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

Adams

SQL

Ihr Plus – digitale Zusatzinhalte!

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

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

plus-6qrL3-2dsm7

Bleiben Sie auf dem Laufenden!

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

www.hanser-fachbuch.de/newsletter

Ralf Adams

SQL

Der Grundkurs für Ausbildung und Praxis

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

5. Auflage

HANSER

Der Autor:

Ralf Adams, Bochum
Kontakt: sqlbuch@ralfadams.de

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.

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
Druck und Bindung: Hubert & Co. GmbH & Co. KG BuchPartner, Göttingen
Printed in Germany

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

Vorwort zur 5. Auflage

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

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

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

Primärziel ist ein möglichst umfangreicher Ersteinstieg, der dann durch berufliche Praxis ausgebaut werden kann. Trotzdem vertiefe ich an vielen Stellen im Buch den Einblick in SQL oder den MySQL Server – zum einen, um zu zeigen, dass ich auch ein bisschen was draufhabe, zum anderen, um Neugierde und Jagdtrieb beim Leser[1] zu wecken.

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

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

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

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

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

Wenn Sie auf

plus.hanser-fachbuch.de

den Code

`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 Vorauflagen – angestoßen und vorangetrieben hat. Frau Rothe und Frau Gottmann haben sprachliche Ausrutscher und allzu flapsige Formulierungen glatt gebügelt. Das Layout wurde von Frau Irene Weilhart betreut.

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

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

Ralf Adams, September 2023

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 vorkommen[1]:

- *Tabellen:* Bei einer relationalen Datenbank werden die Daten in Tabellen organisiert (Kundentabelle, Artikeltabelle, Filmtabelle usw.). Deshalb sind die Tabellen das Herz-

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

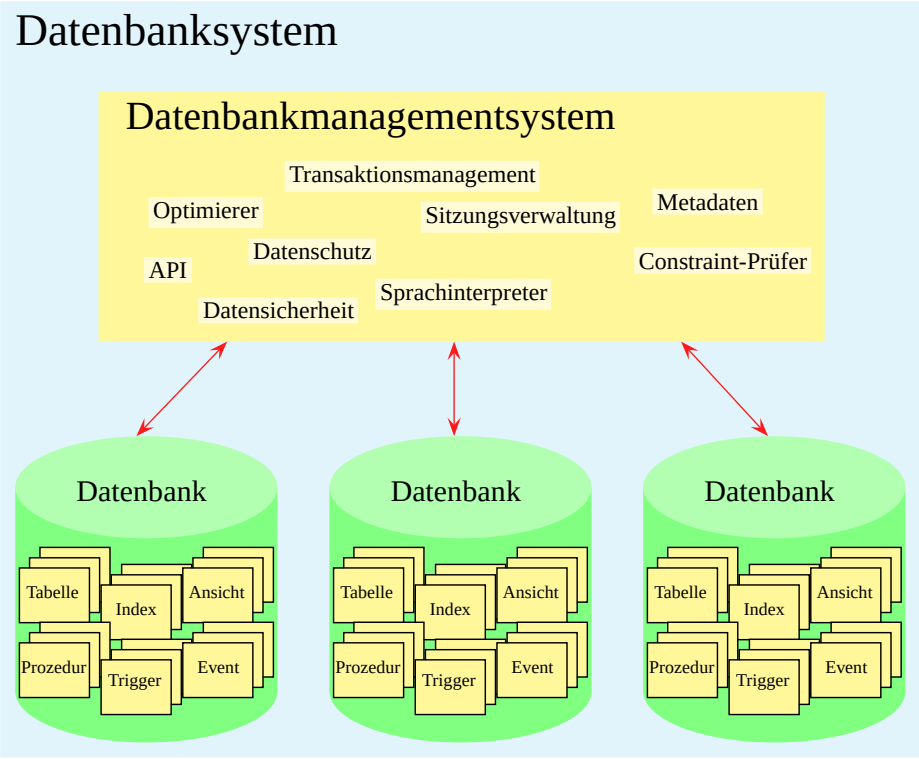

Bild 1.1 Aufbau eines Datenbanksystems

stück einer Datenbank. Alle anderen Datenbankobjekte sind aus diesen Tabellen abgeleitet oder verwenden diese.

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

Dies sind die Aufgaben eines DBMS[2]:

- *Sprachinterpreter*: Herzstück des DBMS ist der Interpreter[3]. 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.

[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 ;-)

- *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 GAU[4] 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 JDBS[6] und ODBC[7] 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*.

[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.

1.2 Im Buch verwendete Server

Kurzporträts der verwendeten SQL Server: Hersteller und Geschichte

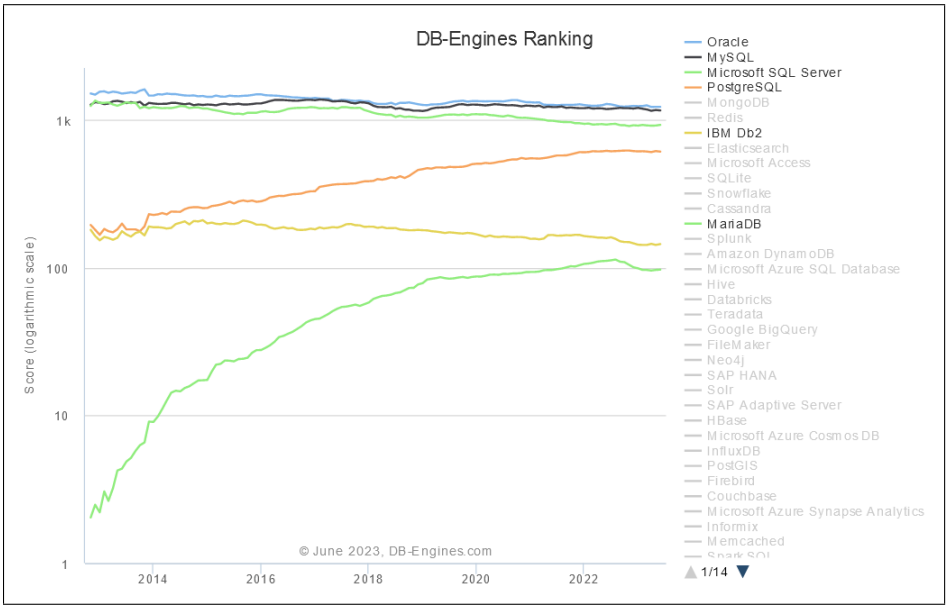

Bild 1.2 Ranking einiger RDBMSe, Quelle [DE23]

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 vom 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 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.

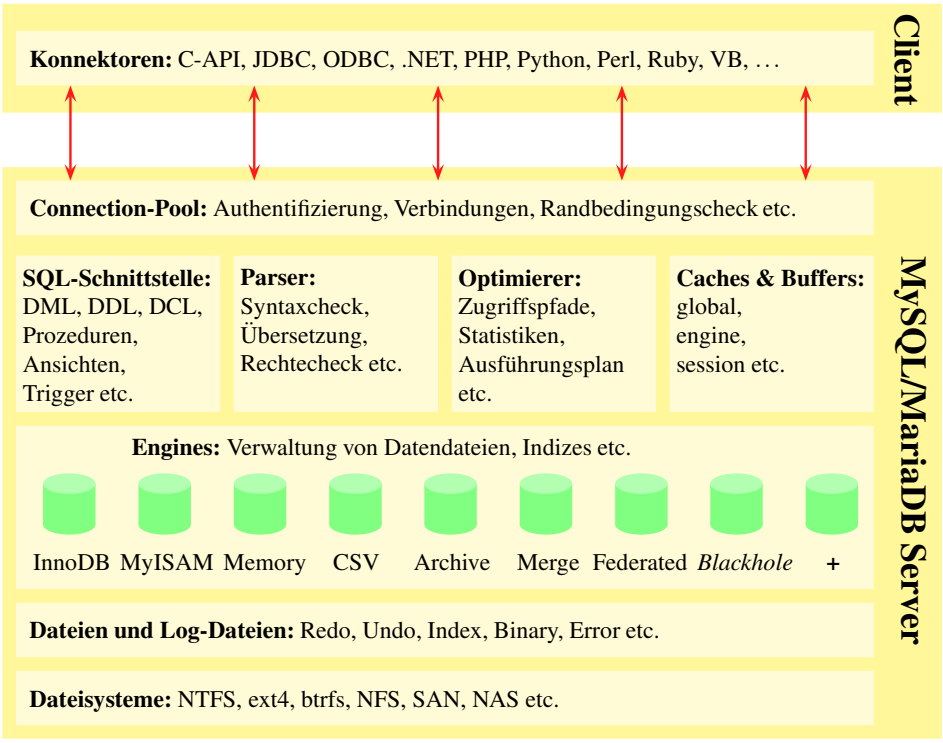

Bild 1.3 Komponenten von MySQL und MariaDB

- *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 wer-

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

- *SQL-Schnittstelle*: Hier werden die SQL-Befehle entgegengenommen. Sie werden dann zum Parser weitergereicht.
- *Parser*: Der Parser überprüft die Syntax eines Befehls und ob man die Ausführungsrechte für diesen Befehl hat.
- *Optimizer* (Optimierer): Anhand von Schätzungen, Statistiken und Algorithmen wird für nicht triviale Befehle ein Ausführungsplan erstellt. Dieser Ausführungsplan berücksichtigt ggf. im Cache vorhandene Ergebnisse.
- *Caches und Buffers* (Zwischenspeicher): Daten und Ergebnisse können in Zwischenspeichern aufgehoben werden.[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, ETX4 etc.) abgelegt. Bis auf die Frage, ob das Dateisystem zwischen Groß- und Kleinschreibung unterscheidet, spielt dieses für die SQL-Programmierung keine Rolle.
- *Management Services & Utilities*: Parallel dazu gibt es die vielen kleinen Helferlein, ohne die nichts geht: für das Sichern und Wiederherstellen von Daten, Datenreplikation, Administration und Konfiguration, Datenmigration und Metadaten.

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 Stonebraker[9] 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,

[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 – Illustra – zu Informix, welche den Fork in den Universal Server integrierte. Auch diese Datenbank besteht heute noch.

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*.

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 der Name.* Nun ist aber der englische Begriff für Beziehung *relationship* und nicht *relation*; somit kann das nicht die Bedeutung sein.

Relation ist ein wohldefinierter Fachbegriff und kann umgangssprachlich als *Tabelle* übersetzt werden (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 Division[1] verwendet wird, angegeben:

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

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

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

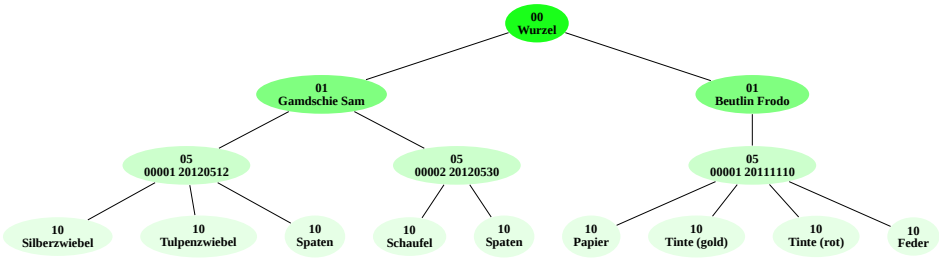

Bild 2.1 Hierarchische Darstellung der Bestelldaten

Die ersten beiden Ziffern sind die sogenannte *Satzkennzeichen*, welches dem verarbeitenden Programm mitteilt, was für eine Art von Information in der Zeile steht. Irgendwo im System müssen diese Satzkennzeichen z.B. in 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] Das Format ist hier vereinfacht und mischt Stufennummern mit Satzkennzeichen. Es lohnt sich, mal einen Blick in die Spezifikation zu werfen: [Wik23d].

```
<?xml version="1.0" encoding="iso-8859-15"?>
<Rechnung>
  <Kunde>
    <Kundennummer>12345</Kundennummer>
    <Nachname>Gamdschie</Nachname>
    <Vorname>Sam</Vorname>
    <Strasse>Beutelhaldenweg</Strasse>
    <Hausnnummer>5</Hausnnummer>
    <Postleitzahl>67676</Postleitzahl>
    <Ort>Hobbingen</Ort>
    <Kundenart>Privatkunde</Kundenart>
    <Bestellung>
      <Bestellnummer>00001</Bestellnummer>
      <Bestelldatum>20120512 12:30:00</Bestelldatum>
      <Status>bezahlt</Status>
      <Position>
        <Artikelnummer>7856</Artikelnummer>
        <Artikelname>Silberzwiebel</Artikelname>
        <Menge>15</Menge>
        <Preis>6.13</Preis>
        <Währung>EUR</Währung>
      </Position>
      <Position>
        <Artikelnummer>7863</Artikelnummer>
        <Artikelname>Tulpenzwiebel</Artikelname>
        <Menge>10</Menge>
        <Preis>32.90</Preis>
        <Währung>EUR</Währung>
      </Position>
     [...]
</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 (*entity*), Objekt, Datensatz (*record*)
Spalte	Attribut, Feld, Datenfeld, Item, Eigenschaft (*property*)
Wertebereich	Domäne
Tabelle	Matrix, Entitätentyp (*entity type*), Relation, Klasse, Datenmenge (*recordset*)
Datenbank	Schema

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

der 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 gelöscht[2].

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

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

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 `kennzeichen`[3] geben. Schnell haben Sie überprüft, dass diese Spalte ein Schlüssel ist. Jetzt gibt es aber auch die Spalte `fahrgestellnummer`[4]. 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 Artikeltabelle erfasst sind. Diese Doppelung der Information nennt man Redundanz:

[3] Siehe [Wik23o].
[4] Siehe [Wik23j].

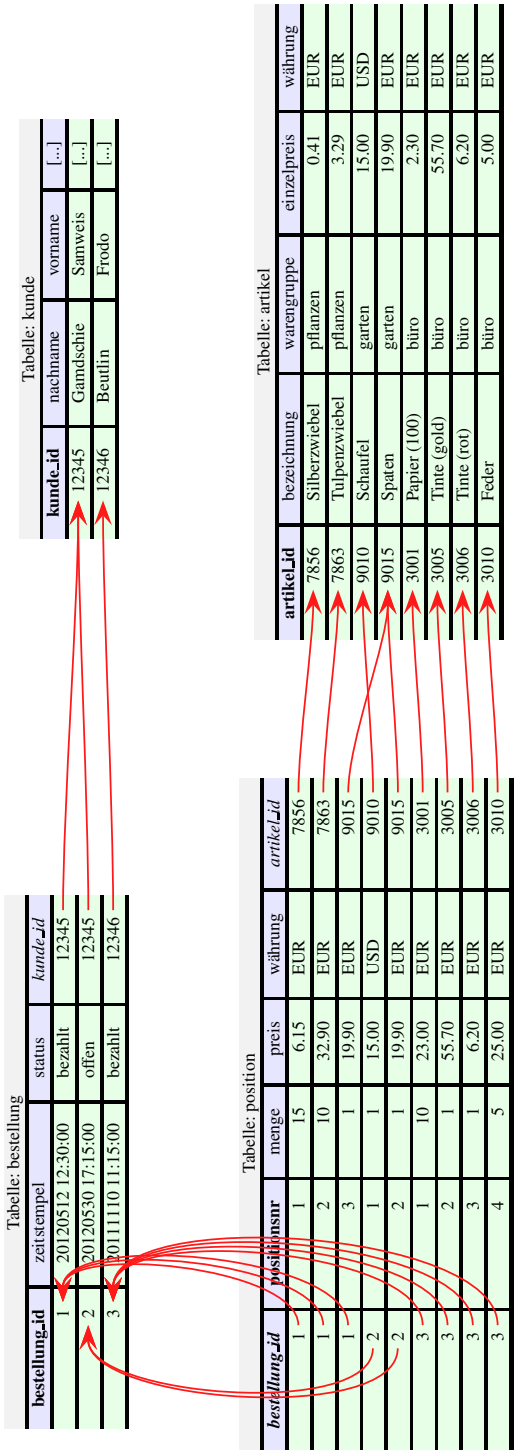

Bild 2.9 Bestellung: Primär-Fremdschlüsselbeziehungen

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: Zum einen verschwendet man Speicherplatz, und zum anderen müssen Änderungen – hier beispielsweise an der Artikelbezeichnung – mehrfach vorgenommen werden (*Wartungsinstabilität*). Auch können unter bestimmten Umständen Performanceverluste auftreten, da mehr Datenmenge verarbeitet werden muss. Wir werden später aber noch sehen, dass Redundanzen auch gewollt und sinnvoll sein können[5].

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.

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

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 einen 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.

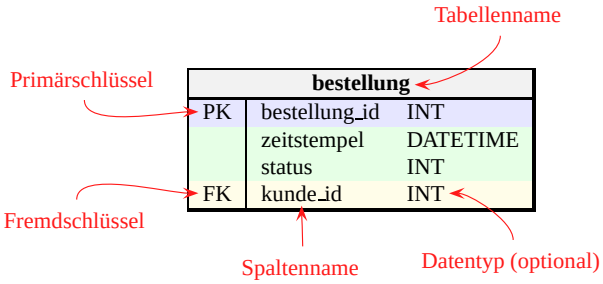

Bild 2.10 Darstellung einer Tabelle in der Krähenfuß-Notation

Wird die Darstellung in CASE-Tools[6] 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.

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

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 dokumentiert[7].

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 einfachste[8].

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).

[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.

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.

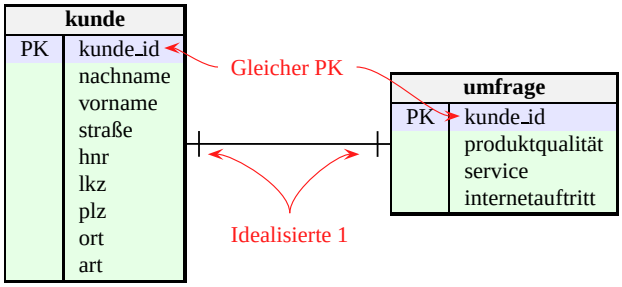

Bild 2.11 Einfache 1:1-Darstellung in Krähenfuß-Notation

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.

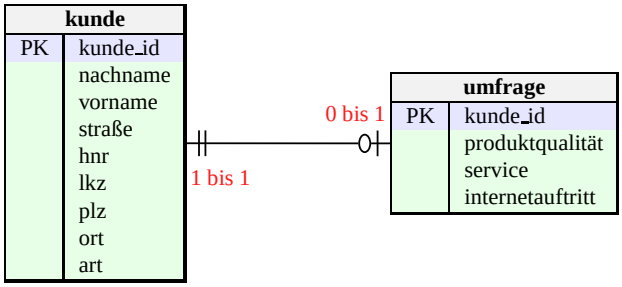

Bild 2.12 Erweiterte 1:1-Darstellung in Krähenfuß-Notation

- 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.

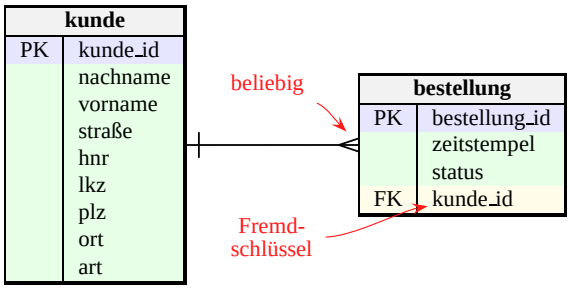

Bild 2.13 Einfache Darstellung der *1:n*-Verknüpfung in Krähenfuß-Notation

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

2.2.3.3 Kann 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 muss[9], würde anstelle der idealisierten 0 wieder ein senkrechter Strich stehen.

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

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.

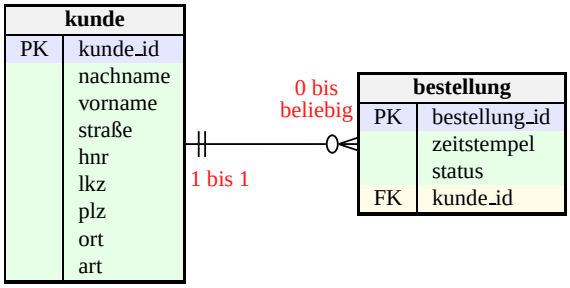

Bild 2.14 Erweiterte Darstellung der *1:n*-Verknüpfung in Krähenfuß-Notation

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.

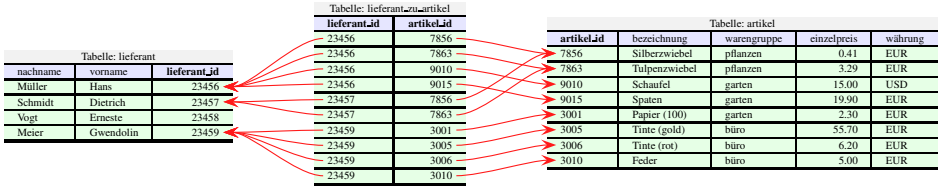

Bild 2.15 Aufbau einer $n:m$-Beziehung

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, so müsste man komplizierte Dinge tun: Die Liste in `lieferant_id` in `artikel` müsste in seine einzelnen Werte aufgeteilt werden. Anschließend muss dieser Lieferant aus der Liste gelöscht und die Liste wieder zusammenmontiert zurückgeschrieben werden. Analoges müsste mit der Liste `artikel_id` in `lieferant` geschehen[10].

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 billiger[11].

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.

[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.

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.

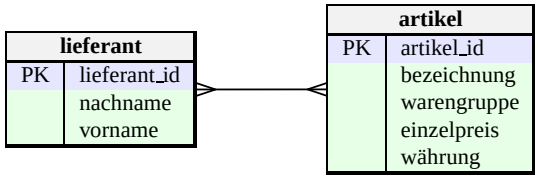

Bild 2.16 Einfache Darstellung der *n:m*-Verknüpfung in Krähenfuß-Notation

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.

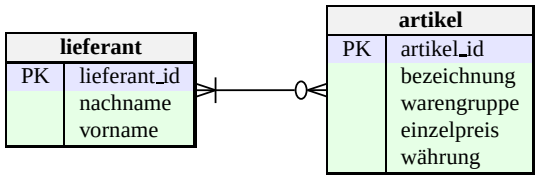

Bild 2.17 Erweiterte Darstellung der *n:m*-Verknüpfung in Krähenfuß-Notation

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 verarbeitet[12].

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 *Änderungsweitergabe* oder *kaskadierende Änderung* verwenden.

Die Folgen sind zwar in der Regel nicht ganz so katastrophal wie bei der Löschweitergabe, können aber auch schon gehörigen Schaden anrichten: Die Kundennummer hat sich geändert. In allen archivierten Schriftverkehren steht aber die alte Kundennummer. Ein Anruf seitens des Kunden könnten dann echte Verwirrung auslösen, weil auf seiner Rechnung eine Kundennummer steht, die es im System gar nicht mehr gibt, oder noch viel schlimmer: nun einem 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 Zeile[13].

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

[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.

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).

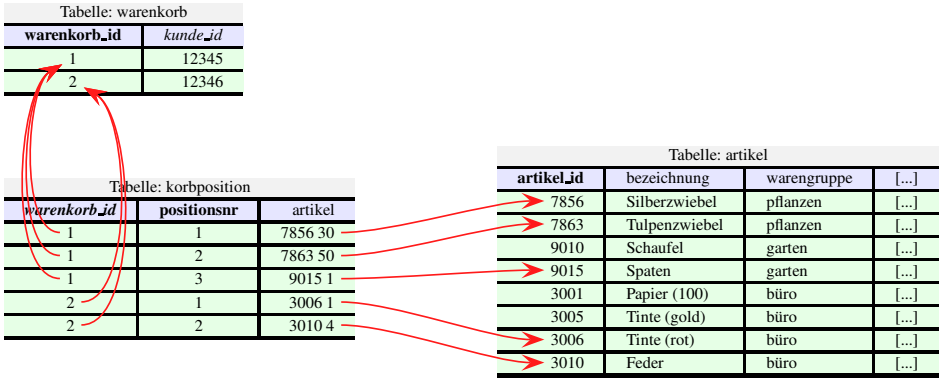

Bild 2.20 Tabelle `warenkorb` ohne Wiederholungsgruppen

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.

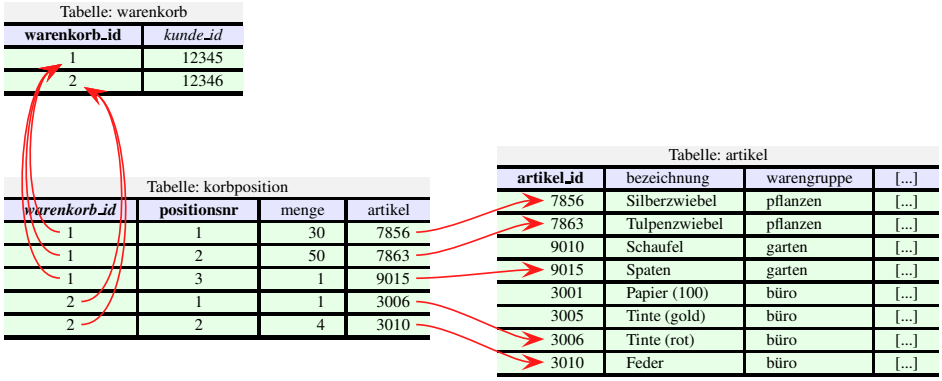

Bild 2.21 Tabelle `warenkorb` in der 1. Normalform

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

kunde_id	**bankverbindung_nr**	kontonr	iban	*blz*
12345	1	1234506789	[...]	50041597
12345	2	5432109876	[...]	50287667
12346	1	5432109880	[...]	50287667

Tabelle: bank

blz	bankname	lkz
50041597	Sparkasse Aulenland	DE
50287667	Volksbank Eriador	DE

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 an-

deren, 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-Codd[14]-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 Literatur[15] 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.

[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

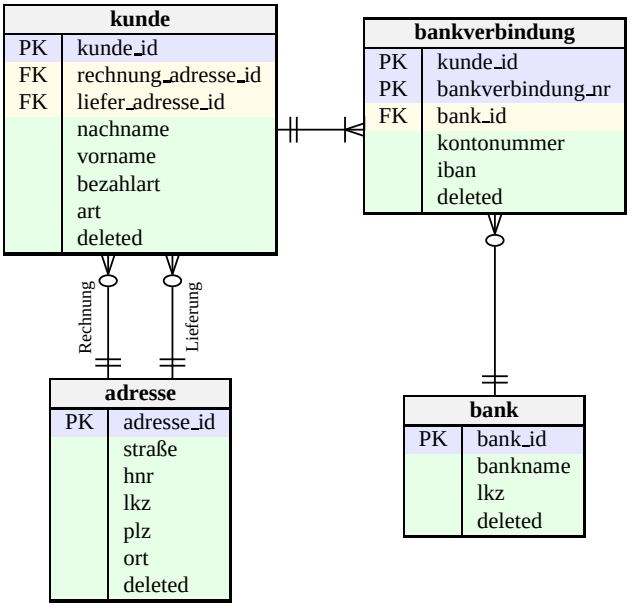

Bild 3.1 ER-Modell: Kundenverwaltung

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

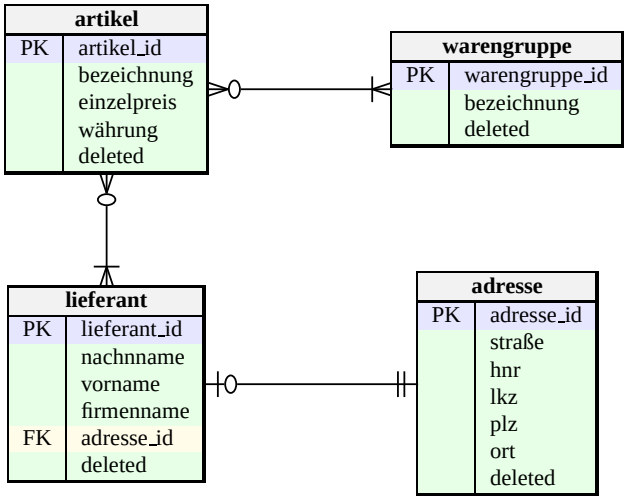

Bild 3.2 ER-Modell: Artikelverwaltung

eine *1:1*-Verknüpfung, die nicht wie oben in Abschnitt 2.2.2.1 beschrieben, durch einen gemeinsamen Primärschlüssel gelöst wird.

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

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.

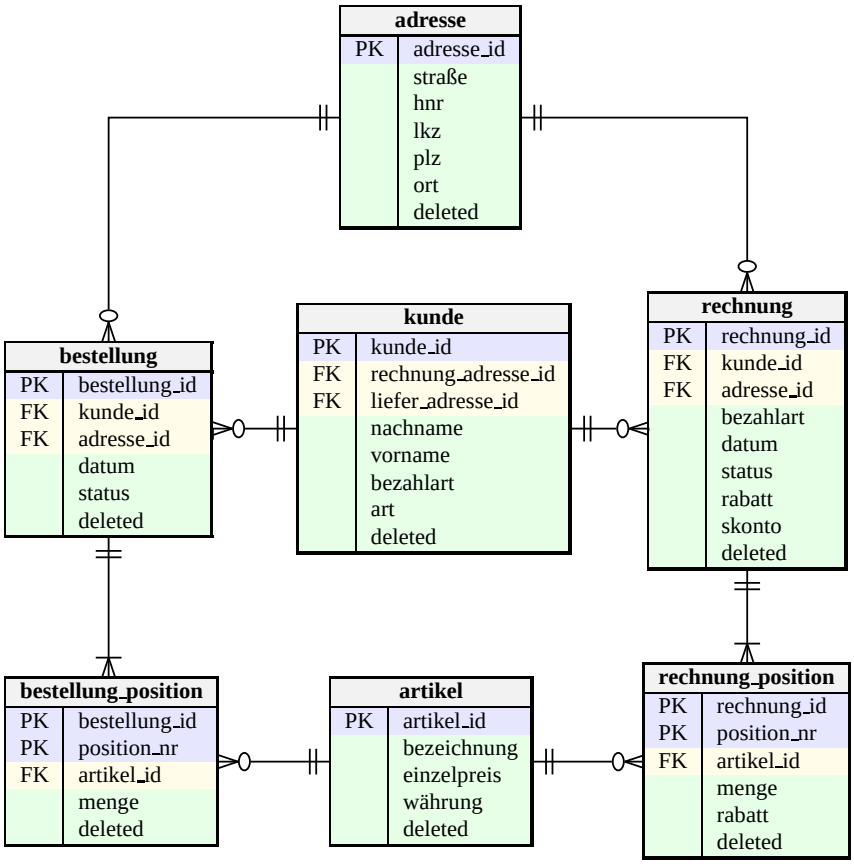

Bild 3.3 ER-Modell: Bestellwesen

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.

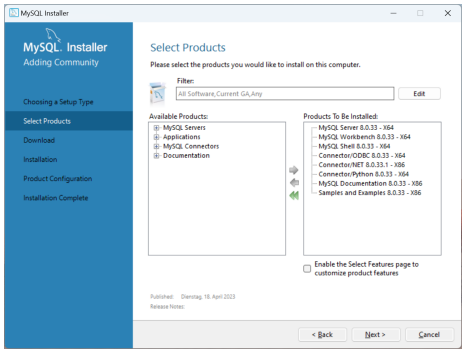

Bild 4.1 Schritt 2: Produktauswahl

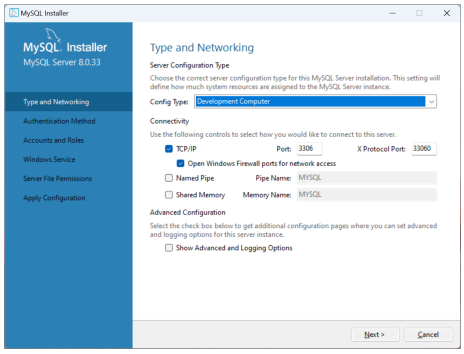

Bild 4.2 Schritt 4: Servertyp und Netzwerk