel_
id = 9015;
14
15 -- Start Lock
16 BEGIN WORK; -- Lock innerhalb einer Trans
aktion
17 SELECT 'LOCK (Abschnitt 18)' Hinweis;
18 LOCK TABLE artikel IN ACCESS EXCLUSIVE MO
DE;
19 SELECT COUNT(*) FROM artikel;
20 -- Ein SELECT COUNT(*) FROM warengruppe;
von einer anderen Transaktion aus würde eine F
M provozieren.
21 COMMIT WORK;

```

```

22  -- Ende Lock
23  \echo 'Ende listing14. sql //////////////////////////////////
////////////////////////////////'

```

Listing 28.40 postgresql/listing15.sql

```

1  -- Ausführen der vorherigen Befehle
2  \i listing14.sql
3  \c oshop
4  \echo 'Start listing15. sql //////////////////////////////////
////////////////////////////////'
5
6  SELECT 'Das Problem (Abschnitt 19.1)' Hin
weis;
7  INSERT INTO artikel
8  (bezeichnung, einzelpreis, waehrung)
9  VALUES
10 ('Säge', 17.85, 'EUR')
11 ;
12 INSERT INTO artikel_nm_warengruppe
13 VALUES
14 (3, CURRVAL('artikel_artikel_id_seq')),
15 (4, CURRVAL('artikel_artikel_id_seq'))
16 ;
17
18 UPDATE lagerbestand
19 SET menge_aktuell = menge_aktuell - 1
20 WHERE artikel_id = 9015
21 ;
22 UPDATE bestellung SET status = 'versendet
' WHERE bestellung_id = 1;
23 SELECT 'Isolationsebenen (Abschnitt 19.3)
' Hinweis;
24 -- AUTOCOMMIT wird in PostgreSQL nicht lä
nger unterstützt.
25
26 -- Start read uncommitted
27 SELECT 'READ UNCOMMITTED (Abschnitt 19.3.

```

```
1)' Hinweis;  
28  -- Session 1  
29  START TRANSACTION ISOLATION  
30  LEVEL REPEATABLE READ;  
31  SELECT artikel_id, menge_aktuell  
32  FROM lagerbestand  
33  WHERE artikel_id = 9015  
34  ;  
35  UPDATE lagerbestand  
36  SET menge_aktuell = menge_aktuell - 1  
37  WHERE artikel_id = 9015  
38  ;  
39  
40  SELECT artikel_id, menge_aktuell  
41  FROM lagerbestand  
42  WHERE artikel_id = 9015  
43  ;  
44  ROLLBACK;  
45  SELECT artikel_id, menge_aktuell  
46  FROM lagerbestand  
47  WHERE artikel_id = 9015  
48  ;  
49  -- Session 2  
50  START TRANSACTION  
51  ISOLATION LEVEL READ UNCOMMITTED;  
52  
53  
54  SELECT artikel_id, menge_aktuell  
55  FROM lagerbestand  
56  WHERE artikel_id = 9015  
57  ;  
58  SELECT artikel_id, menge_aktuell  
59  FROM lagerbestand  
60  WHERE artikel_id = 9015  
61  ;  
62  SELECT artikel_id, menge_aktuell  
63  FROM lagerbestand  
64  WHERE artikel_id = 9015
```



```
65 ;
66 -- Ende read uncommitted
67
68 -- Start read committed
69 SELECT 'READ COMMITTED (Abschnitt 19.3.2)
' Hinweis;
70 -- Session 1
71 SELECT artikel_id, menge_aktuell
72 FROM lagerbestand
73 WHERE artikel_id = 9015
74 ;
75 UPDATE lagerbestand
76 SET menge_aktuell = menge_aktuell - 1
77 WHERE artikel_id = 9015
78 ;
79 SELECT artikel_id, menge_aktuell
80 FROM lagerbestand
81 WHERE artikel_id = 9015
82 ;
83 COMMIT;
84 SELECT artikel_id, menge_aktuell
85 FROM lagerbestand
86 WHERE artikel_id = 9015;
87 -- Session 2
88 SET SESSION TRANSACTION ISOLATION LEVEL R
EAD COMMITTED;
89 SELECT artikel_id, menge_aktuell
90 FROM lagerbestand
91 WHERE artikel_id = 9015
92 ;
93 SELECT artikel_id, menge_aktuell
94 FROM lagerbestand
95 WHERE artikel_id = 9015
96 ;
97 SELECT artikel_id, menge_aktuell
98 FROM lagerbestand
99 WHERE artikel_id = 9015
100 ;
```

```
101  -- Ende read committed
102
103  -- Start repeatable read
104  SELECT 'REPEATABLE READ (Abschnitt 19.3.3
105  )' Hinweis;
106  -- Session 1
107  SELECT artikel_id, menge_aktuell
108  FROM lagerbestand
109  WHERE artikel_id = 9015
110  ;
111  UPDATE lagerbestand
112  SET menge_aktuell = menge_aktuell - 1
113  WHERE artikel_id = 9015
114  ;
115  COMMIT;
116  UPDATE lagerbestand
117  SET menge_aktuell = menge_aktuell - 1
118  WHERE artikel_id = 9015
119  ;
120  COMMIT;
121  SELECT artikel_id, menge_aktuell
122  FROM lagerbestand
123  WHERE artikel_id = 9015
124  ;
125  -- Session 2
126  SET SESSION TRANSACTION ISOLATION LEVEL R
127  EPEATABLE READ;
128  START TRANSACTION;
129  SELECT artikel_id, menge_aktuell
130  FROM lagerbestand
131  WHERE artikel_id = 9015
132  ;
133  SELECT artikel_id, menge_aktuell
134  FROM lagerbestand
135  WHERE artikel_id = 9015
136  ;
137  COMMIT;
138  SELECT artikel_id, menge_aktuell
```

```

137 FROM lagerbestand
138 WHERE artikel_id = 9015
139 ;
140 -- Ende repeatable read
141
142 -- Start Serializable
143 SELECT 'SERIALIZABLE (Abschnitt 19.3.4)'
Hinweis;
144 -- Session 1
145 SELECT COUNT(*) FROM artikel;
146 INSERT INTO artikel VALUES (1, 'X', 0.0,
'EUR', 0);
147 COMMIT;
148
149 -- Session 2
150 SET SESSION TRANSACTION ISOLATION LEVEL S
ERIALIZABLE;
151 DELETE FROM artikel WHERE artikel_id = 1;
152 DELETE FROM artikel WHERE artikel_id = 1;
153 SELECT COUNT(*) FROM artikel;
154 -- Ende Serializable
155 SET SESSION TRANSACTION ISOLATION LEVEL R
EPEATABLE READ;
156 \echo 'Ende listing15. sql //////////////////////////////////
/////////////////////////////////'

```

28.3 Microsoft SQL Server

28.3.1 Quelltexte zu Teil II

Listing 28.41 mssql/listing01.sql

```

1 -- Warum formatiere ich den Quelltext so
   ganz anders, als beispielsweise diese Beautyfi

```

er?

```
2  -- * http://www.dpriver.com/pp/sqlformat.
htm
3  -- * https://codebeautify.org/sqlformatte
r
4  -- Ich finde die Arbeit dieser Programme
sehr gut und auch die Ergebnisse sind ansprech
end,
5  -- aber sie widersprechen meinem Anspruch
an die leichte Verständlichkeit der Befehle.
6  -- Zwei Beispiele:
7  -- * Es werden keine Einrückungen eing
efügt, die den inhaltlichen Zusammenhang
8  -- zwischen dem Befehl und seinen Paramet
ern abbilden:
9  -- SELECT blz,
10 -- bankname
11 -- FROM bank
12 -- ORDER BY blz DESC
13 -- LIMIT 1;
14 --
15 -- anstelle von
16 --
17 -- SELECT
18 -- blz, bankname
19 -- FROM bank
20 -- ORDER BY blz DESC
21 -- LIMIT 1
22 -- ;
23
24 -- * Hinter dem FROM wird ein Tabellennam
e gestellt, obwohl die Auswertung aus dem ganz
en JOIN erfolgt:
25 -- SELECT k.kunde_id,
26 -- k.nachname,
27 -- k.vorname,
28 -- b.datum
29 -- FROM bestellung b
```

```

30  -- INNER JOIN kunde k USING (kunde_id)
31  -- ORDER BY k.nachname,
32  -- k.vorname;
33
34  -- anstelle von
35
36  -- SELECT
37  -- k.kunde_id, k.nachname, k.vorname, b.d
atum
38  -- FROM
39  -- bestellung b INNER JOIN kunde k USING
(kunde_id)
40  -- ORDER BY
41  -- k.nachname, k.vorname
42  -- ;
43
44  -- Ich werde daher meine Darstellung beib
ehalten.
45
46
47  -- Anlegen der Datenbank auf der grünen W
iese
48  SELECT 'Start listing01. sql //////////////////
////////////////////////////////////////';
49  SELECT 'Löschen einer ggf. vorhandenen Da
tenbank oshop (Abschnitt 5.2.3)';
50  USE master;
51  GO
52  DROP DATABASE IF EXISTS oshop;
53  GO
54
55  SELECT 'Ermitteln aller Collations und da
mit auch alle Zeichenkodierungen (
Abschnitt 5.2.4)';
56  -- SELECT name FROM fn_helpcollations();
57  -- SELECT name FROM sys.fn_helpcollations
() WHERE name LIKE '%latin%';
58  -- SELECT name FROM sys.fn_helpcollations

```

```

() WHERE name LIKE '%german%';
59  -- GO
60
61  SELECT 'Anlegen der Datenbank mit Zeichen
satz und Sortierung (Abschnitt 5.2.4)';
62  -- Phonebook: Sortierung nach DIN 5007-2
63  -- CI: case insensitive, ohne Berücksicht
igung der Groß- und Kleinschreibung
64  -- AI: accent insensitive, ohne Berücksic
htigung von Akzenten
65  -- (KS): kanatype sensitive: Es werden ve
rschiedene japanische Schriftzeichen berü cksi
chtigt.
66  -- (WS): width sensitive: Die Kodierungs
länge wird berücksichtigt
67  CREATE DATABASE oshop
68  COLLATE German_PhoneBook_CI_AI
69  ;
70  GO
71
72  SELECT 'Anzeigen aller Datenbanken (Absch
nitt 5.2.5)';
73  SELECT
74  LEFT(name, 10) AS dbname, collation_name
75  FROM sys.databases
76  ;
77  GO
78
79  SELECT 'Ende listing01. sql //////////////////////////////////
////////////////////////////////////////';
80  GO

```

Listing 28.42 mssql/listing02.sql

```

1  -- Ausführen der vorherigen Befehle
2  :r listing01.sql
3
4  -- Hinweise: T-SQL kennt nicht:

```

```

5  -- UNSIGNED
6  -- ENUM
7  SELECT 'Start listing02. sql //////////////////////////////////
////////////////////////////////////////';
8  GO
9  -- Erspare mir ein wenig Tipparbeit
10 USE oshop; -- Abschnitt 5.3.2
11 GO
12 -- adresse
13 SELECT 'Tabelle adresse anlegen (Abschnitt 5.3.5)';
14 GO
15 CREATE TABLE adresse (
16     adresse_id INT IDENTITY (1,1),
17     strasse NVARCHAR(255) NOT NULL DEFAULT ''
18 ,
19     hnr NVARCHAR(255) NOT NULL DEFAULT '',
20     lkz NCHAR(2) NOT NULL DEFAULT '',
21     plz NCHAR(9) NOT NULL DEFAULT '',
22     ort NVARCHAR(255) NOT NULL DEFAULT '',
23     deleted TINYINT NOT NULL DEFAULT 0,
24     PRIMARY KEY(adresse_id)
25 );
26 -- kunde
27 SELECT 'Tabelle kunde mit Fremdschlüssel
anlegen (Abschnitt 5.3.6)';
28 GO
29 CREATE TABLE kunde (
30     kunde_id INT IDENTITY (1,1),
31     nachname NVARCHAR(255) NOT NULL DEFAULT '
32 ,
33     vorname NVARCHAR(255) NOT NULL DEFAULT ''
34 ,
35     rechnung_adresse_id INT,
36     liefer_adresse_id INT,
37     bezahlart INT NOT NULL DEFAULT 0,
38     art INT NOT NULL DEFAULT 0,

```

```

37  deleted TINYINT NOT NULL DEFAULT 0,
38  PRIMARY KEY(kunde_id),
39  FOREIGN KEY(rechnung_adresse_id)
40  REFERENCES adresse(adresse_id)
41  ON UPDATE CASCADE
42  ON DELETE NO ACTION,
43  FOREIGN KEY(liefer_adresse_id)
44  REFERENCES adresse(adresse_id)
45  ON UPDATE NO ACTION
46  ON DELETE NO ACTION
47  );
48  GO
49
50  -- bank
51  SELECT 'Tabelle bank anlegen (Abschnitt 5
.3.2)';
52  GO
53
54  CREATE TABLE bank (
55  bank_id NCHAR(12),
56  bankname NVARCHAR(255) NOT NULL DEFAULT '
',
57  lkz NCHAR(2) NOT NULL DEFAULT 'DE',
58  deleted TINYINT NOT NULL DEFAULT 0,
59  PRIMARY KEY(bank_id)
60  );
61  GO
62
63  -- bankverbindung
64  SELECT 'Tabelle bankverbindung mit Fremds
chluessel anlegen (Abschnitt 5.3.2)';
65  CREATE TABLE bankverbindung (
66  kunde_id INT,
67  bankverbindung_nr INT,
68  bank_id NCHAR(12) NOT NULL,
69  kontonummer NCHAR(25) NOT NULL DEFAULT ''
',
70  iban NCHAR(34) NOT NULL DEFAULT '',

```



```

71 deleted TINYINT NOT NULL DEFAULT 0,
72 PRIMARY KEY(kunde_id,bankverbindung_nr),
73 FOREIGN KEY(kunde_id)
74 REFERENCES kunde(kunde_id)
75 ON UPDATE NO ACTION
76 ON DELETE NO ACTION,
77 FOREIGN KEY(bank_id)
78 REFERENCES bank(bank_id)
79 ON UPDATE NO ACTION
80 ON DELETE NO ACTION
81 );
82 GO
83
84 -- artikel
85 SELECT 'Tabelle artikel anlegen';
86 CREATE TABLE artikel (
87 artikel_id INT IDENTITY (1,1),
88 bezeichnung NVARCHAR(255) NOT NULL DEFAULT
T '',
89 einzelpreis DECIMAL(14,6) NOT NULL DEFAULT
T 0.0, -- Wegen des SQL-Kompatibilitaet wird n
icht MONEY verwendet.
90 waehrung NCHAR(3) DEFAULT 'EUR',
91 deleted TINYINT NOT NULL DEFAULT 0,
92 PRIMARY KEY(artikel_id)
93 );
94 GO
95
96 -- warengruppe
97 SELECT 'Tabelle warengruppe anlegen';
98 CREATE TABLE warengruppe (
99 warengruppe_id INT IDENTITY (1,1),
100 bezeichnung NVARCHAR(255) NOT NULL DEFAULT
T '',
101 deleted TINYINT NOT NULL DEFAULT 0,
102 PRIMARY KEY(warengruppe_id)
103 );
104 GO

```

```

105
106 -- lieferant
107 SELECT 'Tabelle lieferant mit Fremdschlue
ssel anlegen';
108 CREATE TABLE lieferant (
109 lieferant_id INT IDENTITY (1,1),
110 firmenname NVARCHAR(255) NOT NULL DEFAULT
'',
111 adresse_id INT NOT NULL DEFAULT 0,
112 deleted TINYINT NOT NULL DEFAULT 0,
113 PRIMARY KEY(lieferant_id),
114 FOREIGN KEY(adresse_id)
115 REFERENCES adresse(adresse_id)
116 ON UPDATE NO ACTION
117 ON DELETE NO ACTION
118 );
119 GO
120
121 -- Hilfstabelle artikel zu warengruppe
122 SELECT 'Hilfstabelle artikel_nm_warengrup
pe anlegen';
123 CREATE TABLE artikel_nm_warengruppe (
124 warengruppe_id INT,
125 artikel_id INT,
126 PRIMARY KEY(warengruppe_id, artikel_id),
127 FOREIGN KEY(warengruppe_id)
128 REFERENCES warengruppe(warengruppe_id)
129 ON UPDATE NO ACTION
130 ON DELETE NO ACTION,
131 FOREIGN KEY(artikel_id)
132 REFERENCES artikel(artikel_id)
133 ON UPDATE NO ACTION
134 ON DELETE NO ACTION
135 );
136 GO
137
138 -- Hilfstabelle artikel zu lieferant
139 SELECT 'Hilfstabelle artikel_nm_lieferant

```

```

    anlegen';
140 CREATE TABLE artikel_nm_lieferant (
141     lieferant_id INT,
142     artikel_id INT,
143     PRIMARY KEY(lieferant_id, artikel_id),
144     FOREIGN KEY(lieferant_id)
145     REFERENCES lieferant(lieferant_id)
146     ON UPDATE NO ACTION
147     ON DELETE NO ACTION,
148     FOREIGN KEY(artikel_id)
149     REFERENCES artikel(artikel_id)
150     ON UPDATE NO ACTION
151     ON DELETE NO ACTION
152 );
153 GO
154
155 -- bestellung
156 SELECT 'Tabelle bestellung mit Fremdschlüssel anlegen';
157 CREATE TABLE bestellung (
158     bestellung_id INT IDENTITY (1,1),
159     kunde_id INT NOT NULL DEFAULT 0,
160     adresse_id INT NOT NULL DEFAULT 0,
161     datum DATETIME NOT NULL DEFAULT 0,
162     status INT NOT NULL DEFAULT 0,
163     deleted TINYINT NOT NULL DEFAULT 0,
164     PRIMARY KEY(bestellung_id),
165     FOREIGN KEY(kunde_id)
166     REFERENCES kunde(kunde_id)
167     ON UPDATE NO ACTION
168     ON DELETE NO ACTION,
169     FOREIGN KEY(adresse_id)
170     REFERENCES adresse(adresse_id)
171     ON UPDATE NO ACTION
172     ON DELETE NO ACTION
173 );
174 GO
175

```

```

176 -- position der bestellung
177 SELECT 'Tabelle bestellung_position mit F
remdschluessel anlegen';
178 CREATE TABLE bestellung_position (
179 bestellung_id INT,
180 position_nr INT,
181 artikel_id INT NOT NULL DEFAULT 0,
182 menge DECIMAL(14,6) NOT NULL DEFAULT 0.0,
183 deleted TINYINT NOT NULL DEFAULT 0,
184 PRIMARY KEY(bestellung_id, position_nr),
185 FOREIGN KEY(bestellung_id)
186 REFERENCES bestellung(bestellung_id)
187 ON UPDATE NO ACTION
188 ON DELETE NO ACTION,
189 FOREIGN KEY(artikel_id)
190 REFERENCES artikel(artikel_id)
191 ON UPDATE NO ACTION
192 ON DELETE NO ACTION
193 );
194 GO
195
196 -- rechnung
197 SELECT 'Tabelle rechnung mit Fremdschlues
sel anlegen';
198 CREATE TABLE rechnung (
199 rechnung_id INT IDENTITY (1,1),
200 kunde_id INT NOT NULL DEFAULT 0,
201 bestellung_id INT NOT NULL DEFAULT 0,
202 adresse_id INT NOT NULL DEFAULT 0,
203 bezahlart INT NOT NULL DEFAULT 0,
204 datum DATETIME NOT NULL DEFAULT 0,
205 status INT NOT NULL DEFAULT 0,
206 rabatt DECIMAL(6,2) NOT NULL DEFAULT 0.0,
207 skonto DECIMAL(6,2) NOT NULL DEFAULT 0.0,
208 deleted TINYINT NOT NULL DEFAULT 0,
209 PRIMARY KEY(rechnung_id),
210 FOREIGN KEY(kunde_id)
211 REFERENCES kunde(kunde_id)

```

```

212 ON UPDATE NO ACTION
213 ON DELETE NO ACTION,
214 FOREIGN KEY(adresse_id)
215 REFERENCES adresse(adresse_id)
216 ON UPDATE NO ACTION
217 ON DELETE NO ACTION
218 );
219 GO
220
221 -- position der rechnung
222 SELECT 'Tabelle rechnung_position mit Fremdschlüssel anlegen';
223 CREATE TABLE rechnung_position (
224 rechnung_id INT,
225 position_nr INT,
226 artikel_id INT NOT NULL DEFAULT 0,
227 steuer DECIMAL(14,6) NOT NULL DEFAULT 19.0,
228 menge DECIMAL(14,6) NOT NULL DEFAULT 0.0,
229 deleted TINYINT NOT NULL DEFAULT 0,
230 PRIMARY KEY(rechnung_id, position_nr),
231 FOREIGN KEY(rechnung_id)
232 REFERENCES rechnung(rechnung_id)
233 ON UPDATE NO ACTION
234 ON DELETE NO ACTION,
235 FOREIGN KEY(artikel_id)
236 REFERENCES artikel(artikel_id)
237 ON UPDATE NO ACTION
238 ON DELETE NO ACTION
239 );
240 GO
241
242 -- Zeige mir das Ergebnis
243 -- SHOW TABLES;
244 SELECT LEFT(name, 20) AS 'tablename', crdate FROM oshop.dbo.sysobjects WHERE xtype='U ' ORDER BY tablename
245 GO

```

```

246  SELECT 'Ende listing02. sql //////////////////////////////////
/////////////////////////////////////////';
247  GO

```

Listing 28.43 mssql/listing37.sql

```

1  -- Ausführen der vorherigen Befehle
2  :r listing02.sql
3
4  SELECT 'Start listing03. sql //////////////////////////////////
/////////////////////////////////////////';
5  GO
6
7  -- Abschnitt 6.1.1
8  SELECT
9  LEFT(name, 30) AS index_name,
10 LEFT(type_desc, 15) As index_type,
11 is_unique,
12 LEFT(OBJECT_NAME(object_id), 30) As table
_name
13 FROM
14 sys.indexes
15 WHERE
16 is_hypothetical = 0 AND
17 index_id != 0 AND
18 object_id = OBJECT_ID('kunde');
19 GO
20
21 -- Anlegen der Indizes nach Liste auf Abschnitt 6.1.2
22 SELECT 'CREATE INDEX idx_kunde_nachname_v
orname';
23 GO
24 CREATE INDEX idx_kunde_nachname_vorname
25 ON kunde (nachname, vorname)
26 ;
27 GO
28

```

```
29  SELECT 'CREATE INDEX idx_bank_bankname';
30  GO
31  CREATE INDEX idx_bank_bankname
32  ON bank (bankname)
33  ;
34  GO
35
36  SELECT 'CREATE INDEX idx_bankverbindung_b
ankid_kontonummer';
37  GO
38  CREATE INDEX idx_bankverbindung_bankid_ko
ntonummer
39  ON bankverbindung (bank_id, kontonummer)
40  ;
41  GO
42
43  SELECT 'CREATE UNIQUE INDEX idx_bankverbi
ndung_iban';
44  GO
45  CREATE UNIQUE INDEX idx_bankverbindung_ib
an
46  ON bankverbindung (iban)
47  ;
48  GO
49
50  SELECT 'CREATE INDEX idx_artikel_bezeichn
ung';
51  GO
52  CREATE INDEX idx_artikel_bezeichnung
53  ON artikel (bezeichnung)
54  ;
55  GO
56
57  SELECT 'CREATE INDEX idx_warengruppe_beze
ichnung';
58  GO
59  CREATE INDEX idx_warengruppe_bezeichnung
60  ON warengruppe (bezeichnung)
```

```
61  ;
62  GO
63
64  SELECT 'CREATE INDEX idx_lieferant_firmen
name';
65  GO
66  CREATE INDEX idx_lieferant_firmenname
67  ON lieferant (firmenname)
68  ;
69  GO
70
71  SELECT 'CREATE INDEX idx_bestellung_kunde
id_datum';
72  GO
73  CREATE INDEX idx_bestellung_kundeid_datum
74  ON bestellung (kunde_id, datum)
75  ;
76  GO
77
78
79  SELECT 'CREATE UNIQUE INDEX idx_adresse_d
ublette (Abschnitt 6.2.2)';
80  GO
81  CREATE UNIQUE INDEX idx_adresse_dublette
82  ON adresse (strasse, hnr, plz);
83  GO
84
85  -- Start Selektivitaet
86  SELECT 'Indexselektivitaet (Abschnitt 6.2
.3)';
87  SELECT 'CREATE TABLE #testdaten_bank';
88  GO
89
90  DROP TABLE IF EXISTS #testdaten_bank;
91  CREATE TABLE #testdaten_bank
92  (
93  blz CHAR(8),
94  merkmal CHAR(1),
```



```

95  bezeichnung VARCHAR(255),
96  plz CHAR(5),
97  ort VARCHAR(255),
98  kurz VARCHAR(255),
99  pan CHAR(5),
100 bic VARCHAR(255),
101 sm CHAR(2),
102 ds CHAR(6),
103 kz CHAR(1),
104 blzl CHAR(1),
105 blzn CHAR(8)
106 );
107 GO
108
109 SELECT 'BULK INSERT';
110 GO
111
112 SELECT 'BULK findet unter Linux Pfad nicht: Bug #9380471 for Microsoft-internal reference.'
113 SELECT ' Ausführung unter Windows mit angepasstem Pfad klappt prima.'
114 GO
115 -- Hier Dateiname und Pfad einfügen
116 -- BULK INSERT #testdaten_bank
117 -- FROM '/home/[...]/src/mssql/blz_20120305.csv'
118 -- WITH (FIRSTROW = 2, FIELDTERMINATOR =
';', ROWTERMINATOR='0x0a', FORMAT='CSV') ;
119 -- GO
120
121 SELECT ' Workaround zum Bug';
122 :r testdaten_blz.sql
123 GO
124
125
126 -- Workaournd
127 -- TSQL kennt keine Einschränkung der Spal

```

```

tenlänge
128 CREATE INDEX idx_testdaten_bank_bezeichnu
ng
129 ON #testdaten_bank(bezeichnung)
130 ;
131 GO
132
133 SELECT
134 LEFT(name, 30) AS index_name,
135 LEFT(type_desc, 15) As index_type,
136 is_unique,
137 LEFT(OBJECT_NAME(object_id), 30) As table
_name
138 FROM
139 sys.indexes
140 WHERE
141 is_hypothetical = 0 AND
142 index_id != 0 AND
143 object_id = OBJECT_ID('#testdaten_bank');
144 GO
145
146 -- Ende Selektivitaet
147
148 SELECT 'Ende listing03. sql //////////////////////////////////
////////////////////////////////////////';
149 GO

```

Listing 28.44 mssql/listing04.sql

```

1 -- Ausführen der vorherigen Befehle
2 :r listing03.sql
3
4 SELECT 'Start listing04. sql //////////////////////////////////
////////////////////////////////////////';
5 GO
6
7 SELECT 'BULK INSERT (Abschnitt 7.1.2)';
8 GO

```

```

9
10  SELECT 'BULK findet unter Linux Pfad nicht: Bug #9380471 for Microsoft-internal reference.'
```

11 SELECT 'Ausführung unter Windows mit angepasstem Pfad klappt prima.'

```

12  GO
13  -- Hier Dateiname und Pfad einfügen
14  -- BULK INSERT artikel
15  -- FROM 'C:/sqls/artikel02.csv'
16  -- WITH (FIRSTROW = 2, FORMATFILE = 'C:/sqls/artikel.fmt', KEEPIDENTITY);
17  GO
18  SELECT 'Workaround zum Bug';
19  -- TRUNCATE TABLE artikel;
20  :r artikel02_insert.sql
21
22  SELECT * FROM artikel;
23  GO
24
25  -- Start INSERT
26
27  SELECT 'Anlegen mit INSERT INTO ... VALUES (Abschnitt 7.2.1)';
28  SELECT 'Warengruppen anlegen';
29  SELECT * FROM warengruppe;
30  GO
31  SET IDENTITY_INSERT warengruppe ON ;
32  INSERT INTO warengruppe (warengruppe_id, bezeichnung)
33  VALUES
34  (1, 'Bürobedarf')
35  ,(2, 'Pflanzen')
36  ,(3, 'Gartenbedarf')
37  ,(4, 'Werkzeug')
38  ;
39  SET IDENTITY_INSERT warengruppe OFF ;
40  SELECT * FROM warengruppe;
```

```
41  GO
42
43  -- T-SQL kennt die SET-Variante nicht.
44  SELECT 'Anlegen mit INSERT INTO ... SET (Abschnitt 7.2.2)';
45  SELECT 'Warengruppen verbinden';
46  GO
47
48  INSERT INTO artikel_nm_warengruppe(artike
l_id, warengruppe_id)
49  VALUES
50  (3001,1)
51  ,(3005,1)
52  ,(3006,1)
53  ,(3007,1)
54  ,(3010,1)
55  ,(7856,2)
56  ,(7856,3)
57  ,(7863,2)
58  ,(7863,3)
59  ,(9010,3)
60  ,(9010,4)
61  ,(9015,3)
62  ,(9015,4)
63  ;
64  GO
65  -- Ende INSERT
66
67  -- Start Kundendaten
68  SELECT 'Testdaten INSERT adresse';
69  GO
70  -- adresse
71  SET IDENTITY_INSERT adresse ON ;
72  GO
73  INSERT INTO adresse (adresse_id, strasse,
hnr, lkz, plz, ort)
74  VALUES
75  (1, 'Beutelhaldenweg', '5', 'AL', '67676'
```

```

, 'Hobbingen')
76  ,(2, 'Beutelhaldenweg', '1', 'AL', '67676
', 'Hobbingen')
77  ,(3, 'Auf der Feste', '1', 'GO', '54786',
'Minas Tirith')
78  ,(4, 'Letztes Haus', '4', 'ER', '87567',
'Bruchtal')
79  ,(5, 'Baradur', '1', 'MO', '62519', 'Lugb
urz')
80  ,(10, 'Hochstrasse', '4a', 'DE', '44879',
'Bochum')
81  ,(11, 'Industriegebiet', '8', 'DE', '44878
', 'Bochum')
82  ;
83  GO
84  SET IDENTITY_INSERT adresse OFF ;
85  GO
86
87  SELECT 'Testdaten INSERT kunde';
88  GO
89  -- kunde
90  SET IDENTITY_INSERT kunde ON ;
91  GO
92  INSERT INTO kunde (kunde_id, rechnung_adr
esse_id, nachname, vorname, art)
93  VALUES
94  (1, 1, 'Gamdschie', 'Samweis', 1)
95  ,(2, 2, 'Beutlin', 'Frodo', 1)
96  ,(3, 2, 'Beutlin', 'Bilbo', 1)
97  ,(4, 3, 'Telcontar', 'Elessar', 1)
98  ,(5, 4, 'Earendilionn', 'Elrond', 3)
99  ;
100 GO
101
102 SET IDENTITY_INSERT kunde OFF ;
103 GO
104
105 SELECT 'Lieferadresse bei zwei Adressen a

```

```

endern';
106  GO
107  UPDATE kunde SET liefer_adresse_id = 2 WH
ERE kunde_id = 1;
108  UPDATE kunde SET liefer_adresse_id = 4 WH
ERE kunde_id = 3;
109  GO
110  -- Ende Kundendaten
111
112  -- Start Aufbau von 2 Bestellungen
113  SELECT 'Aufbau von 2 Bestellungen: Bestel
ldaten';
114  GO
115
116  INSERT INTO bestellung (kunde_id, adresse
_id, datum)
117  VALUES
118  (1, 1, CAST(N'2012-03-24T17:41:00' AS Dat
eTime))
119  ,(2, 2, CAST(N'2012-03-23T16:11:00' AS Da
teTime))
120  ;
121  GO
122
123  SELECT 'Aufbau von 2 Bestellungen: Bestel
lpositionen';
124  GO
125
126  INSERT INTO bestellung_position (bestellu
ng_id, position_nr, artikel_id, menge)
127  VALUES
128  (1, 1, 7856, 30)
129  ,(1, 2, 7863, 50)
130  ,(1, 3, 9015, 1)
131  ,(2, 1, 7856, 10)
132  ,(2, 2, 9010, 5)
133  ;
134  GO

```

```
135
136  -- Ende Aufbau Bestellung
137
138  -- SELECT 'Drei Fehlermeldung wegen Const
raints (Abschnitt 7.2.3)!';
139  SELECT bestellung_id, kunde_id, adresse_i
d, datum FROM bestellung;
140  GO
141  -- INSERT INTO bestellung (datum) VALUES
(NOW());
142  -- INSERT INTO bestellung (adresse_id, da
tum) VALUES (10, NOW());
143  -- INSERT INTO adresse (adresse_id, stras
se) VALUES (10, 'Oberer Weg');
144
145  -- Bildttabelle
146  SELECT 'Bildttabelle anlegen (Abschnitt 7.
2.4)';
147  GO
148  CREATE TABLE bild
149  (
150  bild_id INT IDENTITY (1,1) PRIMARY KEY,
151  bild VARBINARY(max),
152  dateiname NVARCHAR(255),
153  dateityp NVARCHAR(255),
154  dateigroesse BIGINT,
155  artikel_id INT REFERENCES artikel(artikel
_id)
156  );
157  GO
158
159  SELECT 'Daten kopieren (Abschnitt 7.3)';
160  GO
161
162  -- T-SQL kennt diese Art, eine Tabelle zu
erstellen nicht.
163  -- DROP TABLE IF EXISTS #kunde_beutlin;
164  -- CREATE TABLE #kunde_beutlin LIKE kunde
```

```

;
165 -- INSERT INTO #kunde_beutlin
166 -- SELECT * FROM kunde WHERE nachname='Be
utlin';
167 -- SELECT * FROM #kunde_beutlin;
168 -- GO
169
170 -- Das gleiche Ergebnis kann aber wie fol
gt erreicht werden.
171 SELECT *
172 INTO #kunde_beutlin
173 FROM kunde
174 WHERE nachname='Beutlin'
175 ;
176 SELECT * FROM #kunde_beutlin;
177 GO
178
179
180 SELECT 'Ende listing04. sql //////////////////
////////////////////////////////////////';
181 GO

```

28.3.2 Quelltexte zu Teil III

Listing 28.45 mssql/listing05.sql

```

1 -- Ausführen der vorherigen Befehle
2 :r listing04.sql
3
4 SELECT 'Start listing05. sql //////////////////
////////////////////////////////////////';
5 GO
6 SELECT 'Aktuelle Datenbankeinstellung (Ab
schnitt 8.1)';
7 GO
8 SELECT

```



```

 9  LEFT(name, 10) AS dbname, collation_name
10  FROM sys.databases
11  ;
12  GO
13
14  SELECT 'DATABASE löschen (Abschnitt 8.2)'
;
15  GO
16  CREATE DATABASE wurstbrot;
17  SELECT
18  LEFT(name, 10) AS dbname, collation_name
19  FROM sys.databases
20  ;
21  GO
22
23  DROP DATABASE wurstbrot;
24  SELECT
25  LEFT(name, 10) AS dbname, collation_name
26  FROM sys.databases
27  ;
28  GO
29
30  -- Start Tabellen umbenennen
31  SELECT 'Tabellen umbenennen (Abschnitt 8.3.1)';
32  USE oshop;
33  GO
34
35  DROP TABLE IF EXISTS wurstbrot;
36  SELECT LEFT(name, 20) AS 'tabname', crdate
FROM oshop.dbo.sysobjects WHERE xtype='U ' ORDER BY tabname
37  GO
38
39  EXEC sp_rename adresse, wurstbrot;
40  SELECT LEFT(name, 20) AS 'tabname', crdate
FROM oshop.dbo.sysobjects WHERE xtype='U ' ORDER BY tabname

```

```

41 GO
42
43 SELECT
44 ordinal_position,
45 LEFT(column_name, 20) AS 'column_name',
46 LEFT(data_type, 20) AS 'data_type',
47 character_maximum_length,
48 is_nullable
49 FROM information_schema.columns
50 WHERE table_name = 'kunde'
51 ;
52 GO
53
54 EXEC sp_rename wurstbrot, adresse;
55 GO
56 -- Ende Tabellen umbenennen
57
58 -- Start Spalten verändern
59 SELECT 'Spalte hinzufügen (Abschnitt 8.3.2)';
60 GO
61 SELECT
62 ordinal_position,
63 LEFT(column_name, 20) AS 'column_name',
64 LEFT(data_type, 20) AS 'data_type',
65 character_maximum_length,
66 is_nullable
67 FROM information_schema.columns
68 WHERE table_name = 'kunde'
69 ;
70 GO
71 ALTER TABLE kunde
72 ADD firmenname VARCHAR(255) NOT NULL DEFAULT ''
73 ;
74 GO
75 SELECT
76 ordinal_position,

```

```
77  LEFT(column_name, 20) AS 'column_name',
78  LEFT(data_type, 20) AS 'data_type',
79  character_maximum_length,
80  is_nullable
81  FROM information_schema.columns
82  WHERE table_name = 'kunde'
83  ;
84  GO
85
86  SELECT 'Spalte Datentyp ändern (Abschnitt 8.3.3)';
87  GO
88  SELECT 'Kein enum in TSQL'
89  GO
90  -- ALTER TABLE kunde
91  -- MODIFY
92  -- bezahlart ENUM('rechnung', 'bankeinzug', 'nachname') DEFAULT 'rechnung'
93  -- ;
94
95
96  SELECT 'Spaltenlänge ändern (Abschnitt 8.3.4)';
97  GO
98
99  DROP TABLE IF EXISTS #bla ;
100 GO
101 SELECT * INTO #bla FROM artikel;
102 -- ALTER TABLE #bla
103 -- ALTER COLUMN bezeichnung CHAR(10);
104 SELECT MAX(LEN(bezeichnung)) AS 'maxlen'
FROM artikel;
105 GO
106
107
108 SELECT 'Datum- und Zeitspalten ändern (Abschnitt 8.3.6)';
109 GO
```

```
110  -- Die Beispiele lassen sich so nicht in
111  TSQL nachbauen.
112  -- Das Abändern erfolgt über CAST- oder C
113  ONVERT-Funktionen
114  -- mit einem gleichzeitigen Neuaufbau der
115  Tabellen.
116
117  DROP TABLE IF EXISTS wurstbrot ;
118  GO
119  SELECT * INTO wurstbrot FROM adresse;
120  GO
121  SELECT
122  ordinal_position,
123  LEFT(column_name, 20) AS 'column_name',
124  LEFT(data_type, 20) AS 'data_type',
125  character_maximum_length,
126  is_nullable
127  FROM information_schema.columns
128  WHERE table_name = 'wurstbrot'
129  ;
130  GO
131  SELECT 'Spalten löschen (Abschnitt 8.3.9)
132  ' ;
133  ALTER TABLE wurstbrot
134  DROP COLUMN deleted
135  ;
136  ALTER TABLE wurstbrot
137  DROP COLUMN strasse
138  ;
139  ALTER TABLE wurstbrot
140  DROP COLUMN ort
141  ;
142  SELECT
143  ordinal_position,
144  LEFT(column_name, 20) AS 'column_name',
```

```

144 LEFT(data_type, 20) AS 'data_type',
145 character_maximum_length,
146 is_nullable
147 FROM information_schema.columns
148 WHERE table_name = 'wurstbrot'
149 ;
150 GO
151 SELECT 'Ende listing05. sql //////////////////////////////////
////////////////////////////////////////';
152 GO

```

Listing 28.46 mssql/listing06.sql

```

1  -- Ausführen der vorherigen Befehle
2  :r listing05.sql
3
4  SELECT 'Start listing06. sql //////////////////////////////////
////////////////////////////////////////';
5  GO
6  SELECT 'WHERE mit TRUE / FALSE (Abschnitt 9.1.1)';
7  SELECT 'TSQL kennt keine boolschen Konstanten';
8  -- SELECT * FROM artikel WHERE TRUE;
9  -- SELECT * FROM artikel WHERE FALSE;
10
11 -- Start Groß- Kleinschreibung
12 SELECT 'Gross- Kleinschreibung (Abschnitt 9.1.2)';
13 GO
14 DROP TABLE IF EXISTS #wurstbrot;
15 GO
16 CREATE TABLE #wurstbrot (
17 name NVARCHAR(15)
18 );
19 GO
20
21 INSERT INTO #wurstbrot

```

```

22  VALUES
23  ('Jaqueline')
24  ,('Kevin')
25  ,('kevin')
26  ,('jaqueline')
27
28  SELECT * FROM #wurstbrot WHERE name = 'ke
vin';
29  GO
30
31  ALTER TABLE #wurstbrot
32  ALTER COLUMN name NVARCHAR(15) COLLATE SQ
L_Latin1_General_CP1_CS_AS
33  ;
34  GO
35
36  SELECT * FROM #wurstbrot WHERE name = 'ke
vin';
37  GO
38  -- Ende Groß- Kleinschreibung
39
40  -- Start Wertberechnung
41  SELECT 'Wertberechnung (Abschnitt 9.2.2)'
Hinweis;
42  SELECT * FROM artikel;
43  GO
44  UPDATE artikel SET einzelpreis = einzelpr
eis + einzelpreis / 100.0;
45  SELECT * FROM artikel;
46  GO
47  UPDATE artikel SET einzelpreis = ROUND(ei
nzelpreis, 2);
48  SELECT * FROM artikel;
49  GO
50  -- Ende Wertberechnung
51
52  -- Start Gebastelte Zeichenketten
53  SELECT 'Gebastelte Zeichenketten: Anrede

```

```

(Abschnitt 9.2.3)' Hinweis;
54 GO
55 ALTER TABLE kunde ADD anrede NVARCHAR(255
);
56 GO
57 UPDATE kunde
58 SET anrede = CONCAT('Sehr geehrte/r Frau/
Herr ', nachname)
59 WHERE nachname <> ''
60 ;
61 SELECT * FROM kunde;
62 GO
63 -- Ende Gebastelte Zeichenketten
64
65 -- Start DELETE
66 SELECT 'Löschen mit DELETE (Abschnitt 9.3
)';
67 GO
68 SELECT * FROM bestellung_position;
69 GO
70 DELETE FROM bestellung_position WHERE bes
tellung_id = 1;
71 SELECT * FROM bestellung_position;
72 GO
73
74 -- Start Wiederherstellen der Daten
75 INSERT INTO bestellung_position (bestellu
ng_id, position_nr, artikel_id, menge)
76 VALUES
77 (1, 1, 7856, 30)
78 ,(1, 2, 7863, 50)
79 ,(1, 3, 9015, 1)
80 ;
81 GO
82 -- Ende Wiederherstellen der Daten
83
84 -- SELECT 'Löschversuch mit Constraint. E
rwarte eine Fehlermeldung! (

```

```

Abschnitt 9.3.1)';
85  -- DELETE FROM kunde WHERE kunde_id = 1;
86
87  SELECT 'Test IDENTITY(1,1) (Abschnitt 9.3
.2\)';
88  GO
89  SELECT kunde_id, nachname FROM kunde;
90  GO
91  DELETE FROM kunde WHERE kunde_id IN (3,5)
;
92  GO
93  SELECT kunde_id, nachname FROM kunde;
94  GO
95  INSERT INTO kunde (nachname, vorname) VAL
UES ('Eichenschild', 'Thorin');
96  SELECT kunde_id, nachname FROM kunde;
97  GO
98  -- Start Wiederherstellen der Daten
99  SET IDENTITY_INSERT kunde ON ;
100 INSERT INTO kunde (kunde_id, rechnung_adr
esse_id, nachname, vorname, art)
101 VALUES
102 (3, 2, 'Beutlin', 'Bilbo', 1)
103 ,(5, 4, 'Earendilinn', 'Elrond', 3)
104 ;
105 SET IDENTITY_INSERT kunde OFF ;
106 SELECT kunde_id, nachname FROM kunde;
107 GO
108
109 SELECT 'Tabelle leeren (Abschnitt 9.3.4)'
Hinweis;
110 GO
111 -- DELETE FROM bestellung_position;
112 -- GO
113 -- TRUNCATE bestellung_position;
114 -- GO
115 -- Ende DELETE
116 SELECT 'Ende listing06. sql //////////////////////////////////'

```



```
/////////////////////////////////////////';  
117  GO
```

28.3.3 Quelltexte zu Teil IV

Listing 28.47 mssql/listing07.sql

```
1  -- Ausführen der vorherigen Befehle  
2  :r listing06.sql  
3  
4  SELECT 'Start listing07. sql //////////////////////////////////  
/////////////////////////////////////////';  
5  GO  
6  SELECT 'Ausdrücke (Abschnitt 10.1.1)';  
7  GO  
8  
9  SELECT 5, 'Beginn der Auswertung';  
10 GO  
11 SELECT -5, 9 + 4, 9 - 4, 9 * 4, 9 / 4, 9.  
0 / 4.0, 9 % 4;  
12 GO  
13 SELECT 3 * 4 / 3, (3 * 4) / 3, 3 * (4 / 3  
)  
14 GO  
15 SELECT 9 * NULL;  
16 GO  
17  
18 SELECT 'Zufallszahlen (Abschnitt 10.1.3)'  
;  
19 GO  
20 SELECT RAND(1);  
21 GO  
22 SELECT RAND(1);  
23 GO  
24 SELECT RAND(CONVERT(VARBINARY, NEWID()))  
FROM artikel;
```

```

25  GO
26
27  SELECT 'Variablen (Abschnitt 10.1.4)';
28  GO
29  DECLARE @x INT
30  SELECT POWER(2,8) INTO @x;
31  SELECT @x - 1;
32  GO
33
34  DECLARE @artnr INT
35  SELECT @artnr=3010;
36  SELECT * FROM artikel WHERE artikel_id=@a
rtnr;
37  GO
38
39  SELECT 'Spalten- und Zeilenwahl (Abschnitt 10.2)';
40  GO
41
42  SELECT * FROM artikel;
43  GO
44
45  SELECT
46  artikel_id, bezeichnung, einzelpreis, wae
hrung, deleted
47  FROM artikel
48  ;
49  GO
50
51  SELECT
52  artikel_id + 10000 AS Unsinn, deleted AS
Löschkennzeichen
53  FROM artikel
54  ;
55  GO
56
57  SELECT
58  artikel_id, bezeichnung

```

```
59  FROM artikel
60  WHERE einzelpreis BETWEEN 10 AND 100
61  ;
62  GO
63
64  SELECT 'Sortierung (Abschnitt 10.3)';
65  GO
66  SELECT
67  artikel_id, bezeichnung, einzelpreis
68  FROM artikel
69  ORDER BY einzelpreis ASC
70  ;
71  GO
72
73  SELECT
74  nachname, vorname
75  FROM kunde
76  ORDER BY nachname, vorname
77  ;
78  GO
79
80  SELECT
81  nachname, vorname
82  FROM kunde
83  ORDER BY nachname DESC, vorname ASC
84  ;
85  GO
86
87  SELECT 'TSQL liefert in diesem Fall die g
leiche Sortierung' Hinweis;
88  GO
89  DROP TABLE IF EXISTS #wurstbrot;
90  CREATE TABLE #wurstbrot ( name VARCHAR(25
5) COLLATE SQL_AltDiction_CP850_CI_AI);
91  INSERT INTO #wurstbrot
92  VALUES
93  ('winfried')
94  ,('achim')
```

```

95      ,('olga')
96      ,('zechine')
97      ,('ägidius')
98      ;
99      SELECT * FROM #wurstbrot ORDER BY name;
100     GO
101
102     DROP TABLE IF EXISTS #wurstbrot;
103     CREATE TABLE #wurstbrot ( name VARCHAR(25
104 5) COLLATE SQL_Latin1_General_CP1_CS_AS);
105     INSERT INTO #wurstbrot
106     VALUES
107     ('winfried')
108     ,('achim')
109     ,('olga')
110     ,('zechine')
111     ,('ägidius')
112     ;
113     SELECT * FROM #wurstbrot ORDER BY name;
114     GO
115
116     DROP TABLE IF EXISTS #wurstbrot;
117     GO
118     CREATE TABLE #wurstbrot (name VARCHAR(255
119 ) COLLATE SQL_Latin1_General_CP1_CI_AS);
120     INSERT INTO #wurstbrot
121     VALUES
122     ('winfried')
123     ,('achim')
124     ,('olga')
125     ,('Olga')
126     ,('zechine')
127     ;
128     SELECT * FROM #wurstbrot ORDER BY name;
129     SELECT * FROM #wurstbrot ORDER BY name DE
130 SC;
131     GO
132
133

```

```
130 DROP TABLE IF EXISTS #wurstbrot;
131 GO
132 CREATE TABLE #wurstbrot (name VARCHAR(255
) COLLATE SQL_Latin1_General_CP1_CS_AS);
133 INSERT INTO #wurstbrot
134 VALUES
135 ('winfried')
136 ,('achim')
137 ,('olga')
138 ,('Olga')
139 ,('zechine')
140 ;
141 SELECT * FROM #wurstbrot ORDER BY name;
142 SELECT * FROM #wurstbrot ORDER BY name DE
SC;
143 GO
144
145 DROP TABLE IF EXISTS #wurstbrot;
146 GO
147
148 SELECT
149 bestellung_id, datum
150 FROM bestellung
151 ORDER BY datum;
152 GO
153
154 SELECT
155 bestellung_id, CONVERT(TIME, datum) Uhrze
it
156 FROM bestellung
157 ORDER BY uhrzeit;
158 GO
159
160 SET SHOWPLAN_ALL ON
161 GO
162 SELECT * FROM kunde ORDER BY nachname, vo
rname;
163 GO
```

```
164 SET SHOWPLAN_ALL OFF
165 GO
166
167 SELECT 'Mehrfachausgaben unterbinden (Abschnitt 10.4)';
168 GO
169
170 SELECT ort FROM adresse ORDER BY ort;
171 SELECT DISTINCT ort FROM adresse ORDER BY
    ort;
172 SELECT DISTINCT nachname FROM kunde ORDER
    BY nachname;
173 GO
174
175
176 -- Das Fallbeispiel Bankimport müsste in
    TSQL ganz anders als
177 -- in MySQL oder MariaDB gelöst werden.
178 -- Daher hier nur der INSERT, damit die B
    ankdaten vorliegen.
179 SELECT 'Bankdaten importieren (Abschnitt
    10.4.1)';
180 GO
181 ALTER TABLE bank ADD blz NCHAR(12) NOT NU
    LL DEFAULT '';
182 GO
183 :r blz_20120305.sql
184 GO
185
186 CREATE INDEX idx_bank_blzbankname
187 ON bank (blz, bankname)
188 ;
189 GO
190
191 -- Beginn Für die vorhandenen Kunden eine
    Bankverbindung bauen
192
193 INSERT INTO bankverbindung (kunde_id, ban
```

```

kverbindung_nr, bank_id, kontonummer, iban )
194  VALUES
195  (1, 1, 2, '1111111111', '1001001011111111
11'),
196  (1, 2, 2, '1111111112', '1001001011111111
12'),
197  (2, 1, 35, '2222222221', '10060198222222
221'),
198  (3, 1, 35, '3333333331', '10060198333333
331'),
199  (4, 1, 90, '4444444441', '12070000444444
441'),
200  (5, 1, 90, '5555555551', '12070000555555
551')
201  ;
202  GO
203  -- Ende Für die vorhandenen Kunden eine B
ankverbindung bauen
204
205  -- Ende Bankimport
206
207  SELECT 'Ausschneiden mit TOP (Abschnitt 1
0.5)';
208  GO
209  -- SELECT * FROM bank;
210  SELECT TOP(3) blz, bankname FROM bank;
211  GO
212  SELECT TOP(1)
213  blz, bankname
214  FROM bank
215  ORDER BY blz DESC
216  ;
217  GO
218
219  DECLARE @blzmin BIGINT;
220  SELECT TOP(1) @blzmin =
221  blz
222  FROM bank

```

```

223 ORDER BY blz DESC
224 ;
225 SELECT
226 bankname
227 FROM bank
228 WHERE blz = @blzmin
229 ;
230 GO
231
232 SELECT blz, bankname FROM bank ORDER BY b
lz OFFSET 1 ROWS FETCH NEXT 2 ROWS ONLY;
233 SELECT TOP(1) blz, bankname FROM bank ORD
ER BY blz ;
234 SELECT blz, bankname FROM bank ORDER BY b
lz OFFSET 0 ROWS FETCH NEXT 1 ROWS ONLY;
235 GO
236
237 SELECT 'Ende listing07. sql //////////////////
////////////////////////////////////////';
238 GO

```

Listing 28.48 mssql/listing08.sql

```

1  -- Ausführen der vorherigen Befehle
2  :r listing07.sql
3
4  SELECT 'Start listing08. sql //////////////////
////////////////////////////////////////';
5  GO
6
7  -- Start INNER JOIN
8  SELECT 'CROSS JOIN (Abschnitt 11.1)';
9  GO
10 SELECT
11 kunde_id, nachname, vorname, rechnung_adr
esse_id, adresse_id
12 FROM
13 kunde, adresse

```



```

14  ;
15  GO
16
17  SELECT 'CROSS JOIN mit WHERE (Abschnitt 1
1.1)';
18  GO
19  SELECT
20  kunde_id, nachname, vorname, rechnung_adr
esse_id, adresse_id
21  FROM
22  kunde, adresse
23  WHERE
24  rechnung_adresse_id = adresse_id
25  ;
26  GO
27
28  SELECT 'INNER JOIN (Abschnitt 11.2)';
29  GO
30  SELECT
31  kunde_id, nachname, vorname, rechnung_adr
esse_id, adresse_id
32  FROM
33  kunde INNER JOIN adresse
34  ON rechnung_adresse_id = adresse_id
35  ;
36  GO
37
38  SELECT
39  kunde_id, kontonummer, blz, bankname
40  FROM
41  bankverbindung INNER JOIN bank
42  ON bankverbindung.bank_id = bank.bank_id
43  ;
44  GO
45  SELECT
46  nachname, vorname, kontonummer
47  FROM
48  bankverbindung INNER JOIN kunde ON bankve

```

```

rbindung.kunde_id = kunde.kunde_id
49  ;
50  GO
51
52  SELECT 'Abkürzung mit USING (Abschnitt 11.2.2)';
53  SELECT 'TSQL kennt keine Abkürzung mit USING oder eine vergleichbare';
54  GO
55
56  SELECT 'Abkürzung mit NATURAL JOIN (Abschnitt 11.2.2)';
57  SELECT 'TSQL kennt keine NATURAL JOIN';
58  GO
59  -- Ende INNER JOIN
60
61  -- Start Aufbau Lieferanten
62  SELECT 'Aufbau der Lieferanten';
63  GO
64  INSERT INTO lieferant (firmenname, adresse_id)
65  VALUES
66  ('Gartenbedarf AllesGrün', 10)
67  ,('Office International', 11)
68  ,('Bürohengst GmbH', 11)
69  ;
70
71  INSERT INTO artikel_nm_lieferant (lieferant_id, artikel_id)
72  VALUES
73  (1, 7856)
74  ,(1, 7863)
75  ,(1, 9010)
76  ,(1, 9015)
77  ,(3, 3001)
78  ,(3, 3005)
79  ,(3, 3006)
80  ,(3, 3007)

```

```

81      ,(3, 3010)
82      ;
83      GO
84      -- Ende Aufbau Lieferanten
85
86      -- Start OUTER JOIN
87      SELECT 'OUTER JOIN (Abschnitt 11.4)';
88      GO
89      SELECT
90      k.kunde_id, k.nachname, k.vorname, b.datu
m
91      FROM
92      bestellung b INNER JOIN kunde k ON b.kund
e_id = k.kunde_id
93      ORDER BY
94      k.nachname, k.vorname
95      ;
96      GO
97      SELECT
98      k.kunde_id, k.nachname, k.vorname, b.datu
m
99      FROM
100     bestellung b RIGHT OUTER JOIN kunde k ON
b.kunde_id = k.kunde_id
101     ORDER BY
102     k.nachname, k.vorname
103     ;
104     GO
105     SELECT
106     k.kunde_id, k.nachname, k.vorname, b.datu
m
107     FROM
108     kunde k LEFT OUTER JOIN bestellung b ON k
.kunde_id = b.kunde_id
109     ORDER BY
110     k.nachname, k.vorname
111     ;
112     GO

```

```

113  SELECT
114  l.lieferant_id, l.firmenname, a.artikel_i
d, a.bezeichnung
115  FROM
116  artikel_nm_lieferant nm INNER JOIN artike
l a ON nm.artikel_id = a.artikel_id
117  INNER JOIN lieferant l ON nm.lieferant_id
= l.lieferant_id
118  ORDER BY firmenname
119  ;
120  GO
121  SELECT
122  l.lieferant_id, l.firmenname, a.artikel_i
d, a.bezeichnung
123  FROM
124  artikel_nm_lieferant nm INNER JOIN artike
l a ON nm.artikel_id = a.artikel_id
125  RIGHT JOIN lieferant l ON nm.lieferant_id
= l.lieferant_id
126  ORDER BY firmenname
127  ;
128  GO
129  SELECT
130  l.firmenname
131  FROM
132  artikel_nm_lieferant nm INNER JOIN artike
l a ON nm.artikel_id = a.artikel_id
133  RIGHT JOIN lieferant l ON nm.lieferant_id
= l.lieferant_id
134  WHERE a.artikel_id IS NULL
135  ORDER BY firmenname
136  ;
137  GO
138  SELECT k.kunde_id, k.nachname, k.vorname
139  FROM
140  bestellung b RIGHT JOIN kunde k ON k.kund
e_id = b.kunde_id
141  WHERE b.bestellung_id IS NULL

```

```
142 ;
143 GO
144 -- Ende OUTER JOIN
145
146 -- Start Aufbau Forum
147 SELECT 'SELF JOIN (Abschnitt 11.5)';
148 GO
149 DROP TABLE IF EXISTS beitrag;
150 DROP TABLE IF EXISTS account;
151 GO
152 SELECT 'CREATE account';
153 GO
154 CREATE TABLE account (
155     kunde_id INT NOT NULL DEFAULT 0,
156     name NVARCHAR(255) NOT NULL DEFAULT '',
157     passwort NCHAR(34) NOT NULL DEFAULT '',
158     status INT NOT NULL DEFAULT '0',
159     kommentar TEXT ,
160     admin TINYINT NOT NULL DEFAULT 0,
161     deleted TINYINT NOT NULL DEFAULT 0,
162     PRIMARY KEY(kunde_id),
163     FOREIGN KEY (kunde_id)
164     REFERENCES kunde(kunde_id)
165     ON UPDATE NO ACTION
166     ON DELETE NO ACTION
167 );
168 GO
169
170 SELECT 'CREATE beitrag';
171 GO
172 CREATE TABLE beitrag (
173     beitrag_id INT NOT NULL IDENTITY(1,1),
174     account_id INT NOT NULL,
175     bezug_beitrag_id INT NOT NULL DEFAULT 1,
176     zeitstempel DATETIME NOT NULL DEFAULT '10
177 00-01-01 00:00:00',
177     nachricht TEXT ,
178     deleted TINYINT NOT NULL DEFAULT 0,
```

```

179 PRIMARY KEY(beitrag_id),
180 FOREIGN KEY (account_id)
181 REFERENCES account(kunde_id)
182 ON UPDATE NO ACTION
183 ON DELETE NO ACTION,
184 FOREIGN KEY (bezug_beitrag_id)
185 REFERENCES beitrags(beitrag_id)
186 ON UPDATE NO ACTION
187 ON DELETE NO ACTION
188 );
189 GO
190
191 SELECT 'account daten';
192 GO
193 INSERT INTO account (kunde_id, name, pass
wort, admin)
194 VALUES
195 (1, 'admin', CONVERT(NCHAR(34), HASHBYTES
('MD5', 'rosi'), 2), 1)
196 ,(2, 'frodo', CONVERT(NCHAR(34), HASHBYTE
S('MD5', 'elbereth'), 2), 0)
197 ,(3, 'bilbo', CONVERT(NCHAR(34), HASHBYTE
S('MD5', 'blitzerschlagen'), 2), 0)
198 ,(5, 'elle', CONVERT(NCHAR(34), HASHBYTES
('MD5', 'feanor'), 2), 1)
199 ;
200 GO
201
202
203 SELECT 'beitrag daten';
204 SET IDENTITY_INSERT beitrags ON ;
205 INSERT INTO beitrags (beitrag_id, account_
id, bezug_beitrag_id, zeitstempel, nachricht)
206 VALUES
207 (1, 1, 1, CONVERT(DATETIME, '1753-01-01 0
0:00:00'), '')
208 ,(2, 2, 1, CONVERT(DATETIME, '2011-05-01
14:13:00'), 'Der Lieferservice ist super .')

```

```

209  ,(3, 3, 2, CONVERT(DATETIME, '2011-05-02
11:45:00')), 'Das finde ich auch.')
```

```

210  ,(4, 5, 2, CONVERT(DATETIME, '2011-05-01
17:01:00')), 'Aber ein wenig langsam.')
```

```

211  ,(5, 2, 4, CONVERT(DATETIME, '2011-05-01
17:15:00')), 'Finde ich nicht.')
```

```

212  ,(6, 5, 1, CONVERT(DATETIME, '2011-06-12
09:07:00')), 'Angebot könnte besser sein .')
```

```

213  ;
```

```

214  SET IDENTITY_INSERT beitrags OFF ;
```

```

215  GO
```

```

216  -- Ende Aufbau Forum
```

```

217
```

```

218  -- Start SELF JOIN
```

```

219  SELECT 'SELF JOIN (Abschnitt 11.5)';
```

```

220  GO
```

```

221  SELECT
```

```

222  kunde_id, name, status, admin
```

```

223  FROM
```

```

224  account
```

```

225  ;
```

```

226
```

```

227  SELECT
```

```

228  beitrags_id, account_id, bezug_beitrags_id,
```

```

LEFT(CONVERT(NVARCHAR(255), nachricht), 20) k
```

```

urz
```

```

229  FROM
```

```

230  beitrags
```

```

231  ;
```

```

232  GO
```

```

233  -- SELECT 'Erwarte eine Fehlermeldung (Ab
```

```

schnitt 11.5)';
```

```

234  -- SELECT nachricht
```

```

235  -- FROM
```

```

236  -- beitrags INNER JOIN beitrags ON bezug_be
```

```

itrags_id = beitrags_id
```

```

237  -- WHERE beitrags_id = 2
```

```

238  -- ;
```

```
239  -- GO
240
241  SELECT ant.nachricht 'Antwort'
242  FROM
243  beitrags ant INNER JOIN beitrags orig ON ant
244  t.bezugsbeitrags_id = orig.beitrags_id
245  WHERE orig.beitrags_id = 2
246  ;
247  GO
248
249  SELECT 'CTE rekursiv (Abschnitt 11.5)';
250  GO
251  WITH ant_auf AS
252  (
253  -- nicht rekursiver Teil
254  SELECT * FROM beitrags WHERE bezugsbeitrags
255  _id = 2
256  UNION ALL
257  -- rekursiver Teil
258  SELECT ant.*
259  FROM
260  beitrags ant INNER JOIN ant_auf orig ON ant
261  t.bezugsbeitrags_id = orig.beitrags_id
262  )
263  SELECT beitrags_id, bezugsbeitrags_id, nach
264  richt FROM ant_auf;
265  GO
266  -- Ende SELF JOIN
267
268  -- Start Redundant
269  SELECT 'Beschleunigen mit Redundanzen (Ab
270  schnitt 11.6)';
271  GO
272  ALTER TABLE kunde
273  ADD r_strasse NVARCHAR(255)
274  ;
275  ALTER TABLE kunde
276  ADD r_ort NVARCHAR(255)
```



```

272 ;
273 ALTER TABLE kunde
274 ADD r_aktuell TINYINT NOT NULL DEFAULT 1
275 ;
276 ALTER TABLE kunde
277 ADD l_strasse NVARCHAR(255)
278 ;
279 ALTER TABLE kunde
280 ADD l_ort NVARCHAR(255)
281 ;
282 ALTER TABLE kunde
283 ADD l_aktuell TINYINT NOT NULL DEFAULT 1
284 ;
285 GO
286 UPDATE kunde
287 SET
288 r_strasse = CONCAT(strasse, ' ', hnr),
289 r_ort = CONCAT(lkz, '-', plz, ' ', ort),
290 r_aktuell = 1
291 FROM kunde INNER JOIN adresse ON rechnung
_adresse_id = adresse_id
292 ;
293 GO
294 SELECT
295 kunde_id, r_strasse, r_ort, r_aktuell
296 FROM kunde
297 ;
298 GO
299 -- Ende Redundant

```

Listing 28.49 mssql/listing09.sql

```

1 -- Ausführen der vorherigen Befehle
2 :r listing08.sql
3
4 SELECT 'Start listing09. sql //////////////////////////////////
////////////////////////////////////////';
5 GO

```

```

6  SELECT 'Aggregatfunktionen (Abschnitt 12.1)';
7  GO
8  SELECT AVG(einzelpreis) FROM artikel;
9  SELECT COUNT(*) FROM kunde;
10 SELECT COUNT(liefer_adresse_id) FROM kunde;
11 SELECT COUNT(DISTINCT(rechnung_adresse_id)) FROM kunde;
12 SELECT MAX(einzelpreis), MIN(einzelpreis) FROM artikel;
13 SELECT SUM(menge) FROM bestellung_position;
14 SELECT
15 bp.menge, a.einzelpreis
16 FROM
17 bestellung_position bp INNER JOIN artikel a ON bp.artikel_id = a.artikel_id
18 ;
19 SELECT
20 bp.menge * a.einzelpreis 'Positionswert'
21 FROM
22 bestellung_position bp INNER JOIN artikel a ON bp.artikel_id = a.artikel_id
23 ;
24 GO
25 SELECT
26 SUM(bp.menge * a.einzelpreis)
27 FROM
28 bestellung_position bp INNER JOIN artikel a ON bp.artikel_id = a.artikel_id
29 ;
30 GO
31 -- Start Aufbau Lagerverwaltung
32 SELECT 'Aufbau Lagerbestand (Abschnitt 12.1)';
33 SELECT 'CREATE lagerbestand';
34 GO

```

```

35 DROP TABLE IF EXISTS lagerbestand;
36 GO
37 CREATE TABLE lagerbestand (
38     artikel_id INT NOT NULL DEFAULT 0,
39     menge_mindest DECIMAL(14,6) NOT NULL DEFA
ULT 0.0,
40     menge_aktuell DECIMAL(14,6) NOT NULL DEFA
ULT 0.0,
41     deleted TINYINT NOT NULL DEFAULT 0,
42     PRIMARY KEY(article_id),
43     FOREIGN KEY (artikel_id)
44     REFERENCES artikel(article_id)
45     ON UPDATE NO ACTION
46     ON DELETE NO ACTION
47 );
48 GO
49 SELECT 'Lagerbestand festlegen';
50 GO
51 INSERT INTO lagerbestand (artikel_id, men
ge_aktuell, menge_mindest)
52 VALUES
53 (7856, 500, 100)
54 ,(7863, 400, 100)
55 ,(9010, 30, 10)
56 ,(9015, 25, 10)
57 ,(3001, 5000, 1000)
58 ,(3005, 250, 200)
59 ,(3006, 250, 200)
60 ,(3007, 149, 200)
61 ,(3010, 3, 100)
62 ;
63 GO
64 -- Ende Aufbau Lagerverwaltung
65
66 SELECT 'GROUP BY (Abschnitt 12.2)';
67 GO
68 -- Start Group by
69 SELECT

```

```
70  COUNT(*)
71  FROM
72  bestellung_position
73  ;
74  GO
75  SELECT
76  bestellung_id Bestellnummer, COUNT(*) 'An
zahl der Positionen'
77  FROM
78  bestellung_position
79  GROUP BY
80  bestellung_id
81  ;
82  GO
83  SELECT
84  a.bezeichnung, bp.menge
85  FROM
86  bestellung_position bp INNER JOIN artikel
a ON bp.artikel_id = a.artikel_id
87  ;
88  GO
89  SELECT
90  a.bezeichnung 'Artikelname', SUM(bp.menge
) 'Anzahl bestellter Artikel'
91  FROM
92  bestellung_position bp INNER JOIN artikel
a ON bp.artikel_id = a.artikel_id
93  GROUP BY
94  a.bezeichnung
95  ;
96  GO
97  SELECT
98  a.bezeichnung 'Artikelname', SUM(bp.menge
) 'Anzahl bestellter Artikel'
99  FROM
100 bestellung_position bp RIGHT JOIN artikel
a ON bp.artikel_id = a.artikel_id
101 GROUP BY
```

```

102 a.bezeichnung
103 ;
104 GO
105 SELECT
106 a.bezeichnung 'Artikelname',
107 CASE
108 WHEN SUM(bp.menge) IS NULL THEN 0
109 ELSE SUM(bp.menge)
110 END AS 'Anzahl bestellter Artikel'
111 FROM
112 bestellung_position bp RIGHT JOIN artikel
113 a ON bp.artikel_id = a.artikel_id
114 GROUP BY
115 a.bezeichnung
116 ;
117 GO
118 -- SELECT 'Eine Fehlermeldung! (Abschnitt 12.3)';
119 -- SELECT
120 -- a.bezeichnung 'Artikelname', SUM(bp.menge) 'Anzahl bestellter Artikel'
121 -- FROM
122 -- bestellung_position bp INNER JOIN artikel a USING (artikel_id)
123 -- WHERE
124 -- SUM(bp.menge) > 10
125 -- GROUP BY
126 -- artikel_id
127 -- ;
128 SELECT 'Gruppenergebnisse mit ROLLUP summieren (Abschnitt 12.2)';
129 GO
130 SELECT
131 a.artikel_id, SUM(bp.menge)
132 FROM
133 bestellung_position bp RIGHT JOIN artikel
134 a ON bp.artikel_id = a.artikel_id

```

```
134  GROUP BY
135  ROLLUP(a.artikel_id)
136  ;
137  GO
138
139  SELECT 'Gruppenergebnisse mit HAVING filt
ern (Abschnitt 12.3)';
140  GO
141  SELECT
142  a.bezeichnung 'Artikelname', SUM(bp.menge
) 'Anzahl bestellter Artikel'
143  FROM
144  bestellung_position bp INNER JOIN artikel
a ON bp.artikel_id = a.artikel_id
145  GROUP BY
146  a.bezeichnung
147  HAVING
148  SUM(bp.menge) > 10
149  ;
150  GO
151  SELECT
152  a.bezeichnung 'Artikelname', SUM(bp.menge
) 'Anzahl bestellter Artikel'
153  FROM
154  bestellung_position bp INNER JOIN artikel
a ON bp.artikel_id = a.artikel_id
155  WHERE
156  a.bezeichnung LIKE 'Silber%'
157  GROUP BY
158  a.bezeichnung
159  HAVING
160  SUM(bp.menge) > 10
161  ;
162  GO
163
164  SELECT 'Fragen zu Gruppen (Abschnitt 12.4
.1)';
165  GO
```

```

166  SELECT
167  SUBSTRING(blz, 1, 1) 'Clearinggebiet', CO
UNT(*) 'Anzahl'
168  FROM
169  bank
170  GROUP BY
171  SUBSTRING(blz, 1, 1)
172  ORDER BY 'Anzahl' DESC
173  ;
174  GO
175
176  -- Start Bestellungen erweitern
177  SELECT 'Bestellungen erweitern';
178  GO
179  SET IDENTITY_INSERT bestellung ON ;
180  INSERT INTO bestellung (bestellung_id, ku
nde_id, adresse_id, datum)
181  VALUES
182  (3, 1, 1, '2011-01-15 16:43:00')
183  ,(4, 1, 1, '2011-01-16 09:15:00')
184  ,(5, 1, 1, '2011-01-16 09:16:00')
185  ,(6, 2, 2, '2012-04-01 13:11:00')
186  ;
187  SET IDENTITY_INSERT bestellung OFF ;
188  INSERT INTO bestellung_position (bestellu
ng_id, position_nr, artikel_id, menge)
189  VALUES
190  (3, 1, 7856, 10)
191  ,(3, 2, 7863, 10)
192  ,(4, 1, 3006, 1)
193  ,(4, 2, 3010, 4)
194  ,(5, 1, 3001, 100)
195  ,(5, 2, 3010, 5)
196  ,(5, 3, 3006, 1)
197  ,(5, 4, 3005, 4)
198  ;
199  GO
200  -- Ende Bestellungen erweitern

```

```

201
202  SELECT
203  LEFT(CONCAT(FORMAT(datum, 'yyyy', 'de-de'
), '/', FORMAT(datum, 'MMMM', 'de-de')), 20) A
S 'Monat', COUNT(*) 'Anzahl'
204  FROM bestellung
205  GROUP BY
206  FORMAT(datum, 'yyyy', 'de-de'), FORMAT(da
tum, 'MMMM', 'de-de')
207  ORDER BY
208  FORMAT(datum, 'yyyy', 'de-de') DESC, FORM
AT(datum, 'MMMM', 'de-de') DESC
209  ;
210  GO
211  -- Ende Group by
212  SELECT 'Ende listing09. sql //////////////////
////////////////////////////////////////';
213  GO

```

Listing 28.50 mssql/listing10.sql

```

1  -- Ausführen der vorherigen Befehle
2  :r listing09.sql
3
4  SELECT 'Start listing10. sql //////////////////
////////////////////////////////////////';
5  GO
6
7  SELECT 'Start Aufbau Rechnungswesen';
8  GO
9  USE oshop;
10 GO
11
12 INSERT INTO rechnung
13 (kunde_id, bestellung_id, adresse_id, dat
um)
14 SELECT kunde_id, bestellung_id, adresse_i
d, DATEADD(DAY, 1, datum) FROM bestellung ;

```



```

15  GO
16
17  SET IDENTITY_INSERT rechnung ON ;
18  INSERT INTO rechnung (rechnung_id, kunde_
id, bestellung_id, adresse_id, datum, bezahlar
t, status)
19  VALUES
20  (7, 3, 0, 2, CONVERT(DATETIME, '2012-04-0
6 12:11:00'), 3, 2)
21  ,(8, 3, 0, 2, CONVERT(DATETIME, '2012-04-
07 13:12:00'), 3, 3)
22  ,(9, 5, 0, 4, CONVERT(DATETIME, '2012-04-
08 14:13:00'), 2, 5)
23  ,(10, 5, 0, 4, CONVERT(DATETIME, '2012-04
-09 15:14:00'), 2, 6)
24  ,(11, 5, 0, 4, CONVERT(DATETIME, '2012-04
-10 16:15:00'), 2, 6)
.....
25  ;
26  SET IDENTITY_INSERT rechnung OFF ;
27  GO
28
29  UPDATE rechnung SET status = 4 WHERE rech
nung_id = 2;
30  GO
31  INSERT INTO rechnung_position (rechnung_i
d, position_nr, artikel_id, menge)
32  SELECT bestellung_id, position_nr, artike
l_id, menge FROM bestellung_position;
33  GO
34  INSERT INTO rechnung_position
35  (rechnung_id, position_nr, artikel_id, me
nge)
36  VALUES
37  (7, 1, 3001, 5)
38  ,(7, 2, 3005, 1)
39  ,(8, 1, 3006, 7)
40  ,(8, 2, 3007, 1)
41  ,(10, 1, 3010, 15)

```

```

42  ,(10, 2, 3001, 5)
43  ,(11, 1, 3005, 20)
44  ;
45  GO
46
47  UPDATE rechnung SET rabatt = 7 WHERE kund
e_id = 1;
48  UPDATE rechnung SET skonto = 3 WHERE kund
e_id = 5;
49  GO
50
51  SELECT 'Ende Aufbau Rechnungswesen';
52  GO
53  SELECT 'Problem und Lösung (Abschnitt 13.1)';
54  GO
55  SELECT
56  bp.bestellung_id, SUM(bp.menge * a.einzel
preis) 'Bestellwert'
57  FROM
58  bestellung_position bp INNER JOIN artikel
a ON bp.artikel_id = a.artikel_id
59  GROUP BY
60  bp.bestellung_id
61  ;
62  GO
63  SELECT
64  k.nachname, k.vorname, SUM(bp.menge * a.e
inzelpreis) 'Bestellwert'
65  FROM
66  bestellung_position bp INNER JOIN artikel
a ON bp.artikel_id = a.artikel_id
67  INNER JOIN bestellung b ON bp.bestellung_
id = b.bestellung_id
68  INNER JOIN kunde k ON b.kunde_id = k.kund
e_id
69  GROUP BY
70  bp.bestellung_id, k.nachname, k.vorname

```

```

71 ;
72 GO
73 DROP TABLE IF EXISTS #tmp_bestellwert;
74 GO
75 SELECT
76 k.nachname, k.vorname, SUM(bp.menge * a.e
inzelpreis) 'bestellwert'
77 INTO #tmp_bestellwert
78 FROM
79 bestellung_position bp INNER JOIN artikel
a ON bp.artikel_id = a.artikel_id
80 INNER JOIN bestellung b ON bp.bestellung_
id = b.bestellung_id
81 INNER JOIN kunde k ON b.kunde_id = k.kund
e_id
82 GROUP BY
83 bp.bestellung_id, k.nachname, k.vorname
84 ;
85 GO
86 SELECT * FROM #tmp_bestellwert;
87 GO
88 SELECT
89 nachname, vorname, SUM(bestellwert) 'Umsa
tz'
90 FROM #tmp_bestellwert
91 GROUP BY nachname, vorname
92 ;
93 GO
94 SELECT
95 bw.nachname, bw.vorname, SUM(bw.bestellwe
rt) 'Umsatz'
96 FROM
97 (
98 SELECT
99 k.nachname, k.vorname, SUM(bp.menge * a.e
inzelpreis) 'bestellwert'
100 FROM
101 bestellung_position bp INNER JOIN artikel

```

```

a ON bp.artikel_id = a. artikel_id
102  INNER JOIN bestellung b ON bp.bestellung_
id = b. bestellung_id
103  INNER JOIN kunde k ON b.kunde_id = k.kund
e_id
104  GROUP BY
105  bp.bestellung_id, k.nachname, k.vorname
106  ) AS bw
107  GROUP BY nachname, vorname
108  ;
109  GO
110
111  -- Start nicht korrelierende Unterabfrage
112  SELECT 'Nicht korrelierende Unterabfrage
(Abschnitt 13.2.1.1)';
113  GO
114  SELECT MAX(blz) FROM bank;
115  GO
116  SELECT bankname
117  FROM bank
118  WHERE blz = '37010050'
119  ;
120  GO
121  SELECT bankname
122  FROM bank
123  WHERE blz =
124  (
125  SELECT MAX(blz)
126  FROM bank
127  )
128  ;
129  GO
130  SELECT
131  AVG(einzelpreis) AS 'durchschnittspreis'
132  FROM artikel
133  ;
134  GO
135  SELECT *
```

```
136 FROM artikel
137 WHERE
138 einzelpreis > 30
139 ;
140 GO
141 SELECT *
142 FROM artikel
143 WHERE
144 einzelpreis >
145 (
146 SELECT
147 AVG(einzelpreis) AS 'durchschnittspreis'
148 FROM artikel
149 )
150 ;
151 GO
152 SELECT
153 bp.bestellung_id, SUM(bp.menge * a.einzel
preis) bestellwert
154 FROM
155 bestellung_position bp INNER JOIN artikel
a ON bp.artikel_id = a.artikel_id
156 GROUP BY
157 bp.bestellung_id
158 ;
159 GO
160 -- Leider kann TSQL Alias nicht in WHERE-
Klauseln wiederverwenden!!!
161 SELECT AVG(bw.bestellwert)
162 FROM (
163 SELECT
164 bp.bestellung_id, SUM(bp.menge * a.einzel
preis) bestellwert
165 FROM
166 bestellung_position bp INNER JOIN artikel
a ON bp.artikel_id = a.artikel_id
167 GROUP BY
168 bp.bestellung_id
```

```
169 ) AS bw
170 ;
171 GO
172 SELECT
173 bp1.bestellung_id, SUM(bp1.menge * a1.einzelpreis) bestellwert1
174 FROM
175 bestellung_position bp1 INNER JOIN artikel
176 l a1 ON bp1.artikel_id = a1.artikel_id
177 GROUP BY
178 bp1.bestellung_id
179 HAVING SUM(bp1.menge * a1.einzelpreis) >
180 100
181 ;
182 GO
183 SELECT
184 bp1.bestellung_id, SUM(bp1.menge * a1.einzelpreis) bwert1
185 FROM
186 bestellung_position bp1 INNER JOIN artikel
187 l a1 ON bp1.artikel_id = a1.artikel_id
188 GROUP BY
189 bp1.bestellung_id
190 HAVING
191 SUM(bp1.menge * a1.einzelpreis) >
192 (
193 SELECT AVG(bw.bwert2)
194 FROM
195 (
196 SELECT
197 bp2.bestellung_id, SUM(bp2.menge * a2.einzelpreis) bwert2
198 FROM
199 bestellung_position bp2 INNER JOIN artikel
200 l a2 ON bp2.artikel_id = a2.artikel_id
201 GROUP BY
202 bp2.bestellung_id
203 ) AS bw
```

```
200 )
201 ;
202 GO
203 SELECT bestellung_id FROM bestellung;
204 GO
205 SELECT rechnung_id
206 FROM rechnung
207 WHERE bestellung_id IN
208 (
209 SELECT bestellung_id FROM bestellung
210 )
211 ;
212 GO
213 SELECT a.einzelpreis
214 FROM
215 artikel_nm_warengruppe nm INNER JOIN arti
216 kel a ON nm.artikel_id = a.artikel_id
217 INNER JOIN warengruppe w ON nm.warengrupp
218 e_id = w.warengruppe_id
219 WHERE
220 w.bezeichnung = 'Gartenbedarf'
221 ;
222 GO
223 SELECT
224 a.bezeichnung, a.einzelpreis
225 FROM
226 artikel a
227 WHERE
228 a.einzelpreis > ALL(SELECT 10)
229 ;
230 GO
231 SELECT
232 a.bezeichnung, a.einzelpreis
233 FROM
234 artikel a
235 WHERE
236 a.einzelpreis > ALL
237 (
```

```

236  SELECT a.einzelpreis
237  FROM
238  artikel_nm_warengruppe nm INNER JOIN arti
kel a ON nm.artikel_id = a. artikel_id
239  INNER JOIN warengruppe w ON nm.warengrupp
e_id = w. warengruppe_id
240  WHERE
241  w.bezeichnung = 'Gartenbedarf'
242  )
243  ;
244  GO
245  SELECT
246  r.rechnung_id, r.kunde_id, SUM(a.einzelpr
eis*rp.menge) AS 'umsatz'
247  FROM
248  rechnung_position rp INNER JOIN artikel a
ON rp.artikel_id = a.artikel_id
249  INNER JOIN rechnung r ON rp.rechnung_id =
r.rechnung_id
250  GROUP BY
251  r.rechnung_id, r.kunde_id
252  ;
253  GO
254  SELECT SUM(umsatz) AS 'umsatzsumme'
255  FROM
256  (
257  SELECT
258  r.rechnung_id, r.kunde_id, SUM(a.einzelpr
eis*rp.menge) AS 'umsatz'
259  FROM
260  rechnung_position rp INNER JOIN artikel a
ON rp.artikel_id = a.artikel_id
261  INNER JOIN rechnung r ON rp.rechnung_id =
r.rechnung_id
262  GROUP BY
263  r.rechnung_id, r.kunde_id
264  ) AS ksum
265  GROUP BY

```



```

266 ksum.kunde_id
267 ;
268 GO
269 SELECT SUM(umsatz) AS 'umsatzsumme'
270 FROM
271 (
272 SELECT
273 r.rechnung_id, k.kunde_id, SUM(a.einzelpr
eis*rp.menge) AS 'umsatz'
274 FROM
275 rechnung_position rp INNER JOIN artikel a
ON rp.artikel_id = a.artikel_id
276 INNER JOIN rechnung r ON rp.rechnung_id =
r.rechnung_id
277 INNER JOIN kunde k ON r.kunde_id = k.kund
e_id
278 WHERE nachname = 'beutlin'
279 GROUP BY
280 r.rechnung_id, k.kunde_id
281 ) AS ksum
282 GROUP BY
283 ksum.kunde_id
284 ;
285 GO
286 SELECT ksum.kunde_id, SUM(umsatz) AS 'ums
atzsumme'
287 FROM
288 (
289 SELECT
290 r.rechnung_id, r.kunde_id, SUM(a.einzelpr
eis*rp.menge) AS 'umsatz'
291 FROM
292 rechnung_position rp INNER JOIN artikel a
ON rp.artikel_id = a.artikel_id
293 INNER JOIN rechnung r ON rp.rechnung_id =
r.rechnung_id
294 GROUP BY
295 r.rechnung_id, r.kunde_id

```

```

296 ) AS ksum
297 GROUP BY
298 ksum.kunde_id
299 HAVING SUM(umsatz) > ALL (SELECT 100)
300 ;
301 GO
302 SELECT k.kunde_id, k.nachname, k.vorname,
303        SUM(umsatz) AS 'umsatzsumme'
304 FROM
305 (
306  SELECT
307    r.rechnung_id, r.kunde_id, SUM(a.einzelpr
308    eis*rp.menge) AS 'umsatz'
309  FROM
310    rechnung_position rp INNER JOIN artikel a
311    ON rp.artikel_id = a.artikel_id
312  INNER JOIN rechnung r ON rp.rechnung_id =
313    r.rechnung_id
314  GROUP BY
315    r.rechnung_id, r.kunde_id
316 ) AS ksum
317 INNER JOIN kunde k ON ksum.kunde_id = k.k
318 unde_id
319 GROUP BY
320 k.kunde_id, k.nachname, k.vorname
321 HAVING SUM(umsatz) > ALL
322 (
323  SELECT SUM(umsatz) AS 'umsatzsumme'
324  FROM
325  (
326    SELECT
327      r.rechnung_id, r.kunde_id, SUM(a.einzelpr
328      eis*rp.menge) AS 'umsatz'
329    FROM
330      rechnung_position rp INNER JOIN artikel a
331      ON rp.artikel_id = a.artikel_id
332    INNER JOIN rechnung r ON rp.rechnung_id =
333      r.rechnung_id

```

```
326  INNER JOIN kunde k ON r.kunde_id = k.kund
e_id
327  WHERE k.nachname = 'beutlin'
328  GROUP BY
329  r.rechnung_id, r.kunde_id
330  ) AS ksum
331  GROUP BY
332  ksum.kunde_id
333  )
334  ;
335  GO
336  SELECT a.einzelpreis
337  FROM
338  artikel_nm_warengruppe nm INNER JOIN arti
kel a ON nm.artikel_id = a.artikel_id
339  INNER JOIN warengruppe w on nm.warengrupp
e_id = w. warengruppe_id
340  WHERE
341  w.bezeichnung = 'Bürobedarf'
342  ;
343  GO
344  SELECT
345  a.bezeichnung, a.einzelpreis
346  FROM
347  artikel a
348  WHERE
349  a.einzelpreis > ANY(SELECT 10)
350  ;
351  GO
352  SELECT
353  a.bezeichnung, a.einzelpreis
354  FROM
355  artikel a
356  WHERE
357  a.einzelpreis > ANY
358  (
359  SELECT a.einzelpreis
360  FROM
```

```
361  artikel_nm_warengruppe nm INNER JOIN arti
kel a ON nm.artikel_id = a.artikel_id
362  INNER JOIN warengruppe w ON nm.warengrupp
e_id = w. warengruppe_id
363  WHERE
364  w.bezeichnung = 'Bürobedarf'
365  )
366  ;
367  GO
368  SELECT
369  a.bezeichnung, a.einzelpreis
370  FROM
371  artikel a
372  WHERE
373  a.artikel_id NOT IN
374  (
375  SELECT
376  artikel_id
377  FROM
378  artikel_nm_warengruppe nm INNER JOIN ware
ngruppe w ON nm.warengruppe_id = w. warengrupp
e_id
379  WHERE w.bezeichnung = 'Bürobedarf'
380  )
381  AND
382  a.einzelpreis > ANY
383  (
384  SELECT a.einzelpreis
385  FROM
386  artikel_nm_warengruppe nm INNER JOIN arti
kel a ON nm.artikel_id = a.artikel_id
387  INNER JOIN warengruppe w ON nm.warengrupp
e_id = w. warengruppe_id
388  WHERE
389  w.bezeichnung = 'Bürobedarf'
390  )
391  ;
392  GO
```

```
393  -- Leider kann in TSQL kein Prädikat der
Art (v1, v2) IN (tabspalte1, tabspalte12) gema
cht werden.
394
395  -- Ende nicht korrelierende Unterabfrage
396
397  -- Start korrelierende Unterabfrage
398  SELECT 'Korrelierende Unterabfrage (Absch
nitt 13.3.1)';
399  SELECT rechnung_id
400  FROM rechnung r
401  WHERE
402  (
403  SELECT
404  COUNT(position_nr)
405  FROM rechnung_position rp
406  WHERE
407  rp.rechnung_id = r.rechnung_id
408  ) >= 3
409  ;
410  GO
411  SELECT rechnung_id
412  FROM rechnung r
413  WHERE
414  EXISTS
415  (
416  SELECT
417  bestellung_id
418  FROM bestellung b
419  WHERE b.bestellung_id = r.bestellung_id
420  )
421  ;
422  GO
423  -- Ende korrelierende Unterabfrage
424
425  -- Start CTE
426  SELECT 'CTE vs SUBSELECT (Abschnitt 13.5)
';
```

```

427  -- Start CTE
428  WITH
429  rsum AS (
430  SELECT
431  rechnung.rechnung_id, kunde.kunde_id, nachname, vorname, SUM(einzelpreis*menge) AS umsatz_r
432  FROM
433  rechnung_position INNER JOIN artikel ON rechnung_position.artikel_id = artikel.artikel_id
434  INNER JOIN rechnung ON rechnung_position.rechnung_id = rechnung.rechnung_id
435  INNER JOIN kunde ON rechnung.kunde_id = kunde.kunde_id
436  GROUP BY rechnung.rechnung_id, kunde.kunde_id, nachname, vorname
437  ),
438  ksum AS (
439  SELECT
440  kunde_id, nachname, vorname, SUM(umsatz_r) AS umsatz_k
441  FROM rsum
442  GROUP BY kunde_id, nachname, vorname
443  ),
444  bsum AS (
445  SELECT umsatz_k
446  FROM ksum
447  WHERE nachname='Beutlin'
448  )
449  SELECT kunde_id, nachname, vorname, umsatz_k
450  FROM ksum
451  WHERE umsatz_k > ALL (SELECT * FROM bsum)
452  ;
453  GO
454  -- Ende CTE
455

```

```

456 -- Start Fallstudie
457 SELECT 'Fallstudie Datenimport (Abschnitt
13.5)';
458 GO
459 DROP TABLE IF EXISTS #tmp_import;
460 GO
461 CREATE TABLE #tmp_import
462 (
463     nachname NVARCHAR(255) COLLATE German_PhoneBook_CI_AI,
464     vorname NVARCHAR(255) COLLATE German_PhoneBook_CI_AI,
465     strasse NVARCHAR(255) COLLATE German_PhoneBook_CI_AI,
466     hnr NVARCHAR(255) COLLATE German_PhoneBook_CI_AI,
467     lkz NVARCHAR(255) COLLATE German_PhoneBook_CI_AI,
468     plz NVARCHAR(255) COLLATE German_PhoneBook_CI_AI,
469     ort NVARCHAR(255) COLLATE German_PhoneBook_CI_AI,
470     ktnr NVARCHAR(255) COLLATE German_PhoneBook_CI_AI,
471     blz NVARCHAR(255) COLLATE German_PhoneBook_CI_AI
472 );
473
474 /*
475 BULK INSERT #tmp_import
476 FROM 'X:\[...]kunden01.csv'
477 WITH (FIRSTROW = 2, FIELDTERMINATOR = ';'
, ROWTERMINATOR='0x0a', FORMAT='CSV') ;
478 */
479 :r kunden01.sql
480 SELECT LEFT(nachname,10), LEFT(vorname,10),
LEFT(strasse,10), LEFT(hnr,10), LEFT(plz,10),
LEFT(ort,10) FROM #tmp_import;

```

```
481
482  INSERT INTO kunde (nachname, vorname)
483  SELECT DISTINCT nachname, vorname
484  FROM #tmp_import
485  WHERE
486  CONCAT(TRIM(#tmp_import.vorname), TRIM(#t
mp_import.nachname)) NOT IN
487  (
488  SELECT CONCAT(TRIM(vorname), TRIM(nachnam
e)) FROM kunde
489  )
490  ;
491  GO
492
493  INSERT INTO adresse (strasse, hnr, lkz, p
lz, ort)
494  SELECT DISTINCT strasse, hnr, lkz, plz, o
rt
495  FROM #tmp_import tmp
496  WHERE
497  CONCAT(TRIM(tmp.strasse), TRIM(tmp.hnr),
TRIM(tmp.lkz), TRIM(tmp.plz), TRIM(tmp.ort))
NOT IN
498  (
499  SELECT CONCAT(TRIM(strasse), TRIM(hnr), T
RIM(lkz), TRIM(plz), TRIM(ort))
500  FROM adresse
501  )
502  ;
503  GO
504
505  ALTER TABLE #tmp_import ADD kunde_id INT
;
506  ALTER TABLE #tmp_import ADD adresse_id IN
T ;
507  ALTER TABLE #tmp_import ADD bank_id NCHAR
(12) COLLATE German_PhoneBook_CI_AI;
508  GO
```



```
509
510 SELECT 'UPDATE';
511 GO
512 UPDATE #tmp_import
513 SET
514 #tmp_import.kunde_id = k.kunde_id
515 FROM
516 (
517 SELECT kunde_id, nachname, vorname
518 FROM kunde
519 ) k
520 WHERE
521 CONCAT(TRIM(#tmp_import.vorname), TRIM(#t
mp_import.nachname))
522 =
523 CONCAT(TRIM(k.vorname), TRIM(k.nachname))
524 ;
525
526 GO
527 UPDATE #tmp_import
528 SET #tmp_import.adresse_id = a.adresse_id
529 FROM
530 (
531 SELECT adresse_id, strasse, hnr, plz, ort
532 FROM adresse a
533 ) a
534 WHERE
535 CONCAT(TRIM(#tmp_import.strasse), TRIM(#t
mp_import.hnr), TRIM(#tmp_import.plz), TRIM(#t
mp_import.ort))
536 =
537 CONCAT(TRIM(a.strasse), TRIM(a.hnr), TRIM
(a.plz), TRIM(a.ort))
538 ;
539
540 -- SELECT k.kunde_id, a.adresse_id, CONCA
T(TRIM(LEFT(k.vorname, 20)), TRIM(LEFT(k. nach
name, 20)), TRIM(LEFT(a.strasse, 20)), TRIM(LE
```

```

FT(a.hnr, 20))) FROM # tmp_import INNER JOIN k
unde k ON #tmp_import.kunde_id = k.kunde_id IN
NER JOIN adresse a ON #tmp_import.adresse_id =
a.adresse_id ORDER BY kunde_id, adresse_id
541 GO
542 UPDATE #tmp_import
543 SET #tmp_import.bank_id = b.bank_id
544 FROM (SELECT bank_id, blz FROM bank) b
545 WHERE TRIM(#tmp_import.blz) = TRIM(b.blz)
546 ;
547 GO
548
549 INSERT INTO
550 bankverbindung (kunde_id, bankverbindung_
nr, bank_id, kontonummer, iban)
551 SELECT DISTINCT kunde_id, 1, bank_id, ktn
r, CONCAT(blz, ktnr)
552 FROM #tmp_import
553 WHERE 0 = (
554 SELECT COUNT(*)
555 FROM bankverbindung bv
556 WHERE
557 #tmp_import.kunde_id = bv.kunde_id
558 AND
559 #tmp_import.bank_id = bv.bank_id
560 AND
561 #tmp_import.ktnr = bv.kontonummer
562 )
563 ;
564 GO
565
566 ALTER TABLE #tmp_import
567 ADD bankverbindung_nr INT NOT NULL DEFAULT 1
568 ;
569 GO
570
571 UPDATE kunde

```

```

572 SET
573 kunde.rechnung_adresse_id = #tmp_import.a
    dresse_id
574 FROM kunde INNER JOIN #tmp_import ON #tmp
    _import.kunde_id = kunde.kunde_id
575 WHERE k.rechnung_adresse_id IS NULL
576 ;
577 GO
578 SELECT kunde_id, TRIM(LEFT(nachname, 20))
    , TRIM(LEFT(vorname, 20)), TRIM(LEFT( strasse,
    20)), TRIM(LEFT(ort, 20))
579 FROM
580 kunde LEFT JOIN adresse ON kunde.rechnung
    _adresse_id = adresse_id
581 ;
582 GO
583
584 SELECT 'Ende listing10. sql //////////////////////////////////
    //////////////////////////////////';
585 GO

```

Listing 28.51 mssql/listing11.sql

```

1  -- Ausführen der vorherigen Befehle
2  :r listing10.sql
3
4  SELECT 'Start listing11. sql //////////////////////////////////
    //////////////////////////////////';
5  GO
6
7  SELECT 'UNION (Abschnitt 14.1)';
8  GO
9  SELECT
10 kunde_id, LEFT(vorname, 10), LEFT(strasse
    , 10), LEFT(hnr, 10), LEFT(lkz, 10), LEFT (plz
    , 10), LEFT(ort, 10)
11 FROM
12 kunde INNER JOIN adresse ON rechnung_adre

```

```
sse_id = adresse_id
13  ;
14  GO
15
16  SELECT
17  strasse, hnr, lkz, plz, ort
18  FROM
19  lieferant l INNER JOIN adresse a ON l.adr
esse_id = a.adresse_id
20  ;
21  GO
22  SELECT
23  strasse, hnr, lkz, plz, ort
24  FROM
25  kunde INNER JOIN adresse ON rechnung_adre
sse_id = adresse_id
26  UNION
27  SELECT
28  strasse, hnr, lkz, plz, ort
29  FROM
30  lieferant l INNER JOIN adresse a ON l.adr
esse_id = a.adresse_id
31  ;
32  GO
33  SELECT
34  strasse, hnr, lkz, plz, ort
35  FROM
36  kunde INNER JOIN adresse ON rechnung_adre
sse_id = adresse_id
37  UNION ALL
38  SELECT
39  strasse, hnr, lkz, plz, ort
40  FROM
41  lieferant l INNER JOIN adresse a ON l.adr
esse_id = a.adresse_id
42  ;
43  GO
44
```

```

45  SELECT 'INTERSECT (Abschnitt 14.2.1)';
46  GO
47  SELECT 'Schnittmenge mit Unterabfrage (Abschnitt 14.2.2)';
48  GO
49  SELECT kunde_id FROM rechnung WHERE YEAR(
datum) = '2011';
50  GO
51  SELECT DISTINCT kunde_id
52  FROM rechnung
53  WHERE YEAR(datum) = '2012' AND kunde_id I
N (1, 2, 3)
54  ;
55  GO
56  SELECT DISTINCT kunde_id
57  FROM rechnung
58  WHERE
59  YEAR(datum) = '2012' AND kunde_id IN
60  (
61  SELECT DISTINCT kunde_id FROM rechnung WH
ERE YEAR(datum) = '2011'
62  )
63  ;
64  GO
65
66  SELECT 'EXCEPT (Abschnitt 14.3.1)';
67  GO
68
69  SELECT 'Differenz mit Unterabfrage (Absch
nitt 14.3.1)';
70  SELECT
71  strasse, hnr, lkz, plz, ort
72  FROM
73  adresse
74  ;
75  GO
76  SELECT
77  strasse, hnr, lkz, plz, ort

```

```

78  FROM
79  kunde INNER JOIN adresse
80  ON liefer_adresse_id = adresse_id
81  ;
82  GO
83  SELECT kunde_id
84  FROM rechnung
85  WHERE YEAR(datum) <> '2011' AND YEAR(datum) = '2012'
86  ;
87  GO
88  SELECT kunde_id FROM rechnung WHERE YEAR(
datum) = '2011';
89  GO
90  SELECT DISTINCT kunde_id
91  FROM rechnung
92  WHERE
93  YEAR(datum) = '2012' AND kunde_id NOT IN
(1, 2, 3)
94  ;
95  GO
96  SELECT DISTINCT kunde_id
97  FROM rechnung
98  WHERE
99  YEAR(datum) = '2012' AND kunde_id NOT IN
100 (
101 SELECT DISTINCT kunde_id FROM rechnung WH
ERE YEAR(datum) = '2011'
102 )
103 ;
104 SELECT 'Ende listing11. sql //////////////////////////////////
////////////////////////////////////////';
105 GO

```

Listing 28.52 mssql/listing12.sql

```

1  -- Ausführen der vorherigen Befehle
2  :r listing11.sql

```

```

3
4  SELECT 'Start listing12. sql //////////////////////////////////
////////////////////////////////////////';
5  GO
6
7  SELECT kunde_id, nachname, vorname, art F
ROM kunde;
8  GO
9
10 SELECT 'Start Aufbau einer temporären art
-Tabelle (Abschnitt 15.1)';
11 GO
12
13 DROP TABLE IF EXISTS #tmp_art;
14 CREATE TABLE #tmp_art
15 (
16  art INT,
17  k_art NVARCHAR(255)
18 )
19 ;
20 GO
21
22 INSERT INTO #tmp_art
23 VALUES
24 (1, 'Privatkunde')
25 ,(2, 'Geschäftskunde')
26 ,(0, 'Unbekannt')
27 ;
28
29 SELECT kunde_id, nachname, vorname, k_art
30 FROM
31 kunde INNER JOIN #tmp_art ON kunde.art =
#tmp_art.art
32 ;
33 SELECT 'Ende Aufbau einer temporären art-
Tabelle';
34 GO
35

```

```

36  SELECT 'Lösung mit UNIONs (Abschnitt 15.1
)';
37  GO
38
39  SELECT kunde_id, nachname, vorname, k_art
40  FROM
41  kunde INNER JOIN
42  (
43  SELECT 1 AS art, 'Privatkunde' AS k_art
44  UNION
45  SELECT 2 AS art, 'Geschäftskunde' AS k_ar
t
46  UNION
47  SELECT 0 AS art, 'Unbekannt' AS k_art
48  ) t
49  ON kunde.art = t.art
50  ;
51  GO
52
53  SELECT 'Nö (Abschnitt 15.1)';
54  SELECT kunde_id, nachname, vorname,
55  CASE art
56  WHEN 1 THEN 'Privatkunde'
57  WHEN 2 THEN 'Geschäftskunde'
58  ELSE 'Unbekannt'
59  END AS k_art
60  FROM kunde
61  ;
62  GO
63
64  -- Start Einfacher CASE
65  SELECT 'Einfacher CASE (Abschnitt 15.2)';
66  SELECT artikel_id, bezeichnung,
67  CASE waehrung
68  WHEN 'EUR' THEN CONCAT(ROUND(einzelpreis,
2), ' e')
69  WHEN 'USD' THEN CONCAT(ROUND(einzelpreis,
2), ' $')

```



```

70 ELSE '?????'
71 END AS preis
72 FROM artikel
73 ;
74 GO
75
76 UPDATE artikel SET waehrung = 'XYZ' WHERE
artikel_id = 3001;
77 SELECT artikel_id, bezeichnung,
78 CASE waehrung
79 WHEN 'EUR' THEN CONCAT(ROUND(einzelpreis,
2), ' e')
80 WHEN 'USD' THEN CONCAT(ROUND(einzelpreis,
2), ' $')
81 END AS preis
82 FROM artikel
83 ;
84 GO
85
86 UPDATE artikel SET waehrung = 'EUR' WHERE
artikel_id = 3001;
87 GO
88 -- Ende Einfacher CASE
89
90 -- Start Searched CASE
91 SELECT 'Searched CASE (Abschnitt 15.3)';
92 GO
93 SELECT beitrag_id,
94 CASE
95 WHEN bezug_beitrag_id > 1 THEN 'Antwort'
96 WHEN bezug_beitrag_id <= 1 THEN 'Keine An
twort'
97 ELSE '???????'
98 END AS Typ
99 FROM beitrag
100 ORDER BY beitrag_id
101 ;
102 GO

```

```

103
104 SELECT beitrags_id,
105 CASE
106 WHEN bezug_beitrags_id > 1
107 THEN CONCAT(nachricht, ' Antwort auf ', b
ezug_beitrags_id, ': >',
108 (
109 SELECT nachricht
110 FROM beitrags b2
111 WHERE b2.beitrags_id = b1.bezug_beitrags_id
112 )
113 ) -- Ende CONCAT
114 WHEN bezug_beitrags_id <= 1 THEN nachricht
115 ELSE '???????'
116 END AS Inhalt
117 FROM beitrags b1
118 WHERE beitrags_id > 1
119 ;
120 GO
121
122 SELECT beitrags_id,
123 CASE
124 WHEN bezug_beitrags_id >= 0 THEN 'Antwort'
125 WHEN bezug_beitrags_id > 1 THEN 'Keine Ant
wort'
126 ELSE '???????'
127 END AS Typ
128 FROM beitrags
129 ORDER BY beitrags_id
130 ;
131 GO
132 -- Ende Searched CASE
133
134 -- Start Fallbeispiele
135 SELECT 'Fallbeispiel Lagerbestand (Abschn
itt 15.4.1)';
136 GO
137 SELECT * FROM lagerbestand;

```

```

138 GO
139 SELECT a.bezeichnung,
140 CASE
141 WHEN l.menge_aktuell <= l.menge_mindest
142 THEN 'Artikel nachbestellen'
143 ELSE 'Bestand ausreichend'
144 END AS lagerstand
145 FROM lagerbestand l INNER JOIN artikel a
ON l.artikel_id=a.artikel_id
146 ORDER BY l.menge_aktuell / l.menge_mindest
147 ;
148 GO
149
150 SELECT 'Fallbeispiel Kundengruppen (Abschnitt 15.4.2)';
151 GO
152 SELECT DISTINCT k.kunde_id, LEFT(k.nachname, 20), LEFT(k.vorname, 20),
153 CASE
154 WHEN 3 <
155 (
156 SELECT COUNT(rechnung_id) FROM rechnung r
157 WHERE r.kunde_id = k.kunde_id
158 ) THEN 'Prämiumkunde'
159 WHEN 2 <
160 (
161 SELECT COUNT(rechnung_id) FROM rechnung r
162 WHERE r.kunde_id = k.kunde_id
163 ) THEN 'Guter Kunde'
164 ELSE 'Kleinkunde'
165 END kundenart
166 FROM
167 rechnung r RIGHT OUTER JOIN kunde k ON r.
kunde_id = k.kunde_id
168 ;
169 GO
170 SELECT k.kunde_id, LEFT(k.nachname, 20),

```

```

LEFT(k.vorname, 20),
171  (
172  SELECT SUM(a.einzelpreis * rp.menge)
173  FROM rechnung r INNER JOIN rechnung_posit
174  ion rp
175  ON rp.rechnung_id = r.rechnung_id
176  INNER JOIN artikel a
177  ON rp.artikel_id=a.artikel_id
178  WHERE r.kunde_id = k.kunde_id
179  ) rechnungswert,
180  CASE
181  WHEN
182  kunde_id NOT IN (SELECT kunde_id FROM rec
183  hnung)
184  THEN 'Kleinkunde'
185  WHEN
186  (
187  SELECT SUM(a.einzelpreis * rp.menge) rech
188  nungswert
189  FROM rechnung r INNER JOIN rechnung_posit
190  ion rp
191  ON rp.rechnung_id = r.rechnung_id
192  INNER JOIN artikel a
193  ON rp.artikel_id=a.artikel_id
194  WHERE r.kunde_id = k.kunde_id
195  ) BETWEEN 0 AND 300 THEN 'Kleinkunde'
196  WHEN
197  (
198  SELECT SUM(a.einzelpreis * rp.menge) rech
199  nungswert
200  FROM rechnung r INNER JOIN rechnung_posit
201  ion rp
202  ON rp.rechnung_id = r.rechnung_id
203  INNER JOIN artikel a
204  ON rp.artikel_id=a.artikel_id
205  WHERE r.kunde_id = k.kunde_id
206  ) BETWEEN 300 AND 1000 THEN 'Guter Kunde'
207  ELSE 'Prämiumkunde'

```

```
202  END AS kundenart
203  FROM kunde k
204  ORDER BY
205  (
206  SELECT SUM(a.einzelpreis * rp.menge) rech
nungswert
207  FROM rechnung r INNER JOIN rechnung_posit
ion rp
208  ON rp.rechnung_id = r.rechnung_id
209  INNER JOIN artikel a
210  ON rp.artikel_id=a.artikel_id
211  WHERE r.kunde_id = k.kunde_id
212  ) DESC
213  ;
214  GO
215
216  DROP TABLE IF EXISTS #tmp_umsatz;
217  GO
218  SELECT kunde_id, SUM(a.einzelpreis * rp.m
enge) rechnungswert
219  INTO #tmp_umsatz
220  FROM rechnung r INNER JOIN rechnung_posit
ion rp ON rp.rechnung_id = r.rechnung_id
221  INNER JOIN artikel a ON rp.artikel_id = a
.artikel_id
222  GROUP BY kunde_id
223  ;
224  GO
225  SELECT k.kunde_id, LEFT(k.nachname, 20),
LEFT(k.vorname, 20), #tmp_umsatz. rechnungswer
t,
226  CASE
227  WHEN
228  #tmp_umsatz.kunde_id IS NULL THEN 'Kleink
unde'
229  WHEN
230  rechnungswert BETWEEN 0 AND 300 THEN 'Kle
inkunde'
```

```

231 WHEN
232 rechnungswert BETWEEN 300 AND 1000 THEN '
Guter Kunde'
233 ELSE 'Premiumkunde'
234 END AS kundenart
235 FROM kunde k LEFT JOIN #tmp_umsatz ON k.k
unde_id = #tmp_umsatz.kunde_id
236 ORDER BY rechnungswert DESC
237 ;
238 GO
239
240 WITH tmp_umsatz (kunde_id, rechnungswert)
241 AS (
242 SELECT kunde_id, SUM(a.einzelpreis * rp.m
enge) rechnungswert
243 FROM rechnung r INNER JOIN rechnung_posit
ion rp ON rp.rechnung_id = r.rechnung_id
244 INNER JOIN artikel a ON rp.artikel_id = a
.artikel_id
245 GROUP BY kunde_id
246 )
247 SELECT k.kunde_id, LEFT(k.nachname, 20),
LEFT(k.vorname, 20), tmp_umsatz. rechnungswert
,
248 CASE
249 WHEN
250 tmp_umsatz.kunde_id IS NULL THEN 'Kleinku
nde'
251 WHEN
252 rechnungswert BETWEEN 0 AND 300 THEN 'Kle
inkunde'
253 WHEN
254 rechnungswert BETWEEN 300 AND 1000 THEN '
Guter Kunde'
255 ELSE 'Premiumkunde'
256 END AS kundenart
257 FROM kunde k LEFT JOIN tmp_umsatz ON k.ku
nde_id = tmp_umsatz.kunde_id

```

```

258 ORDER BY rechnungswert DESC
259 ;
260 GO
261
262 SELECT 'Fallbeispiel Lieferanten (Abschnitt 15.4.3)';
263 GO
264 SELECT LEFT(l.firmenname, 20),
265 CASE
266 WHEN
267 l.lieferant_id IN
268 (
269 SELECT lieferant_id FROM artikel_nm_lieferant
270 ) THEN 'aktiv'
271 ELSE 'inaktiv'
272 END AS status
273 FROM lieferant l
274 ORDER BY firmenname
275 ;
276 GO
277
278 SELECT LEFT(a.bezeichnung, 20),
279 CASE
280 WHEN l.menge_aktuell <= l.menge_mindest THEN 'Artikel sofort nachbestellen'
281 WHEN l.menge_aktuell <= l.menge_mindest +
282 0.3 * l.menge_mindest THEN 'Artikel bald nachbestellen'
283 ELSE 'Bestand ausreichend' END AS lagerstatus
284 FROM lagerbestand l INNER JOIN artikel a ON l.artikel_id = a.artikel_id
285 ORDER BY l.menge_aktuell / l.menge_mindest
286 ;
287 -- Ende Fallbeispiele
288 SELECT 'Ende listing12. sql //////////////////////////////////////
////////////////////////////////////////';

```

Listing 28.53 mssql/listing13.sql

```
1  -- Ausführen der vorherigen Befehle
2  -- :r listing12.sql
3
4  SELECT 'Start listing13. sql //////////////////////////////////
////////////////////////////////////////';
5  GO
6  SELECT 'Meine erste Ansicht (Abschnitt 16
.1.1)';
7  GO
8  CREATE VIEW
9  view_kundenrechnungsadresse (id, nachnam
e, vorname, strasse, ort)
10 AS
11 SELECT
12     kunde_id, nachname, vorname,
13     CONCAT_WS(' ', strasse, hnr),
14     CONCAT_WS(' ', lkz, plz, ort)
15 FROM
16     kunde INNER JOIN adresse
17     ON rechnung_adresse_id = adresse_id
18 WHERE kunde.deleted = 0
19 ;
20 GO
21
22 SELECT * FROM view_kundenrechnungsadresse
;
23 GO
24
25 SELECT 'VIEW-Verarbeitung (Abschnitt 16.1
.2)';
26 GO
27 CREATE VIEW view_artikel_aktiv
28 AS
29     SELECT * FROM artikel
```



```

30     WHERE deleted = 0
31 ;
32 GO
33 CREATE VIEW view_artikel_aktiv_tmp -- Nur
um es später zu löschen!
34 AS
35     SELECT * FROM artikel
36     WHERE deleted = 0
37 ;
38 GO
39
40 SELECT FLOOR(artikel_id / 1000) gruppe, A
VG(einzelpreis)
41     FROM view_artikel_aktiv
42     GROUP BY FLOOR(artikel_id / 1000)
43 ;
44 GO
45
46 -- EXPLAIN SELECT FLOOR(artikel_id / 1000
) gruppe, AVG(einzelpreis)
47     -- FROM view_artikel_aktiv
48     -- GROUP BY FLOOR(artikel_id / 1000)
49 -- GO
50 -- ;
51
52
53 SELECT artikel_id, bezeichnung FROM
54     view_artikel_aktiv
55     WHERE
56     artikel_id BETWEEN 7000 AND 9000
57 ;
58 GO
59
60 -- EXPLAIN SELECT artikel_id, bezeichnung
FROM
61     -- view_artikel_aktiv
62     -- WHERE
63     -- artikel_id BETWEEN 7000 AND 9000

```

```
64  -- GO
65
66
67  -- Start DROP VIEW
68  SELECT 'Ansicht loeschen (Abschnitt 16.1.3)';
69  GO
70  DROP VIEW view_artikel_aktiv_tmp;
71  GO
72
73  DROP DATABASE IF EXISTS tmp1;
74  GO
75  CREATE DATABASE tmp1;
76  GO
77  USE tmp1;
78  GO
79  CREATE TABLE a (ai INT);
80  GO
81  CREATE TABLE b (bi INT);
82  GO
83  CREATE VIEW ab
84  AS
85      SELECT * FROM a INNER JOIN b ON ai = bi
86  ;
87  GO
88  DROP TABLE a;
89
90  -- SELECT 'Erwarte eine Fehlermeldung';
91  -- SELECT * FROM ab;
92  -- GO
93  -- Ende DROP VIEW
94  USE oshop;
95  GO
96  DROP DATABASE tmp1;
97  GO
98
99  SELECT 'Anwendungsgebiet: Vereinfachung (Abschnitt 16.2)';
```

```

100 GO
101 CREATE OR ALTER
102     VIEW view_artikel_aktiv
103     AS
104         SELECT * FROM artikel
105             WHERE deleted = 0
106     ;
107 GO
108 CREATE OR ALTER VIEW view_artikel_verfueg
bar
109 AS
110     SELECT a.artikel_id, bezeichnung
111     FROM artikel a INNER JOIN lagerbestand
l ON a.artikel_id = l.artikel_id
112     WHERE menge_aktuell >= menge_mindest
113 -- ORDER BY a.artikel_id In T-SQL nicht m
öglich
114 ;
115 GO
116 SELECT * FROM view_artikel_verfuegbar;
117 GO
118 CREATE OR ALTER
119     VIEW view_lagerbestand_aktiv
120     AS
121         SELECT * FROM lagerbestand
122             WHERE deleted = 0
123     ;
124 GO
125 CREATE OR ALTER VIEW view_lagerbestand_ar
tikelbezeichnung
126 AS
127     SELECT l.*, a.bezeichnung
128     FROM
129         view_lagerbestand_aktiv l INNER JOIN v
iew_artikel_aktiv a
130         ON l.artikel_id = a.artikel_id
131     ;
132 GO

```

```
133  SELECT * FROM view_lagerbestand_artikelbe  
zeichnung;  
134  GO  
135  
136  SELECT 'Ende listing13. sql //////////////////////////////////  
////////////////////////////////////////';
```

Literatur

- [Ada14] ADAMS, Ralf: *Bug #71244 –Wrong result computation* [com/bug.php?id=71244](https://bug.php.net/bug.php?id=71244), März 2014. – [Online: Stand 07.2023]
- [Ada23] ADAMS, Ralf: *Unexpected behaviour of FULL JOIN*. <https://www.php.net/bug.php?id=71244>, Juli 2023. – [Online: Stand 07.2023]
- [Ahm22] AHMED, Shahzeb: *MariaDB vsMySQL: A Detailed Comparison* [mariadb-vs-mysql/](https://www.mariadb.com/kb/en/mariadb-vs-mysql/), November 2022. – [Online: Stand 07.2023]
- [Aka19] AKAMAI: *Web Attacks and Gaming Abuse*. <https://www.akamai.com/en/press-room/press-releases/2019/04/24/akamai-releases-state-of-the-internet-soti-security-web-attacks-and-gaming-abuse-report>, April 2019. – [Online: Stand 07.2023]
- [Arb17] ARBEZZANO, Gianluca: *Grundkurs Docker: Eine praktische Einführung* <https://jaxenter.de/einfuehrung-docker-tutorial-concept/>, April 2017. – [Online: Stand 07.2023]
- [Ass76] ASSOCIATION, American S.: *American Standard Code for Information Interchange: ASA X3.4 (1976)*, Juni 1976.
- [Bam20] BAMBROUGH, Billy: *New YorkMan Charged In \$100M Bitcoin Case* <https://www.forbes.com/sites/billybambrough/2020/01/10/new-york-man-charged-in-100-million-dollar-bitcoin-case/?sh=2343063bc90e,M>, 10. Januar 2020.
- [Bun16] BUNDESLAND, Nordrhein-Westfalen: *Verordnung über die Ausführung der Aufgaben der Bundesagentur für Arbeit*, April 2016.

BUNDESBANK, Deutsche: *Bankleitzahlen*.
http://www.bundesbank.de/Redaktion/DE/Standard/bankleitzahlen_download.html, Januar 2016. – [Online unter <https://www.bundesbank.de/de/aufgaben/unzahlungsverkehr/serviceangebot/bankleitzahlen/download> werden.]

- [Cal22] CALBIMONTE, Daniel: *SQL Boolean Tutorial*.
- [Che76] <https://www.sqlshack.com/sql-boolean-tutorial/>, No
Peter Pin-Shan: The Entity-Relationship Model – *Transactions on Database Systems* 1 (1976), März, Nr. 1, S. 9–36
- [Cod70] CODD, Edgar F.: A Relational Model of Data for Large
ACM (1970), 13. Juni, S. 377–387. – ISSN 0001-0782
- [Cor22] CORPORATION, Microsoft: *BULK INSERT (Transact-SQL)*
<https://docs.microsoft.com/en-us/sql/statements/bulk-insert-transact-sql?view=sql-server-150>
- [Dai20] DAITYARI, Shaumik: *SQL Injection: Eine Anfängeranleitung*
<https://kinsta.com/de/blog/sql-injection/>, Februar 2020
- [DE23] DB-ENGINES: *DB-Engines Ranking - Trend Popularity*
Juni 2023. – [Online: Stand 07.2023]
- [Doc21] DOCKERHUB: *Build and Ship any Application Anywhere*
[Online: Stand 07.2023]
- [EFH+] EDLICH, Stefan; FRIEDLAND, Achim; HAMPE, Jens; E
Einstieg in die Welt nicht relationaler Web 2.0 Datenbanken
Hanser Verlag. – ISBN 978-3-446-42855-3
- [Eis23a] EISENTRAUT, Peter: *PostgreSQL and SQL:2023*.
<http://peter.eisentraut.org/blog/2023/04/18/postgresql-and-sql-2023/>
[Online: Stand 07.2023]

- [Eis23b] EISENTRAUT, Peter: *SQL:2023 is finished: Here is what*
<https://peter.eisentraut.org/blog/2023/04/04/sql-2023-is-finished/>
[Online: Stand 07.2023]
- [Fou22] FOUNDATION, OWASP: *Top 10 Web Application Security*
[top-ten/](https://owasp.org/Top10/), November 2022. – [Online: Stand 07.2023]
- [Fri23] FRIENDS, Apache: *Apache Friends – Herunterladen.*
https://www.apachefriends.org/de/download_success.html
- [Gar16] GARDEN, Euan: *SQL MythBusters – SQL Server is really*
<https://blogs.msdn.microsoft.com/euanga/2006/01/11/sql-server-is-not-a-microsoft-product-not-a-microsoft-one/>, Januar 2016. –
- [Har] HARRISON, Guy: *Next Generation Databases: NoSQL*
Apress. – ISBN 978-1-14842-1330-8
- [Hel19] HELP, Software T.: *JSON Tutorial: Introduction and*
<https://www.softwaretestinghelp.com/json-tutorial/>
- [Inc20] INC., Sentient D.: *4 Types of Data Breaches to Know*
<https://www.entrustsolutions.com/2020/09/02/4-types-of-data-breaches-to-know/>
[Online: Stand 07.2023]
- [Inf23] INFORMATIONSTECHNIK, Bundesamt für Sicherheit
https://www.bsi.bund.de/DE/Themen/Verbraucherinformation/Empfehlungen/Cyber-Sicherheitsempfehlungen/Accomplishing-sichere-passwoerter-erstellen_node.html, 2023. – [O
- [Int17] INTERNATIONAL, ECMA: *The JSON Data Interchange*
international.org/publications-and-standards/standards
- [ISO85] ISO/TC97: *Information processing; Documentation systems*
system flowcharts, program network charts and systems
der DIN 66001

- [JRH+04] *Kapitel 3.4.7 Assoziationen.* In: JECKLE, Mario; RUPP QUEINS, Stefan: *UML 2 glasklar*. 1. Auflage. München: ISBN 3-446-22575-7, S. 89 f.

- [KE01] KEMPER, Alfons; EICKLER, André: *Datenbanksystem* ISBN 3-486-25706-4

- [Kir14] KIRK, Jeremy: *SQL injection flaw in Wall Street Journal* <https://www.csoonline.com/article/2455696/sql-injection-breach.html>, Juli 2014. – [Online: Stand 07.2023]

- [Knu81] KNUTH, Donald E.: *The Art of Computer Programming* Edition. Addison-Wesley, 1981. – ISBN 0-201-03822-1

- [Mar16] MARIADB: *MariaDB versus MySQL -- Kompatibilität.* <https://mariadb.com/kb/en/compatibility/#inkompatibilit%C3%A4ten-zwischen-mysql> Stand 07.2023]

- [Mar23a] MARIADB: *MariaDB versus MySQL - Features.* <https://mariadb.com/kb/en/features/> Juli 2023. – [Online: Stand 07.2023]

- [Mar23b] MARIADB: *Query Cache.* <https://mariadb.com/kb/en/query-cache/> 07.2023]

- [Mar23c] MARIADB: *Storage Engines.* <https://mariadb.com/kb/en/storage-engines/> Stand 07.2023]

- [Mik10] MIKISOFT: *MySQL Injection – Simple Load File and In* [db.com/papers/14635](https://mariadb.com/papers/14635), August 2010. – [Online: Stand 07.2023]

- [MOP06] MÖHRING, Rolf H.; OELLRICH, Martin ; PANKRATH, Roman: *man alle Primzahlen bis eine Milliarde berechnen?* <https://www.heise.de/inf/woche-das-sieb-des-eratosthenes-wie-schnell-kann-berechnen.html>, August 2006. – [Online: Stand 07.2023]

- [MyS18a] MySQL: 12.3.3 Logical Operators. <https://dev.mysql.com/doc/refman/8.0/en/logical-operators.html>, April 2018. – [Online: Stand 07.2023]
- [MyS18b] MySQL: 12.6.2 Mathematical Functions. <https://dev.mysql.com/doc/refman/8.0/en/mathematical-functions.html>, April 2018. – [Online: Stand 07.2023]
- [MyS18c] MySQL: 8.10.3 The MySQL Query Cache. <https://dev.mysql.com/doc/refman/8.0/en/query-cache.html>, August 2018. – [Online: Stand 07.2023]
- [MyS19a] MySQL: 13.2.20 WITH (Common Table Expressions). <https://dev.mysql.com/doc/refman/8.0/en/with.html>, April 2019. – [Online: Stand 07.2023]
- [MyS19b] MYSQLTUTORIAL: An Introduction to MySQL CTE. <http://www.mysqltutorial.org/mysql-cte/>, April 2019. – [Online: Stand 07.2023]
- [MyS21a] MySQL: 12.7 Date and Time Functions. <https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html>, April 2021. – [Online: Stand 07.2023]
- [MyS21b] MySQL: 13.2.11.4 Subqueries with ALL. <https://dev.mysql.com/doc/refman/8.0/en/subqueries-with-all.html>, April 2021. – [Online: Stand 07.2023]
- [MyS21c] MySQL: Chapter 16 Alternative Storage Engines. <http://dev.mysql.com/doc/refman/8.0/en/storage-engines.html>, 2021. – [Online: Stand 07.2023]
- [MyS21d] MySQL: Chapter 21 InnoDB Cluster. <https://dev.mysql.com/doc/refman/8.0/en/innodb-cluster-introduction.html>, März 2021. – [Online: Stand 07.2023]
- [MyS21e] MySQL: Chapter 23 Writing a Custom Storage Engine. <https://dev.mysql.com/doc/refman/8.0/en/writing-a-custom-storage-engine.html>, 2021. – [Online: Der ursprüngliche Artikel <https://web.archive.org/web/20220422033459/https://dev.mysql.com/doc/refman/8.0/en/writing-a-custom-storage-engine.html> noch nachgelesen werden.]
- [MyS23] MySQL: 12.8.2 Regular Expressions. <https://dev.mysql.com/doc/refman/8.0/en/regular-expressions-syntax.html>, Juli 2023. – [Online: Stand 07.2023]
- [Nor85]

- NORMEN, Deutsches I.: *DIN 66261, Sinnbilder für Str*
1985. – auch: Nassi-Shneidermann-Diagramm
- [Nor09] NORMEN, Deutsches I.: *DIN 69901-5, Projektmanage*
Begriffe. Januar 2009. – u.A. Definition von Lasten- u
- [Oes06] *Kapitel 4.4.2 Assoziationen*. In: OESTERREICH, Bernd:
Oldenbourg, 2006. – ISBN 978-3-486-57926-0, S. 271
- [SEL23] SELFHTML E.V.: *SELFHTML*. <https://wiki.selfhtml.org/>,
- [Sha05] SHAFRANOVICH, Y.: *Common Format and MIME Type*
<http://tools.ietf.org/html/rfc4180>, Oktober 2005. – [C
- [Sta18] STACKOVERFLOW: *Foreign key constraint may cause*
<https://stackoverflow.com/questions/851625/foreign-cascade-paths>, Mai 2018. – [Online: Stand 07.2023]
- [SZT+09] *Kapitel Optimale Datentypen auswählen*. In: SCHW/ ZAWDNOY, Jeremy D.; LENTZ, Arjen ; BALLING, Dere
O'Reilly, 2009. – ISBN 987-3-89721-889-5, S. 87 f.
- [The10] THE OPEN GEOSPATIAL CONSORTIUM: *OpenGIS Imp*
– Simple feature access – Part 2: SQL option. <https:///>
07.2023]
- [The23a] THE PHP GROUP: *PHP Handbuch*. <https://www.php.>
07.2023]
- [The23b] THE PHP GROUP: *Quick Start Guide*. <https://www.ph>
2023. – [Online: Stand 07.2023]
- [Vij19] VIJAYAN, Jai: *SQL Injection Attacks Represent Two-T*
<https://www.darkreading.com/attacks-breaches/sql-app-attacks/d/d-id/1334960>, Juni 2019. – [Online: St

- [Whe19] WHEAT, Jake: *Document*. <https://jakewheat.github.io/grammar.html>, Juli 2019. – [Online: Stand 07.2023]
- [Wik21] WIKIPEDIA: *Rufnummer*. <https://de.wikipedia.org/wi>
- [Wik23a] WIKIPEDIA: *B-Baum*. <http://de.wikipedia.org/wiki/B->
- [Wik23b] WIKIPEDIA: *Bankleitzahl*. <http://de.wikipedia.org/wi>
- [Wik23c] WIKIPEDIA: *Chen-Notation*. <http://de.wikipedia.org/> 07.2023]
- [Wik23d] WIKIPEDIA: *Cobol*. <http://de.wikipedia.org/wiki/Cobol> 07.2023]
- [Wik23e] WIKIPEDIA: *Data-Warehouse*. <https://de.wikipedia.o> 07.2023]
- [Wik23f] WIKIPEDIA: *Deklarative Programmierung*. <https://de> 2023. – [Online: Stand 07.2023]
- [Wik23g] WIKIPEDIA: *Eierkennzeichnung*. <https://de.wikipedia> Stand 07.2023]
- [Wik23h] WIKIPEDIA: *Eierkennzeichnung*. <https://de.wikipedia> 07.2023]
- [Wik23i] WIKIPEDIA: *Entity-Relationship-Modell*. <http://de.wik> 07.2023]
- [Wik23j] WIKIPEDIA: *Fahrzeug-Identifizierungsnummer*. <http://de.wikipedia.org/wiki/Fahrzeug-Identifizierungsnummer>, 2023. – [Online: Stand 07.2023]
- [Wik23k] WIKIPEDIA: *IDEF1X*. <http://de.wikipedia.org/wiki/IDEF1X>
- [Wik23l] WIKIPEDIA: *Include*. [http://de.wikipedia.org/wiki/Co](http://de.wikipedia.org/wiki/Include)

- [Wik23m] WIKIPEDIA: *Internationale Bankkontonummer*. https://de.wikipedia.org/wiki/Internationale_Bankkontonummer
- [Wik23n] WIKIPEDIA: *Internationale Standardbuchnummer*. https://de.wikipedia.org/wiki/Internationale_Standardbuchnummer. [Online: Stand 07.2023]
- [Wik23o] WIKIPEDIA: *Kraftfahrzeugkennzeichen*. <http://de.wikipedia.org/wiki/Kraftfahrzeugkennzeichen>. [Online: Stand 07.2023]
- [Wik23p] WIKIPEDIA: *Martin-Notation*. <https://de.wikipedia.org/wiki/Martin-Notation>. [Online: Stand 07.2023]
- [Wik23q] WIKIPEDIA: *Rainbow Table*. https://de.wikipedia.org/wiki/Rainbow_Table. [Online: Stand 07.2023]
- [Wik23r] WIKIPEDIA: *Relationale Datenbank*. https://de.wikipedia.org/wiki/Relationale_Datenbank, 2023. – [Online: Stand 07.2023]
- [Wik23s] WIKIPEDIA: *SELECT (SQL)*. [https://en.wikipedia.org/wiki/SELECT_\(SQL\)](https://en.wikipedia.org/wiki/SELECT_(SQL)). [Online: Stand 07.2023]
- [Wik23t] WIKIPEDIA: *SQL*. <http://de.wikipedia.org/wiki/SQL>, 2023. – [Online: Stand 07.2023]

Font License

Zurück zum Impressum

Copyright 2010, 2012, 2014 Adobe Systems Incorporated (<http://www.adobe.com/>), with Reserved Font Name 'Source'. All Rights Reserved. Source is a trademark of Adobe Systems Incorporated in the United States and/or other countries.

This Font Software is licensed under the SIL Open Font License, Version 1.1.

This license is copied below, and is also available with a FAQ at: <http://scripts.sil.org/OFL>

SIL OPEN FONT LICENSE Version 1.1 - 26 February 2007

PREAMBLE

The goals of the Open Font License (OFL) are to stimulate worldwide development of collaborative font projects, to support the font creation efforts of academic and linguistic communities, and to provide a free and open framework in which fonts may be shared and improved in partnership with others.

The OFL allows the licensed fonts to be used, studied, modified and redistributed freely as long as they are not sold by themselves. The fonts, including any derivative works, can be bundled, embedded, redistributed and/or sold with any software provided that any reserved

names are not used by derivative works. The fonts and derivatives, however, cannot be released under any other type of license. The requirement for fonts to remain under this license does not apply to any document created using the fonts or their derivatives.

DEFINITIONS

"Font Software" refers to the set of files released by the Copyright Holder(s) under this license and clearly marked as such. This may include source files, build scripts and documentation.

"Reserved Font Name" refers to any names specified as such after the copyright statement(s).

"Original Version" refers to the collection of Font Software components as distributed by the Copyright Holder(s).

"Modified Version" refers to any derivative made by adding to, deleting, or substituting -- in part or in whole -- any of the components of the Original Version, by changing formats or by porting the Font Software to a new environment.

"Author" refers to any designer, engineer, programmer, technical writer or other person who contributed to the Font Software.

PERMISSION & CONDITIONS

Permission is hereby granted, free of charge, to any person obtaining a copy of the Font Software, to use, study, copy, merge, embed, modify, redistribute, and sell modified and unmodified co

pies of the Font

Software, subject to the following conditions:

1) Neither the Font Software nor any of its individual components, in Original or Modified Versions, may be sold by itself.

2) Original or Modified Versions of the Font Software may be bundled, redistributed and/or sold with any software, provided that each copy contains the above copyright notice and this license. These can be included either as stand-alone text files, human-readable headers or in the appropriate machine-readable metadata fields within text or binary files as long as those fields can be easily viewed by the user.

3) No Modified Version of the Font Software may use the Reserved Font Name(s) unless explicit written permission is granted by the corresponding Copyright Holder. This restriction only applies to the primary font name as presented to the users.

4) The name(s) of the Copyright Holder(s) or the Author(s) of the Font Software shall not be used to promote, endorse or advertise any Modified Version, except to acknowledge the contribution(s) of the Copyright Holder(s) and the Author(s) or with their explicit written permission.

5) The Font Software, modified or unmodified, in part or in whole, must be distributed entirely under this license, and must not be

distributed under any other license. The requirement for fonts to remain under this license does not apply to any document created using the Font Software.

TERMINATION

This license becomes null and void if any of the above conditions are not met.

DISCLAIMER

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.