

DBI11

Theoretische Grundlagen von DB-Systemen

**Prof. Dr. rer. nat.
Matthias Schubert**

Das Studienheft und seine Teile sind urheberrechtlich geschützt. Jede Nutzung in anderen als den gesetzlich zugelassenen Fällen ist nicht erlaubt und bedarf der vorherigen schriftlichen Zustimmung des Rechteinhabers. Dies gilt insbesondere für das öffentliche Zugänglichmachen via Internet, Vervielfältigungen und Weitergabe. Zulässig ist das Speichern (und Ausdrucken) des Studienheftes für persönliche Zwecke.

Falls wir in unseren Studienheften auf Seiten im Internet verweisen, haben wir diese nach sorgfältigen Erwägungen ausgewählt. Auf Inhalt und Gestaltung haben wir jedoch keinen Einfluss. Wir distanzieren uns daher ausdrücklich von diesen Seiten, soweit darin rechtswidrige, insbesondere jugendgefährdende oder verfassungsfeindliche Inhalte zutage treten sollten.

Theoretische Grundlagen von DB-Systemen

Inhaltsverzeichnis

Vorwort	1
Kapitel 1	
1 Warum braucht man Datenbanksysteme?	3
1.1 Ein privates Beispiel	3
1.2 Ein öffentliches Beispiel	6
Aufgaben zur Selbstüberprüfung	7
Kapitel 2	
2 Relationale und andere Datenbanksysteme	9
2.1 Tabellen und Relationen	9
2.2 Ein erster Tabellenentwurf für unser Adressbuch	10
2.3 Eine genauere Festlegung unserer Begriffe	10
2.4 Eine erste Charakterisierung relationaler Datenbanksysteme	12
Aufgaben zur Selbstüberprüfung	13
Kapitel 3	
3 Eine Beispieldatenbank	14
3.1 Das relationale Modell – eine erste vorläufige Charakterisierung	14
3.2 Primärschlüssel	14
3.3 Primärschlüssel als Referenzen	16
3.4 Warum Primärschlüssel?	18
3.5 Attribute: Definition und Eigenschaften	18
3.6 Ein unerfreuliches Thema: NULL-Werte	19
3.7 Primärschlüssel, Fremdschlüssel und weitere Bedingungen	20
3.8 Die restlichen Tabellen der Datenbank „LieferBar“	21
3.8.1 Die Tabelle PERSON	21
3.8.2 Die Tabelle KUNDE	23
3.8.3 Die Tabelle LIEFERANT	23
3.8.4 Die Tabelle BESTELLUNGEN	23
3.8.5 Die Tabelle ERLEDIGTEBESTELLUNGEN	25
3.9 Die Beziehungen zwischen den Tabellen der Datenbank „LieferBar“	25
Aufgaben zur Selbstüberprüfung	26

Kapitel 4

4	Tabellen und Relationen: Eine produktive Kontroverse.....	28
4.1	Beispiele für Mengen und das Kreuzprodukt.....	28
4.2	Relationen: Die Auswahl „sinnvoller“ Tupel aus einem Kreuzprodukt.....	32
4.3	Datenbankrelationen können mehr als Tabellen.....	36
4.4	Tabellen können mehr als Datenbankrelationen.....	37
4.5	Konsequenzen für Abbildungen und Operatoren	38
4.6	Nomenklatur.....	39
4.7	Das relationale Modell – Die abschließende Charakterisierung	39
	Aufgaben zur Selbstüberprüfung	40

Kapitel 5

5	Relationale Operatoren als Grundlage aller manipulativen Operationen	41
5.1	Unsere grundlegende Definition und einige Bemerkungen.....	41
5.2	Acht relationale Operatoren	42
5.2.1	Der Operator UNION oder: Die Vereinigung	42
5.2.2	Der Operator INTERSECTION oder: Der Durchschnitt.....	43
5.2.3	Der Operator DIFFERENCE oder: Die Differenz.....	43
5.2.4	Der Operator PRODUCT oder: Das kartesische Produkt.....	44
5.2.5	Der Operator RESTRICT oder: Die Restriktion.....	47
5.2.6	Der Operator PROJECT oder: Die Projektion.....	49
5.2.7	Die JOIN-Operatoren oder: Verbindungen von Tabellen.....	51
5.2.8	Der DIVIDE-Operator	55
5.3	Der Nutzen dieser Operatoren	55
5.4	Anforderungen an eine Datensprache.....	56
	Aufgaben zur Selbstüberprüfung	56

Kapitel 6

6	Die Integrität einer Datenbank und Schlüssel aller Art.....	58
6.1	Schlüsselkandidaten (Candidate Keys)	58
6.2	Primärschlüssel.....	59
6.3	Fremdschlüssel und referentielle Integrität.....	60
6.4	Regeln beim Umgang mit Fremdschlüsseln	64
6.5	Ein Verbot für NULL-Werte – aber nur in Primärschlüsseln	65
6.6	Abschließende Definition der Integrität einer relationalen Datenbank	66
	Aufgaben zur Selbstüberprüfung	68

Kapitel 7

7	Das Entity/Relationship-Modell	69
7.1	Unser Plan	69
7.2	Entitäten und Entitätstypen	70
7.3	Eine Analyse unseres Versandhauses „LieferBar & Co“	73
7.4	Die Abbildung der Eigenschaften im E/R-Diagramm	74
7.5	Beziehungen zwischen Entitätstypen mit Tabellenentwürfen ...	74
7.6	Der Datenbankentwurf mithilfe des E/R-Modells	87
7.7	Das allgemeine Vorgehen beim Datenbankentwurf	88
	Aufgaben zur Selbstüberprüfung	89

Kapitel 8

8	Normalisierungen	90
8.1	Funktionale Abhängigkeit	90
8.2	Die Erste Normalform	91
8.3	Die zweite Normalform	91
8.4	Die Dritte Normalform und die Boyce/Codd-Normalform	93
	Aufgaben zur Selbstüberprüfung	95

Anhang

A.	Lösungen der Aufgaben zur Selbstüberprüfung	98
B.	Literaturverzeichnis	105
C.	Abbildungsverzeichnis	106
D.	Tabellenverzeichnis	108
E.	Sachwortverzeichnis	110
F.	Einsendeaufgabe	113

Vorwort

Die Menge der Daten, die in einzelnen Anwendungen, auf einzelnen Rechnern, in Netzwerken, in verteilten Anwendungen verarbeitet werden muss, wächst sekundlich. Daten müssen schnell gefunden, schnell bereitgestellt und ausgewertet, schnell verdichtet und synchronisiert werden, denn auf ihrer Grundlage werden wichtige, manchmal lebenswichtige Entscheidungen getroffen. Daten bilden die Grundlage jeden Online-Handels; unser Finanzsystem ist ohne Daten nicht zu denken. Steuerbehörden, Einwohnermeldeämter, Touristikunternehmen, Banken, Krankenhäuser, Navigationssysteme – sie alle haben als Grundlage ihrer Arbeit eine Datenbasis, die in Datenbanken organisiert ist.

Eine Informatikerin oder ein Informatiker muss mit Datenbanken umgehen können. Was bedeutet das und was muss er dazu lernen? Und wie muss sie oder er es lernen?

Sie müssen natürlich zunächst einige Grundlagen über Datenbanken lernen und wir müssen ein gemeinsames Vokabular für weitergehende Untersuchungen erarbeiten. Das werden wir im ersten Kapitel tun.

Insbesondere müssen Sie lernen, was relationale Datenbanken sind. Das ganze Heft handelt von diesem Thema; wir beginnen diese Diskussion in Kapitel 2.

Eine der wichtigsten Qualifikationen einer Informatikerin oder eines Informatikers ist die Fähigkeit zu einem guten Entwurf einer Datenbankstruktur, mit der die Daten einer konkreten Anwendungssituation möglichst optimal verwaltet und erfasst werden. In diesem Heft sollen Sie lernen, was die Kriterien für „optimal“ sind und wie man solche Entwürfe erstellen kann. Dazu stelle ich Ihnen im dritten Kapitel erst einmal eine fertige Datenbank vor.

Dann besprechen wir im vierten Kapitel die Theorie der zentralen Bausteine der relationalen Datenbanken, die Relationen und Tabellen.

Und im fünften Kapitel werden dann die theoretischen Grundlagen behandelt für alle Abfrageoperationen, die Sie später auf relationalen Datenbanken durchführen werden.

Daten müssen – sowohl bei der Erfassung als auch noch nach vielen Jahren der Bearbeitung, nach Änderungen aller Art – korrekt bleiben. Bei relationalen Datenbanken stellt sich, wie Sie sehen werden, dieses Problem auf eine besondere Weise. Darum gibt es hier auch zu diesem Zweck ein gutes und mächtiges Instrumentarium an Begriffen und Mechanismen zur Sicherung dieser Korrektheit (bei relationalen Datenbanken spricht man von „Konsistenz“). Das ist das Thema unseres sechsten Kapitels.

Endlich sprechen wir über den eigentlichen Entwurf. Wir beginnen damit im siebten Kapitel, wo wir Analysetechniken und erste Entwurfsschritte diskutieren.

Vorwort

Im abschließenden achten Kapitel lernen Sie dann Tests und Überprüfungsverfahren kennen für Entwurfsentscheidungen, die Sie treffen wollen oder getroffen haben.

Bitte nehmen Sie die Diskussion der Beispiele und die Aufgaben zur Selbstüberprüfung sehr ernst. Ohne eigene Versuche und eigenes Nachdenken über Probleme kann man einen solchen Stoff nicht lernen. Im Anhang finden Sie Lösungen zu allen Aufgaben.

Und nun viel Spaß und Erfolg beim Erlernen eines sehr interessanten und wichtigen Gebiets aus der Informatik!

Ihre Studienleitung

Kapitel 1

1 Warum braucht man Datenbanksysteme?

Wir untersuchen zuerst, wozu man eigentlich Datenbanksysteme braucht. Während wir noch überlegen, was an Notizbüchern bzw. großen Aktenansammlungen schlecht ist oder welche Mängel auch solch mächtigen Werkzeuge wie Excel-Listen in bestimmten Situationen haben, bekommen wir einen ersten Eindruck, worum es sich bei Datenbanksystemen eigentlich handelt.

1.1 Ein privates Beispiel

Betrachten Sie ein „gutes, altes“ papierenes persönliches Adressbuch. Dort steht vielleicht:

Inspektor Cluseau, Luisenstr. 6, 53024 Bonn, 0228/922323

und

Edith Piaf, Rue Rivoli 2, Paris, 03278/6543

und noch vieles mehr.

Solch ein persönliches Adressbuch war sehr wichtig, es durfte keinesfalls verloren gehen, Rekonstruktionsversuche waren meistens sehr unvollständig und voller Irrtümer.

Auch eine Datenbank mit Adressdaten auf einem Rechner muss dauerhaft gespeichert werden. Das wird unsere erste Forderung an eine Datenbank:

Die Daten müssen persistent gespeichert werden.



Die Eigenschaft der Persistenz ist beiden Adressbüchern – dem papierenen und dem digitalen – eigen. Was aber sind die Vorteile der elektronischen Speicherung?

Da ist zunächst die Möglichkeit komfortabler Datensuche:

In den handschriftlichen Adressbüchern müssen Sie sich für genau eine Sortierung der Einträge entscheiden: meistens alphabetisch nach Nachnamen. Viele teilen ihre Kontakte in Gruppen auf: dienstliche und private Adressen, Kontakte aus Inland und Ausland usw. Der Sinn dieser Maßnahmen besteht natürlich darin, seine Einträge jeweils möglichst schnell und komfortabel finden zu können. Für unsere Datenbank bedeutet das:

Es muss die unterschiedlichsten Suchzugriffe auf den Datenbestand geben.



Ich möchte meine Datenbank beispielsweise fragen können:

- Zeige mir den Eintrag zu meinem Freund Charlie Chaplin.
- Zeige mir alle Einträge, wo der Ort „Regensburg“ ist.
- Zeige mir, ob es in meiner Datenbank einen Eintrag mit der Telefonnummer „069/123456“ gibt.
- Zeige mir alle Einträge – sortiert nach Namen.
- Zeige mir alle Einträge – sortiert nach Wohnorten, Straßen und Hausnummern.
- *Und vieles andere mehr ...*

Je komfortabler die Suchfunktionalitäten sind, über die eine Datenbank verfügt, desto mehr ist sie dem papierenen Adressbuch überlegen und desto klarer wird es einem potenziellen Benutzer, dass eine elektronische Datenspeicherung sinnvoll und vorzuziehen ist.

Außerdem möchte man eine Datenbank durchsuchen und verändern können – einfach dadurch, dass man ihr das gewünschte Ergebnis mitteilt. Das ist ein völlig anderer Standpunkt als der, den Sie beim Programmieren in einer der Standardprogrammiersprachen wie Java, C++ oder C# einnehmen. Dort würden Sie programmieren:

1. Lies den „ersten“ Eintrag der Datenbank.
2. Prüfe, ob der Eintrag angezeigt werden soll, und zeige ihn gegebenenfalls an.
3. Lies den nächsten Eintrag der Datenbank.
4. Prüfe, ob der Eintrag angezeigt werden soll, und zeige ihn gegebenenfalls an.
5. Und so weiter ...

Genauer gesagt bedeutet das also:



Ich brauche **Auswahl-** bzw. **Suchfunktionen** für meine Datenbank, mit denen ich durch Beschreibung des gewünschten Ergebnisses die gewünschten Anzeigen erhalte. Des Weiteren brauchen wir sogenannte **Verwaltungsfunktionen** für unsere Daten, die ebenfalls ergebnisorientiert formuliert werden.

Denken Sie daran, was Sie alles mit einem Adressbuch bzw. den darin enthaltenen Daten machen können:

- Sie können sich ein neues Adressbuch kaufen, das zunächst völlig leer ist.
- Sie können neue Einträge in dieses Adressbuch **einfügen**.
- Sie können einzelne Einträge in diesem Adressbuch **verändern**.
- Sie können Einträge herausstreichen, d.h. **löschen**.
- Sie können das gesamte Adressbuch **vernichten** oder **wegwerfen**.

All dies soll auch mit den Daten Ihrer Datenbank möglich sein.

Ein weiteres Beispiel:

Die telefonische Vorwahl eines Ortes ändere sich beispielsweise von 01234 in 09876. Dann müssten Sie Ihre handschriftlichen oder anderweitigen papierenen Aufzeichnungen mühsam durchgehen, um diese Änderungen bei allen Positionen vorzunehmen. Und natürlich werden Sie Fälle übersehen oder vergessen. Ihrer Datenbank hingegen wollen Sie lediglich mitteilen können:

- Ändere alle Einträge mit der Vorwahl 01234 in Einträge mit der Vorwahl 09876.

Bedenken Sie bitte: Auch hier kommunizieren Sie mit Ihrer Datenbank ergebnisorientiert.

Damit alle diese Funktionen eine Chance haben zu funktionieren, müssen aber einige andere Dinge so streng wie möglich beachtet werden.

Ich selber wohne in Frankfurt am Main und habe in meinem Adressbuch viele Einträge von Personen aus dieser Stadt. Wenn ich für alle diese Personen immer wieder neu die Angabe „Frankfurt am Main“ durchführen müsste, kämen unter anderem sicherlich die folgenden Varianten vor:

Frankfurt am Main, Frankfurt/Main, Frankfurt, Frankfurt a.M., FFM, Frnkft.

Bei einer solchen Vielfalt der Darstellung ein- und derselben Eigenschaft bzw. ein- und desselben Attributs haben natürlich alle oben beschriebenen Such- und Verwaltungsfunktionen keine Chance der Realisierung. Wie wollen Sie hier z. B. Ihrer Datenbank mitteilen: „Zeige mir alle Personen aus Frankfurt“?

Die Lösung dieses Problems besteht darin, dass es uns gelingen muss, unsere Datenbank so zu konzipieren, dass der Text „Frankfurt am Main“ nur ein einziges Mal in ihr gespeichert wird und bei allen Personen aus Frankfurt auf diesen einen Eintrag verwiesen wird. Durch ein solches Vorgehen vermeidet man die **Redundanz** (d.i. das mehrfache unterschiedliche Vorkommen derselben Sache) von Daten und damit mögliche Fehlerquellen.

Das ist bei einfachen Listen oder bei Excel-Tabellen nicht möglich. Hier sehen Sie einen entscheidenden Unterschied zwischen Listen und Datenbanksystemen, der enorme Konsequenzen hat.

Um die Sache noch komplizierter zu machen:

Eine Adresse ist beispielsweise unter dem Vornamen einsortiert worden und wird außerdem auch noch unter dem Nachnamen eingetragen. Vielleicht hat man ihn zu oft am falschen Platz gesucht. Also:

Unter „C“ steht: Connery, Sean,, Handy: 0175/007

Unter „S“ steht: Sean Connery,, Handy: 0175/007

Nun ändert sich bei dieser Person beispielsweise die Handynummer. Dann kann es leicht passieren, dass man die Änderung nur bei einem Eintrag vornimmt. Ihr Adressbuch enthält dann die Daten:

Unter „C“: Connery, Sean,, Handy: 0175/0815

Unter „S“: Sean Connery,, Handy: 0175/007

Was Sie jetzt haben nennt man Dateninkonsistenz und ist – gerade im kommerziellen Bereich – der Untergang jeder korrekten Datenhaltung. Diese Inkonsistenz kann ebenfalls durch die Vermeidung von Datenredundanz, in diesem Fall das mehrfache Abspeichern ein und derselben Person, verhindert werden.

Wir fordern also von unserer Datenbank:



Sie soll so konstruiert sein, dass man die Daten redundanzfrei speichern kann.

1.2 Ein öffentliches Beispiel

Wenn Sie z. B. bei einem Online-Händler ein Buch suchen und bestellen wollen, greifen Sie auf eine Datenbank dieses Online-Händlers zu, in der die Bücher dieses Händlers mit allen möglichen Eigenschaften gespeichert sind. Gleichzeitig mit Ihnen greifen möglicherweise noch viele andere potenzielle Kunden ebenfalls auf diese Datenbank und auch auf exakt dieselben Informationen zu. Unter Umständen nimmt zur selben Zeit gerade der Verantwortliche für diese Datenbank bei dem Online-Händler das Buch, das Sie ansehen, aus der Datenbank heraus oder verändert seinen Preis oder die Verfügbarkeitsangaben für dieses Buch. All das muss gleichzeitig möglich sein. Und es muss ohne Systemprobleme und ohne unlogische Präsentationsabläufe vonstatten gehen können.

Unsere Forderung lautet:



Ein Datenbanksystem muss für einen **Mehr-Benutzer-Betrieb** ausgelegt sein.

Zusammenfassung

Wir haben uns an mehreren Beispielen klargemacht, welche Eigenschaften ein Anwender von einem Datenbanksystem erwartet:

- Es muss einen Kern von persistent gespeicherten Daten enthalten.
- Auf diese Daten müssen die verschiedenartigsten Suchzugriffe möglich sein.
- Diese Daten müssen verändert werden können. Es muss komfortable Einfüge- Veränderungs- und Löschooperationen geben.
- Redundanz der Datenhaltung und die Gefahr von Dateninkonsistenzen müssen soweit wie möglich vermieden werden.
- Es muss die Möglichkeit gleichzeitiger, konkurrierender Zugriffe von mehreren Seiten auf den Datenbestand geben.

Aufgaben zur Selbstüberprüfung

1.1 In einer Datenbank seien eine Datei ADRESS mit Adressdaten und eine leere Datei mit derselben Struktur gegeben. Wir nennen die leere Datei ERGEBNIS. Betrachten Sie die folgende Verarbeitung:

- Lies den ersten Satz der Datei ADRESS.
- Wiederhole die folgende Verarbeitung, solange noch nicht das Ende der Datei erreicht wurde:
 - Falls der Satz die Daten einer Person enthält, die in *Frankfurt* wohnt, kopiere diesen Satz in die Datei ERGEBNIS.
 - Lies den nächsten Satz in der Datei ADRESS
- Gib die Daten der Datei ERGEBNIS aus.

Beschreiben Sie diese Verarbeitung mit **einem** ergebnisorientiert formulierten Befehl.

1.2 Es sei wieder eine Adressdatei ADRESS gegeben. Beschreiben Sie die Verarbeitungen, die hinter dem ergebnisorientiert formulierten Befehl „Zeige mir die Daten von Sean Connery“ stecken, wie Beispiel aus Aufgabe 1.1.

1.3 In einer Bibliothek will man eine Datenbank einrichten, in welcher der Buchbestand verwaltet wird. Sehen Sie in der Tabelle 1.1 die Datei, die man dazu implementiert. In der Struktur dieser Datei finden sich mehrere Entwurfsfehler, die zu einer erheblichen Redundanz bei der Speicherung von Daten führen können. Beschreiben Sie diese Entwurfsfehler.

Tabelle 1.1: Bibliotheksdaten

Id	Autor	Titel	Verlag
12	Goll	JAVA als erste Programmiersprache	Teubner
14	Johnson	Entwurfsmuster	Addison-Wesley
16	Weiß	JAVA als erste Programmiersprache	Teubner
17	Gamma	Entwurfsmuster	Addison-Wesley
25	Zschiegner	Diskrete Mathematik	Vieweg
76	Dieker	Datenstrukturen und Algorithmen	Teubner
136	Beutelspacher	Diskrete Mathematik	Vieweg
183	Beutelspacher	Kryptologie	Vieweg
212	Helms	Entwurfsmuster	Addison-Wesley
815	Müller	JAVA als erste Programmiersprache	Teubner

- 1.4 (Fortsetzung von Aufgabe 1.3) Was für einen Dateientwurf würden Sie dem Datenbankverantwortlichen dieser Bibliothek stattdessen empfehlen? Hinweis: Man **muss** hier mit mehreren Dateien arbeiten.
- 1.5 In einer Lagerbestandsdatei einer Firma werden für die Artikel des Artikelsortiments dieser Firma die Bestandsmengen dieser Artikel geführt. Auf diese Datei haben verschiedene Mitarbeiter von verschiedenen Rechnern aus Zugriff. Ich nenne jetzt eine Veränderung von Daten dieser Datei eine Buchung.
- Entwerfen Sie einige Szenarien, bei denen durch konkurrierende Buchungen falsche Bestandszahlen in der Datei gespeichert werden, bzw. sogar gar nicht (mehr) existierende Artikel in ihren Bestandszahlen verändert werden oder existierende Artikel nicht gefunden werden.
 - Stellen Sie Überlegungen an, wie man sich gegen solche fehlerhaften Abspeicherungen schützen kann.

Kapitel 2

2 Relationale und andere Datenbanksysteme

Wir lernen einige Begriffe aus der Theorie der Datenbanksysteme, insbesondere der relationalen Datenbanksysteme. Wir untersuchen den Begriff der Tabelle und geben eine erste Charakterisierung relationaler Datenbanken.

2.1 Tabellen und Relationen

In der Mathematik ist eine Funktion eine Abbildung $f: A \rightarrow B$ von der Menge A in die Menge B , wobei f jedem Element $x \in A$ genau ein Element $f(x) \in B$ zuordnet. Funktionen sind auch Relationen, ganz offensichtlich besteht eine Relation zwischen dem Wert x und dem zugehörigen Funktionswert $f(x)$. Exakte Definitionen werden wir später untersuchen.

Funktionen kann man auf verschiedene Weise beschreiben:

1. Allgemein durch eine Zuordnungsvorschrift oder Gleichung:

z.B. $f: \mathbb{R} \rightarrow \mathbb{R}$ mit $f(x) = x^2$

2. Durch eine (Werte-)Tabelle, in diesem Fall z.B.

x	$f(x) = x^2$
-2	4
-1	1
0	0
0,5	0,25
1,5	2,25
7	49
usw.	usw.

Also:

Tabellen sind eine Veranschaulichung von Funktionen oder Relationen. In einer Tabelle stehen die Werte, die zu einer Funktion oder Relation gehören.



Genauso – und zwar exakt genauso – ist es bei den relationalen Datenbanken:

Das allgemeine Konzept, das jeweils einer Tabelle zugrunde liegt, ist das einer Relation. Aber man kann solche Relationen nur in Form ihrer Wertetabellen beschreiben, d.h. anhand der Datensätze, die zu solch einer Relation gehören und die man in diese Tabellen einträgt.

2.2 Ein erster Tabellenentwurf für unser Adressbuch

Wir beginnen mit der Frage:

- Welche Eigenschaften haben unsere Adresseinträge? Diese Eigenschaften werden die Spalten unserer Tabelle.

In Kategorien eines Datenbankentwurfs gedacht, müsste man fragen:

- Welche Attribute haben die Datensätze?

Ich beginne mit folgendem Entwurf:

Tabelle 2.1: Ein erster Entwurf einer Adresstabelle

<i>Name</i>	<i>Vorname</i>	<i>Straße</i>	<i>Nr</i>	<i>Plz</i>	<i>Ort</i>	<i>Land</i>	<i>Telefon</i>
Schubert	Matthias	Baumweg	35	60316	Frankfurt	Deutschland	
Mammon	Manfred					Polarkreis	
Mozart	Wolfgang	Tonikastraße	32	A10010	Wien	Österreich	004568
...

Sie sehen: Wir müssen jetzt für jeden Datensatz alle in dem Adressbuch insgesamt möglichen Attribute vorsehen – das führt dazu, dass nicht immer alle Spalten der Tabelle gefüllt sind. Beispielsweise möchte man zu vielen Orten die Postleitzahl speichern. Für andere Orte – beispielsweise in anderen Ländern – weiß man die Postleitzahl gar nicht oder es gibt dort nichts Vergleichbares. In solchen Fällen kann dann dieser Spalteneintrag nicht gefüllt werden.

Sie sehen außerdem: Ich habe mir bei diesem Tabellenentwurf noch keinerlei Mühe gegeben, die Datenredundanz zu vermeiden. Niemand hindert mich hier, für „Österreich“ auch einmal „Austria“ zu schreiben, statt „USA“ einfach „Amerika“ usw. Wir werden später Techniken kennenlernen, wie man solche Entwurfsfehler bei Tabellen aufspürt bzw. vermeidet.

2.3 Eine genauere Festlegung unserer Begriffe

Lassen Sie uns etwas genauer betrachten, was wir eigentlich gemacht und welche Begriffe wir verwendet haben.

- Wir wollen Informationen über Personen speichern. Diese Personen bezeichnen wir dabei als **Objekte** oder **Entitäten**.
- Wir müssen nun unsere Objekte so klassifizieren, dass sie alle aus denselben Eigenschaftskomponenten bestehen. Alle unsere Personen haben in unserer gespeicherten Welt die Eigenschaftskomponenten:
 - *Name*
 - *Vorname*
 - *Straße*

- *Hausnummer*
 - *Postleitzahl*
 - *Wohnort*
 - *Land*
 - *Telefonnummer*
- Diese Eigenschaftskomponenten nennen wir **Attribute**.

Definition 1:

Gegeben sei eine Menge von Objekten. Ein **Attribut** ist eine Eigenschaftsart, die alle Objekte dieser Menge besitzen. Jedem Objekt dieser Menge ist genau ein Wert dieses Attributs, der sogenannte **Attributwert** zugeordnet. Ein Attribut muss einen Namen haben, den sogenannten **Attributnamen**. Dieser Attributname muss unter allen Attributnamen der betrachteten Objekte eindeutig sein.

Wir können jetzt schon eine ziemlich genaue Definition des Begriffes der Relation geben:

Definition 2: (erste, noch vorläufige) Definition einer Relation

Eine Menge von Objekten, die durch die Werte einer einheitlichen Attributkombination vollständig und eindeutig beschrieben werden, heißt **Relation**

Weiter stellen wir fest:

- Diese Objekte werden nun in **Tabellen** gespeichert. Eine **Zeile** dieser Tabelle repräsentiert genau ein **Objekt**. Wir nennen solch eine Zeile auch einen **Datensatz**.
- Die **Spalten** dieser Tabelle entsprechen den einzelnen **Eigenschaftskomponenten** der Objekte. Sie repräsentieren die **Attribute**. Die Spaltenüberschriften sind identisch mit den Attributnamen. Für jedes Objekt steht in seiner Zeile der Tabelle in der entsprechenden Spalte der zugehörige Attributwert. Er definiert sich aus dem konkreten Objekt (die Zeile) und der Eigenschaftsart, dem Attribut, das in der Spalte beschrieben wird.

Ein Beispiel:

Mozart	Wolfgang	Tonikastrasse	32	A10010	Wien	Österreich	004568
--------	----------	---------------	----	--------	------	------------	--------

ist eine Zeile aus der oben dargestellten Tabelle, also ein Datensatz und „Wolfgang“ ist beispielsweise der Wert des Attributs **Vorname**:

2.4 Eine erste Charakterisierung relationaler Datenbanksysteme

Es gab und gibt die verschiedensten Ansätze zur Entwicklung von Datenbanken. Seit dem Ende der Siebzigerjahre existieren relationale Datenbanksysteme, die heute sicherlich am meisten auf dem Markt verbreitet sind. Das relationale Modell, das diesen Systemen zugrunde liegt, ist eine der wichtigsten Entwicklungen in der gesamten Geschichte der Datenbanken. Wir werden dieses Modell noch sehr eingehend untersuchen, aber wir können bereits jetzt das **entscheidende Charakteristikum des relationalen Ansatzes** formulieren:



Der Benutzer einer relationalen Datenbank sieht die Daten nur in Form von Tabellen. Alle Operationen, die dem Benutzer einer relationalen Datenbank zur Verfügung stehen (z. B. zum Einfügen, Ansehen, Ändern und Löschen von Daten), operieren auf der Basis von Tabellen. Sie generieren neue (Teil-)Tabellen, löschen diese usw.

Und vergessen Sie bitte nicht, dass Tabellen nichts anderes sind als die konkrete Beschreibung von Relationen mithilfe der einheitlichen Kombination von Attributwerten (man sagt auch: Wertetupel), die man für diese Relationen entworfen hat.

Bei nicht-relationalen Systemen sieht der Benutzer neben den Tabellen stets noch andere oder auch nur andere Datenstrukturen. Ich erwähne als ein Beispiel für ein „vor-relationales“ System die hierarchischen Datenbanksysteme (z. B. IMS von der IBM), in denen der Benutzer vor allem in Baumstrukturen herum navigieren muss. Auch nach dem relationalen Modell hat es noch etliche andere Weiterentwicklungen gegeben. Ein wichtiger Ansatz dazu kommt von den objektorientierten Systemen her.

Zusammenfassung

In diesem Kapitel ging es darum, eine erste Vorstellung von der abstrakten Struktur gespeicherter Daten zu bekommen. Wir wollten mit dem Begriff der Tabelle vertraut werden.

- Wir begannen mit der Beschreibung von Funktionen durch Tabellen, um Sie auf die Beschreibung von Relationen durch Tabellen vorzubereiten.
- Dann wurde eine Tabelle vorgestellt, in der man die Informationen unseres Adressbuches abspeichern kann. Dieser Tabellenentwurf hat noch viele Mängel, aber er zeigt die grundsätzliche Architektur, mit der relationale Datenbanksysteme arbeiten.
- Dann haben wir die dazugehörigen theoretischen Begriffe für dieses Arbeiten mit Tabellen festgelegt:

- Objekte bzw. Entitäten
- Attribute, Attributwerte, Attributnamen
- einheitliche Attributkombinationen
- Relationen
- Tabellen
- Datensätze und Einträge in den Tabellen

Hier erfolgte auch eine erste Definition einer Relation.

- Mithilfe dieser Tabellen wurde schließlich eine erste Charakterisierung relationaler Datenbanksysteme versucht.

Aufgaben zur Selbstüberprüfung

- 2.1 Für unser Adressbuch sind einzelne Personen die Objekte. Für eine Tabelle über die Völkerwanderungsbewegungen zur Zeit des römischen Kaiserreichs sind offensichtlich nicht mehr einzelne Personen, sondern ganze Stämme oder Volksgruppen die „richtigen“ Objekte. Finden Sie Beispiele für die folgenden Analyseergebnisse für eine Tabelle:
- a) Einzelne Buchstaben sind die „richtigen“ Objekte.
 - b) Ganze Wörter sind die „richtigen“ Objekte.
 - c) Bücher sind die „richtigen“ Objekte.
- 2.2 Entwerfen Sie für Ihre private Sammlung von Musik-CDs ein Attributras-ter zur eindeutigen Kennzeichnung jeder CD. Damit haben Sie diese Men-ge zu einer Relation gemacht, die Sie in einer Tabelle darstellen können.
- 2.3 Nehmen Sie an, Sie hätten eine relationale Datenbank für Ihre Adress-tabelle zur Verfügung. Warum widerspricht die Anfrage
- „Wie lautet die Telefonnummer von Karl Engels“
- unserer Charakterisierung relationaler Datenbanksysteme? Wie müsste eine relational korrekte Formulierung dieser Anfrage lauten?

Kapitel 3

3 Eine Beispieldatenbank

Wir werden jetzt eine Beispieldatenbank aufbauen und dabei eine Menge über relationale Datenbanken lernen. Diese Beispieldatenbank werden wir in den folgenden Abschnitten immer wieder benutzen. Die vollständige Liste der Beispielsätze können Sie im Online-Campus „StudyOnline“ herunterladen.

3.1 Das relationale Modell – eine erste vorläufige Charakterisierung

Das **relationale Modell** ist durch die folgenden drei Punkte charakterisiert:



1. Die Daten können nur in Form von Tabellen betrachtet und verändert werden. Das wird oft der **strukturelle Aspekt** genannt.
2. Alle Operatoren, die dem Benutzer zur Manipulation von Daten zur Verfügung stehen, operieren grundsätzlich nur auf Tabellen und ergeben auch wieder neue Tabellen. Ein relationales System enthält mindestens die Operatoren RESTRICT, PROJECT und JOIN. Man nennt diesen zweiten Punkt den **manipulativen Aspekt** des relationalen Modells. (Dieser zweite Punkt wird im fünften Kapitel dieses Heftes behandelt.)
3. Tabellen müssen gewissen **Integritätsbedingungen** genügen. (Dieser dritte Punkt wird bei der Vorstellung unserer Beispieldatenbank erläutert und im sechsten Kapitel genau diskutiert.)

Diese Grundlage von relationalen Modellen für Datenbank-Systeme geht zurück auf **E. F. Codd**, der diese Ideen im Wesentlichen in den Jahren 1969 und 1970 entwickelt hat. Codd arbeitete zu dieser Zeit als Forscher bei IBM.

Wir wollen nun Tabellen definieren und anlegen und müssen uns überlegen, was man alles für eine Tabelle braucht. Wir beginnen mit den Primärschlüsseln.

3.2 Primärschlüssel

Man braucht für eine Tabelle einen Primärschlüssel.



Der **Primärschlüssel** ist ein Attribut oder eine Kombination von Attributen einer Tabelle, das bzw. die für jeden Satz dieser Tabelle eindeutig ist.

Diese Eindeutigkeit muss gewährleistet sein, ganz gleichgültig wie viele Sätze in diese Tabelle noch aufgenommen werden. Es muss also eine „prinzipielle“ Eindeutigkeit vorliegen. Im Falle einer Attributkombination muss diese Eigenschaft der Eindeutigkeit für diese Kombination verloren gehen, sobald ein Attribut aus dieser die Kombination bildenden Menge von Attributen entfernt wird. Man sagt auch: In Bezug auf die Eigenschaft, eindeutig zu sein, muss die Attributkombination minimal sein.

Dies ist allerdings keine Definition eines Primärschlüssels, denn für viele Tabellen wird es nicht nur einen, sondern mehrere Kandidaten für solch einen Primärschlüssel geben. Man nennt solche Attribute bzw. Attributkombinationen übrigens **Schlüsselkandidaten**. Unter diesen verschiedenen Möglichkeiten sucht sich der Tabellenarchitekt, der Designer, einen „Lieblings“-Kandidaten heraus, den er dann gegenüber dem Datenbankmanagementsystem zum Primärschlüssel erklärt.

Welche Kriterien zeichnen einen „guten“ Primärschlüssel aus?

- **Ein Primärschlüssel sollte grundsätzlich keinerlei Information enthalten, die für den Anwender von Bedeutung ist.** Sie können nie sicher sein, dass sich – auch bei unproblematischer Ausgangssituation – Datenbanklogik und Anwenderlogik nicht auseinander bzw. im Widerspruch zueinander entwickeln werden. Die Konsequenz ist: **Man definiert sich zumeist für eine Tabelle ein zusätzliches Schlüsselfeld** – ich nenne es stets *Id* –, in dem man einen **numerischen Primärschlüsselwert** verwaltet.
- Es gibt weitere Gründe, die gegen die Verwendung anwendungsrelevanter Daten in Primärschlüsseln sprechen: In relationalen Datenmodellen kommt der Primärschlüssel eines Satzes oft in anderen Tabellen als Referenz, als Fremdschlüssel vor. So wird z. B. der Primärschlüssel der Tabelle ARTIKEL auch in einer Tabelle BESTELLUNGEN vorkommen, in der die Bestellungen von Artikeln gespeichert sind. Aber benutzerrelevante Daten ändern sich während der Lebenszeit eines Datenobjektes. Jedes Update einer benutzerrelevanten Eigenschaft, die für den Primärschlüssel mit verwendet wird, hätte auch entsprechende Updates in allen anderen Tabellen zur Folge, in denen dieser Primärschlüssel vorkommt. Das gefährdet sowohl die Performance meiner Anwendungen als auch die Sicherung der Integrität der Daten. **Es ist deshalb ganz allgemein das Ziel, Primärschlüsselwerte zu einem konstanten Attribut eines Datensatzes zu machen, dessen Wert sich niemals ändert.**
- **Schließlich sollte die Struktur eines Primärschlüssels möglichst einfach sein,** denn seine potenzielle oder tatsächliche Verwendung als Referenz in anderen Tabellen transportiert auch diese komplexe Struktur in diese anderen Tabellen mit den entsprechenden Konsequenzen für die Performance von Änderungsverarbeitungen und die Komplexität von Sicherungsmaßnahmen für die Integrität des Datenbestandes.

Die folgende Präsentation einer kleinen Datenbank soll das alles ein bisschen klarer machen. Wir werden uns bei Beispielen und konkreten Erläuterungen immer wieder auf diese Datenbank beziehen. Es ist die Datenbank des Versandhauses „LieferBar & Co“. Die vollständige Liste der Beispielsätze können Sie im Online-Campus „StudyOnline“ herunterladen.

3.3 Primärschlüssel als Referenzen

Unser Versandhaus hat ein Artikelsortiment; dieses Sortiment ist in Gruppen aufgeteilt. Betrachten Sie einen Ausschnitt aus dem Artikelstamm in Tabelle 3.1:

Tabelle 3.1: Einige Artikel, geordnet nach ihren zugehörigen Artikelgruppen

Artikelnr	Artikel.Bezeichnung	Artikelgruppe.Bezeichnung	Bestandsmenge	Preis
A0060002	Der Termin	Bücher	153	16,00 €
A0060001	Gödel Escher Bach	Bücher	640	49,90 €
A0020005	DVD-Player	Elektronik-Fachgerät	848	112,34 €
A0010002	Lampenschirme	Kleinteile	669	10,13 €
A0010004	Schrauben	Kleinteile	689	2,39 €
A0090002	Topfpflanzen	Kokolores	356	20,65 €

Wir legen zunächst eine Tabelle für die Artikelgruppen an. Wir werden die Ihnen jetzt noch unbekannten Begriffe wie Feldtyp, Default usw. im Folgenden erklären.

Tabelle 3.2: Struktur der Tabelle ARTIKELGRUPPE

Feldname	Feldtyp	Primärschlüssel	Nullwerte	Default	Referenz
Id	integer	ja	Nein	nein	
Bezeichnung	varchar(30)	nein	Nein	nein	

Diese Tabelle enthält beispielsweise die folgenden Sätze:

Tabelle 3.3: Einige Sätze der Tabelle ARTIKELGRUPPE

Id	Bezeichnung
1	Kleinteile
2	Elektronik-Fachgerät
6	Bücher
9	Kokolores

Von den Artikeln wollen wir die Artikelnummer, die Bezeichnung, einen Verweis auf die zugehörige Artikelgruppe, einen Verweis auf den Lieferanten des Artikels (für den wir natürlich nachher noch eine Tabelle anlegen werden), die Bestandsmenge im Lager und den Verkaufspreis speichern. Betrachten Sie im Folgenden bitte Format und Inhalt der Tabelle ARTIKEL:

Tabelle 3.4: Struktur der Tabelle ARTIKEL

Feldname	Feldtyp	Primär-schlüssel	Null-werte	Default	Referenz
Id	integer	ja	nein	nein	
Artikelnr	varchar(10)	nein	nein	nein	
Bezeichnung	varchar(30)	nein	nein	nein	
Artikelgruppeld	integer	nein	nein	nein	ARTIKELGRUPPE.Id
LieferantId	integer	nein	nein	nein	LIEFERANT.Id
Bestandsmenge	integer	nein	nein	0	
Preis	float(2)	nein	nein	0	

Das folgende Bild zeigt Ihnen einige Beispielsätze:

Tabelle 3.5: Einige Sätze der Tabelle ARTIKEL

Id	Artikelnr	Bezeichnung	Artikelgruppeld	LieferantId	Bestandsmenge	Preis
1	A0010002	Lampenschirme	1	6	669	10,13 €
3	A0020005	DVD-Player	2	15	848	112,34 €
5	A0090002	Topfpflanzen	9	19	356	20,65 €
10	A0060001	Gödel Escher Bach	6	22	640	49,90 €
12	A0010004	Schrauben	1	10	689	2,39 €
15	A0060002	Der Termin	6	22	153	16,00 €

Zur Erläuterung: Beispielsweise gehören (in dieser Datenbank) *Lampenschirme* zur Artikelgruppe *Kleinteile* und *Topfpflanzen* zur Artikelgruppe *Kokolores*. Diese Referenzinformation wird mit dem Primärschlüssel der Tabelle ARTIKELGRUPPE gegeben, der als ein sogenannter Fremdschlüssel auch noch in der Tabelle ARTIKEL steht.

Sie sehen: Falls dieser Primärschlüssel eine anwenderrelevante und damit im Allgemeinen auch Update-anfällige Größe wäre, wären zur Verwaltung unserer ARTIKELGRUPPE – ARTIKEL – Beziehung im Falle von Änderungen viel mehr Tabellenzugriffe nötig. Unsere Verarbeitung wäre fehleranfälliger und weniger performant, die Integrität der Daten wäre gefährdeter.

3.4 Warum Primärschlüssel?

Das sogenannte Datenbankmanagementsystem (DBMS) verlangt die Primärschlüsselauswahl, weil es mit ihr die Integrität der Beziehungen zwischen Tabellen, so wie wir sie im Beispiel der Artikelgruppen und der Artikel kennengelernt haben, sichert.

Ein Beispiel:

Solange in der Tabelle ARTIKEL noch Sätze stehen, die auf die Artikelgruppe mit der Id 1 verweisen, darf der Satz (1, Kleinteile) aus der ARTIKELGRUPPE nicht gelöscht werden. Das sichert das DBMS und dazu braucht es die Primärschlüssel.

Außerdem braucht jeder Programmierer einer datenbankgestützten Anwendung ein möglichst einfaches Identifikationsattribut für einen Datensatz, um Einfüge-, Veränderungs- und Löschverarbeitungen korrekt programmieren zu können. Primärschlüssel mit den Eigenschaften, die wir besprochen haben, eignen sich dafür vortrefflich.

Schließlich werden Primärschlüssel auch noch benötigt, um die verschiedenen Verfahren zur Zugriffsoptimierung auf die Festplatte organisieren zu können.

3.5 Attribute: Definition und Eigenschaften

Attribute müssen gewissen standardmäßig vorgegebenen Datentypen wie **integer**, **character** usw. zugeordnet sein. Vielfach gibt es die Möglichkeit, eigene Datentypen zu definieren. Man kann auch stets noch zusätzliche Eigenschaften wie das Verbot gewisser Werte oder eine Beziehung zu Attributen in anderen Tabellen festlegen.

Erweiterung der Definitionsbeschreibungen für den Begriff Attribut:



Zusätzlich zu den Begriffen **Attributwert** und **Attributname** (siehe Kapitel 2) definieren wir jetzt noch für Attribute einer Tabelle einer relationalen Datenbank den Begriff des **Attributtyps**. Der Attributtyp ist der Datentyp, der für die Variable vorgesehen ist, auf der die entsprechenden Attributwerte gespeichert werden sollen.

Die Standarddatentypen reichen nicht immer aus, um geeignete Wertebereichen festzulegen. Je unflexibler ein relationales DBMS bei der Festlegung der Wertebereiche ist, desto weiter ist es von einem guten relationalen Modell entfernt und desto größer ist der zusätzliche Aufwand, der für „saubere“, für integere Daten getrieben werden muss. Und desto gefährdeter ist ein fehlerfreier Datenbestand.

Ein **Default-Wert** für ein Attribut ist ein Wert, der vom System standardmäßig zugewiesen wird, falls der Benutzer keine Wertzuweisung vornimmt.

3.6 Ein unerfreuliches Thema: NULL-Werte

Es gibt in allen kommerziellen Datenbankmanagementsystemen die Möglichkeit, zuzulassen, dass bei einem Datensatz einer Tabelle für ein Attribut überhaupt nichts eingetragen wird. Es wird also kein (Werte)-Element aus der Wertebereichsmenge ausgewählt. Das nennt man einen NULL-Wert. Diese Abweichung vom relationalen Modell verschafft eine kurzfristige Erleichterung bei Entwurfsentscheidungen. Beispielsweise kann ich in einer Personentabelle ein Attribut *Telefon* vorsehen, in das, sollte eine Person kein Telefon haben oder die Nummer unbekannt sein, kein Eintrag gemacht wird. Diese Entscheidung ist nicht optimal, obwohl sie mir in der Entwurfsphase Arbeit und lästiges Nachdenken erspart. Aber für diesen einmaligen Vorteil muss ich ein (Tabellen)-Leben lang bezahlen:

NULL-Werte verhalten sich in allen Datenmanipulations-Befehlen äußerst widerspenstig. Um NULL-Werte in irgendeinem Datenmanipulations-Befehl mit zu bearbeiten, muss man immer eine Extra-Klausel hinzufügen, die speziell die NULL-werte behandelt. Beispielsweise würden bei einer Abfrage der Art

„Zeige alle Personen, die nicht in Frankfurt wohnen“

nicht die Personen angezeigt, bei denen im Feld *Ort* kein Eintrag wäre. Man müsste formulieren:

„Zeige alle Personen, die nicht in Frankfurt wohnen oder bei denen der Ortseintrag NULL ist“

Sie können sich vorstellen, dass solche Abfragen beliebig kompliziert werden können, je mehr Attribute zum Vergleich herangezogen werden. Und Sie können sich wahrscheinlich auch vorstellen, dass die Anzahl der Gelegenheiten, wo man solche Klauseln einfach vergisst, beliebig groß werden kann.

Zum anderen zeigen einem mögliche NULL-Werte in einer Tabelle oft, dass es besser wäre, das entsprechende Attribut in eine neue Tabelle auszulagern, zu der man dann auf eine geeignete Weise eine Beziehung definiert.

3.7 Primärschlüssel, Fremdschlüssel und weitere Bedingungen

Fremdschlüssel (englisch „**foreign key**“) sind ein spezielles Beispiel für Bedingungen, die man an den Datenbestand einer Datenbank stellt.

Beispiele für Bedingungen an die Daten einer Datenbank sind die Anforderungen:

- *Id* muss stets ein – für die jeweilige Tabelle – eindeutiger Wert sein.
- Jeder Attributwert in der Spalte *ArtikelgruppeId* der Tabelle ARTIKEL muss auch in der Spalte *Id* der Tabelle ARTIKELGRUPPE vorkommen.
- Der Attributwert für *Bezeichnung* darf sowohl in der Tabelle ARTIKELGRUPPE als auch in der Tabelle ARTIKEL nicht leer sein.
- Der *Preis* eines Artikels muss stets ≥ 0 sein.

Das englische Wort für diese Art Bedingungen an die Korrektheit des Datenbestands ist „**Constraints**“. Sie werden diesen Begriff wahrscheinlich häufiger lesen oder hören als dessen deutsche Übersetzung.

Die jeweiligen Felder mit Namen *Id* in ARTIKELGRUPPE und ARTIKEL sind jeweils der Primärschlüssel. Das Feld *ArtikelgruppeId* in der Tabelle ARTIKEL ist dagegen ein Fremdschlüssel. Die englischen Fachbegriffe hierfür sind: **PRIMARY KEY** und **FOREIGN KEY**. Sie bilden wichtige Elemente zur Definition und Sicherung der **Integrität** des Datenbestandes.

Die Integrität einer relationalen Datenbank ist allgemein folgendermaßen definiert:

Definition 3: Definition der Integrität einer relationalen Datenbank

Die **Integrität** einer relationalen Datenbank wird durch die Erfüllung der folgenden drei Bedingungen definiert:

1. Jeder Satz einer Tabelle hat einen eindeutigen Primärschlüsselwert. Man nennt diese Bedingung auch **Entity-Integrität**.
2. Zu jedem Fremdschlüssel in einer Tabelle T1 gibt es einen identischen Schlüsselwert in einer anderen Tabelle T2, die dafür beim Anlegen von T1 festgelegt wurde. Man nennt diese Bedingung auch **referentielle Integrität**.
3. Alle weiteren Bedingungen, die beim Anlegen einer Tabelle festgelegt wurden, sind erfüllt. Mit anderen Worten: Die restlichen **Constraints** sind erfüllt.

Für unser Beispiel der Tabellen ARTIKEL und ARTIKELGRUPPE bedeutet die Fremdschlüsselbedingung konkret:

- Zu jedem Wert in der Spalte *ArtikelgruppeId* in der Tabelle ARTIKEL muss es einen Satz mit genau diesem Wert als Primärschlüssel in der Spalte *Id* in der Tabelle ARTIKELGRUPPE geben.

Etwas Analoges gilt selbstverständlich für das Feld *LieferantId* in der Tabelle ARTIKEL, das sich auf den Primärschlüssel der Tabelle LIEFERANT bezieht.

3.8 Die restlichen Tabellen der Datenbank „Liefer-Bar“

3.8.1 Die Tabelle PERSON

In der Tabelle PERSON werden sämtliche Personen, seien es Lieferanten oder Kunden, abgespeichert, die für das Versandhaus von Bedeutung sind (siehe Tabelle 3.6).

Wir brauchen nun zwei „Spezifikationen“ dieser Personentabelle. Genauer: Wenn wir diese Personen als Objekte einer Klasse, sagen wir der Klasse PERSON, auffassen, dann möchte ich in meiner Datenbank zwei Klassen modellieren, die von dieser Klasse PERSON abgeleitet sind, nämlich die Klasse KUNDE und die Klasse LIEFERANT. Ein entsprechendes UML-Diagramm sehen Sie in Abb. 3.1.

Tabelle 3.6: Struktur der Tabelle PERSON

Feldname	Feldtyp	Primärschlüssel	Nullwerte	Default	Referenz
Id	integer	ja	Nein	nein	
Name	varchar(30)	nein	Nein	nein	
Vorname	varchar(30)	nein	Nein	nein	
Strasse	varchar(30)	nein	Nein	nein	
Nr	varchar(5)	nein	Nein	nein	
Plz	varchar(10)	nein	Nein	'unbekannt'	
Ort	varchar(30)	nein	Nein	nein	
Land	varchar(30)	nein	Nein	'Deutschland'	

Die folgende Tabelle zeigt einige Datensätze:

Tabelle 3.7: Einige Sätze der Tabelle PERSON

Id	Name	Vorname	Strasse	Nr	Plz	Ort	Land
10	Gauss	Carl Friedrich	Primallee	17	65223	Göttingen	Deutschland
12	Hau	Arnold	Hattersheimer Strasse	11	60309	Frankfurt/Main	Deutschland
20	Mouse	Mickey	Barksweg	46	12344	Entenhausen	Disneyland
22	Ekberg	Anita	Via Dolorosa	66	27364	Rom	Italien

Wenn wir diese Beziehung (Ein Kunde „ist eine“ PERSON, ein Lieferant „ist eine“ PERSON) relational abbilden wollen, müssen wir die jeweiligen Primärschlüssel aufeinander beziehen. Die Art dieses Bezugs muss folgenden Anforderungen genügen:

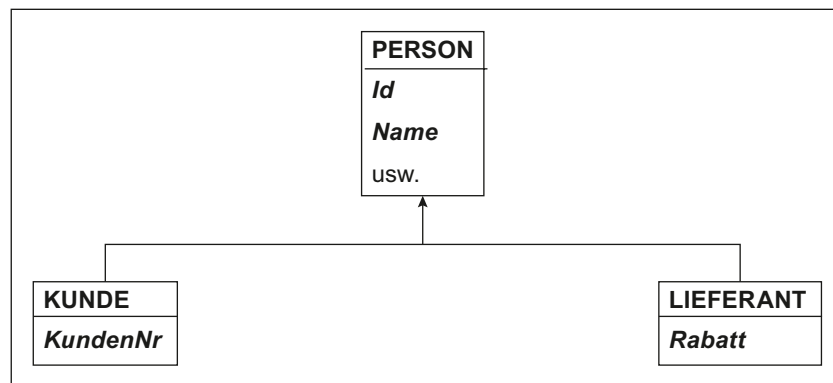


Abb. 3.1: UML-Diagramm zur Vererbungshierarchie der Klassen PERSON, KUNDE und LIEFERANT

- Es darf nicht möglich sein, eine Person zu löschen, solange es Kunden oder Lieferanten mit demselben Primärschlüssel gibt.
- Man kann aber andererseits zulassen, dass es Personen mit Primärschlüsseln gibt, die weder in der Tabelle KUNDE noch in der Tabelle LIEFERANT vorkommen. Mit anderen Worten: Ich will zulassen, dass es Personen gibt, die weder Lieferant noch Kunde sind. (Wer weiß, wozu man sie noch einmal brauchen kann.) Ich brauche also keine Löschoptionen von Kunde/Lieferant in Richtung PERSON festzulegen.

Wir erhalten die folgenden Tabellen:

3.8.2 Die Tabelle KUNDE

Tabelle 3.8: Struktur der Tabelle KUNDE

Feldname	Feldtyp	Primärschlüssel	Nullwerte	Default	Referenz
Id	integer	Ja	nein	nein	PERSON.Id
KundenNr	varchar(15)	Nein	nein	nein	

Ein typischer Datensatz sieht so aus:

Tabelle 3.9: Ein Satz aus der Tabelle KUNDE

Id	KundenNr
12	678912F

3.8.3 Die Tabelle LIEFERANT

Tabelle 3.10: Struktur der Tabelle LIEFERANT

Feldname	Feldtyp	Primärschlüssel	Nullwerte	Default	Referenz
Id	integer	Ja	nein	nein	PERSON.Id
Rabatt	integer	Nein	nein	nein	

Rabatt ist ein Feld, in dem wir den Rabatt (in Prozent) festhalten, den der betreffende Lieferant gewährt. Ein typischer Datensatz sieht so aus:

Tabelle 3.11: Ein Satz aus der Tabelle LIEFERANT

Id	Rabatt
10	13

3.8.4 Die Tabelle BESTELLUNGEN

Diese Tabelle ist eine sogenannte Beziehungstabelle. Hier werden die Bestellungsbeziehungen zwischen Kunden und Artikeln gespeichert. Dazu gehört:

- der Kunde (genauer: die *Id* des Kunden), der die Bestellung gemacht hat
- der Artikel (genauer: die *Id* des Artikels), der bestellt wurde
- die Menge der Artikel, die bestellt wurde
- das Datum des Bestelleingangs
- das Datum der Bestellabwicklung

Tabelle 3.12: Struktur der Tabelle BESTELLUNGEN

Feldname	Feldtyp	Primär-schlüssel	Nullwerte	Default	Referenz
Id	integer	ja	nein	nein	
KundeId	integer	nein	nein	nein	KUNDE.Id
ArtikelId	integer	nein	nein	nein	ARTIKEL.Id
Menge	integer	Nein	nein	0	
Bestelldatum	Date	Nein	nein	Current Date	

Die folgende Tabelle zeigt einige Datensätze:

Tabelle 3.13: Einige Sätze der Tabelle BESTELLUNGEN

Id	KundeId	ArtikelId	Menge	Bestelldatum
2	20	14	8	09.11.2024
11	2	15	10	07.12.2024
19	4	10	20	19.11.2024
20	18	18	15	20.12.2024

Zum guten Schluss brauchen wir noch eine Spezifikation, diesmal eine „Spezifikation“ der Tabelle der Bestellungen. In dieser neuen Tabelle speichern wir alle erledigten Bestellungen. Genauer: Wir speichern die *Id* und das Ausführungsdatum: Ein entsprechendes UML-Diagramm sehen Sie in Abbildung 3.2.

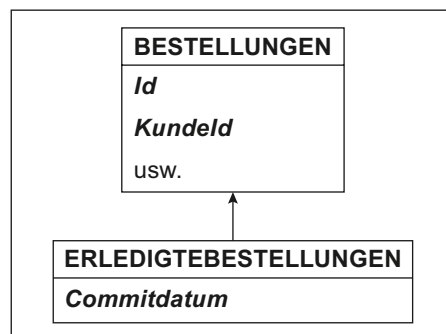


Abb. 3.2: UML-Diagramm zur Vererbungshierarchie der Klassen „Bestellungen“ und „erledigteBestellungen“

Ganz entsprechend dem oben diskutierten Fall der Modellierung einer Vererbung müssen wir wieder deklarieren:

3.8.5 Die Tabelle ERLEDIGTEBESTELLUNGEN

Tabelle 3.14: Struktur der Tabelle ERLEDIGTEBESTELLUNGEN

Feldname	Feldtyp	Primär-schlüssel	Null-werte	Default	Referenz
Id	integer	Ja	nein	nein	BESTELLUNGEN.Id
Commitdatum	date	Nein	nein	nein	

Und ein typischer Datensatz sieht so aus:

Tabelle 3.15: Ein Satz aus der Tabelle ERLEDIGTEBESTELLUNGEN

Id	Commitdatum
2	12.11.2024

Damit haben wir alle Tabellen unserer kleinen Beispieldatenbank diskutiert. Wir werden mit diesen Tabellen viel arbeiten. Die Beziehungen zwischen diesen Tabellen lassen sich beispielsweise so darstellen (wie in Abb. 3.3.)

3.9 Die Beziehungen zwischen den Tabellen der Datenbank „LieferBar“

Sie finden die diskutierten Beziehungen zwischen den sieben Tabellen unserer Datenbank auch in der Grafik (Abb. 3.3) wieder. Lassen Sie sich nicht von den Zahlen und ∞ -Zeichen (∞ steht für: unendlich) irritieren, sie geben die sogenannten **Kardinalitäten** der Beziehungen wieder. Wir werden das bei der Diskussion der ER-Analysetechniken und des relationalen Tabellenentwurfs genauer besprechen.

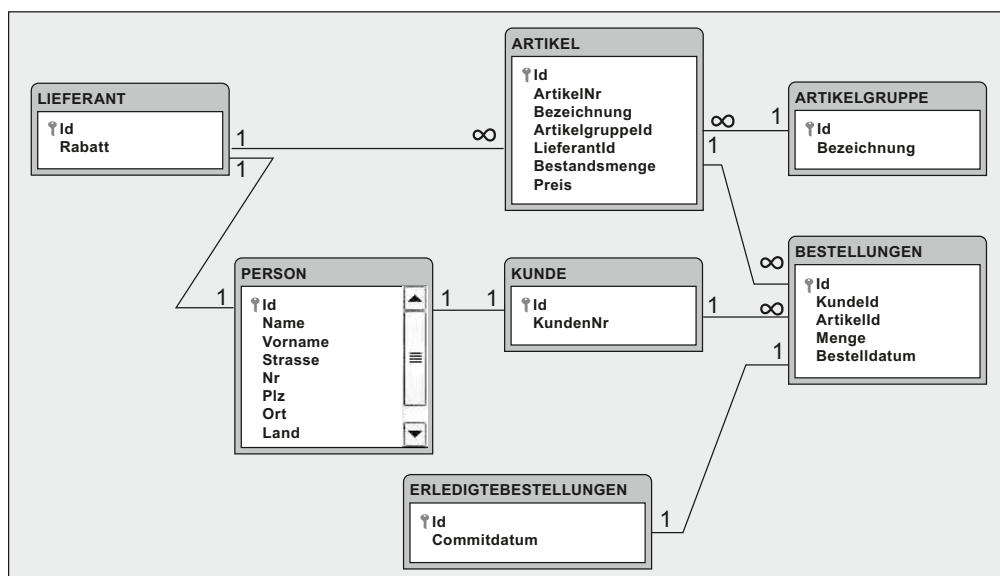


Abb. 3.3: Eine Übersicht über die Tabellen und Beziehungen der Datenbank „LieferBar“

Zusammenfassung

In diesem Kapitel haben wir bei der Vorstellung unserer kleinen Beispieldatenbank viel über die relationalen Datenbanken erfahren können:

- Wir haben eine Charakterisierung des relationalen Modells nach E. F. Codd besprochen.
- Wir haben uns ausführlich mit Primärschlüsseln und ihrer Funktion aus der Sicht des relationalen Modells beschäftigt. Dazu haben wir uns ein Beispiel angesehen, in dem die Werte eines Primärschlüssels in einer anderen Tabelle als Fremdschlüssel auftreten.
- Wir haben Attribute, Wertebereiche und Datentypen als Bausteine der Tabellendefinitionen betrachtet. Dazu haben wir die Attributdefinition aus Kapitel 2 noch etwas erweitert.
- Ich habe Sie vor NULL-Werten gewarnt.
- Wir haben über Datenintegrität gesprochen. Integrität bestimmt sich durch die Eindeutigkeitsanforderung an den Primärschlüssel, die referentielle Integrität und die Erfüllung weiterer Bedingungen. All diese Bedingungen nennt man Constraints. Wir haben diskutiert, wie das DBMS bei den verschiedenen Verletzungsversuchen der Integrität reagieren kann.
- Schließlich haben wir die vollständige Beispieldatenbank „LieferBar“ besprochen.

Aufgaben zur Selbstüberprüfung

- 3.1 Stellen Sie sich vor, die Tabelle BESTELLUNGEN sähe folgendermaßen aus (Sie sehen die ersten sechs Sätze):

Id	KundeId	ArtikelId	LfdNr	Menge	Bestelldatum
1	13	1	1	3	03.10.2024
2	23	2	1	7	17.09.2024
3	13	2	1	5	12.07.2024
4	13	2	2	12	01.02.2025
5	12	2	1	1	01.02.2025
6	3	3	1	1	07.12.2024

Beachten Sie, dass zu den Attributen *KundeId* und *ArtikelId* noch eine sogenannte laufende Nummer – das Attribut *LfdNr* – mitgeführt wird, die anzeigt, die „wievielte“ Bestellung dieses Artikels von diesem Kunden hier vorliegt.

- a) Diese Tabelle hat mehrere mögliche Kandidaten für einen Primärschlüssel. Welche sind das?
 - b) Für welchen würden Sie sich entscheiden? Begründen Sie Ihre Entscheidung.
- 3.2 Formulieren Sie die vollständige Menge der Constraints – der Bedingungen – für die Korrektheit eines Satzes der Tabelle BESTELLUNGEN.
- 3.3 Betrachten Sie noch einmal die zwei „Spezifikationen“ unserer Personentabelle in Abbildung 3.1. Sie haben gelernt: Wenn wir diese Beziehung (Ein Kunde „ist eine“ Person, ein Lieferant „ist eine“ Person) relational abbilden wollen, müssen wir die jeweiligen Primärschlüssel aufeinander beziehen. Die Art dieses Bezugs muss folgenden Anforderungen genügen:
- Es darf nicht möglich sein, eine Person zu löschen, solange es Kunden oder Lieferanten mit demselben Primärschlüssel gibt.
 - Man kann aber andererseits zulassen, dass es Personen mit Primärschlüsseln gibt, die weder in der Tabelle KUNDE noch in der Tabelle LIEFERANT vorkommen. Mit anderen Worten: Ich will zulassen, dass es Personen gibt, die weder Lieferant noch Kunde sind. (Wer weiß, wozu man sie noch einmal brauchen kann.) Ich brauche also keine Löschoptionen von Kunde/Lieferant in Richtung Person festzulegen.

Welche der drei folgenden Möglichkeiten ist die richtige Festlegung?

- a) KUNDE.*Id* ist Fremdschlüssel mit Referenz zu PERSON.*Id*, aber PERSON.*Id* ist **nicht** Fremdschlüssel mit Referenz zu KUNDE.*Id*
- b) KUNDE.*Id* ist **nicht** Fremdschlüssel mit Referenz zu PERSON.*Id*, aber PERSON.*Id* ist Fremdschlüssel mit Referenz zu KUNDE.*Id*
- c) KUNDE.*Id* ist Fremdschlüssel mit Referenz zu PERSON.*Id*, aber PERSON.*Id* ist Fremdschlüssel mit Referenz zu KUNDE.*Id*

Nur eine der drei Möglichkeiten ist korrekt. Hinweis: Überprüfen Sie die referentielle Integrität.

Kapitel 4

4 Tabellen und Relationen: Eine produktive Kontroverse

Das wichtigste Werkzeug zum Speichern und Verwalten von Daten in einer relationalen Datenbank ist die Tabelle. Dem entspricht auf Seiten der Theorie der Begriff der Relation. Wir werden uns diesen Begriff erarbeiten – viele von Ihnen werden ihn schon aus einer Mathematikvorlesung kennen – und dann „hinüber in die Praxis“ zur Tabelle gehen.

Der grundlegende Begriff in dieser Diskussion ist der Begriff der Menge und ich tue so, als wüssten wir alle, was das ist. Leider ist das in Wirklichkeit nicht der Fall, aber alles, was wir hier besprechen, kommt mit der „naiven“ Vorstellung von der Menge als „einer Zusammenfassung von bestimmten, wohlunterschiedenen Objekten zu einem Ganzen“ aus. Die so zusammengefassten Objekte heißen – wie Sie alle wissen – die Elemente der Menge.

4.1 Beispiele für Mengen und das Kreuzprodukt

Lassen Sie mich Ihnen einige Beispiele geben, mit denen wir in diesem Kapitel durchgehend arbeiten werden:

1. drei endliche mathematische Mengen
 - a) $M_1 := \{-3, -1, 2\}$
 - b) $M_2 := \{2, 4\}$
 - c) $M_3 := \{-0.75, -0.25, 2\}^1$
2. inhaltlich definierte Begriffe (wir werden das später „semantisch festgelegte Begriffe“ nennen)
 - a) **Namen:** die Menge aller Nachnamen von Personen
 - b) **Vornamen:** die Menge aller Vornamen von Personen
 - c) **Lebensalter:** die Menge aller ganzen Zahlen zwischen 0 und 150
3. dieselben Begriffe, jetzt ergänzt durch ein Variablenformat, in dem die Daten zu diesen Begriffen gespeichert werden sollen
 - a) **Name: varchar(20):** die Menge aller Nachnamen von Personen im Format von Zeichenketten mit bis zu 20 Zeichen
 - b) **Vorname: varchar(20):** die Menge aller Vornamen im Format von Zeichenketten mit bis zu 20 Zeichen

1. Ich arbeite in diesem Kapitel grundsätzlich mit einem Dezimalpunkt anstatt mit einem Dezimalkomma, um nicht mit den anderen Kommas in der Tupeldarstellung in Konflikt zu geraten.

- c) **Alter: integer:** die Menge aller ganzen Zahlen zwischen 0 und 150 im Format von Integer-Zahlen

Jetzt bilden wir kartesische Produkte von Mengen. Man nennt sie auch Kreuzprodukte. Ich beginne mit einer Definition:

Definition 4:

Sei $n \in \mathbb{N}$, $n > 1$. Seien A_1, A_2, \dots, A_n n Mengen. Dann ist das **kartesische Produkt** $A_1 \times A_2 \times \dots \times A_n$ die Menge aller Elementetupel (x_1, x_2, \dots, x_n) , für die gilt:

$x_1 \in A_1, x_2 \in A_2, \dots, x_n \in A_n$, also

$A_1 \times A_2 \times \dots \times A_n := \{(x_1, x_2, \dots, x_n) \mid x_1 \in A_1, x_2 \in A_2, \dots, x_n \in A_n\}$.

Die Elemente von $A_1 \times A_2 \times \dots \times A_n$ nennt man **Tupel**, genauer **n -Tupel**.

Wir betrachten zwei Beispiele:

Beispiel 1

- Die Menge $M_1 \times M_2 \times M_3$ besteht aus $3 \cdot 2 \cdot 3 = 18$ Dreitupeln, sie sieht folgendermaßen aus:

$\{ (-3, 2, -0.75), (-3, 2, -0.25), (-3, 2, 2),$
 $(-3, 4, -0.75), (-3, 4, -0.25), (-3, 4, 2),$
 $(-1, 2, -0.75), (-1, 2, -0.25), (-1, 2, 2),$
 $(-1, 4, -0.75), (-1, 4, -0.25), (-1, 4, 2),$
 $(2, 2, -0.75), (2, 2, -0.25), (2, 2, 2),$
 $(2, 4, -0.75), (2, 4, -0.25), (2, 4, 2) \}$

Sie sehen: In einem Kreuzprodukt von Mengen gruppiert sich jedes Element einer Menge mit jedem Element der anderen Menge. Die Anzahl der Elemente ist gleich dem Produkt der Anzahlen der Elemente der einzelnen Mengen.

Dieser Tatbestand ist sehr wichtig, er wird uns in einem anderen Zusammenhang noch beschäftigen, und darum gebe ich Ihnen noch ein weiteres Zahlenbeispiel, das Ihnen zeigen soll, wie bei kartesischen Produkten die Mengenmächtigkeit „explodiert“: B_1 habe 30 Elemente, B_2 habe 20 Elemente und B_3 habe 25 Elemente, dann hat $B_1 \times B_2 \times B_3$ schon $30 \cdot 20 \cdot 25 = 15\,000$ Elemente.

Man kann die Elemente von $M_1 \times M_2 \times M_3$ auch in Tabellenform präsentieren. Vergleichen Sie dazu Tabelle 4.1.

Tabelle 4.1: Elemente der Menge $M_1 \times M_2 \times M_3$

<i>Komponente aus M_1</i>	<i>Komponente aus M_2</i>	<i>Komponente aus M_3</i>
-3	2	-0,75
-3	2	-0,25
-3	2	2
-3	4	-0,75
-3	4	-0,25
-3	4	2
-1	2	-0,75
-1	2	-0,25
-1	2	2
-1	4	-0,75
-1	4	-0,25
-1	4	2
2	2	-0,75
2	2	-0,25
2	2	2
2	4	-0,75
2	4	-0,25
2	4	2

Beispiel 2

- Nehmen Sie an, die Mengen **Namen**, **Vornamen** und **Alter** hätten die folgenden Elemente:
 - Namen** = {Chaplin, Einstein, Potter}
 - Vornamen** = {Albert, Charlie, Harry}
 - Alter** = {12, 40}

Dann sieht die Menge **Namen** \times **Vornamen** \times **Alter** folgendermaßen aus:

{ (Chaplin, Albert, 12) , (Chaplin, Albert, 40) ,
 (Chaplin, Charlie, 12), (Chaplin, Charlie, 40),
 (Chaplin, Harry, 12) , (Chaplin, Harry, 12) ,
 (Einstein, Albert, 12) , (Einstein, Albert, 40) ,
 (Einstein, Charlie, 12), (Einstein, Charlie, 40),
 (Einstein, Harry, 12) , (Einstein, Harry, 12) ,
 (Potter, Albert, 12) , (Potter, Albert, 40) ,
 (Potter, Charlie, 12) , (Potter, Charlie, 40) ,
 (Potter, Harry, 12) , (Potter, Harry, 12) }

In der Darstellung einer Tabelle erhalten wir folgendes Bild:

Tabelle 4.2: Elemente der Menge Name \times Vorname \times Alter

Name	Vorname	Alter
Chaplin	Albert	12
Chaplin	Albert	40
Chaplin	Charlie	12
Chaplin	Charlie	40
Chaplin	Harry	12
Chaplin	Harry	40
Einstein	Albert	12
Einstein	Albert	40
Einstein	Charlie	12
Einstein	Charlie	40
Einstein	Harry	12
Einstein	Harry	40
Potter	Albert	12
Potter	Albert	40
Potter	Charlie	12
Potter	Charlie	40
Potter	Harry	12
Potter	Harry	40

Besonders wenn man das letzte Beispiel betrachtet, wird klar, dass das Kreuzprodukt allein noch nicht der richtige Begriff für unsere Datenbanktabellen ist. Wir brauchen eine Konstruktion, die es uns gestattet, nur die „richtigen“ Komponenten zu Tupeln zusammenzufassen. Das aber ist die Relation.

4.2 Relationen: Die Auswahl „sinnvoller“ Tupel aus einem Kreuzprodukt

Definition 5:

Eine **Relation** R auf den Mengen A_1, A_2, \dots, A_n ist eine Teilmenge des kartesischen Produktes $A_1 \times A_2 \times \dots \times A_n$, also $R \subseteq A_1 \times A_2 \times \dots \times A_n$.

Zur umgangssprachlichen Bedeutung des Wortes „Relation“: Während sich in einem Kreuzprodukt jeder mit jedem tummelt – wie wir gesehen haben –, gehören zu einer Relation nur noch bestimmte Tupel. Man sagt dann: Die Tupelkomponenten dieser Tupel „stehen in einer Beziehung bzw. einer Relation zueinander“. Und man gibt dann dieser Beziehung den Namen der betreffenden Relation.

Als Beispiele betrachten wir jetzt Relationen zu unseren beiden Kreuzprodukten:

1. Auf der Menge $M_1 \times M_2 \times M_3$ definieren wir jetzt die Relation QUOTIENT durch:

$$\text{QUOTIENT} := \{(x_1, x_2, x_3) \in M_1 \times M_2 \times M_3 \mid x_3 = x_1 / x_2\}$$

Die Relation QUOTIENT enthält jetzt noch die beiden Elemente

$\{(-3, 4, -0.75), (-1, 4, -0.25)\}$, in Tabellenform:

Tabelle 4.3: Elemente der Relation $\text{QUOTIENT} \subseteq M_1 \times M_2 \times M_3$

Komponente aus M_1 (Zähler)	Komponente aus M_2 (Nenner)	Komponente aus M_3 (Quotient aus Zähler und Nenner)
-3	4	-0.75
-1	4	-0.25

Wir wollen, ehe wir weitermachen, eine allgemeine Sprachregelung festlegen:

Definition 6: Definition zur Tabellendarstellung einer Relation

Sei $R \subseteq A_1 \times A_2 \times \dots \times A_n$ eine Relation. Ein **Tabellenkopf** für eine Darstellung der Elemente dieser Relation besteht aus

1. der **Festlegung eines eindeutigen Namens N_i** für die Mengen A_i für alle $1 \leq i \leq n$. Dieser Name kommt in den Spaltenkopf der Tabellenspalte, in der die Tupelkomponenten aus A_i angezeigt werden;
2. der **Festlegung einer Reihenfolge**, in der die Tabellenspalten notiert werden.

Für unser erstes Beispiel bedeutet das, dass wir es auch folgendermaßen beschreiben können:

Beispiel 1

Sei $M_1 := \{-3, -1, 2\}$, $M_2 := \{2, 4\}$ und $M_3 := \{-0.75, -0.25, 2\}$. Die Relation $\text{QUOTIENT} \subseteq M_1 \times M_2 \times M_3$ sei definiert durch:

$$\text{QUOTIENT} := \{(x_1, x_2, x_3) \in M_1 \times M_2 \times M_3 \mid x_3 = x_1 / x_2\}$$

Es sei weiter

- *Zähler* der Spaltenkopfname für die Elemente aus M_1
- *Nenner* der Spaltenkopfname für die Elemente aus M_2
- *Quotient* der Spaltenkopfname für die Elemente aus M_3

Und die Reihenfolge der Spalten sei *Zähler*, *Nenner*, *Quotient*. Dann erhalten wir folgende Tabellendarstellung für die Relation QUOTIENT :

Tabelle 4.4: Eine Tabellendarstellung der Relation $\text{QUOTIENT} \subseteq M_1 \times M_2 \times M_3$

<i>Zähler</i>	<i>Nenner</i>	<i>Quotient</i>
-3	4	-0.75
-1	4	-0.25

Nun zu unserem zweiten Beispiel:

Beispiel 2

Auf der Menge $\text{Namen} \times \text{Vornamen} \times \text{Alter}$ definieren wir eine Relation FREUNDE durch die Festlegung:

$$\text{FREUNDE} := \{(x_1, x_2, x_3) \in \text{Namen} \times \text{Vornamen} \times \text{Alter} \mid$$

Es gibt eine Person mit dem (Nach-)Namen x_1 ,

dem Vornamen x_2 und dem Alter $x_3\}$

Dann sieht die Menge $\text{Namen} \times \text{Vornamen} \times \text{Alter}$ folgendermaßen aus:

$$\{(\text{Chaplin}, \text{Charlie}, 40), (\text{Einstein}, \text{Albert}, 40), (\text{Potter}, \text{Harry}, 12)\}$$

Es sei jetzt

- *Name* der Spaltenkopfname für die Elemente aus Namen

- *Vorname* der Spaltenkopfname für die Elemente aus **Vornamen**
- *Alter* der Spaltenkopfname für die Elemente aus **Alter**

Und die Reihenfolge der Spalten sei *Name*, *Vorname*, *Alter*. Dann erhalten wir folgende Tabellendarstellung für die Relation FREUNDE:

Tabelle 4.5: Elemente der Menge $\text{Name} \times \text{Vorname} \times \text{Alter}$

Name	Vorname	Alter
Chaplin	Charlie	40
Einstein	Albert	40
Potter	Harry	12

Wenn Sie jetzt – wie bei unserem dritten Mengenbeispiel – zu den Spaltenkopfnamen auch noch die Variablenformate angeben, dann haben Sie alle Informationen, die Sie für die Definition einer Tabelle bzw. einer Relation in einem beliebigen relationalen Datenbankmanagementsystem brauchen.

Offensichtlich möchte man bei den Operationen auf relationalen Datenbanken nicht nur ganze Tupel von Relationen, d.h. vollständige Zeilen von Tabellen bearbeiten, sondern auch einzelne Komponentenwerte, d.h. die Werte einzelner Spalten und man möchte diese Spaltenwerte nicht mit dem Index des Tupelements sondern mit dem Namen der Menge im Tabellenkopf ansprechen.

Man möchte also nicht sagen:

„Erhöhe alle Werte in der 3. Tupelkomponente der Relation FREUNDE um +1“

sondern man möchte sagen:

„Erhöhe alle Elemente der Menge **Alter** in der Relation FREUNDE um +1“

Man möchte nicht sagen:

„Zeige alle voneinander verschiedenen Werte der 3. Tupelkomponente der Relation FREUNDE“

sondern man möchte sagen:

„Zeige alle voneinander verschiedenen Elemente der Menge **Alter** in der Relation FREUNDE“

Um das zu ermöglichen, müssen die Tabellenkopfnamen und damit auch die Namen der Mengen, die an dem Kreuzprodukt einer Relation beteiligt sind, eindeutig sein.

Wir erhalten die wichtige Erweiterung unserer ursprünglichen Definition:

Definition 7:

Eine **Datenbankrelation** R auf den Mengen A_1, A_2, \dots, A_n ist eine Teilmenge des kartesischen Produktes $A_1 \times A_2 \times \dots \times A_n$, also $R \subseteq A_1 \times A_2 \times \dots \times A_n$, wobei die Bezeichnungen für die Mengen A_1, \dots, A_n alle eindeutig und voneinander verschieden sind.

Zum Abschluss dieser Diskussion gehen wir jetzt den umgekehrten Weg: Wir nehmen eine unserer Tabellen und fragen uns: Wie heißt die zugehörige Datenbankrelation?

Erinnern Sie sich: Unsere Tabelle ARTIKEL hatte die folgende Struktur:

Tabelle 4.6: Struktur der Tabelle ARTIKEL

Feldname	Feldtyp	Primär-schlüssel	Null-werte	Default	Referenz
Id	integer	Ja	nein	nein	
Artikelnr	varchar(10)	Nein	nein	nein	
Bezeichnung	varchar(30)	Nein	nein	nein	
ArtikelgruppelId	integer	Nein	nein	nein	ARTIKELGRUPPE.Id
LieferantId	integer	Nein	nein	nein	LIEFERANT.Id
Bestandsmenge	integer	Nein	nein	0	
Preis	float(2)	Nein	nein	0	

Wir definieren dazu eine Datenbankrelation: Es sei

- **Primärschlüssel Id:** integer = die Menge der ganzen Zahlen im integer-Format
- **Artikelnr:** varchar(10) = die Menge von Artikelnummern unseres Artikelsortiments im Format von Zeichenketten mit bis zu 10 Zeichen
- **Bezeichnung:** varchar(30) = die Menge der Bezeichnungen für die Artikel unseres Artikelsortiments im Format von Zeichenketten mit bis zu 30 Zeichen
- **ArtikelgruppeId:** integer = die Menge der Primärschlüssel aus der Tabelle/Datenbankrelation ARTIKELGRUPPE im Integer-Format
- **LieferantId:** integer = die Menge der Primärschlüssel aus der Tabelle/Datenbankrelation LIEFERANT im Integer-Format
- **Bestandsmenge:** integer = die Menge der Bestandsmengenangaben für die einzelnen Artikel des Artikelsortiments im Integer-Format
- **Preis:** float(2) = die Menge der Preise für die Artikel unseres Artikelsortiments im Format von Dezimalzahlen mit 2 Nachkommastellen.

Sei nun BasismengeFürArtikel =
 = Id × Artikelnr × Bezeichnung × ArtikelgruppeId × LieferantId ×
 Bestandsmenge × Preis

Dann ist:

ARTIKEL := $\{(x_1, x_2, x_3, x_4, x_5, x_6, x_7) \in \text{BasismengeFürArtikel} \mid$

$x_1 \in \mathbb{N} \setminus \{0\}$ und x_1 ist in ARTIKEL eindeutig

d.h., kein anderes Element aus ARTIKEL hat diesen x_1 – Wert

und es gibt einen Artikel aus dem Versandhaus LieferBar & Co,

dessen Artikelnummer identisch mit dem String x_2 ist,

dessen Bezeichnung identisch mit dem String x_3 ist,

dessen Artikelgruppe den Primärschlüssel x_4 hat,

dessen Lieferant den Primärschlüssel x_5 hat,

von dem noch x_6 Stück auf Lager sind

und der x_7 Euro kostet}

Merken Sie übrigens, was für einen Unsinn NULL-Werte (d.h. keinerlei Elementauswahl in der entsprechenden Komponente) in der Relationensprache bedeuten? Etwas Entsprechendes kommt hier überhaupt nicht vor.

4.3 Datenbankrelationen können mehr als Tabellen

Unter dieser Überschrift sind zwei Punkte wichtig, die sich beide daraus ergeben, dass eine Datenbankrelation stets ein abstrakteres Objekt ist als die zugehörige Tabelle, die konkret einige oder – falls möglich – gegebenenfalls auch alle Elemente aus der Datenbankrelation aufzählt.

- Eine Datenbankrelation kann unendlich viele oder auch eine völlig unbestimmte endliche Anzahl von Elementen enthalten. Und natürlich verändert sich die Definition einer Datenbankrelation nicht, obwohl sich die Anzahl ihrer Elemente im Laufe der Zeit ständig verändert und auch dramatisch verändern kann. Eine Tabelle dagegen als Darstellungswerkzeug der Elemente einer Datenbankrelation enthält bei jeder Präsentation eine endliche, den Zwecken der Präsentation angepasste Anzahl von Elementzeilen. Diese Anzahl ist stets bekannt und im Allgemeinen kleiner als die Mächtigkeit der Datenbankrelation. **Insbesondere hat jede Datenbanktabelle nur endlich viele Elemente.**

- Außerdem muss sich jede Präsentation einer Datenbankrelation für eine Sortierung entscheiden, in der die Elemente der Datenbankrelation angezeigt werden. Die Datenbankrelation selber als eine Menge ist dazu nicht gezwungen und hat jede Möglichkeit der Sortierung oder Nicht-Sortierung als Option offen. Die Darstellung in einer Tabelle macht diese Freiheit zunichte.

4.4 Tabellen können mehr als Datenbankrelationen

Es gibt im Wesentlichen zwei Gebiete, auf denen Tabellen den Datenbankrelationen „überlegen“ sind. Beide sind für die Zwecke der relationalen Datenbanksysteme geradezu schädlich und wir werden besprechen, wie wir diese Überlegenheit so gründlich wie möglich unterbinden werden.

Im Einzelnen:

- Datenbankrelationen sind Mengen. Elemente von Mengen sind „wohlunterschieden“. Das bedeutet: Ein und dasselbe Element kann nicht mehrmals in einer Menge vorkommen. Bei einer Tabelle kann das aber durchaus der Fall sein. Hier kann ein- und dieselbe Zeile beliebig oft vorkommen. Ich kann solche Tabellen auch beispielsweise mit der Datensprache SQL leicht erzeugen. Sie könnten dort beispielsweise sagen:

„Zeige mir alle Werte aus der Tabellenspalte **Alter** von der Tabelle FREUNDE“

In SQL heißt das (das lernen Sie in einem späteren Kurs noch genauer):

„SELECT **Alter** from FREUNDE“

Und da FREUNDE so aussah,

Tabelle 4.7: Tabelle FREUNDE

Name	Vorname	Alter
Chaplin	Charlie	40
Einstein	Albert	40
Potter	Harry	12

erhalten wir als Ergebnis

Tabelle 4.8: Werte von Alter aus der Tabelle FREUNDE ohne Unterdrückung gleicher Werte

Alter
40
40
12

Nebenbei gesagt: Es gibt in SQL eine Möglichkeit, das Auftreten gleicher Zeilen in einer Ergebnistabelle zu verhindern, man programmiert dann:

„Zeige mir alle verschiedenen Werte aus der Tabellenspalte **Alter** von der Tabelle FREUNDE“

In SQL: „SELECT DISTINCT **Alter** from FREUNDE“

Wir erhalten:

Tabelle 4.9: Werte von **Alter** aus der Tabelle FREUNDE mit Unterdrückung gleicher Werte

Alter
40
12

Das, was wir hier gemacht haben, nennt man eine Projektion – wir werden solche Operationen im nächsten Kapitel genauer besprechen. Wann immer Sie solche Operationen durchführen, sollten Sie daran denken, das Auftreten mehrfach vorkommender Tabellenzeilen zu verhindern. Sie gefährden sonst den Bezug zum Relationenmodell mit oft sehr ärgerlichen Konsequenzen. Beispielsweise ist es in derartigen Fällen nicht mehr möglich, Änderungs- oder Löschbefehle für eindeutig bestimmte Tabellenzeilen durchzuführen.

- Bei Tabellen können einzelne Spalteneinträge leer bleiben – das entspräche NULL-Werten in einem Datensatz. Solche NULL-Werte ergeben bei den Elementen einer Datenbankrelation keinen Sinn, denn ein Element kann nur aus einem Kreuzprodukt von Mengen sein, wenn dieses Element ein Tupel ist, das aus lauter Komponenten aus diesen Mengen besteht. NULL-Werte können bei der Definition von Tabellen für jedes einzelne Attribut, d.h. für jede einzelne Spalte, verboten werden.

4.5 Konsequenzen für Abbildungen und Operatoren

Zum relationalen Modell gehören natürlich nicht nur die Datenbankrelationen, sondern auch die Abbildungen und Operatoren, die auf die Datenbankrelationen angewendet werden. Es gilt:



Alle Abbildungen und Operatoren, die wir auf der Menge der Datenbankrelationen definieren werden, müssen die Eigenschaft haben, dass sie auch wieder Datenbankrelationen als Ergebnis haben.

Unser einfaches Beispiel hat uns gezeigt, dass das gar nicht so selbstverständlich ist und auf Tabellenebene auch nicht automatisch erreicht wird.

4.6 Nomenklatur

Zum Abschluss dieses Kapitels lernen wir noch zwei weitere Begriffe kennen, die im Zusammenhang mit Datenbankrelationen oft genannt werden.

Definition 8:

Die **Kardinalität** einer **Datenbankrelation R** bezeichnet die Anzahl der Elemente einer Datenbankrelation. Kardinalitäten sind offensichtlich – insbesondere im Zusammenhang mit der täglichen Datenbankarbeit, wo eingefügt und gelöscht wird – sehr veränderliche Größen.

Der **Grad** einer **Datenbankrelation R** ist die Anzahl der Komponenten, welche die Tupelemente der Datenbankrelation haben. Zum Beispiel hat der Grad unserer Datenbankrelation ARTIKEL den Wert 7.

Jetzt können wir abschließend definieren:

4.7 Das relationale Modell – Die abschließende Charakterisierung

Wir werden von jetzt ab unter einer Relation immer eine Datenbankrelation verstehen. Das relationale Modell ist dann durch die folgenden drei Punkte charakterisiert:

1. Die Daten können nur in Form von Relationen betrachtet und verändert werden. Das wird oft der **strukturelle Aspekt** genannt.
2. Alle **Operatoren**, die dem Benutzer zur Manipulation von Daten zur Verfügung stehen, operieren grundsätzlich nur auf der Menge der Relationen und ergeben auch wieder neue Relationen. Ein relationales System enthält mindestens die Operatoren RESTRICT, PROJECT und JOIN. Man nennt diesen zweiten Punkt den **manipulativen Aspekt** des relationalen Modells. Dieser zweite Punkt wird im folgenden Kapitel behandelt.
3. Relationen müssen gewissen **Integritätsbedingungen** genügen. Vergleichen Sie dazu bitte Kapitel 3 und Kapitel 6.

Zusammenfassung

Zunächst haben wir – und zwar auf strikt mathematische Weise – definiert, was eine Relation ist. Wir haben aber zusätzlich besprochen, wie sich Relationen bzw. die Elemente von Relationen, die sogenannten Tupel, in Tabellen darstellen lassen. Eine wichtige Rolle spielte dabei der Begriff des Tabellenkopfes. Nach dieser Diskussion der Begriffe „Relation“ und „Tabelle zur Darstellung einer Relation“ konnten wir sie auch miteinander vergleichen. Wir machten dann die Einschränkung, dass wir nur noch Relationen betrachten wollten, bei denen die am Kreuz-

produkt beteiligten Mengen eindeutig und voneinander verschieden sind. Dafür legten wir die Bezeichnung „Datenbankrelation“ fest. Die folgenden fünf Punkte fielen uns dabei auf:

1. Eine Datenbanktabelle besitzt stets eine endliche Kardinalität. Das braucht bei einer Datenbankrelation nicht der Fall zu sein.
2. Die Elemente einer Datenbankrelation sind zunächst nicht in irgendeiner Weise sortiert. Das gibt uns bei einer Datenbankrelation die Option zu jeder beliebigen Sortierung. Sobald man die Elemente einer Datenbankrelation in einer Tabelle dargestellt hat, ist diese Option verloren; man muss sich stets für eine bestimmte Sortierung entscheiden.
3. Jedes Element einer Datenbankrelation kommt nur ein einziges Mal in dieser Datenbankrelation vor. In einer Tabelle kann ich ein und dasselbe Tupel mehrmals eintragen. Das ist eine schlechte Eigenschaft; sie sollte nicht genutzt werden.
4. In einer Datenbankrelation muss bei jedem Element in jeder Tupelkomponente ein Element aus der entsprechenden Menge des Kreuzprodukts ausgewählt sein. Es gibt keine „leeren“ Tupelkomponenten. Dagegen können in einer Tabelle einzelne Spalteneinträge leer bleiben. Diese Möglichkeit zur Verwendung sogenannter NULL-Werte ist eine schlechte Eigenschaft, sie sollte nicht genutzt werden.
5. Die Charakterisierung des relationalen Modells mithilfe der Datenbankrelationen schloss dieses erste theoretische Kapitel ab. Von jetzt ab verstehen wir unter einer Relation immer eine Datenbankrelation.

Aufgaben zur Selbstüberprüfung

- 4.1 Sei $A := \{1, 2, 3\}$, $B := \{\text{eins}, \text{zwei}\}$
 - a) Geben Sie sämtliche Elemente von $A \times B$ an.
 - b) Es sei $R := \{(x, y) \in A \times B \mid y \text{ ist der Name für die Zahl } x\}$. Ist R eine Relation? (Argumentieren Sie mit der Definition einer Relation.)
 - c) Stellen Sie R (vollständig) in einer Tabelle dar. Denken Sie an den Tabellenkopf.
- 4.2 Sei $\text{Name} := \{\text{Arnold}, \text{Euler}, \text{Harder}, \text{Hirzebruch}, \text{Karcher}, \text{Klingenberg}\}$, $\text{Vorname} := \{\text{Carla}, \text{Sonja}, \text{Wilhelm}, \text{Leonard}\}$, $\text{Ort} := \{\text{Bonn}, \text{Wuppertal}\}$. Wie viele Datensätze kann eine Relation, die genau Name, Vorname und Ort als Attribute hat, höchstens enthalten?

Kapitel 5

5 Relationale Operatoren als Grundlage aller manipulativen Operationen

Was wir bisher besprochen haben und was wir hier und im Folgenden besprechen werden, hat eine lange Vorgeschichte. Bereits 1970 veröffentlichte E. F. Codd, ein Mitarbeiter des IBM Research Laboratory (IBM Forschungslabor) in San Jose in Kalifornien einen Artikel, der sehr berühmt werden sollte. Er hatte die Überschrift: ‚A Relational Model for Large Shared Data Banks‘. In diesem Artikel wurden einige abstrakte Prinzipien für ein Datenbankmanagement definiert. C. J. Date hat dann eine Definition des relationalen Modells versucht (vergleichen Sie dazu bitte noch einmal das Ende des vierten Kapitels).

Codd forderte und definierte acht Operatoren für ein relationales Datenbankmanagementsystem. Genau diese Operatoren werden wir in diesem Kapitel besprechen.

5.1 Unsere grundlegende Definition und einige Bemerkungen

Gemäß Punkt 2 unserer Charakterisierung des relationalen Modells definieren wir:

Definition 9:

Sei REL die Menge aller Relationen. Dann ist ein **Operator** ω eine Abbildung von REL nach REL , also $\omega : REL \rightarrow REL$.

Die Bemerkungen:

- Oft verlangt man in der Mathematik mehr von einem Operator: Er soll nicht nur eine Abbildung sein, er soll eine lineare Abbildung sein. Dazu braucht man auf den beiden Mengen, die zu der Abbildung gehören, dem Definitionsbereich und dem Bildbereich eine Vektorraumstruktur oder eine Algebrastruktur, damit diese Anforderung einen Sinn ergibt. Wir müssten also beispielsweise aus REL eine Algebra machen und nachweisen, dass die Operatoren, die wir definieren werden, die Algebraoperationen respektieren. Codd, der Begründer dieser Theorie, ist dementsprechend vorgegangen und deshalb wird auch im Zusammenhang mit den Operatoren in den Lehrbüchern gerne von einer relationalen Algebra gesprochen.
- Bitte erinnern Sie sich (noch einmal), dass die Anforderung, dass unsere Operatoren aus Relationen wieder Relationen machen, genau den Definitionen des relationalen Modells im dritten und im vierten Kapitel entsprechen: Daten tauchen grundsätzlich nur in Form von Tabellen bzw. von Relationen auf, egal was man mit ihnen macht. Man spricht auch von der Abgeschlossenheit

der relationalen Algebra unter all ihren Operatoren. Diese Anforderung bedeutet auch, dass man mehrere Operatoren hintereinander auf Relationen anwenden kann. Denn: Man verlässt niemals die Welt der Relationen.

- Jede Datensprache, genauer: der manipulative Teil (der DML-Teil) einer Datensprache muss Befehle enthalten, mit denen diese manipulativen Operatoren angewendet werden können. Hier haben wir ein weiteres Kriterium zur Qualitätsprüfung aller implementierbaren bzw. implementierten Produkte.

Ich will nun die wichtigsten relationalen Operatoren der Reihe nach mit Ihnen besprechen.

5.2 Acht relationale Operatoren

Die ersten drei Operatoren haben als „Input“ stets zwei Relationen und liefern als „Output“ eine neue Relation. Die Input-Relationen müssen eine bestimmte Kompatibilitätsbedingung erfüllen. Sie lautet folgendermaßen:

Definition 10:

Zwei Relationen R und S heißen **union-kompatibel** oder auch **typ-kompatibel**, wenn R und S Teilmengen desselben Kreuzproduktes $M_1 \times \dots \times M_n$ sind.

5.2.1 Der Operator UNION oder: Die Vereinigung

1. Kurze Beschreibung von UNION

Der Operator UNION bildet die Vereinigungsmenge von zwei Relationen.

2. Definition von UNION

Es seien R und S zwei typ-kompatible Relationen, $R \subseteq M_1 \times \dots \times M_n$ und $S \subseteq M_1 \times \dots \times M_n$. Dann ist $\text{UNION}(R, S)$ gerade die Relation

$$R \cup S \subseteq M_1 \times \dots \times M_n$$

3. Beispiel für UNION

Sei R die Menge aller Artikel, die zur Artikelgruppe mit der *Id* 9 gehören. Sei S die Menge aller Artikel, deren *Preis* > 50.000 € ist. Dann zeigt die folgende Tabelle gerade die Elemente von $\text{UNION}(R, S)$:

Tabelle 5.1: $\text{UNION}(\text{ARTIKEL mit Id} = 9, \text{ARTIKEL mit Preis} > 50000)$:

Id	Artikelnr	Bezeichnung	Artikel-gruppelId	LieferantId	Bestands-menge	Preis
2	A0090001	Hilfsmotoren	9	5	514	99,23 €
4	A0040001	Steinway-Flügel	4	6	7	52.000,00 €
5	A0090002	Topfpflanzen	9	19	356	20,65 €

5.2.2 Der Operator INTERSECTION oder: Der Durchschnitt

1. Kurze Beschreibung von INTERSECTION

Der Operator INTERSECTION bildet die Schnittmenge von zwei Relationen.

2. Definition von INTERSECTION

Es seien R und S zwei typ-kompatible Relationen, $R \subseteq M_1 \times \dots \times M_n$ und $S \subseteq M_1 \times \dots \times M_n$. Dann ist $\text{INTERSECTION}(R, S)$ gerade die Relation $R \cap S \subseteq M_1 \times \dots \times M_n$.

3. Beispiel für INTERSECTION

Sei R die Menge der Personen, die Kunden sind, sei S die Menge der Personen, die Lieferanten sind. Dann zeigt die folgende Tabelle gerade die Elemente von $\text{INTERSECTION}(R, S)$:

Tabelle 5.2: INTERSECTION(KUNDE, LIEFERANT)

Id	Name	Vorname	Straße	Nr	Plz	Ort	Land
15	Hooker	John Lee	Long Road	66	unbekannt	Chicago	USA
20	Mouse	Mickey	Barksweg	46	12344	Entenhausen	Disneyland

5.2.3 Der Operator DIFFERENCE oder: Die Differenz

1. Kurze Beschreibung von DIFFERENCE

Der Operator DIFFERENCE bildet die Differenzmenge von zwei Relationen.

2. Definition von DIFFERENCE

Es seien R und S zwei typ-kompatible Relationen, $R \subseteq M_1 \times \dots \times M_n$ und $S \subseteq M_1 \times \dots \times M_n$. Dann ist $\text{DIFFERENCE}(R, S)$ gerade die Relation $R \setminus S \subseteq M_1 \times \dots \times M_n$ (Sprich: R ohne S) = $\{x \in R \mid x \notin S\}$.

3. Beispiel für DIFFERENCE

Sei R die Menge der Personen, sei S die Menge der Personen, die Lieferanten sind. Dann zeigt die folgende Tabelle gerade die Elemente von $\text{DIFFERENCE}(R, S)$ (also alle Personen, die keine Lieferanten sind):

Tabelle 5.3: DIFFERENCE(PERSON, LIEFERANT): Personen, die keine Lieferanten sind

Id	Name	Vorname	Straße	Nr	Plz	Ort	Land
2	Engels	Karl	Rotlindstraße	12	01848	Wuppertal	Deutschland
3	Mozart	Wolfgang	Tonikastraße	32	70178	Regensburg	Deutschland
4	Picasso	Pablo	Highway	61	unbekannt	New York	USA
7	Lennon	John	Penny Lane	33	unbekannt	New York	USA
8	Sellers	Peter	Luisenstraße	5	53024	Bonn	Deutschland
9	Cluseau	Inspektor	Luisenstraße	5	53024	Bonn	Deutschland
12	Hau	Arnold	Hattersheimer Straße	11	60309	Frankfurt/M.	Deutschland
13	Fellini	Federico	Via Mala	10	unbekannt	Rom	Italien
16	Marx	Groucho	Funny Valentine	12	23154	Augsburg	Deutschland
18	Dylan	Robert	Arndtstraße	41	45634	Nürnberg	Deutschland
23	Mammut	Manfred	Gletscherspalte	7	00005	Spitzbergen	Grönland
24	Zetkin	Clara	Weg	25	32145	Berlin	Deutschland
25	Sharif	Omar	Schiwagoplatz	1	unbekannt	Moskau	Russland
27	Goethe	Wolfgang	Am Platz	1	01804	Weimar	Deutschland
28	Schiller	Friedrich	Am Platz	1	01804	Weimar	Deutschland
30	Euler	Leonhard	Brückenstraße	7	unbekannt	Kaliningrad	Russland

Für die folgenden Operatoren brauchen wir die (starke) Einschränkung der Typ-Kompatibilität nicht mehr – sie sind mit beliebigen Relationen möglich.

5.2.4 Der Operator PRODUCT oder: Das kartesische Produkt

Wie es die Überschrift schon andeutet ist hier mit dem Wort PRODUCT unser „alter Bekannter“, das kartesische Produkt gemeint. Das Produkt ist ein Operator, der aus zwei Relationen $R \subseteq A_1 \times \dots \times A_m$ und $S \subseteq B_1 \times \dots \times B_n$ eine neue Relation $R \times S \subseteq A_1 \times \dots \times A_m \times B_1 \times \dots \times B_n$ macht, deren Basiskreuzprodukt aus allen Mengen besteht, die von R und S herkommen. Um die wichtige Eigenschaft einer Datenbankrelation, die in der Eindeutigkeit der beteiligten Mengen besteht, zu sichern, definieren wir:

Definition 11:

Es seien R und S zwei Relationen, $R \subseteq A_1 \times \dots \times A_m$ und $S \subseteq B_1 \times \dots \times B_n$. Um bei der Produktrelation $R \times S \subseteq A_1 \times \dots \times A_m \times B_1 \times \dots \times B_n$ die Eindeutigkeit der Mengenbezeichnungen zu sichern, schreiben wir gegebenenfalls statt A_i den Namen $R.A_i$ und statt B_j den Namen $S.B_j$. So ist, falls R und S Datenbankrelationen sind, auch stets $R \times S$ eine Datenbankrelation.

Auch wenn wir im Folgenden weiter nur A_i und B_j schreiben, werden wir das so verstehen, dass wir im Falle einer Übereinstimmung der Bezeichnungen A_i und B_j übereinstimmen, stets $R.A_i$ und $S.B_j$ meinen und so unsere geforderte Eindeutigkeit nicht gefährdet ist. Sollten wir beispielsweise das Produkt von PERSON und LIEFERANT bilden, würden wir im Falle des mehrdeutigen Attributnamens **Id** schreiben: PERSON.**Id** bzw. LIEFERANT.**Id**

Wir können nun definieren:

1. **Kurze Beschreibung von PRODUCT**

Der Operator **PRODUCT** bildet das kartesische Produkt von zwei Relationen.

2. **Definition von PRODUCT**

Es seien R und S zwei Relationen, $R \subseteq A_1 \times \dots \times A_m$ und $S \subseteq B_1 \times \dots \times B_n$. Dann ist **PRODUCT**(R, S) gerade die Relation $R \times S \subseteq A_1 \times \dots \times A_m \times B_1 \times \dots \times B_n$, das kartesische Produkt aus R und S .

3. **Beispiel für PRODUCT**

Sei R die Menge aller Artikel und es sei S die Menge aller Artikelgruppen. R hat 30 Elemente, S hat 10 Elemente. Die Relation **PRODUCT**(R, S) = $R \times S$ hat dann $30 \cdot 10 = 300$ Elemente. Sie sehen in der folgenden Tabelle die ersten (elf) Elemente, die man erhält, wenn man zunächst nach **ARTIKEL.Id** und bei gleicher **ARTIKEL.Id** nach **ARTIKELGRUPPE.Id** sortiert. Erinnern Sie sich: Bei kartesischen Produkten ist die Anzahl der Elemente des Produktes stets gleich dem Produkt der Anzahl der Elemente der einzelnen Komponenten.

Tabelle 5.4: PRODUKT(ARTIKEL,ARTIKELGRUPPE): Ein Ausschnitt von elf Sätzen

ARTIKEL.Id	Artikelnr	ARTIKEL.Bezeichnung	ArtikelgruppelId	LieferantId	Bestandsmenge	Preis	ARTIKELGRUPPE.Id	ARTIKELGRUPPE.Bezeichnung
1	A0010002	Lampenschirme	1	6	669	10,13	2	Elektronik-Fachgerät
1	A0010002	Lampenschirme	1	6	669	10,13	3	Haushaltswaren
1	A0010002	Lampenschirme	1	6	669	10,13	4	Musikalien
1	A0010002	Lampenschirme	1	6	669	10,13	5	Lebensmittel
1	A0010002	Lampenschirme	1	6	669	10,13	6	Bücher
1	A0010002	Lampenschirme	1	6	669	10,13	7	Tabakwaren
1	A0010002	Lampenschirme	1	6	669	10,13	8	Larifari
1	A0010002	Lampenschirme	1	6	669	10,13	9	Kokolores
1	A0010002	Lampenschirme	1	6	669	10,13	10	Autozubehör
1	A0010002	Lampenschirme	1	6	669	10,13	1	Kleinteile
2	A0090001	Hilfsmotoren	9	5	514	99,23	7	Tabakwaren

Warum macht man so etwas „Unsinniges“ wie diese Produktbildung? Was für einen Sinn ergibt es, den Artikel „Lampenschirm“ mit der Artikelgruppe „Bücher“ zu koppeln? Bei allem, was wir bisher zu dem Thema „Relationen und Tabellen“ besprochen haben, waren Kreuzprodukte stets die Grundlage unserer Untersuchungen, von denen aus wir zu den uns interessierenden Tabellen vorgestoßen sind. Genauso wird es auch hier sein. **Kreuzprodukte sind der erste Schritt** zu für uns sinnvollen Verbindungen von Tabellen. Der zweite Schritt folgt sogleich.

5.2.5 Der Operator RESTRICT oder: Die Restriktion

1. Kurze Beschreibung von RESTRICT

Der Operator RESTRICT bildet eine Teilmenge einer Relation, die nur noch die Elemente enthält, die eine bestimmte Bedingung erfüllen.

2. Definition von RESTRICT

Es sei R eine Relation und $\chi: R \rightarrow \{\text{false}, \text{true}\}$ sei eine Abbildung. Dann ist $\text{RESTRICT}(R, \chi)$ gerade die Relation

$$\{x \in R \mid \chi(x) = \text{true}\}$$

Man nennt „ $\chi(x) = \text{true}$ “ auch die (restriktive) Bedingung und schreibt – in Anlehnung an den Formalismus in SQL – auch:

$$R \text{ WHERE } \chi(x) = \text{true}$$

Es gilt offensichtlich: $R \text{ WHERE } \chi(x) = \text{true} \subseteq R$

3. Beispiele für RESTRICT

Wir betrachten zwei Beispiele:

Beispiel 1

Sei R die Menge aller Artikel und sei $\Phi: R \rightarrow \{\text{false}, \text{true}\}$ die Abbildung, die folgendermaßen definiert ist:

$\Phi(x) = \text{true}$ genau dann, wenn die *Bezeichnung* von x mit ‚D‘ beginnt.

Die folgende Tabelle zeigt Ihnen sämtliche Elemente von $\text{RESTRICT}(R, \Phi)$

Tabelle 5.5: $\text{RESTRICT}(\text{ARTIKEL}) \text{ WHERE Bezeichnung beginnt mit D}$

Id	Artikelnr	Bezeichnung	ArtikelgruppelId	LieferantId	Bestandsmenge	Preis
3	A0020005	DVD-Player	2	15	848	112,34
14	A0060003	Der Name der Rose	6	6	562	26,21
15	A0060002	Der Termin	6	22	153	16,00
16	A0020004	Drucker	2	14	8	879,99
25	A0020001	DVD-Player	2	22	12	175,35
29	A0020003	Drucker	2	11	0	2.300,00

Beispiel 2

Das zweite Beispiel schließt sich an die Bemerkungen an, die wir bei der Diskussion des Produktoperators gemacht haben. Wir benutzen jetzt die Restriktion, um all das, was uns beim Produkt stört, herauszufiltern. Das geschieht folgendermaßen:

Sei wieder R die Menge aller Artikel und es sei S die Menge aller Artikelgruppen. Wir wollen jetzt auf dem Produkt $R \times S$ eine Restriktion definieren.

Sei dazu $\Phi: R \times S \rightarrow \{\text{false}, \text{true}\}$ die Abbildung, die folgendermaßen definiert ist: $\Phi(x,y) = \text{true}$ genau dann, wenn der Wert von $\text{ARTIKEL.ArtikelgruppelId}$ in x mit dem Wert von ARTIKELGRUPPE.Id in y übereinstimmt.

$\text{RESTRICT}(R \times S, \Phi)$ hat jetzt nur noch 30 Elemente, die anderen 270 Elemente wurden herausgefiltert. Und die folgende Tabelle zeigt Ihnen die ersten zwölf Elemente von $\text{RESTRICT}(R \times S, \Phi)$ – in derselben Sortierung wie unsere Produktelemente im vorletzten Beispiel).

Tabelle 5.6: $\text{PRODUKT}(\text{ARTIKEL}, \text{ARTIKELGRUPPE}) \text{ WHERE } \text{ARTIKEL.ArtikelgruppelId} = \text{ARTIKELGRUPPE.Id}$
Ein Ausschnitt von zwölf Sätzen

ARTIKEL.Id	Artikelnr	ARTIKEL.Bezeichnung	ArtikelgruppelId	LieferantId	Bestandsmenge	Preis	ARTIKELGRUPPE.Id	ARTIKELGRUPPE.Bezeichnung
1	A0010002	Lampenschirme	1	6	669	10,13	1	Kleinteile
2	A0090001	Hilfsmotoren	9	5	514	99,23	9	Kokolores
3	A0020005	DVD-Player	2	15	848	112,34	2	Elektronik-Fachgerät
4	A0040001	Steinway-Flügel	4	6	7	52.000,00	4	Musikalien
5	A0090002	Topfpflanzen	9	19	356	20,65	9	Kokolores
6	A0020002	Akku	2	5	890	2,37	2	Elektronik-Fachgerät
7	A0040003	Altsaxofon	4	19	98	4.000,00	4	Musikalien
8	A0030001	Bilderrahmen	3	22	767	177,85	3	Haushaltswaren
9	A0020006	Computer	2	15	390	1.119,98	2	Elektronik-Fachgerät
10	A0060001	Gödel Escher Bach	6	22	640	49,90	6	Bücher
11	A0040002	Fake Book	4	6	920	42,00	4	Musikalien
12	A0010004	Schrauben	1	10	689	2,39	1	Kleinteile

Etwas stört beim zweiten Beispiel: Wenn Sie die 4. **Tabellenspalte** (hier stehen die Werte des Attributs *ARTIKEL.ArtikelgruppeId*) mit der 8. **Tabellenspalte** (hier stehen die Werte des Attributs *ARTIKELGRUPPE.Id*) vergleichen, stellen Sie fest, dass hier grundsätzlich die gleichen Werte stehen. Schließlich war es ja genau das, was unsere Restriktionsbedingung verlangte. Wir wollen diese Information aber natürlich nur einmal sehen. Um das zu erreichen, brauchen wir einen weiteren Operator, den wir jetzt definieren werden.

5.2.6 Der Operator PROJECT oder: Die Projektion

1. Kurze Beschreibung von PROJECT

Der Operator **PROJECT** projiziert eine Relation auf eine Menge, deren Elemente „schmäler“ sind – die also weniger Komponenten haben.

2. Definition von PROJECT

Es sei R eine Relation, $R \subseteq A_1 \times \dots \times A_n$, weiter seien für $m \leq n$ die m -elementigen Mengen

$$\{i_1, \dots, i_m\} \subseteq \{1, \dots, n\} \quad \text{bzw.} \quad \{A_{i_1}, \dots, A_{i_m}\} \subseteq \{A_1, \dots, A_n\} \quad \text{gegeben}$$

Es sei weiter $S = A_{i_1} \times \dots \times A_{i_m}$ mit $i_1 < \dots < i_m$

dann ist **PROJECT** (R) (A_{i_1}, \dots, A_{i_m}) $\subseteq S$ gerade die Relation

$$\left\{ (x_{i_1}, \dots, x_{i_m}) \in S \mid \begin{array}{l} \text{Es gibt ein } (y_1, \dots, y_n) \in R, \text{ sodass} \\ \text{für alle } i_j \text{ gilt: } x_{i_j} = y_{i_j} \text{ für } 1 \leq j \leq m \end{array} \right\}$$

3. Beispiele für PROJECT

Wir betrachten zwei Beispiele:

Beispiel 1

Sei R die Menge der Artikel und wir wollen die Projektion

PROJECT(**ARTIKEL**) (*Bezeichnung, Bestandsmenge*)

sehen. Die folgende Tabelle zeigt Ihnen die ersten fünf Sätze, die Sie bei einer absteigenden Sortierung nach *Bestandsmenge* erhalten:

Tabelle 5.7: PROJECT(ARTIKEL) (Bezeichnung, Bestandsmenge): Fünf Sätze

Bezeichnung	Bestandsmenge
Playstation	973
Fake Book	920
Akku	890
Fön	887
Joghurt	887

Beispiel 2

Das zweite Beispiel setzt die Bearbeitung unserer Restriktion des Produktes ARTIKEL \times ARTIKELGRUPPE fort. Wir benutzen jetzt die Projektion, um eine der beiden „*ArtikelgruppeId*“-Spalten herauszuprojizieren. Die Formel lautet:

```
PROJECT
(
  PRODUKT(ARTIKEL,ARTIKELGRUPPE)
  WHERE ARTIKEL.ArtikelgruppeId = ARTIKELGRUPPE.Id
)
(
  ARTIKEL.Id, ARTIKEL.Artikelnr, ARTIKEL.Bezeichnung,
  ARTIKEL.ArtikelgruppeId, ARTIKEL.LieferantId,
  ARTIKEL.Bestandsmenge, ARTIKEL.Preis,
  ARTIKELGRUPPE.Bezeichnung)
)
```

(das sind alle Attribute des Produkts mit Ausnahme des doppelten Werts der Id der ARTIKELGRUPPE)

Und die Tabelle 5.8 zeigt Ihnen (wieder) die ersten zwölf Elemente dieser Relation – in derselben Sortierung wie in den vorangegangenen Beispielen. Damit haben Sie schon ein Beispiel für unseren nächsten Operator gesehen: den **Join**.

Tabelle 5.8: PROJECT(PRODUKT(ARTIKEL,ARTIKELGRUPPE) WHERE ARTIKEL.ArtikelgruppelId = ARTIKELGRUPPE.Id) (alles außer ARTIKELGRUPPE.Id): Ein Ausschnitt von zwölf Sätzen

ARTIKEL.Id	Artikelnr	ARTIKEL.Bezeichnung	ArtikelgruppelId	LieferantId	Bestandsmenge	Preis	ARTIKELGRUPPE.Bezeichnung
1	A0010002	Lampenschirme	1	6	669	10,13	Kleinteile
2	A0090001	Hilfsmotoren	9	5	514	99,23	Kokolores
3	A0020005	DVD-Player	2	15	848	112,34	Elektronik-Fachgerät
4	A0040001	Steinway-Flügel	4	6	7	52.000,00	Musikalien
5	A0090002	Topfpflanzen	9	19	356	20,65	Kokolores
6	A0020002	Akku	2	5	890	2,37	Elektronik-Fachgerät
7	A0040003	Altsaxofon	4	19	98	4.000,00	Musikalien
8	A0030001	Bilderrahmen	3	22	767	177,85	Haushaltswaren
9	A0020006	Computer	2	15	390	1.119,98	Elektronik-Fachgerät
10	A0060001	Gödel Escher Bach	6	22	640	49,90	Bücher
11	A0040002	Fake Book	4	6	920	42,00	Musikalien
12	A0010004	Schrauben	1	10	689	2,39	Kleinteile

5.2.7 Die JOIN-Operatoren oder: Verbindungen von Tabellen

Der natürliche Join (the natural join) – zunächst die einfachste Form

Wir beginnen mit unseren Definitionen zunächst bei der einfachsten Form des natürlichen Joins, das ist der Join über nur **ein** Attributpaar. Unmittelbar danach geben wir die Definition für den allgemeinen Fall.

1. Kurze Beschreibung von NATURAL JOIN in der einfachsten Form

Der **NATURAL JOIN** in der einfachsten Form (natürlicher Join) ist eine Hintereinanderausführung von Produkt, Restriktion und Projektion. Ausgangsbasis sind zwei Relationen R und S , die je eine Komponente besitzen, über die man diese beiden Relationen miteinander verbinden möchte. Beispielsweise die ArtikelgruppeId in der Relation ARTIKEL mit der Id in der Relation ARTIKELGRUPPE. Das **Produkt** verbindet jedes Element aus R mit jedem Element aus S . Die anschließende **Restriktion** sorgt dafür, dass nur noch die Sätze „übrig“ bleiben, bei denen die ausgewählte Komponente aus R mit der entsprechenden Komponente aus S übereinstimmt. Bei einem NATURAL JOIN werden diese zwei identischen Attributwerte aber nur einmal in einem Attribut geführt. Die zweite diesbezügliche Spalte wird mit einer **Projektion** unterdrückt.

2. Definition von NATURAL JOIN in der einfachsten Form

Es seien R und S zwei Relationen, $R \subseteq A_1 \times \dots \times A_m$, $S \subseteq B_1 \times \dots \times B_n$

Es seien $i \in \{1, \dots, m\}$ und $j \in \{1, \dots, n\}$ fest vorgegeben.

Für $\Phi : R \times S \rightarrow \{false, true\}$ gelte:

$\Phi(x, y) = true$ genau dann, wenn für den Tupelwert $a_i \in R.A_i$ von x und für den Tupelwert $b_j \in S.B_j$ von y gilt: $a_i = b_j$

Dann ist

$R \text{ NATURAL JOIN } S \text{ ON } \Phi$ bzw.

$R \text{ NATURAL JOIN } S \text{ ON } R.A_i = S.B_j$

gerade die Relation, die man durch Hintereinanderschaltung der folgenden drei Operatoren erhält:

$PROJECT(RESTRICT(PRODUCT(R, S), \Phi))$

(alle Komponenten von R und S , aber ohne $S.B_j$)

3. Beispiel für NATURAL JOIN in der einfachsten Form

Die beliebteste Form des natürlichen Joins findet über die Verbindung zwischen einem Fremdschlüssel in einer Tabelle und dem referenzierten Schlüsselkandidat in der dazugehörigen anderen Relation statt. Genauer erfahren Sie im nächsten Kapitel, aber unser letztes Beispiel ist genau von dieser Art. Für

PROJECT

```
(
  PRODUKT(ARTIKEL,ARTIKELGRUPPE)
  WHERE ARTIKEL.ArtikelgruppeId = ARTIKELGRUPPE.Id
)
(
  ARTIKEL.Id, ARTIKEL.Artikelnr, ARTIKEL.Bezeichnung,
  ARTIKEL.ArtikelgruppeId, ARTIKEL.LieferantId,
```

ARTIKEL.*Bestandsmenge*, ARTIKEL.*Preis*,
 ARTIKELGRUPPE.*Bezeichnung*)
)

können wir genauso schreiben:

ARTIKEL NATURAL JOIN ARTIKELGRUPPE
 ON ARTIKEL.*ArtikelgruppeId* = ARTIKELGRUPPE.*Id*

Der natürliche Join (the natural join) in der allgemeinen Form

Für die allgemeine Form des natürlichen Joins wird die Bedingung, dass genau ein Attributpaar aus den beiden Relationen aus übereinstimmenden Werten bestehen muss, dahingehend verallgemeinert, dass es sich jetzt auch um mehrere Attributpaare handeln kann. Diese Verallgemeinerung brauchen Sie immer dann, wenn beispielsweise ein Fremdschlüssel in einer Tabelle TA, der sich auf einen Primärschlüssel einer Tabelle TB bezieht, aus mehreren Attributen besteht. Dann müssen alle Werte zweier **Attributkombinationen** (einer aus TA und einer aus TB) miteinander übereinstimmen. Ich gebe Ihnen jetzt die formalen Definitionen:

1. Kurze Beschreibung von NATURAL JOIN in der allgemeinen Form

Beim **NATURAL JOIN in der allgemeinen Form** besteht die Ausgangsbasis wieder aus zwei Relationen R und S. Diese werden zunächst mit einem Produkt miteinander „gekreuzt“. Diese beiden Relationen müssen jetzt aber nicht durch ein einzelnes Komponentenpaar miteinander verbunden sein, sondern es ist eine ganze Gruppe solcher Paare erlaubt. Eine anschließende Restriktion sorgt dafür, dass nur die Produktsätze „übrig“ bleiben, bei denen alle Paarwerte dieser Gruppe jeweils identisch sind. Alle zweiten Partner einer solchen Gruppe werden dann durch eine **Projektion** unterdrückt.

2. Definition von NATURAL JOIN in der allgemeinen Form

Es seien R und S zwei Relationen, $R \subseteq A_1 \times \dots \times A_m$, $S \subseteq B_1 \times \dots \times B_n$

Weiter sei $k \in \mathbb{N}$ und es seien in beiden Relationen jeweils k Tupel fest ausgewählt:

Für R die Tupel $\alpha_1, \dots, \alpha_k$ mit $\{\alpha_1, \dots, \alpha_k\} \subseteq \{A_1, \dots, A_m\}$ und

Für S die Tupel β_1, \dots, β_k mit $\{\beta_1, \dots, \beta_k\} \subseteq \{B_1, \dots, B_n\}$

Für $\Phi: R \times S \rightarrow \{\text{false}, \text{true}\}$ gelte:

$\Phi(x, y) = \text{true}$ genau dann, wenn für alle $q \in \mathbb{N}$ mit $1 \leq q \leq k$ gilt: Der Tupelwert $a_q \in R.\alpha_q$ von x und der Tupelwert $b_q \in S.\beta_q$ von y stimmen überein, d.h. es gilt: $a_q = b_q$.

Dann ist

R NATURAL JOIN S ON Φ bzw.

R NATURAL JOIN S

ON $R.\alpha_1 = S.\beta_1$ AND ... AND $R.\alpha_k = S.\beta_k$

gerade die Relation, die man durch Hintereinanderschaltung der folgenden drei Operatoren erhält:

PROJECT(RESTRICT(PRODUCT(R, S), Φ))

(alle Mengen von R und S, aber ohne $S.\beta_1, \dots, S.\beta_k$)

Wenn man beim **NATURAL JOIN** die abschließende Projektion weglässt, erhält man einen **EQUIJOIN**. Dieser Name macht noch klarer, dass das entscheidende Zeichen, das die Restriktion kennzeichnet, die im Anschluss an die Projektion durchgeführt wird, das Gleichheitszeichen ist. Denken Sie an „ $a_q = b_q$ “ oder „für alle $1 \leq q \leq k$ muss gelten: $a_q = b_q$ “

Statt dieses Gleichheitszeichens sind auch andere Zeichen zur Definition einer Restriktion denkbar. Sie können neben dem „=“-Zeichen Vergleichsoperatoren aus der Menge $\{<, <=, !=, >, >=\}$ benutzen, um Restriktionen zu definieren. Beispielsweise führte die Auswahl von „<“ zu Bedingungen wie:

„ $a_q < b_q$ “ oder „für alle $1 \leq q \leq k$ muss gelten: $a_q < b_q$ “

Die Menge der Joins, zu der alle Joins mit all diesen sechs möglichen Vergleichszeichen gehören, nennt man die Menge der **θ -JOINS** (sprich: Theta Joins).

Die *Definition* lautet:

Definition 12:

Es seien R und S zwei Relationen, $R \subseteq A_1 \times \dots \times A_m$, $S \subseteq B_1 \times \dots \times B_n$
 Weiter sei $k \in \mathbb{N}$ und es seien in beiden Relationen jeweils k Tupel fest ausgewählt:
 Für R die Tupel $\alpha_1, \dots, \alpha_k$ mit $\{\alpha_1, \dots, \alpha_k\} \subseteq \{A_1, \dots, A_m\}$ und
 Für S die Tupel β_1, \dots, β_k mit $\{\beta_1, \dots, \beta_k\} \subseteq \{B_1, \dots, B_n\}$
 Es sei $\theta \in \{<, <=, =, !=, >, >=\}$
 Für $\Phi: R \times S \rightarrow \{\text{false}, \text{true}\}$ gelte:
 $\Phi(x, y) = \text{true}$ genau dann, wenn für alle $q \in \mathbb{N}$ mit $1 \leq q \leq k$ gilt: Für die Tupelwerte $a_q \in R.\alpha_q$ von x und $b_q \in S.\beta_q$ von y gilt: $a_q \theta b_q$.
 Dann ist
R θ -JOIN S ON Φ bzw.
R θ -JOIN S ON R. α_1 θ S. β_1 AND ... AND R. α_k θ S. β_k
 gerade die Relation, die man durch Hintereinanderschaltung der folgenden drei Operatoren erhält:
 RESTRICT(PRODUCT(R, S), Φ)

Es fehlt noch ein letzter Operator: der **DIVIDE**-Operator.

5.2.8 Der DIVIDE-Operator

Wir werden den DIVIDE-Operator nicht genau besprechen und definieren. Gegebenenfalls konsultieren Sie die einschlägige Literatur [Schube]. Nur so viel:

Kurze Beschreibung von DIVIDE in der allgemeinen Form

Der DIVIDE-Operator „dividiert“ Tupel aus einer Relation (dem Zähler) mithilfe einer anderen Relation (dem Nenner) heraus. Wie soll das gehen?

Beim Rechnen mit Zahlen ist Ihnen klar: Die **Division ist die Umkehrung der Multiplikation**. Es gilt beispielsweise:

$$\text{DIVIDE}(\text{PRODUCT}(3, 4), 4) = 3 \cdot 4 / 4 = 3$$

Genauso gilt für den Relationsoperator DIVIDE: Seien R, S Relationen. Dann gilt:

$$\text{DIVIDE}(\text{PRODUCT}(R, S), S) = R$$

5.3 Der Nutzen dieser Operatoren

Wie Sie sicher schon gemerkt haben, gehört es zu den Eigentümlichkeiten in relationalen Datenbanken, dass man zur Vermeidung von Datenredundanzen seine Daten auf mehrere kleinere Tabellen und Relationen aufteilt. Deshalb braucht man Werkzeuge, um diese Tabellen wieder zusammenzuführen und benutzerverständliche Auskünfte über den Datenbestand geben zu können. Denn: kein Benutzer auf dieser Welt interessiert sich für den Fremdschlüsselwert der Artikelgruppe „Kleinteile“. Aber jeder Benutzer unserer Datenbank „LieferBar“ will wissen, wie die Artikelgruppe des Artikels „Schrauben“ heißt. Dafür brauchen wir den Join.

Wir brauchen diese Operatoren außerdem unter anderem für die folgenden Zwecke: Für die Definition von

- Teilmengen einer Relation, die gesucht werden sollen
- Teilmengen einer Relation, die woanders eingefügt werden sollen
- Teilmengen einer Relation, die nach einer bestimmten Regel verändert werden sollen
- Teilmengen einer Relation, die gelöscht werden sollen
- Integritätsbedingungen an eine Datenbank (vergleichen Sie dazu bitte das folgende Kapitel 6)
- usw., usw.

5.4 Anforderungen an eine Datensprache

Von daher ist klar, dass wir an jeden Anbieter einer Datensprache die entscheidende Frage stellen müssen:

Können mit dieser Sprache diese acht Operatoren abgebildet werden?

Sie werden in einem anderen Kurs sehen, dass das mit SQL problemlos geht.

Zusammenfassung

Wir haben in diesem Kapitel acht grundlegende Operatoren beschrieben, nämlich

- die Vereinigung UNION
- der Durchschnitt INTERSECTION
- die Differenz DIFFERENCE
- das Produkt PRODUCT
- die Restriktion RESTRICT
- die Projektion PROJECT
- die Verbindung JOIN
- die Division DIVIDE

Anschließend haben wir noch einmal über den Nutzen dieser Operatoren nachgedacht und die wichtigste Anforderung an eine Datensprache für relationale Datenbanken formuliert: Diese acht Operatoren müssen in jeder Datensprache darzustellen sein.

Aufgaben zur Selbstüberprüfung

5.1 Sie wollen alle Attribute der Artikel sehen, die bestellt wurden. Stellen Sie diese Abfrage auf dreierlei Weisen dar:

- a) In der Form ARTIKEL WHERE *Bedingungen*
- b) In der Form


```
PROJECT
(
  ARTIKEL EQUIJOIN BESTELLUNGEN
  ON Bedingung
)
```

 (Alle Attribute von ARTIKEL)

c) In der Form

```
PROJECT  
(  
  ARTIKEL × BESTELLUNGEN  
  WHERE Bedingung  
)  
(Alle Attribute von ARTIKEL)
```

5.2 Bestimmen Sie Grad und Kardinalität von:
ARTIKELGRUPPE × ARTIKEL

5.3 Beschreiben Sie den einzigen sinnvollen natürlichen Join zwischen
ARTIKEL und ARTIKELGRUPPE. Beschreiben Sie ihn in Worten und
auf zwei verschiedene Weisen als Formel mithilfe der Operatoren aus
diesem Kapitel.

Kapitel 6

6 Die Integrität einer Datenbank und Schlüssel aller Art

Man möchte, dass die abgespeicherten Daten einer Datenbank den Ausschnitt der Realität, den sie abbilden sollen, auch korrekt repräsentieren. Dabei gibt es zwei verschiedene Arten von Bedingungen, die solch eine korrekte Datenbank erfüllen muss:

- *Wahrheitsanforderungen, die nur durch den Vergleich mit der Realität überprüft werden können. Zum Beispiel: Wohnt Albert Einstein wirklich in der Raumstraße 2, wie wir in der Tabelle PERSON gespeichert haben? Oder: Ist der Kunde mit der Id 16 wirklich der Besteller des Artikels mit der Id 19 – so, wie es in der Tabelle BESTELLUNGEN gespeichert wurde?*
- *Logische Integritätsanforderungen, die die Gestalt der einzelnen Tabellen und die Beziehungen zwischen den verschiedenen Relationen einer Datenbank betreffen. Wir können uns zum Beispiel genauso gut fragen: Gibt es überhaupt einen Kunden mit der Id 16 in der Tabelle KUNDE, denn schließlich wird auf diese KUNDE.Id ja in der Tabelle BESTELLUNGEN Bezug genommen. Offensichtlich muss diese Frage zuerst geklärt sein, ehe ich die Frage aus Punkt 1 überhaupt erfolgversprechend untersuchen kann.*

Kriterien der letzteren Art, die zwar nicht die vollständige inhaltliche Korrektheit der gespeicherten Daten sichern, sondern nur die Einhaltung bestimmter logischer Grundvoraussetzungen für diese Korrektheit betreffen, nennen wir Integritätsbedingungen. Sie können durch ein DBMS garantiert werden. Es wird sich zeigen, dass man einige Bedingungen so allgemein formulieren kann, dass sie als Kriterien für jede Datenbank dienen können – unabhängig vom konkreten Inhalt und Entwurf der Datenbank. Eine zentrale Rolle spielen dabei die Begriffe Schlüsselkandidat, Primärschlüssel und Fremdschlüssel. Wir werden diese Begriffe jetzt der Reihe nach untersuchen.

6.1 Schlüsselkandidaten (Candidate Keys)

Insbesondere für die Verwaltungen der Beziehungen zwischen zwei Relationen (denken Sie zum Beispiel an die Beziehung zwischen Artikeln und Kunden in der Relation BESTELLUNGEN) braucht man die Möglichkeit, die Elemente einer Relation eindeutig und redundanzfrei zu kennzeichnen. Alle derartigen Möglichkeiten sind Kandidaten für einen Schlüssel und man definiert deshalb:

Definition 13:

R sei eine Relation. Eine Teilmenge K der Attribute von R heißt **Schlüsselkandidat**, wenn die folgenden zwei Bedingungen erfüllt sind:

- **Die Eindeutigkeit:** Jedes Element von R ist eindeutig durch die Werte der Attribute von K charakterisiert. Zwei voneinander verschiedene Elemente in R können nie in den Werten der Attribute aus K übereinstimmen. Diese Tatsache gilt für einen Schlüsselkandidaten prinzipiell – unabhängig davon, wie viele Elemente gerade aktuell zu der Relation gehören.
- **Irreduzibilität:** K verliert die Eigenschaft der Eindeutigkeit, sobald man ein oder mehrere Attribute aus K entfernt.

Diese Irreduzibilität wird oft auch **Minimalität** genannt. Beachten Sie bitte, dass jede Relation mindestens einen Schlüsselkandidaten hat – denn Relationen enthalten keine doppelten Elemente. Also erfüllt die Menge aller Attribute einer Relation schon die Bedingung der Eindeutigkeit. Und diese Menge kann man nun – oft auf mehrere Weisen – so verkleinern, bis sie bei einer weiteren Verkleinerung diese Eigenschaft der Eindeutigkeit verlieren würde – bis sie also irreduzibel ist. Folgende Eigenschaften von Schlüsselkandidaten sollten klar sein:

- Oft haben Relationen wirklich nur einen Kandidaten für einen Schlüssel. Das muss aber nicht so sein. Es kann durchaus mehrere Schlüsselkandidaten geben.
- **Schlüsselkandidaten** können Mengen von Attributen sein, die nur ein einziges Element enthalten. Dann heißen sie **einfach**. Falls sie aus mehreren Attributen bestehen, heißen sie **zusammengesetzt**.

6.2 Primärschlüssel

Über Primärschlüssel haben wir in diesem Heft schon mehrfach gesprochen. Das Interessante ist: Für Schlüsselkandidaten gibt es ein theoretisches Kriterium, quasi eine mathematische Formel, die Sie auf eine Relation legen können und diese Formel sagt Ihnen, welche Attributkombinationen Schlüsselkandidaten sind. Nicht so bei Primärschlüsseln.

Definition 14:

Der **Primärschlüssel** einer Relation ist derjenige Schlüssel unter den Schlüsselkandidaten, für den sich der Datenbankadministrator beim Entwurf der Tabelle als passende Charakterisierung der Elemente der Tabelle entscheidet. Der Primärschlüssel ist wichtig für das Datenbankmanagementsystem. Es benutzt ihn zum raschen Lokalisieren der entsprechenden Sätze auf dem Festspeichermedium und zur Verwaltung der Beziehungen zwischen verschiedenen Tabellen. Wichtige Kriterien für einen „guten“ Primärschlüssel sind:

- Der Primärschlüssel sollte eine Attributkombination oder ein Attribut sein, das keinerlei Bedeutung für den Anwender hat. Nur so können Sie sicher sein, dass die datenbankinterne Logik und die anwenderspezifische Logik nicht miteinander in Konflikt kommen können.
- Außerdem sollte die Attributkombination des Schlüsselkandidaten, den Sie zum Primärschlüssel machen, eine möglichst einfache Gestalt haben.

Der Primärschlüssel wird mehr als alle anderen Schlüsselkandidaten in anderen Relationen benutzt, um sich auf Elemente der ursprünglichen Relation zu beziehen. So wird zum Beispiel der Primärschlüssel *Id* der Relation KUNDE benutzt, um in der Relation BESTELLUNGEN einen bestimmten Kunden zu kennzeichnen, der die betreffende Bestellung ausgeführt hat. Oder anders gesagt: Der Kunde, der zu dem Element aus der Relation BESTELLUNGEN gehört, wird über den Primärschlüssel KUNDE.*Id* referenziert.

Es gilt darüber hinaus: Viele Datenbankmanagementsysteme können die jeweiligen Ausprägungen der referenziellen Integrität nur dann sichern und verwalten, wenn die entsprechenden Beziehungen mithilfe eines Primärschlüssels erstellt wurden. Sie sehen, der Begriff des Primärschlüssels führt uns automatisch zu unserem nächsten Thema, dem Fremdschlüssel.

6.3 Fremdschlüssel und referentielle Integrität

Definition 15:

Sei R eine Relation. Eine Menge FS von Attributen von R heißt **Fremdschlüssel**, falls die folgende Bedingung erfüllt ist:

- Es gibt eine Relation S mit einem Schlüsselkandidaten SK , sodass jede Wertbelegung für die Attribute in FS mit einer Wertbelegung für die Attribute in SK übereinstimmt.

Betrachten Sie noch einmal ein Beispiel aus unserer Beispieldatenbank, nämlich die Relation ARTIKEL, dargestellt als Tabelle 6.1, wo Sie einige Sätze sehen.

Wir hatten schon in Kapitel 3 festgestellt:

- **Id** ist der Primärschlüssel. Das heißt: Einer der möglicherweise mehreren Schlüsselkandidaten ist die Menge mit dem Attribut **Id**. Wir machen diesen Schlüsselkandidaten, der sehr einfach strukturiert ist, zum Primärschlüssel. Um noch einmal den Begriff des Schlüsselkandidaten zu erläutern: Auch die Attributkombination (**Id**, **Bezeichnung**) ist eindeutig, aber sie ist **kein** Schlüsselkandidat, denn diese Attributkombination verliert das Charakteristikum der Eindeutigkeit nicht, wenn man das Attribut **Bezeichnung** aus ihr entfernt. Diese Attributkombination ist also **nicht minimal** in Bezug auf die Anforderung der Eindeutigkeit.

Tabelle 6.1: Die ersten zehn Artikel bei alphabetischer Sortierung nach der **Bezeichnung**

Id	Artikelnr	Bezeichnung	ArtikelgruppelId	LieferantId	Bestandsmenge	Preis
6	A0020002	Akku	2	5	890	2,37
7	A0040003	Altsaxofon	4	19	98	4.000,00
24	A0050002	Banane	5	10	521	0,31
8	A0030001	Bilderrahmen	3	22	767	177,85
9	A0020006	Computer	2	15	390	1.119,98
14	A0060003	Der Name der Rose	6	6	562	26,21
15	A0060002	Der Termin	6	22	153	16,00
29	A0020003	Drucker	2	11	0	2.300,00
16	A0020004	Drucker	2	14	8	879,99
3	A0020005	DVD-Player	2	15	848	112,34

- **ArtikelgruppeId** ist ein Fremdschlüssel, der die Werte des Primärschlüssels der Tabelle ARTIKELGRUPPE annimmt. Es besteht eine Referenz von **ARTIKEL.ArtikelgruppeId** auf **ARTIKELGRUPPE.Id**. Damit wird gespeichert, zu welcher Artikelgruppe der Artikel gehört.
- **LieferantId** ist ein Fremdschlüssel, der die Werte des Primärschlüssels der Tabelle LIEFERANT annimmt. Es besteht eine Referenz von **ARTIKEL.LieferantId** auf **LIEFERANT.Id**. Damit wird der Lieferant des Artikels gespeichert. Jeder Artikel hat höchstens einen Lieferanten in diesem Versandhaus.

Nun können wir eine der wichtigsten relationalen Integritätsbedingungen formulieren, die sogenannte **referentielle Integrität**:

Definition 16: Definition der referentiellen Integrität

Zu jedem Wert eines Fremdschlüssels muss es ein Element in der referenzierten Relation geben, dessen referenzierter Schlüsselkandidat genau diesen Wert hat.

Diese referentielle Integrität kann auf verschiedene Weise verletzt werden und Sie haben verschiedene Möglichkeiten, in Ihrem Datenbankmanagementsystem festzulegen, wie es darauf reagieren soll:

1. Diese referentielle Integrität kann beim Hinzufügen von Elementen in die Relation, die den Fremdschlüssel enthält, verletzt werden. Falls Sie einen Satz hinzufügen, bei dem der Fremdschlüssel einen Wert hat, der nicht in dem entsprechenden Schlüsselkandidaten bei der referenzierten Tabelle vorkommt, haben Sie die referentielle Integrität verletzt.

Beispiel:

Sie fügen in die Relation ARTIKEL das Element

(31, 'A0990001', 'Fehlerfreie Software', 99, 15, 7, 33.00)

hinzu, obwohl es die Artikelgruppe mit der *Id* 99 gar nicht gibt. Dann ist die Integrität Ihrer Datenbank verletzt. Sie sollten deshalb Ihr Datenbankmanagementsystem anweisen können, solche Einfügeoperationen zu verbieten. Hier gibt es für ein Datenbankmanagementsystem nur diese eine Möglichkeit zu reagieren.

2. Die referentielle Integrität kann beim Löschen von Elementen aus der Relation, in der der Schlüsselkandidat steht, auf den von anderen Relationen hin referenziert wird, verletzt werden. Sie dürfen offensichtlich keine Elemente entfernen, deren Schlüsselkandidaten noch als Fremdschlüssel in anderen Relationen auftauchen.

Beispiel:

Sobald Sie in der Relation ARTIKELGRUPPE das Element

(5, 'Lebensmittel')

löschen, verlieren alle Elemente in der Relation ARTIKEL, bei denen als *ArtikelgruppeId* der Wert 5 angegeben war, ihre Beziehung zur Relation ARTIKELGRUPPE. Die Datenbank ist nicht mehr in einem „integeren“ Zustand. Sie haben bei den meisten relationalen Datenbankmanagementsystemen zwei verschiedene Möglichkeiten der Reaktion, die festgelegt werden können:

- Das Löschen der Artikelgruppe mit der *Id* 5 wird verboten, solange es noch Artikel aus dieser Artikelgruppe gibt.

- Alle Artikel, die zu der Artikelgruppe 5 gehören, werden mitgelöscht. Aber Vorsicht: So kann man seine Datenbank sehr schnell sehr kleinkriegen, denn: In unserem Fall bedeutet das auch: Alle Bestellungen von Artikeln der Artikelgruppe 5 müssen gelöscht werden. Und alle erledigten Bestellungen.
3. Schließlich können Sie die referentielle Integrität durch das Ändern von Elementen verletzen. Entweder, indem Sie den Fremdschlüssel eines Elementes in einer Relation auf einen Wert ändern, dem kein Schlüsselkandidatenwert in der referenzierten Tabelle entspricht. Oder, indem Sie den Schlüsselkandidaten des Elementes einer Relation verändern, auf den sich Fremdschlüssel aus anderen Relationen beziehen.

Beispiele:

Sie ändern in der Relation ARTIKEL das Element

(11, 'A0040002', 'Fake Book', 4, 6, 920, 42.00)

in

(11, 'A0040002', 'Fake Book', **99**, 6, 920, 42.00)

Dann haben Sie dasselbe Problem wie bei unserem Einfügebeispiel. Wieder gibt es für ein Datenbankmanagement nur eine Möglichkeit, zu reagieren – wenn es denn überhaupt etwas tun soll: durch Ablehnung.

Oder aber: Sie ändern in der Relation ARTIKELGRUPPE beispielsweise das Element

(5, Lebensmittel) in (55, Lebensmittel)

Dann entsteht dieselbe problematische Situation wie bei unserem Löscheispiel. Jetzt hat ein Datenbankmanagementsystem wieder zwei Möglichkeiten zu reagieren:

- Entweder wird jedwede Änderung eines Primärschlüssels, auf den sich andere Fremdschlüssel beziehen, verboten, d.h., die obige Änderung wird verweigert.
- Oder alle Fremdschlüsselwerte in der Tabelle ARTIKEL, die den Wert 5 haben (die sich also auf 'Lebensmittel' beziehen) werden mit auf den Wert 55 geändert.

Wir wollen das, was wir hier an einem konkreten Beispiel besprochen haben, noch einmal allgemein formulieren:

6.4 Regeln beim Umgang mit Fremdschlüsseln

Eine Warnung vorneweg: Einige Datenbankmanagementsysteme sind nur mit unzureichenden Mitteln zur Sicherung der referentiellen Integrität ausgestattet. Je nach dem System, mit dem Sie arbeiten, werden Sie zusätzliche Programmierarbeit leisten müssen, um die im Folgenden beschriebenen Vorgehensweisen zu realisieren.

Der Datenbankdesigner muss sich beim Entwurf eines Fremdschlüssels stets mit den folgenden Fragen befassen:

1. Wie soll sich das System verhalten, wenn versucht wird, einen Satz aus einer Tabelle zu löschen, die einen Schlüsselkandidaten besitzt, auf den sich ein Fremdschlüssel aus einer anderen Tabelle bezieht? Es gibt (zunächst) zwei Möglichkeiten der Reaktion:

- **Eingeschränkt (engl. restricted):**

Das bedeutet, das System schränkt die Möglichkeit, einen Satz zu löschen, auf die Fälle ein, in denen es keine Bezüge durch andere Fremdschlüssel gibt.

- **Das sogenannte Kaskadenlöschen (engl. cascades):**

In diesem Falle werden die Sätze in den anderen Tabellen, die einen Fremdschlüsselwert enthalten, der sich auf den zu löschenden Satz bezieht, mitgelöscht. Beachten Sie, dass das unter Umständen eine Kettenreaktion von Löschvorgängen auslösen kann.

Die andere Frage, mit der sich ein Datenbankdesigner beim Entwurf eines Fremdschlüssels befassen muss, lautet:

2. Wie soll sich das System verhalten, wenn der Endbenutzer versucht, einen Schlüsselkandidaten eines Satzes aus einer Tabelle zu ändern, auf den sich ein Fremdschlüssel aus einer anderen Tabelle bezieht? Es gibt wieder zwei Möglichkeiten der Reaktion:

- **Eingeschränkt (engl. restricted):**

Das bedeutet, das System schränkt die Möglichkeit, einen Schlüsselkandidaten zu ändern, auf die Fälle ein, in denen es keine Bezüge durch andere Fremdschlüssel aus den entsprechenden anderen Tabellen auf diesen Schlüsselkandidaten gibt.

- **Das sogenannte Kaskadenändern (engl. cascades):**

In diesem Falle werden die Fremdschlüssel in den anderen Tabellen, die sich auf den zu ändernden Schlüsselkandidaten beziehen, mitgeändert.

6.5 Ein Verbot für NULL-Werte – aber nur in Primärschlüsseln

Ich habe schon mehrmals in diesem Heft gegen NULL-Werte polemisiert. Eine minimale Vorsichtsmaßnahme gegen die Verwirrungen, die NULL-Werte stiften, ist die sogenannte Integrität der Entitäten, die **Entity-Integrität**. Da NULL-Werte für Elemente von Relationen keinen Sinn ergeben, spreche ich für den Rest dieses Kapitels nur noch von Tabellen:

Definition 17: Definition der Integrität der Entitäten

Keine Komponente des Primärschlüssels einer Tabelle darf für irgendein Element dieser Tabelle einen NULL-Wert enthalten.

Das bedeutet: Selbst wenn man in seinem Datenbankdesign NULL-Werte zulässt, **muss** es in jeder Tabelle immer mindestens einen Schlüsselkandidaten geben, der keinerlei NULL-Werte enthält.

Mithilfe der NULL-Werte kann man die Regeln für die Fremdschlüssel folgendermaßen abschwächen:

Definition 18: Erweiterte Definition des Fremdschlüssels

Sei R eine Tabelle. Eine Menge FS von Attributen von R heißt **Fremdschlüssel**, falls die folgende Bedingung erfüllt ist:

- Es gibt eine Tabelle S mit einem Schlüsselkandidaten SK, sodass jede Wertbelegung für die Attribute in FS entweder NULL ist oder mit einer Wertbelegung für die Attribute in SK übereinstimmt.

Und wir erhalten auch eine erweiterte Form der **referentiellen Integrität**:

Definition 19: Referentielle Integrität (erweiterte Version)

Zu jedem Wert eines Fremdschlüssels, der nicht NULL ist, muss es ein Element in der referenzierten Tabelle geben, bei dem der referenzierte Schlüsselkandidat genau diesen Wert hat.

Das bedeutet, dass man – wenn man sich auf dieses Spiel mit den NULL-Werten einlässt – gerne sein Datenbankmanagement sowohl für die Operation des Löschs als auch für Update-Operationen zu einer weiteren Form der Reaktion veranlassen können möchte:

Die Option „Setze auf NULL“ (Nullify):

- Jedes Mal, wenn versucht wird, einen Satz aus einer Tabelle zu löschen, die einen Schlüsselkandidaten besitzt, auf den sich ein Fremdschlüssel aus einer anderen Tabelle bezieht, wird der Fremdschlüssel in dieser anderen Tabelle auf NULL gesetzt.
- Und genauso gilt: Wenn man versucht, einen Schlüsselkandidaten eines Satzes aus einer Tabelle zu ändern, auf den sich ein Fremdschlüssel aus einer anderen Tabelle bezieht, wird der Fremdschlüssel in dieser anderen Tabelle auf NULL gesetzt.

In unserer Tabelle BESTELLUNGEN würde die Option NULLIFIES z.B. bedeuten, es gäbe Bestellungen ohne einen Kunden. Sie könnten auch versucht sein, bei dem Fremdschlüssel *ArtikelgruppeId* in der Tabelle ARTIKEL NULL-Werte zuzulassen – z.B. für den Fall, dass für einen neuen Artikel eine neue Artikelgruppe einzurichten wäre, was aus den verschiedensten Gründen erst zu einem späteren Zeitpunkt stattfinden könnte.

In all diesen Fällen würde ich stets lieber mit einer Artikelgruppe der Art „Bezeichnung noch unbekannt“, die eine konkrete *Id* hat, arbeiten. Mögliche NULL-Werte in Fremdschlüsseln werden bei späteren Abfragen, Sortierungen und anderen Verarbeitungen immer wieder Ärger machen. Mein Rat bleibt immer derselbe: Vermeiden Sie NULL-Werte, wo es irgend geht.

6.6 Abschließende Definition der Integrität einer relationalen Datenbank

Mit den jetzt entwickelten Begriffen können wir – nach unserem ersten Versuch im dritten Kapitel – abschließend definieren:

Definition 20: Definition der Integrität einer relationalen Datenbank

Die **Integrität** einer relationalen Datenbank wird durch die Erfüllung der folgenden drei Bedingungen für alle beteiligten Tabellen definiert:

1. die **Entity-Integrität**
2. die **referentielle Integrität**
3. alle weiteren **Constraints** sind erfüllt

Zusammenfassung

Wir haben in diesem Kapitel die folgenden Punkte besprochen:

- Datenbankmanagementsysteme können nur die logische Stimmigkeit, die Integrität einer Datenbank, sichern helfen.
- Die Menge von Attributkombinationen, aus der man den Primärschlüssel auswählt, besteht aus den sogenannten Schlüsselkandidaten. Schlüsselkandidaten sind jeweils eindeutig identifizierend für ein Element der Relation und verlieren diese Eigenschaft, wenn man sie verkleinert.
- Primärschlüssel sollten frei von Anwenderbedeutung und so einfach wie möglich strukturiert sein. Sie werden vom Datenbankmanagementsystem zur Optimierung von Suchzugriffen auf der Festplatte und zur Verwaltung von Beziehungen zwischen verschiedenen Relationen gebraucht.
- Die Beziehung zwischen zwei Tabellen wird über einen Schlüsselkandidaten und einen Fremdschlüssel, der sich auf diesen Schlüsselkandidaten bezieht, geregelt. Die Korrektheit dieser Beziehung nennt man referentielle Integrität.
- Die referentielle Integrität kann bei Einfüge-, Lösch- und Update-Operationen verletzt werden. Es gibt für das Datenbankmanagementsystem drei Möglichkeiten darauf zu reagieren: Mit der Verbotsoption „Restricted“, mit der Weitergabeoption „Cascades“ und mit der Option „Nullify“, die Fremdschlüsselwerte ohne korrespondierende Schlüsselkandidatenpartner auf NULL setzt.
- Die Entity-Integrität verbietet NULL-Werte in Primärschlüsseln.

Mithilfe dieser Punkte war dann eine neue Definition der Integrität einer relationalen Datenbank möglich, die knapper und präziser war als unser erster Definitionsversuch im dritten Kapitel.

Aufgaben zur Selbstüberprüfung

6.1 (Warnung: Wie Sie bereits wissen, ist die folgende Tabelle voller Entwurfsfehler. Wir werden sie im 8. Kapitel genauer untersuchen.)

- a) In einer Bibliothek will man eine Datenbank einrichten, in welcher der Buchbestand verwaltet wird. Sehen Sie in der folgenden Abbildung die Tabelle, die man dazu implementiert:

	Id	Autor	Titel	Verlag
	12	Goll	JAVA als erste Programmiersprache	Teubner
	14	Johnson	Entwurfsmuster	Addison-Wesley
	16	Weiß	JAVA als erste Programmiersprache	Teubner
	17	Gamma	Entwurfsmuster	Addison-Wesley
	25	Zschiegner	Diskrete Mathematik	Vieweg
	76	Dieker	Datenstrukturen und Algorithmen	Teubner
	136	Beutelspacher	Diskrete Mathematik	Vieweg
	183	Beutelspacher	Kryptologie	Vieweg
	212	Helms	Entwurfsmuster	Addison-Wesley
	815	Müller	JAVA als erste Programmiersprache	Teubner

Warum ist die Attributkombination (*Id*, *Autor*) kein Schlüsselkandidat?

- b) Diese Tabelle hat zwei Schlüsselkandidaten. Welche sind das?
- c) Für welchen dieser Schlüsselkandidaten würden Sie sich als Primärschlüssel entscheiden? Begründen Sie Ihre Entscheidung.
- 6.2 Nehmen Sie an, wir hätten in unserer Datenbank sämtliche DELETE-Optionen auf CASCADE gesetzt. Nun löschen wir aus der Tabelle ARTIKELGRUPPE den **einen** Satz mit der *Id* 1 und der *Bezeichnung* *Kleinteile*. Wie viele Sätze werden dann insgesamt aus der Datenbank gelöscht? Welche Tabellen sind alle betroffen?
- 6.3 Nehmen Sie nun an, wir hätten in unserer Datenbank sämtliche UPDATE-Optionen auf CASCADE gesetzt. Überlegen Sie, was dann die Veränderung der *Id* der Artikelgruppe *Kleinteile* für Konsequenzen hätte.
- 6.4 (Fortsetzung von Aufgabe 6.3) Überlegen Sie (oder besser noch: Diskutieren Sie mit Kommilitonen oder Kollegen), wie man „default“-mäßig die Delete-Optionen und wie man „default“-mäßig die Update-Optionen bei möglichen Verletzungen der referentiellen Integrität setzen sollte. Sollte man bei Delete und Update identisch verfahren? Betrachten Sie die Situation aus Aufgabe 6.2 dabei als ein repräsentatives Beispiel.

Kapitel 7

7 Das Entity/Relationship-Modell

Bisher hatten wir immer vorgegebene Tabellen, Beziehungen und Constraints, an denen wir die grundlegenden Begriffe aus der Theorie der Datenbanken besprochen haben. In diesem Kapitel (und im folgenden) wird es um Techniken gehen, wie man im Angesicht der oft chaotischen Anwenderwelt zu einem möglichst guten relationalen Datenbankentwurf kommt. „Gut“ heißt: Möglichst frei von Redundanzen, leicht zu warten und unproblematisch zu erweitern. Fehler beim Datenbankentwurf werden vom „System“ leider nie verzeihen und können sehr teuer werden.

7.1 Unser Plan

Zunächst wird es darum gehen, wie man durch eine Analyse der Begriffe aus der Realität eine Struktur modellieren kann, die zur Grundlage des späteren Tabellenentwurfs für die relationale Datenbank wird. Solch eine Struktur besteht aus Entitäten, genauer gesagt: Entitätstypen und den Beziehungen zwischen ihnen. So erklärt sich der Name dieses Modells. Da es hier um die Semantik, d.h. die Bedeutung der Typen bzw. Entitäten generierenden Begriffe geht, spricht man auch von dem **semantischen Datenmodell**.

Leider ist die Sprache der Informatiker in diesem Bereich nicht immer exakt, was es der oder dem Lernenden oft noch zusätzlich erschwert, in diesem wichtigen Gebiet sicherer zu werden. Eine Analogie zur objektorientierten Programmierung verdeutlicht den korrekten Sachverhalt:

E/R-Modellierung	Objektorientierter Entwurf	Beispiel
Entität	Objekt	Jimi Hendrix
Entitätstyp <i>oder auch</i> Entitätsklasse	Klasse	PERSON

Das heißt: **Entitäten** sind „eigentlich“ die einzelnen Objekte unserer Datenbankwelt, die „Typisierung“ oder „Klassifizierung“ dieser einzelnen Objekte in Relationen oder Tabellen sind **Entitätstypen** oder **Entitätsklassen**. Beide Begriffe (Entitätstypen und Entitätsklassen) werden verwendet, sie meinen dasselbe. Entitätsklassen unterscheiden sich von objektorientierten Klassen dadurch, dass sie nicht über Methoden verfügen. Und um es noch schlimmer zu machen: Oft sprechen die Informatiker von „Entität“, wenn sie eigentlich Entitätstypen bzw. Entitätsklassen meinen. Wir werden in diesem Heft exakt bleiben, wir werden ab jetzt nur noch von Entitäten und Entitätstypen reden und wir werden sie nicht wechseln.

Die bekannteste Methode zur Modellierung von Datenstrukturen ist das **Entity/Relationship-Modell** von Peter Pi-Shan **Chen**, im Folgenden stets **E/R-Modell** genannt. Ein wichtiger Bestandteil dieses E/R-Modells ist eine Diagrammtechnik, mit der man die analysierten Strukturen veranschaulichen kann.

Wir werden – um diese Technik verstehen zu können – die beiden Begriffe Entity und Relationship zunächst gesondert diskutieren. Die Entitätstypen, die man bei der semantischen Analyse einer Anwenderwelt findet, werden Grundlage für die ersten Tabellenentwürfe liefern. Oft wird ein Entitätstyp gerade einer Tabelle entsprechen.

Dieser erste Teil ist die sogenannte **Entity-Analyse**.

Das nächste, was man untersuchen muss, sind die Beziehungen zwischen den verschiedenen bis dahin gefundenen Tabellen/Entitätstypen. Wir werden zeigen, dass es für gewisse Beziehungen nötig ist, eigene Beziehungs-Tabellen anzulegen. Dieser Teil der Untersuchung heißt nach dem englischen Wort für Beziehung: **Relationship-Analyse**.

Beides: Die Untersuchung der Entitäten und die Untersuchung der Beziehungen sind Teil der **semantischen Datenanalyse**.

Definition 21:

Es soll eine Teilmenge der realen Welt, die aus Objekten, Eigenschaften dieser Objekte und Beziehungen zwischen diesen Objekten besteht, in einer Datenbank abgebildet werden. Dann nennen wir die Gesamtheit dieser Objekte, Objekteigenschaften und Beziehungen die **Anwendungswelt**.

7.2 Entitäten und Entitätstypen

Wir wenden jetzt diesen Begriff sofort weiter an:

Definition 22:

Gegeben sei eine Anwendungswelt, die analysiert werden soll. Dann ist jedes Ding,

- das in dieser Anwendungswelt mehrere Eigenschaften hat und
- aufgrund dieser Eigenschaften eindeutig identifiziert werden kann,

eine **Entität** (dieser Anwendungswelt). Wir nennen – in Übereinstimmung mit unserer früheren Definition in Kapitel 2 – die Eigenschaften dieses Dinges seine **Attribute**, die jeweils die entsprechenden **Attributwerte** annehmen.

Diese Anwendungswelt definiert nicht nur, dass gewisse andere Dinge dieser Welt fremd sind und nicht hineingehören. Sondern sie definiert auch die Genauigkeit, mit der die Dinge angesehen werden – die Stärke der Lupe, mit der die Dinge betrachtet werden.

Dazu zwei Beispiele:

Beispiel 1

Sie wollen eine Datenbank über Multinationale Konzerne anlegen. In dieser Anwendungswelt gibt es Personen, aber sie haben keinerlei Eigenschaften, die eine eindeutige Identifizierung erlauben. Die „Lupe“, mit der ich in diesem System auf die Personen schaue, ist viel zu unscharf, um einzelne Individuen zu erkennen. Also sind in dieser Anwendungswelt die meisten Personen, die hier vorkommen, keine Entitäten.

Beispiel 2

Sie wollen eine Adressdatenbank anlegen. Dann gibt es in der Anwendungswelt Personen. Diese Personen haben unterschiedliche Eigenschaften, die sie zu wohl-unterschiedenen, zu distinkten Objekten werden lassen. Sie haben z.B. unterschiedliche Adressen. Adressen gehören auf jeden Fall zur Anwendungswelt. Dasselbe gilt für Telefonnummern. Also ist hier jede Person ein Ding, das eindeutig von allen anderen Dingen unterschieden werden kann. Also ist in dieser Anwendungswelt jede Person eine Entität.

Und: Solche „Dinge“ wie Hausnummern oder Telefonnummern sind in diesem Realitätsausschnitt natürlich auch eindeutig identifizierbar, aber sie werden keine Entitäten, denn sie haben – wieder in **dieser** Anwendungswelt – keine Attribute, sie sind selber Attribute.

Gruppenbildung oder besser: Typenbildung

In der Analyse werden gewisse Entitäten zu Gruppen oder zu Typen zusammengefasst. Diese Typen sind dann die ersten Kandidaten für Tabellen unserer Datenbank. Wie passiert das? In unserer Sprache passiert andauernd etwas Ähnliches. Winston Churchill, Bismarck, Hans-Dietrich Genscher bezeichnen wir – je nach dem für uns interessanten Blickwinkel auf die Realität – als Personen, als Männer, als Politiker oder als Memoirenschriftsteller oder, oder, oder ... Alle diese Begriffe sind Typenbegriffe, Abstraktionen, die stets eine Menge von Entitäten umfassen.

Wie findet nun die Typisierung in unserer Entitäten-Analyse einer Anwendungswelt statt? Es gilt:

Alle Entitäten, die mit einem **gemeinsamen Begriff** aus der Anwendungswelt charakterisiert werden können und die mit der gleichen Kombination von Attributen beschrieben werden, werden zu einem Entitätstyp zusammengefasst.



Beispiel:

In unserem Warenhaus „LieferBar“ gibt es u. a. die Dinge:

Altsaxofon, Computer, Drucker, Joghurt, Schreibtisch.

Alle diese Dinge haben in unserer Anwendungswelt die folgenden Eigenschaften:

Sie sind **alle** Artikel aus dem Artikelsortiment des Warenhauses „LieferBar“, sie können also alle mit dem gemeinsamen Begriff *Artikel* charakterisiert werden.

Darüber hinaus haben sie in unserer Anwendungswelt alle dieselbe Kombination von Attributen:

1. Sie haben alle eine Artikelnummer.
2. Sie haben alle eine Bezeichnung.
3. Sie gehören alle jeweils einer der zehn Artikelgruppen an.
4. Sie werden alle jeweils von einem der Lieferanten des Warenhauses „LieferBar & Co.“ geliefert.
5. Für jedes dieser Dinge wird eine Bestandsmenge geführt.
6. Für jedes dieser Dinge ist ein Preis festgelegt.

Das bedeutet: Diese Entitäten definieren also einen Entitätstyp *Artikel*, zu dem all diesen konkreten Entitäten gehören.

Wir werden dieses Beispiel jetzt weiter ausbauen bis wir eine vollständige E/R-Analyse unserer Warenhaus-Welt durchgeführt haben. Am Beginn einer solchen Analyse muss eine möglichst gute Beschreibung der Anwendungswelt stehen. Diese sollte zunächst noch unstrukturiert, ohne voreilige Typisierungen aus der Sicht des Anwenders erfolgen.

Die ersten Schritte zur Erstellung eines E/R-Modells lauten also:

1. Beschreiben Sie möglichst vollständig, aber unstrukturiert die Anwendungswelt.
2. Beschreiben Sie Attribute und Entitäten dieses Realitätsausschnitts.
3. Fassen Sie die Welt Ihrer Entitäten zu Entitätstypen zusammen.

7.3 Eine Analyse unseres Versandhauses „LieferBar & Co“

Bei der Analyse unseres Versandhauses „LieferBar & Co“ könnten wir z.B. die folgenden Entitätstypen bestimmen:

- ARTIKEL
- KUNDE
- LIEFERANT

Bei näherer Betrachtung fällt uns Folgendes auf:

1. Die Entitätstypen KUNDE und LIEFERANT haben sehr viele gemeinsame Attribute; wir entwerfen deshalb (wie im objektorientierten Entwurf) einen Basistyp PERSON, für den die Entitätstypen KUNDE und LIEFERANT Spezialisierungen sind.
2. Bei weiterem Nachdenken entscheiden wir uns dafür, einen eigenen Entitätstyp bzw. eine eigene Tabelle für die Artikelgruppen vorzusehen, da wir nur so eine einheitliche Speicherungslogik bei den Bezeichnungen der Artikelgruppen garantieren können. Nur wenn „Kleinteile“ bei **allen** Artikeln dieser Artikelgruppe „Kleinteile“ und nicht einmal „kleine Teile“ oder „Kleint.“ oder ähnlich heißen, können den Artikelgruppen später auch andere Eigenschaften oder Verarbeitungslogiken zugeordnet werden. Durch das Anlegen eines eigenen Entitätstyps ARTIKELGRUPPE vermeiden wir Redundanzen in unserer Datenbank, jede Bezeichnung einer Artikelgruppe kommt nur genau einmal in unserer Datenbank vor.

Wir beginnen also mit den folgenden Entitätstypen:

- ARTIKELGRUPPE
- ARTIKEL
- PERSON
- KUNDE
- LIEFERANT

In der entsprechenden E/R-Diagrammtechnik zeichnet man die Entitätstypen als Rechtecke. Unsere Vererbungshierarchie kennzeichnen wir dabei mit den aus der UML gebräuchlichen Symbolen. Wir „starten“ also mit dem Diagramm aus Abbildung 7.1. Dieses Diagramm werden wir im Verlaufe unserer Diskussion weiter vervollständigen.

7.4 Die Abbildung der Eigenschaften im E/R-Diagramm

Eigenschaften bzw. Attribute können:

- einfach oder – wie z.B. die Adresse – aus anderen einfachen Eigenschaften zusammengesetzt sein
- ein Schlüssel sein
- bei einigen Entitäten eines Entitätstyps nicht vorliegen (unsere berühmte berückichtigten NULL-Werte)

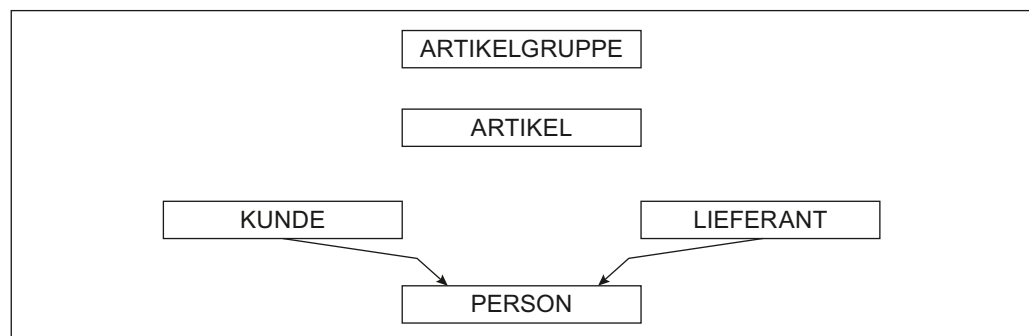


Abb. 7.1: Erster Schritt eines E/R-Diagramms für unser Warenhaus „LieferBar“ – Entitätstypen ohne Attribute und Beziehungen

Für die Diagrammtechnik gelten die folgenden Regeln:

- Eigenschaften werden als Ellipsen dargestellt, in denen der Name der Eigenschaft steht und die mit der Entität (oder Beziehung), zu der sie gehören, durch einen Strich verbunden sind.
- Falls die Eigenschaft zusammengesetzt ist, werden die einzelnen Komponenten wieder als Ellipsen gezeichnet, die mit dem zusammengesetzten Namen verbunden werden.
- Schlüssel-Eigenschaften werden unterstrichen.
- Wertebereiche werden nicht gezeichnet.

Wir wollen unser Beispieldiagramm jetzt um einige Eigenschaften erweitern. Betrachten Sie dazu Abb. 7.2. Ich habe dort aus Platzgründen nicht alle Attribute der einzelnen Entitätstypen eingezeichnet.

7.5 Beziehungen zwischen Entitätstypen mit Tabellenentwürfen

Zwischen Entitätstypen können Beziehungen bestehen. Offensichtlich besteht zum Beispiel eine Beziehung zwischen den Entitäten aus ARTIKEL und ARTIKELGRUPPE. Die Anzahl der Entitätstypen, die an einer Beziehung beteiligt sind, nennt man den **Grad dieser Beziehung**. Das E/R-Modell unterscheidet drei Arten von Beziehungen (wir diskutieren sie zunächst nur am Beispiel

von Beziehungen vom Grad 2). Das sind 1:1-Beziehungen, 1:n- bzw. n:1-Beziehungen und m:n-Beziehungen. Man nennt ganz allgemein bei einer a:b-Beziehung a und b die **Kardinalitäten** dieser Beziehung. In den Diagrammen des E/R-Modells wird jede Beziehung als eine Raute gezeichnet, die mit dem Namen der Beziehung versehen wird. Die zu der Beziehung gehörigen Entitätstypen werden durch Linien mit dieser Raute verbunden. Es gibt nun verschiedene Möglichkeiten, diese Linien zu beschriften, je nachdem für welche Art der Diagrammtechnik man sich entscheidet. Wir werden im Folgenden drei Arten kennenlernen.

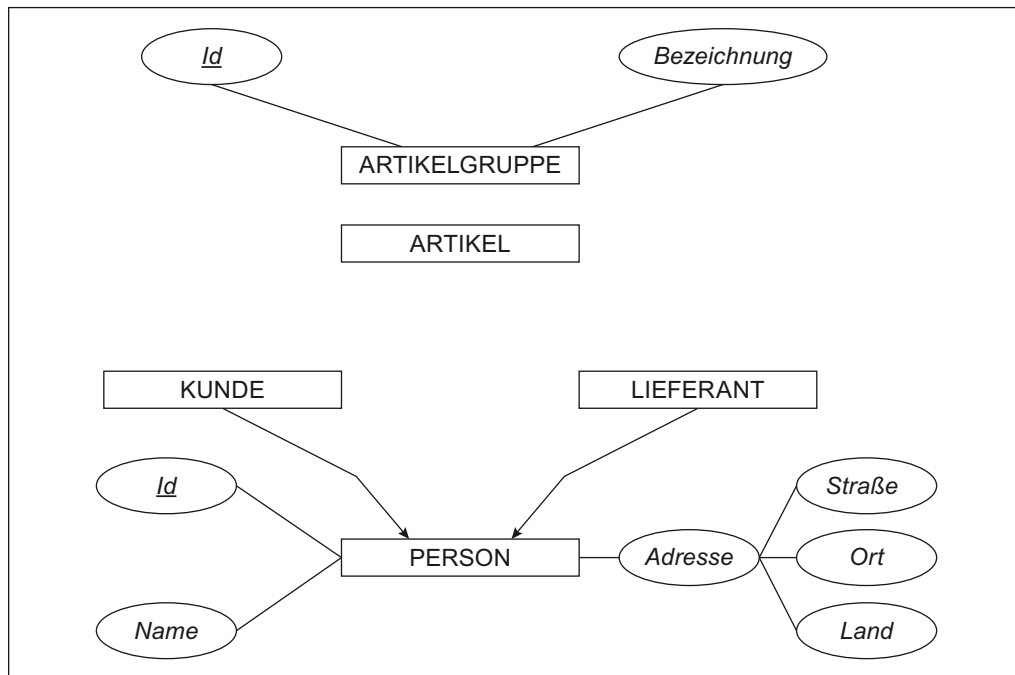


Abb. 7.2: Erster Schritt eines E/R-Diagramms für unser Warenhaus „LieferBar“. Entitätstypen ohne Attribute und Beziehungen

1 : 1-Beziehungen

Zwischen zwei Entitätstypen E1 und E2 besteht eine 1:1-Beziehung genau dann, wenn zu jedem Exemplar aus E1 nur höchstens ein Exemplar des Entitätstyps E2 gehört und umgekehrt wenn zu jedem Exemplar aus E2 nur höchstens ein Exemplar des Entitätstyps E1 gehört.

Definition 23:

Eine Beziehung, bei der jeder Entität aus E1 höchstens eine Entität aus E2 zugeordnet ist, heißt **funktional**.

Offensichtlich sind 1:1-Beziehungen in beiden Richtungen (von E1 zu E2 und von E2 zu E1) funktional. Bei genauerer Betrachtung ergeben sich einige unterschiedliche Fälle:

1. Zu jedem Exemplar aus E1 gibt es **genau ein Exemplar** aus E2, das mit ihm in Beziehung steht und umgekehrt; zu jedem Exemplar aus E2 gibt es **genau ein Exemplar** aus E1, das mit ihm in Beziehung steht. Dies ist eine (nicht konditionale) 1:1-Beziehung. Ein Beispiel ist die Beziehung zwischen der Menge aller Projekte, die in einer Firma durchgeführt werden, und der Menge aller Dokumentationen, die in einer speziellen Firmenbibliothek verwaltet werden. Die erste Art der Anzeige gibt einfach die Kardinalitäten 1 und 1 wieder (Abb. 7.3).

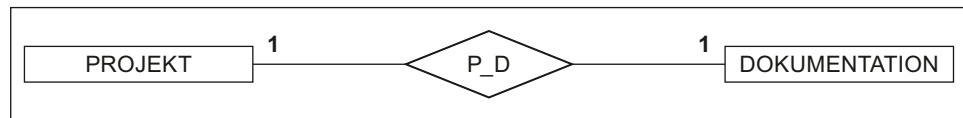


Abb. 7.3: Einfaches E/R-Diagramm zur Darstellung einer 1:1-Beziehung

Die zweite Art der Diagrammdarstellung gibt die Kardinalitäten nicht als Zahl an, sondern sie kennzeichnet die Funktionalität der Beziehung durch Pfeile (Abb. 7.4).

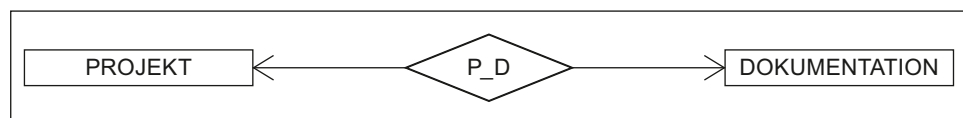


Abb. 7.4: Einfaches E/R-Pfeildiagramm zur Darstellung einer 1:1-Beziehung

Am explizitesten ist die folgende Art der Diagrammdarstellung, bei der in der Form [Minimum, Maximum] überall immer die kleinst- und größtmögliche Kardinalität eingetragen wird (Abb. 7.5):

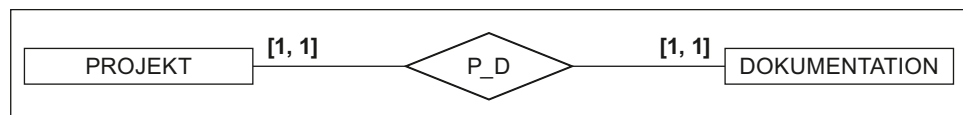


Abb. 7.5: E/R-Diagramm in [min, max]-Notation zur Darstellung einer 1:1-Beziehung

Hier gibt es die folgenden Möglichkeiten des Tabellenentwurfs:

Tabelle 7.1: Schema für 1:1-Beziehung

Charakterisierung	eine einzige Tabelle						
Schema	<table><tr><th>Id</th><th>Projektdaten</th><th>Dokumentationsdaten</th></tr><tr><td>...</td><td>...</td><td>...</td></tr></table>	Id	Projektdaten	Dokumentationsdaten
Id	Projektdaten	Dokumentationsdaten					
...					
Bemerkungen	Diesen Entwurf sollte man wählen, wenn die Daten der beiden Entitätstypen meistens gemeinsam verarbeitet werden.						

Oder aber auch:

Tabelle 7.2: Schema für 1:1-Beziehung

Charakterisierung	zwei Tabellen mit gegenseitiger Referenzierung		
Schema, bestehend aus Projekt-tabelle und Dokumentationstabelle			
	Id	Projektdaten	DokumentationsId

	Id	Dokumentationsdaten	ProjektId

Bemerkungen	Diesen Entwurf sollte man wählen, wenn die Daten der beiden Entitätstypen meistens separat verarbeitet werden. In diesem Fall sollte es auch möglich sein, in beiden Richtungen von der einen Entität zum Beziehungspartner der anderen Entität zu wechseln, darum habe ich hier zwei Fremdschlüssel vorgeschlagen.		

Die nächsten beiden Fälle nennt man **konditionale** 1:1-Beziehungen. Das bedeutet, dass entweder für eine oder auch für beide Entitätstypen E1 und E2 nicht stets ein Partner vorhanden sein muss.

- Es gelte: Zu jedem Exemplar aus E1 gibt es **genau ein Exemplar** aus E2, das mit ihm in Beziehung steht, aber zu jedem Exemplar aus E2 gibt es **höchstens ein** Exemplar aus E1, das mit ihm in Beziehung steht. Als Beispiel betrachten Sie eine Firma, in der jede Abteilung genau einen Abteilungsleiter bzw. eine Abteilungsleiterin hat, wo aber jeder Angestellte höchstens eine Abteilung leitet. Solch eine Beziehung nennt man **einseitig konditional**. Man definiert:
 - In der einfachen Darstellung kennzeichnet man die Kardinalität auf der Seite, wo der konditional beteiligte Entitätstyp steht, zusätzlich mit einem c – dem Anfangsbuchstaben des englischen bzw. lateinischen Wortes „conditional“.
 - Im Pfeildiagramm kennzeichnet man den Pfeil auf der Seite, wo der konditional beteiligte Entitätstyp steht, mit einem Querstrich.

Wir erhalten die folgenden drei Diagramme (Abb. 7.6, 7.7 und 7.8):

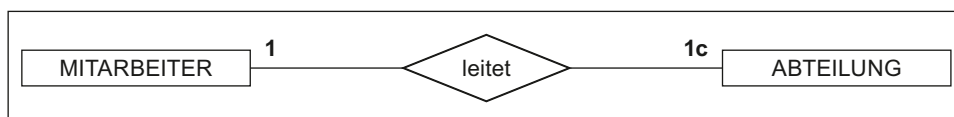


Abb. 7.6: Einfaches E/R-Diagramm zur Darstellung einer 1:1-Beziehung

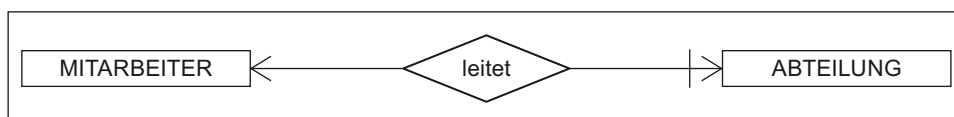


Abb. 7.7: Einfaches E/R-Pfeildiagramm zur Darstellung einer 1:1-Beziehung

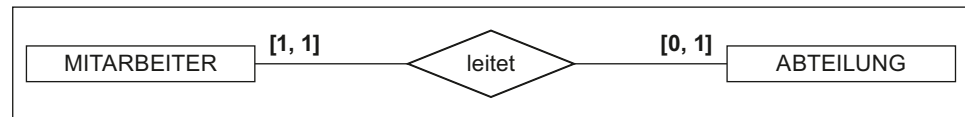


Abb. 7.8: E/R-Diagramm in $[min, max]$ -Notation zur Darstellung einer 1:1-Beziehung

Hier ist der folgende Tabellenentwurf am sinnvollsten:

Tabelle 7.3: Schema für 1:1-Beziehung

Charakterisierung	zwei Tabellen mit einem Fremdschlüssel auf der nicht konditionalen Seite								
Schema, bestehend aus Abteilungstabelle und Mitarbeitertabelle	<table><tr><td>Id</td><td>Abteilungsdaten</td><td>LeiterId</td></tr><tr><td>...</td><td>...</td><td>...</td></tr></table>			Id	Abteilungsdaten	LeiterId
	Id	Abteilungsdaten	LeiterId						
						
	<table><tr><td>Id</td><td>Mitarbeiterdaten</td></tr><tr><td>...</td><td>...</td></tr></table>			Id	Mitarbeiterdaten		
Id	Mitarbeiterdaten								
...	...								

3. Beachten Sie bitte, dass unsere Generalisierung-Spezialisierungsbeziehung ebenfalls eine einseitig konditionale Beziehung ist. Das entsprechende Diagramm am Beispiel der Beziehung von PERSON und KUNDE lautet:

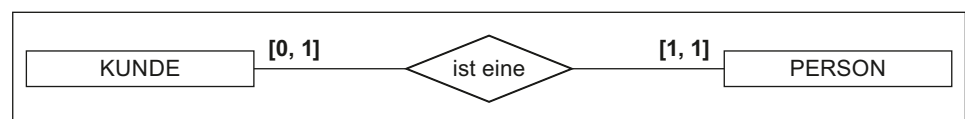


Abb. 7.9: E/R-Diagramm in $[min, max]$ -Notation zur Darstellung einer 1:1-Beziehung

Allerdings ist die Notation der Vererbung aussagekräftiger, sie enthält mehr Informationen, darum behalten wir sie bei. Den Tabellenentwurf für diesen Spezialfall haben wir bereits diskutiert.

4. Nun gelte: Zu jedem Exemplar aus E1 gibt es **höchstens ein Exemplar** aus E2, das mit ihm in Beziehung steht, und zu jedem Exemplar aus E2 gibt es **höchstens ein Exemplar** aus E1, das mit ihm in Beziehung steht. Als Beispiel betrachten Sie eine Firma, in der es interne und externe Mitarbeiter gibt. Die internen Mitarbeiter haben alle ein eigenes Arbeitszimmer für sich allein, die externen haben kein Arbeitszimmer und es gibt Räume, die (bislang) unbesetzt sind. Es sei E1 der Entitätstyp MITARBEITER, E2 der Entitätstyp RAUM. Dann heißt die 1:1-Beziehung zwischen ihnen **beidseitig konditional**. Wir erhalten die folgenden drei Diagramme (Abb. 7.10, 7.11 und 7.12):

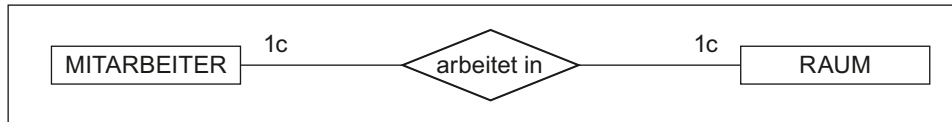


Abb. 7.10: Einfaches E/R-Diagramm zur Darstellung einer 1:1-Beziehung

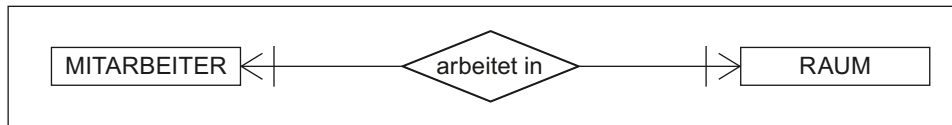


Abb. 7.11: Einfaches E/R-Pfeildiagramm zur Darstellung einer 1:1-Beziehung

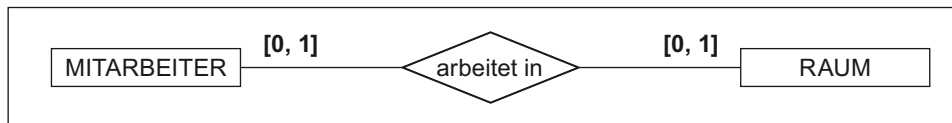


Abb. 7.12: E/R-Diagramm in [min, max]-Notation zur Darstellung einer 1:1-Beziehung

Obwohl wir hier eine 1:1-Beziehung haben, schlage ich Ihnen eine Drei-Tabellen-Lösung vor, ansonsten würden wir NULL-Werte in den Fremdschlüsselfeldern bekommen.

Tabelle 7.4: Schema für 1:1-Beziehung

Charakterisierung	Drei Tabellen: zwei Tabellen und eine Verbindungstabelle mit Fremdschlüsseln														
Schema, bestehend aus Mitarbeitertabelle, Raumtabelle und Verbindungstabelle, die angibt, wer in welchem Raum sitzt.	<table><tr><th>Id</th><th>Mitarbeiterdaten</th><th>Id</th><th>Raumdaten</th></tr><tr><td>...</td><td>...</td><td>...</td><td>...</td></tr></table> <table><tr><th>MitarbeiterId</th><th>RaumId</th><th>Verbindungsdaten</th></tr><tr><td>...</td><td>...</td><td>...</td></tr></table>	Id	Mitarbeiterdaten	Id	Raumdaten	MitarbeiterId	RaumId	Verbindungsdaten
Id	Mitarbeiterdaten	Id	Raumdaten												
...												
MitarbeiterId	RaumId	Verbindungsdaten													
...													
Bemerkungen	In der Verbindungstabelle ist (MitarbeiterId, RaumId) der Primärschlüssel.														

Solch eine Drei-Tabellen-Lösung kann natürlich zu Performanceproblemen führen, weil für Abfragen immer aufwendige JOINS durchgeführt werden müssen. Gegebenenfalls muss man sich dann doch mit einer Zwei-Tabellen-Lösung und NULL-Werten oder künstlichen Schlüsselwerten für einen Raum mit der Bezeichnung „Raum nicht vorhanden“ behelfen. Sie werden immer wieder erleben, wie sich die reine Theorie und die gewünschte Schnelligkeit in der Verarbeitung zueinander „feindlich“ verhalten. Tabelle 7.4 zeigt Ihnen den theoretisch schönsten Tabellenentwurf.

5. Beziehungen können auch rekursiv sein. Das bedeutet, dass Entitäten desselben Typs miteinander in Beziehung stehen können. Ich gebe Ihnen ein Beispiel einer rekursiven, beidseitig konditionalen 1:1-Beziehung. E sei die Entität der Einwohner einer Stadt, die Beziehung sei der (aktuell bestehende) Bund der Ehe von einem Einwohner zu einer anderen Person in der Stadt. Das sieht in unserem Diagramm dann folgendermaßen aus (Abb. 7.13):

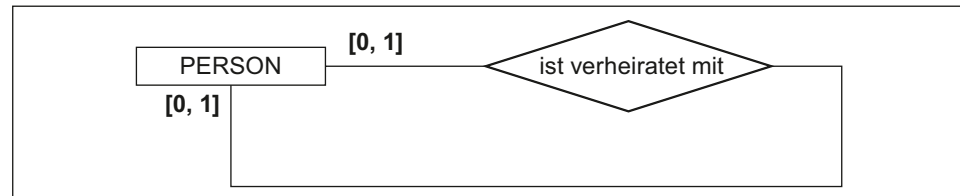


Abb. 7.13: E/R-Diagramm in $[min, max]$ -Notation zur Darstellung einer rekursiven 1:1-Beziehung

Die anderen Darstellungen (einfach bzw. mit Pfeil) sehen völlig analog aus. Wie bei unserem vorherigen Beispiel erhalten wir als Tabellenentwurf das Schema aus Tabelle 7.5:

Tabelle 7.5: Schema für rekursive 1:1-Beziehung

Charakterisierung	eine Tabelle und eine Verbindungstabelle mit den Fremdschlüsseln, die beide auf Id in der Personentabelle referenzieren										
Schema, bestehend aus Personentabelle und Verbindungstabelle, die angibt, wer mit wem verheiratet ist	<table><tr><th>Id</th><th>Personendaten</th></tr><tr><td>...</td><td>...</td></tr></table> <table><tr><th>EhefrauId</th><th>EhemannId</th><th>Heiratsdaten</th></tr><tr><td>...</td><td>...</td><td>...</td></tr></table>	Id	Personendaten	EhefrauId	EhemannId	Heiratsdaten
Id	Personendaten										
...	...										
EhefrauId	EhemannId	Heiratsdaten									
...									
Bemerkungen	In der Verbindungstabelle ist (EhefrauId, EhemannId) der Primärschlüssel.										

Wie Sie sehen, haben alle diese Fälle unterschiedliche Konsequenzen für die Modellierung der Daten in Tabellenform, darum ist es wichtig, dass man sie deutlich machen kann.

1:n-Beziehungen bzw. n:1-Beziehungen:

Zwischen zwei Entitätstypen E1 und E2 besteht eine 1:n-Beziehung genau dann, wenn zu jedem Exemplar aus E1 mehrere Exemplare des Entitätstyps E2 gehören können, aber umgekehrt zu jedem Exemplar aus E2 höchstens ein Exemplar des Entitätstyps E1 gehört. Auch hier müssen wir Fallunterscheidungen diskutieren.

1. Zu jedem Exemplar aus E1 gibt es n Exemplare aus E2, $n > 0$, und zu jedem Exemplar aus E2 gibt es **genau ein Exemplar** aus E1. Dies ist eine (nicht konditionale) 1:n-Beziehung. Ein Beispiel ist die Beziehung zwischen dem Entitätstyp Schüler einer Schule und der Beziehung „besucht“ zum Entitätstyp Schulklasse. Abb. 7.14 zeigt Ihnen das entsprechende Diagramm in der einfachen Darstellung, Abb. 7.15 zeigt es als Pfeildiagramm und Abb. 7.16 die [min, max]-Notation. Wenn man das Maximum nicht explizit als Zahl angeben kann, symbolisiert man diese Tatsache oft durch einen Stern *. Im Pfeildiagramm wird so eine mehrwertige Abhängigkeit (von einer Klasse sind mehrere Schüler „abhängig“) durch einen Pfeil mit zwei Spitzen symbolisiert.

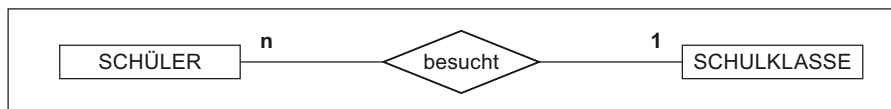


Abb. 7.14: Einfaches E/R-Diagramm zur Darstellung einer n:1-Beziehung

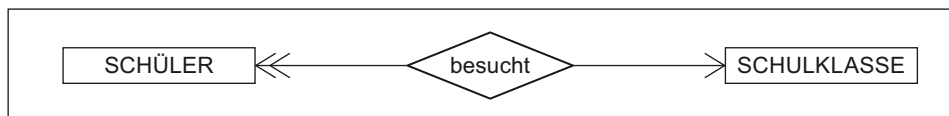


Abb. 7.15: E/R-Diagramm als Pfeildiagramm zur Darstellung einer n:1-Beziehung

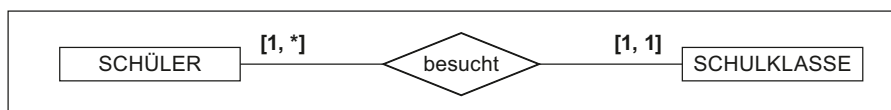


Abb. 7.16: E/R-Diagramm in [min, max]-Notation zur Darstellung einer n:1-Beziehung

Hier und übrigens auch in dem ersten der folgenden konditionalen Fälle ist der folgende Tabellenentwurf am sinnvollsten:

Tabelle 7.6: Schema für 1:n-Beziehung

Charakterisierung	zwei Tabellen mit einem Fremdschlüssel auf der n-Seite		
Schema, bestehend aus Schülertabelle und Schulklassentabelle	Id	Schülerdaten	SchulklassenId

	Id	Schulklassendaten	
	

Auch im Falle von 1:n-Beziehungen sind konditionale Fälle möglich. Wir müssen jetzt **zwei** einseitig konditionale Fälle untersuchen.

2. Zu jedem Exemplar aus E1 gibt es **n Exemplare** aus E2, $n \geq 0$, und zu jedem Exemplar aus E2 gibt es **genau ein Exemplar** aus E1. Dies ist eine einseitig konditionale 1:n-Beziehung. Ich betrachte als Beispiel die Beziehung zwischen dem Entitätstyp ARTIKELGRUPPE und dem Entitätstyp ARTIKEL in unserer Datenbank LieferBar, wo wir durchaus den Fall zulassen wollen, dass es Artikelgruppen gibt, die (noch nicht oder nicht mehr) einen Artikel enthalten, aber jeder Artikel unbedingt zu einer Artikelgruppe gehören muss. Die Abbildungen 7.17, 7.18 und 7.19 zeigen die entsprechenden Diagramme:



Abb. 7.17: Einfaches E/R-Diagramm zur Darstellung einer konditionalen n:1-Beziehung



Abb. 7.18: E/R-Diagramm als Pfeildiagramm zur Darstellung einer konditionalen n:1-Beziehung



Abb. 7.19: E/R-Diagramm in [min, max]-Notation zur Darstellung einer n:1-Beziehung

Wie Sie schon wissen und wie angekündigt, ist hier ein zum vorherigen Fall analoger Tabellenentwurf am sinnvollsten:

Tabelle 7.7: Schema für 1:n-Beziehung

Charakterisierung	zwei Tabellen mit einem Fremdschlüssel auf der n-Seite										
Schema, bestehend aus Schülertabelle und Schulklassentabelle	<table><tr><th>Id</th><th>Artikeldaten</th><th>ArtikelgruppenId</th></tr><tr><td>...</td><td>...</td><td>...</td></tr></table> <table><tr><th>Id</th><th>Artikelgruppendaten</th></tr><tr><td>...</td><td>...</td></tr></table>	Id	Artikeldaten	ArtikelgruppenId	Id	Artikelgruppendaten
Id	Artikeldaten	ArtikelgruppenId									
...									
Id	Artikelgruppendaten										
...	...										

Im Folgenden werde ich Ihnen nur noch die E/R-Diagramme in der [min, max]-Notation zeigen, die anderen Darstellungen ergeben sich sofort daraus.

3. Zu jedem Exemplar aus E1 gibt es **n Exemplare** aus E2, **$n > 0$** , und zu jedem Exemplar aus E2 gibt es **höchstens ein Exemplar** aus E1. Dies ist ebenfalls eine einseitig konditionale 1:n-Beziehung. Betrachten Sie als Beispiel die Abgeordneten eines Parlaments, für die man ihre Parteizugehörigkeit speichert:

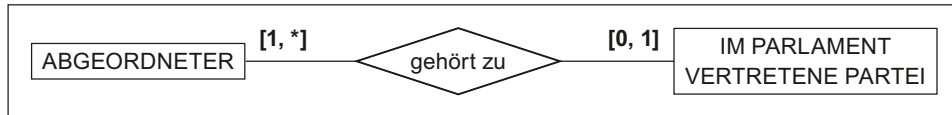


Abb. 7.20: E/R-Diagramm in $[min, max]$ -Notation zur Darstellung einer $n:1$ -Beziehung

Es werden hier nur die im Parlament vertretenen Parteien geführt, d.h. zu jeder Partei gibt es mindestens einen Abgeordneten. Andererseits gibt es aber auch parteilose Abgeordnete.

Tabelle 7.8: Schema für 1:n-Beziehung

Charakterisierung	Drei Tabellen: zwei Tabellen und eine Verbindungstabelle mit Fremdschlüsseln														
Schema, bestehend aus Abgeordnetentabelle, Parteitabelle und Verbindungstabelle, die angibt, welcher Abgeordnete zu welcher Partei gehört	<table><tr><td>Id</td><td>Abgeordnetendaten</td><td>Id</td><td>Parteidaten</td></tr><tr><td>...</td><td>...</td><td>...</td><td>...</td></tr></table> <table><tr><td>AbgeordnetenId</td><td>ParteiId</td><td>Verbindungsdaten</td></tr><tr><td>...</td><td>...</td><td>...</td></tr></table>	Id	Abgeordnetendaten	Id	Parteidaten	AbgeordnetenId	ParteiId	Verbindungsdaten
Id	Abgeordnetendaten	Id	Parteidaten												
...												
AbgeordnetenId	ParteiId	Verbindungsdaten													
...													
Bemerkung	In der Verbindungstabelle ist (AbgeordnetenId, ParteiId) der Primärschlüssel.														

Abb. 7.20 zeigt das entsprechende Diagramm. Auch hier muss man mit drei Tabellen arbeiten, wenn man NULL-Werte bei Fremdschlüsseln vermeiden will.

4. Um den Fall der beidseitig konditionalen 1:n-Beziehung zu diskutieren, bleiben wir bei unserem Beispiel. Zu jedem Exemplar aus E1 gibt es **n Exemplare** aus E2, **$n \geq 0$** , und zu jedem Exemplar aus E2 gibt es **höchstens ein Exemplar** aus E1. E2 sei jetzt wieder der Entitätstyp der Abgeordneten eines Parlaments, für die man ihre Parteizugehörigkeit speichert. Jetzt werden aber alle zur Parlamentswahl zugelassenen Parteien in E1 gespeichert, auch die, die gar nicht in das Parlament hereingekommen sind. Abb. 7.21 zeigt das entsprechende Diagramm:

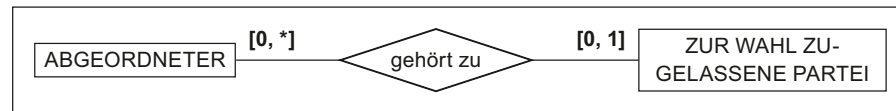


Abb. 7.21: E/R-Diagramm in $[min, max]$ -Notation zur Darstellung einer $n:1$ -Beziehung

Und das Tabellenschema sieht wieder genauso aus wie in Tabelle 7.8 skizziert.

5. Auch $1:n$ -Beziehungen können rekursiv sein. Immer, wenn man gerichtete Baumstrukturen abspeichert, hat man solch eine Architektur. Denken Sie nur an den Baum, den der Windows Explorer mit seinen Ordnern und Unterordnern darstellt. Die Verwaltung, insbesondere das Suchen und Navigieren in solchen Strukturen ist bei relationalen Datenbanken ein schwieriges Problem. Ich bleibe bei diesem Beispiel, mein Entitätstyp sei der (Unterordner), der jeweils beliebig viele weitere Unterordner haben kann, aber immer nur genau einen übergeordneten Ordner.

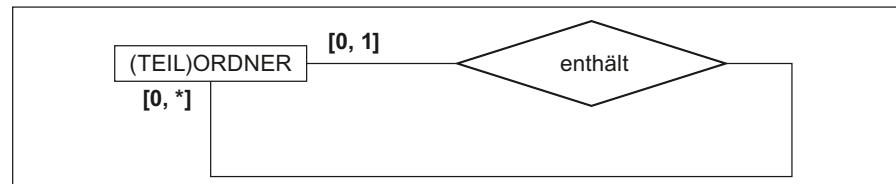


Abb. 7.22: E/R-Diagramm in $[min, max]$ -Notation zur Darstellung einer rekursiven $1:n$ -Beziehung

Aber es gibt auch einen Ordner, der keinen „Vorgänger“ hat, das ist das Laufwerk. Wir erhalten das Diagramm aus Abb. 7.22. In diesem Beispiel haben wir nur einen einzigen Fall, wo es keinen oberen Ordner gibt, der den Teilordner enthält, das ist das Laufwerk selbst oder der Desktop – je nachdem, wie Sie arbeiten. Hier ist ein Fremdschlüssel mit NULL-Wert sehr sinnvoll zur Charakterisierung dieser einzigartigen Situation. Wir kommen also in einer der kompliziertesten Anwendungen mit einer einzigen Tabelle aus:

Tabelle 7.9: Schema für eine rekursive $n:1$ -Beziehung

Charakterisierung	eine einzige Tabelle, in der ein Fremdschlüssel auf einen Satz aus derselben Tabelle verweist						
Schema	<table><tr><th>Id</th><th>Ordnerdaten</th><th>ElternordnerId</th></tr><tr><td>...</td><td>...</td><td>...</td></tr></table>	Id	Ordnerdaten	ElternordnerId
Id	Ordnerdaten	ElternordnerId					
...					

m:n-Beziehungen

Zwischen zwei Entitätstypen E1 und E2 besteht eine m:n-Beziehung genau dann, wenn zu jedem Exemplar aus E1 mehrere Exemplare des Entitätstyps E2 gehören können, und umgekehrt, wenn zu jedem Exemplar aus E2 mehrere Exemplare des Entitätstyps E1 gehören können. Ein Beispiel für eine m:n-Beziehung ist bei uns die Beziehung BESTELLUNGEN zwischen den Entitätstypen ARTIKEL und KUNDE: Ein Artikel kann von mehreren Kunden bestellt werden, genauso wie ein Kunde mehrere Artikel bestellen kann. Ich diskutiere hier nicht die verschiedenen konditionalen und nicht-konditionalen Sonderfälle, weil diese Unterschiede keinerlei Konsequenzen für den anschließenden Tabellenentwurf haben. Im folgenden Diagramm sehen Sie aber noch eine andere Tatsache: Nicht nur die (Stamm-)Entitätstypen, damit meine ich die Partner einer Beziehung, sondern auch die Beziehungen selber können natürlich Eigenschaften haben. Das ist (zumindest bei m:n-Beziehungen) in den allermeisten Fällen so.

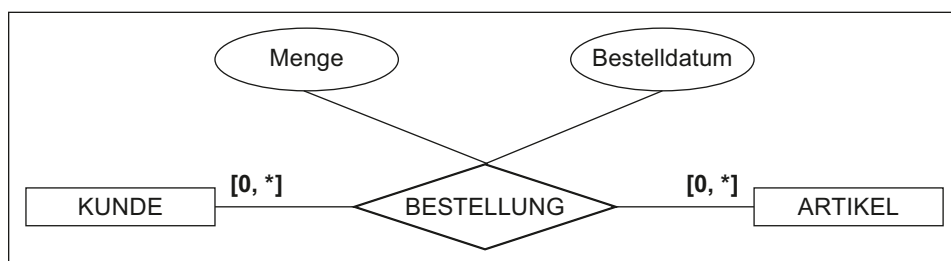


Abb. 7.23: E/R-Diagramm in [min, max]-Notation zur Darstellung einer m:n-Beziehung mit Attributen der Beziehungsentitäten

Grundsätzlich braucht man, egal ob konditional oder nicht, bei m:n-Beziehungen drei Tabellen: die beiden Partnerrelationen und eine Verbindungstabelle. In unserem Beispiel sieht das so aus:

Tabelle 7.10: Schema für 1:n-Beziehung

Charakterisierung	Drei Tabellen: zwei Tabellen und eine Verbindungstabelle mit Fremdschlüsseln																
Schema, bestehend aus Kundentabelle, Artikeltabelle und Verbindungstabelle „Bestellungen“, die angibt, welcher Kunde welchen Artikel wann und in welcher Menge bestellt hat	<table><tr><th>Id</th><th>Kundendaten</th><th>Id</th><th>Artikeldaten</th></tr><tr><td>...</td><td>...</td><td>...</td><td>...</td></tr></table> <table><tr><th>Id</th><th>KundenId</th><th>ArtikelId</th><th>Bestelldaten</th></tr><tr><td>...</td><td>...</td><td>...</td><td>...</td></tr></table>	Id	Kundendaten	Id	Artikeldaten	Id	KundenId	ArtikelId	Bestelldaten
Id	Kundendaten	Id	Artikeldaten														
...														
Id	KundenId	ArtikelId	Bestelldaten														
...														
Bemerkungen	In der Verbindungstabelle ist (Id) der Primärschlüssel.																

In der darauf folgenden Abb. 7.24, die Ihnen das gesamte E/R-Modell unseres Kaufhauses „LieferBar“ zeigen soll, sind die 1:1-Beziehungen zwischen PERSON und KUNDE, bzw. PERSON und LIEFERANT als Beziehungen zwischen Basisentitätentyp und abgeleitetem Entitätentyp charakterisiert.

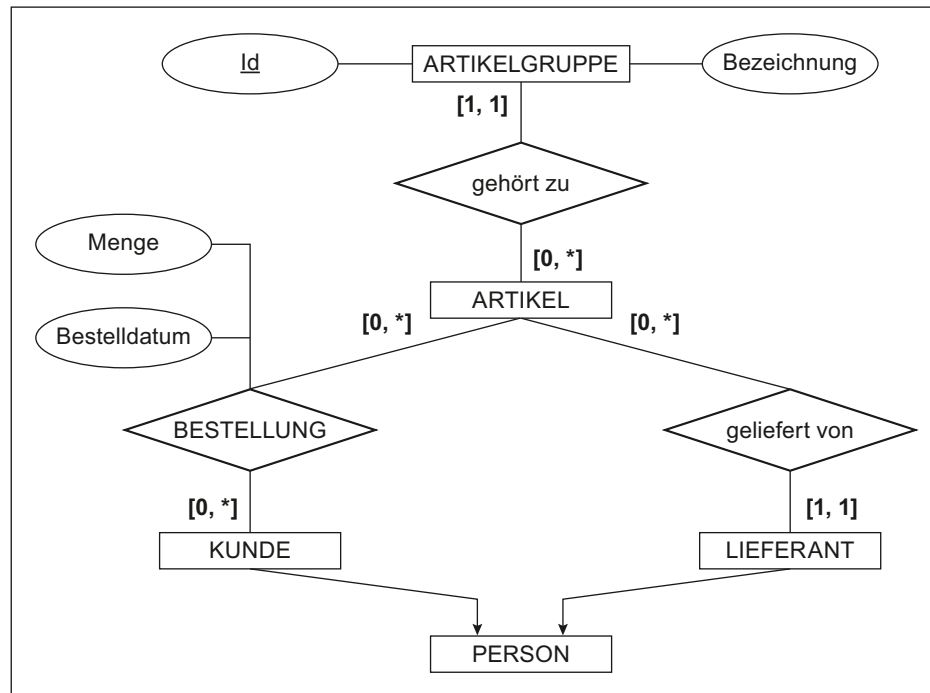


Abb. 7.24: Dritter Schritt eines E/R-Diagramms für unser Warenhaus „LieferBar“ – Entitäten mit (einigen) Attributen und mit Beziehungen

Ich gebe Ihnen zum Abschluss dieses Abschnitts noch einmal eine Übersicht über die besprochenen Kardinalitätendarstellungen bzw. die entsprechenden Symbole im Pfeildiagramm:

Tabelle 7.11: Schema für 1:n-Beziehung

Abhängigkeit	konditional (ja, nein)	einfache Darstellung	Pfeil- diagramm	[min, max]- Notation
einfach	nein	1	→	[1, 1]
einfach	ja	1 c	→+	[0, 1]
mehrwertig	nein	n bzw. *	→>	[1, n] bzw. [1, *]
mehrwertig	ja	n c bzw. * c	→+>	[0, n] bzw. [0, *]

7.6 Der Datenbankentwurf mithilfe des E/R-Modells

Das meiste haben wir schon besprochen. Sie können nach den folgenden Regeln verfahren:

1. Aus jedem Entitätstyp machen Sie eine Relation, also eine Tabelle mit den analysierten atomaren Eigenschaften als Attributen. Die unterstrichenen Eigenschaften werden zu den Schlüsselkandidaten bzw. zum Primärschlüssel. Eigenschaften, deren Wert eine Entität aus einem anderen Entitätstyp ist, werden durch Fremdschlüssel repräsentiert.

So erhalten Sie in unserem Beispiel zunächst die Tabellen:

ARTIKELGRUPPE, ARTIKEL, KUNDE, LIEFERANT und PERSON

2. 1:1-Beziehungen zwischen Entitäten sollten Sie sorgfältig analysieren. Je nach der Art der Konditionalität ergeben sich unterschiedliche Entwürfe. Außerdem können noch Vererbungsbeziehungen vorliegen, wie Sie hier bei KUNDE und LIEFERANT vorliegen, die sich beide auf den Entitätstyp PERSON als Basistyp beziehen. Hier empfehle ich, den Primärschlüssel der abgeleiteten Klasse gleichzeitig zum Fremdschlüssel mit Referenz zum Primärschlüssel der Basisklasse zu machen.
3. Auch 1:n-Beziehungen bzw. n:1-Beziehungen müssen sorgfältig analysiert werden. Im Falle von „[0, 1] zu n“-Beziehungen empfehle ich drei Tabellen, d. h., alles was mit der Beziehung zu tun hat, wird in einer Verbindungstabelle abgespeichert. So vermeidet man NULL-Werte in den Fremdschlüsseln. Dagegen kommt man bei „[1, 1] zu n“-Beziehungen grundsätzlich mit zwei Tabellen aus. Hier platziert man einen Fremdschlüssel auf der „n-Seite“. Die Beziehung zwischen ARTIKEL und ARTIKELGRUPPE ist z. B. so, dass der referenzierende Fremdschlüssel *ArtikelgruppeId* in der Tabelle ARTIKEL steht. Außerdem müssen Sie die geeigneten Entscheidungen über die DELETE- und UPDATE-Optionen treffen.
4. Dagegen legt man für alle m:n-Beziehungen eine eigene Tabelle an, welche die analysierten Eigenschaften dieser Beziehung enthält. Man spricht auch von einer Beziehungsentität bzw. von einem Beziehungsentitätstyp. Der Hauptgrund für dieses Vorgehen ist die Vermeidung von Redundanzen. Die Teilnehmer dieser Beziehungen werden in dem jeweiligen Beziehungsentitätstyp durch referenzierende Fremdschlüssel repräsentiert. Wieder sind DELETE- und UPDATE-Optionen festzulegen. Diese Optionen sind nicht im E/R-Diagramm notiert. In unserem Beispiel gibt es nur eine m:n-Beziehung – das sind die BESTELLUNGEN.

Sie sollten jetzt in der Lage sein, mithilfe unseres Diagramms ein vollständiges Tabellendesign für die Welt unseres Versandhauses „LieferBar & Co“ durchzuführen.

7.7 Das allgemeine Vorgehen beim Datenbankentwurf

Allgemein sollte das Vorgehen beim Datenbankentwurf das folgende sein:

- Man benutzt eine Top-Down-Methode wie das E/R-Modell, um einen groben Überblick über Entitäten, Entitätstypen, Eigenschaften und Beziehungen des Systems zu entwickeln, und macht daraus einen ersten Relationen- bzw. Tabellenentwurf.
- Man analysiert und verfeinert diesen Entwurf mithilfe der Normalformen. Man nennt diesen Teil auch Normalisierung.

Was unter diesem zweiten Punkt genau zu verstehen ist, erfahren Sie im nächsten Kapitel. Ich mache Sie nur darauf aufmerksam, dass wir bisher noch nicht den abgeleiteten Typ ERLEDIGTEBESTELLUNGEN „gefunden“ haben (bearbeiten Sie dazu bitte die Aufgabe 7.2).

Zusammenfassung

Wir haben in diesem Kapitel die semantische Datenanalyse mithilfe des Entity/Relationship-Modells von Chen besprochen. Unsere Untersuchung umfasste dabei die folgenden Punkte:

- Entitäten und Entitätstypen
- Diskussion dieser Begriffe am Beispiel unserer Datenbank „LieferBar“
- Darstellung der Entitätstypen in einem E/R-Diagramm
- Darstellung von Attributen in einem E/R-Diagramm
- Verschiedene Arten der Beziehungen zwischen Entitätstypen und ihre Notierung in einem E/R-Diagramm
- Der Datenbankentwurf mithilfe des E/R-Modells
- Der Datenbankentwurf mithilfe des E/R-Modells braucht noch die „Normalisierung“

Aufgaben zur Selbstüberprüfung

- 7.1 In einer Firma sollen für die Angehörigen des Entitätstyps MITARBEITER die Projekte, in denen sie mitarbeiten, gespeichert werden. Eine Person kann in keinem, einem oder mehreren Projekten mitarbeiten. In einem Projekt arbeiten ein oder mehrere Mitarbeiter mit. Es soll außerdem die Art der Mitarbeit (einfach, Projektleitung usw.) und die Dauer der Mitarbeit (Anfangs- und Enddatum) gespeichert werden. Erstellen Sie für diese Situation ein E/R-Modell in der [min, max]-Notation.
- 7.2 Vervollständigen Sie unsere E/R-Analyse bis hin zu dem Entitätstyp ERLEDIGTEBESTELLUNGEN. Viele Informatiker würden diesen weiteren Entitätstyp **nicht** vorsehen, sondern stattdessen in dem Entitätstyp BESTELLUNGEN ein weiteres Attribut *Commitdatum* (und eventuell noch ein logisches Attribut *IsCommitted*) führen. Auch für diese Lösung gibt es gute Gründe. Beschreiben Sie genau, was Ihrer Meinung nach für und was gegen den Wegfall des Typs ERLEDIGTEBESTELLUNGEN spricht.
- 7.3 Betrachten Sie das folgende E/R-Diagramm:

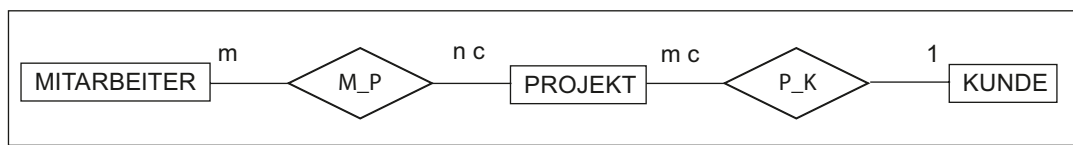


Abb. 7.25: E/R-Diagramm

- Interpretieren Sie die Kardinalitätenangaben in Ihren eigenen Worten.
- Zeichnen Sie dieses Diagramm auch als Pfeildiagramm und in der [min, max]-Darstellung.
- Erstellen Sie zu dem E/R-Diagramm ein Tabellendesign. Es gilt: Ein Mitarbeiter hat eine *Id*, *Name* und *Vorname*, ein Projekt hat eine *Id*, eine *Bezeichnung* und ein *Anfangsdatum*, ein Kunde hat eine *Id* und eine *Firmenbezeichnung*, es gibt für Mitarbeiter ein *Eintrittsdatum* und ein *Austrittsdatum* aus jedem Projekt.

Kapitel 8

8 Normalisierungen

Nachdem man – z. B. mithilfe einer E/R-Analyse ein erstes Datenbankmodell erstellt hat, geht man an die genauere Überprüfung dieses Modells mithilfe der Normalformen. Sie helfen dabei, die „richtigen“ Schlüssel zu definieren, und vor allem helfen Sie dabei, Redundanzen im Datenbankentwurf zu vermeiden. Vor allem die Normalformen sorgen für die Aufspaltung der Datenwelt in viele kleinere Tabellen, die miteinander in Beziehung stehen.

8.1 Funktionale Abhängigkeit

Zunächst brauchen wir den Begriff der **funktionalen Abhängigkeit**:

Definition 24:

Sei R eine Relation und seien X und Y beliebige Teilmengen der Attribute von R . Dann heißt Y **funktional abhängig** von X genau dann, wenn zu jeder Wertekombination der Attributwerte aus X genau eine Wertekombination der Attributwerte aus Y gehört.

Man sagt auch: X ist eine **funktionale Determinante** für Y bzw. X **determiniert** Y **funktional**. Man schreibt: $X \rightarrow Y$. X heißt auch eine **Determinante**.

Falls die Menge X nicht mehr verkleinert werden kann, ohne dass die funktionale Abhängigkeit verloren geht, heißt X eine **minimale Determinante**.

Falls Y eine Teilmenge von X ist, heißt die funktionale Abhängigkeit $X \rightarrow Y$ **trivial**. (Und das ist sie ja auch.)

Lassen Sie uns einige Beispiele betrachten. Ich nehme als Relation R immer unsere Relation $PERSON$. Die vollständige Attributmenge dieser Relation lautet:

$$A = \{Id, Name, Vorname, Strasse, Nr, Plz, Ort, Land\}$$

- Sei $X = \{Id\}$. Dann besteht die funktionale Abhängigkeit $X \rightarrow A$. Denn Id war Schlüsselkandidat, den wir sogar als Primärschlüssel ausgewählt hatten. Diese Abhängigkeit ist nicht-trivial.
- Sei $X = \{Name, Vorname\}$. Dann besteht keine funktionale Abhängigkeit $X \rightarrow A$, denn Sie können nicht ausschließen, dass Sie zwei verschiedene Personen mit denselben Namen und Vornamen in Ihrer Tabelle haben.
- Sei $X = \{Name, Vorname, Nr, Ort\}$, $Y = \{Name, Ort\}$. Dann besteht die triviale funktionale Abhängigkeit $X \rightarrow Y$.
- Sei $X = \{Strasse, Nr, Ort, Land\}$, $Y = \{Plz\}$. Dann besteht die nicht-triviale funktionale Abhängigkeit $X \rightarrow Y$. X ist also eine nicht-triviale Determinante.

Nun wissen wir genug, um die ersten drei Normalformen zu charakterisieren:

8.2 Die Erste Normalform

Die erste Normalform beschreibt eine grundlegende Eigenschaft von Relationen, die für alle manipulativen Operationen auf den Tabellen einer Datenbank – seien es Suchabfragen, seien es Einfüge-, Update- oder Löschoperationen – unerlässlich ist. Sie lautet:

Erste Normalform

Eine Relation befindet sich in der **ersten Normalform**, wenn zu jedem Attribut dieser Relation ein Wertebereich gehört, der nur **atomare** Werte enthält. Das bedeutet: Solch ein Wertebereich darf keine Werte enthalten, die aus mehreren Einzelwerten bestehen. Genauer: Er darf kein **Array** und kein mengenwertiger Bereich (**Set**) sein.



Eine Relation in der Ersten Normalform heißt **normalisiert** oder auch **normal**. Der Hauptgrund für diese Anforderung ist, dass nur auf Relationen mit dieser Eigenschaft sinnvoll Restriktionen mit WHERE-Bedingungen angewendet werden können. Ohne diesen Operator wäre z.B. kein JOIN möglich. Auch die Möglichkeiten der Sortierung nach Attributen und die Schlüsselfestlegung hängen von dieser Eigenschaft ab.

8.3 Die zweite Normalform

Nehmen Sie bitte für einen Moment an, die Tabelle BESTELLUNGEN sähe wie in Abb. 8.1 aus. Zur Erklärung:

1. Der Primärschlüssel für die Tabelle BESTELL_2_NF ist hier im Wesentlichen die Attributkombination aus *KundeId* und *ArtikelId*. Für den Fall, dass der gleiche Artikel von demselben Kunden mehrmals bestellt wurde, fügen wir noch zum Primärschlüssel eine sogenannte laufende Nummer ein; ich nenne dieses Attribut: *LfdNr*.
2. Zusätzlich wurde „aus Performancegründen“ die *ArtikelBezeichnung* mit in die Relation BESTELL_2_NF aufgenommen. In unserem Fall wollte der Anwender zu einer Bestellung sehr schnell die Bezeichnung des bestellten Artikels sehen und die Verarbeitung, die erst auf die Tabelle ARTIKEL zugreifen muss, um die Artikelbezeichnung einzulesen, dauerte ihm zu lange. Eine solche Modellierung birgt sehr große Risiken in sich: Zum Beispiel müssen Sie, wenn sich die Bezeichnung eines Artikels ändert, nicht nur das entsprechende UPDATE in der Tabelle ARTIKEL vornehmen, sondern Sie müssen die gesamte Tabelle BESTELL_2_NF durchgehen, um bei allen Sätzen, wo dieser Artikel auftaucht, das entsprechende Update vorzunehmen. Sie haben Informationen redundant gespeichert, etwas, das Sie stets vermeiden sollten. Hier verlieren Sie unter Umständen mehr Zeit, als Sie durch dieses Tabellendesign an anderer Stelle gewonnen haben. Und Sie verlieren Größenordnungen in der Zuverlässigkeit, dass Ihre Daten konsistent und korrekt sind.

Primärschlüssel					
Kundeld	Artikelld	LfdNr	Artikelbezeichnung	Menge	Bestelldatum
13	2	1	Hilfsmotoren	5	12.07.24
13	2	2	Hilfsmotoren	12	01.02.25
12	2	1	Hilfsmotoren	1	01.02.25
3	3	1	Video-Recorder	1	07.12.24
12	4	1	Steinway-Flügel	9	17.03.25
12	6	1	Akku	4	21.01.25
4	7	1	Altsaxofon	2	04.02.25
4	9	1	Computer	3	23.03.25
3	10	1	Software Engineering	12	20.10.24
4	10	1	Software Engineering	20	19.11.24

Abb. 8.1: Zehn Sätze aus der Tabelle *BESTELL_2_NF* in einer nach der *Artikelld* sortierten Ansicht

Die formale Regel, mit der man solchen Design-Fehlern auf die Schliche kommt, ist die **Zweite Normalform**. Sie lautet:



Eine Relation ist in der **Zweiten Normalform**, wenn sie

- die Erste Normalform erfüllt und
- jedes Attribut, das nicht zu irgendeinem Schlüsselkandidaten gehört, zwar von jedem vollständigen Schlüsselkandidaten funktional abhängig ist, jedoch von keiner echten Teilmenge von Attributen irgendeines Schlüsselkandidaten funktional abhängig ist.

Insbesondere darf kein Attribut, das nicht zum Primärschlüssel gehört, von einer echten Teilmenge von Attributen des Primärschlüssels funktional abhängig sein.

In unserem Beispiel war $X = \{Kundeld, Artikelld, LfdNr\}$ der einzige Schlüsselkandidat und daher auch der Primärschlüssel, es bestand aber die funktionale Abhängigkeit:

$$\{Artikelld\} \rightarrow \{Artikelbezeichnung\}$$

eine offensichtliche Verletzung der Zweiten Normalform.

8.4 Die Dritte Normalform und die Boyce/Codd-Normalform

Die Zweite Normalform ist nur dann für die Überprüfung einer Tabelle von Nutzen, wenn der Primärschlüssel zusammengesetzt ist, wenn er also aus mehr als einem Attribut besteht. Unser performance-orientierter Designer aus Abschnitt 8.3 weiß das auch und überrascht uns mit einem überarbeiteten Entwurf der Tabelle BESTELLUNGEN. Wir nennen ihn BESTELL_3_NF. Betrachten Sie dazu Abb. 8.2:

PK					
Id	Kundeld	Artikelld	Artikelbezeichnung	Menge	Bestelldatum
3	13	1	Lampenschirme	3	03.10.24
29	23	2	Hilfsmotoren	7	17.09.24
27	13	2	Hilfsmotoren	5	12.07.24
12	13	2	Hilfsmotoren	12	01.02.25
18	12	2	Hilfsmotoren	1	01.02.25
13	3	3	Video-Recorder	1	07.12.24
15	12	4	Steinway-Flügel	9	17.03.25
1	12	6	Akku	4	21.01.25
6	4	7	Altsaxofon	2	04.02.25
9	4	9	Computer	3	23.03.25
4	3	10	Software Engineering	12	20.10.24
19	4	10	Software Engineering	20	19.11.24

Abb. 8.2: Die ersten zwölf Sätze aus der Tabelle BESTELL_3_NF – in einer nach der Artikelld sortierten Ansicht

Dieser überarbeitete Entwurf hat jetzt einen neuen Primärschlüssel, der nur noch aus einem Attribut **Id** besteht. Das Attribut *LfdNr* wird nicht mehr gebraucht, es entfällt. Der Rest bleibt unverändert. Betrachten Sie bitte Abb. 8.2. Dort sehen Sie die ersten zwölf Sätze – wieder bei einer Sortierung nach der *Artikelld*.

Jetzt scheint alles in Ordnung zu sein, der Primärschlüssel **Id** ist nicht mehr zusammengesetzt und die Zweite Normalform ist automatisch erfüllt. Trotzdem gelten unsere Einwände, die wir beim vorherigen Entwurf erhoben haben, noch genauso. Wir brauchen eine weitergehende Regel, um den Design-Fehlern bei diesem Entwurf auf die Schliche zu kommen. Das ist die sogenannte Dritte Normalform. Um sie definieren zu können, brauchen wir einen neuen Begriff: die **transitive Abhängigkeit**.

Definition 25:

Sei R eine Relation und seien X_1 , X_2 und X_3 beliebige Teilmengen der Attribute von R . Dann heißt X_3 **transitiv abhängig** von X_1 genau dann, wenn die folgenden Bedingungen erfüllt sind:

1. $X_1 \rightarrow X_2$
2. aber es gilt nicht $X_2 \rightarrow X_1$ (d.h. X_1 und X_2 sind nicht äquivalent)
3. und $X_2 \rightarrow X_3$.

Wir schreiben dafür auch: $X_1 \rightarrow X_2 \rightarrow X_3$

Falls mindestens eine der drei funktionalen Abhängigkeiten $X_1 \rightarrow X_2$, $X_2 \rightarrow X_3$ oder $X_1 \cup X_2 \rightarrow X_3$ trivial ist, heißt die transitive Abhängigkeit $X_1 \rightarrow X_2 \rightarrow X_3$ **trivial**.

Jetzt können wir die Dritte Normalform definieren. Sie lautet:

Definition 26: Dritte Normalform

Eine Relation ist in der **Dritten Normalform**, wenn sie

- die Zweite Normalform erfüllt und
- in der Relation kein Attribut, das nicht zu einem Schlüsselkandidaten gehört, auf nicht-trivialen Weise von diesem Schlüsselkandidaten transitiv abhängig ist.

Insbesondere dürfen keine nicht-trivialen transitiven Abhängigkeiten vom Primärschlüssel vorliegen.

Und die nicht-triviale transitive Abhängigkeit

$\{\text{Id}\} \rightarrow \{\text{ArtikelId}\} \rightarrow \{\text{ArtikelBezeichnung}\}$

verletzt gerade diese Dritte Normalform.

Eine im Sinne der relationalen Theorie exakter formulierte Bedingung an eine Relation ist die Boyce/Codd-Normalform. Sie ist etwas weitgehender als die Dritte Normalform:

**Boyce/Codd-Normalform**

Eine Relation in der Ersten Normalform erfüllt die **Boyce/Codd-Normalform**, wenn jede nicht-triviale, minimale Determinante ein Schlüsselkandidat ist.

In unserem letzten Beispiel ist das Attribut *ArtikelId* eine (natürlich minimale) Determinante für die *ArtikelBezeichnung*, aber *ArtikelId* ist kein Schlüsselkandidat. Denn dieses Attribut ist nicht eindeutig. Also ist bei diesem Entwurf auch die Boyce/Codd-Normalform verletzt.

Es gilt allgemein:

1. Eine Relation, die der Boyce/Codd-Normalform genügt, befindet sich offensichtlich in der Zweiten Normalform.
2. Eine Relation, die der Boyce/Codd-Normalform genügt, befindet sich in der Dritten Normalform.
3. Die Boyce/Codd-Normalform ist stärker als die Dritte Normalform, d.h. es gibt Relationen, die in der Dritten Normalform sind, die jedoch die Boyce/Codd-Normalform nicht erfüllen.

Zusammenfassung

In diesem Kapitel haben wir untersucht, wie man einen bereits vorliegenden Relationenentwurf bzw. einen bereits vorliegenden Tabellenentwurf auf möglichst gründliche Weise redundanzfrei hält.

Diese Analyse wird durch drei Normalformen unterstützt.

- Wir begannen mit der Definition der funktionalen Abhängigkeit, die wir bei der Diskussion der Zweiten und Dritten Normalform entscheidend brauchten.
- Es folgte die Erste Normalform, die eine der Haupttechniken zur Redundanzvermeidung, nämlich Aufteilungen von großen Relationen in kleinere und deren Rekonstruktion durch JOINS, erst möglich macht.
- Die Zweite und die Dritte Normalform schlossen sich an.
- Wir diskutierten außerdem die Boyce/Codd-Normalform und ihre Beziehung zur Dritten Normalform.

Aufgaben zur Selbstüberprüfung

- 8.1 (Fortsetzung von Aufgabe 7.1) Ich erstelle für die Mitarbeiterdatei folgenden Tabellenentwurf (Tabelle 8.1):

Tabelle 8.1: Entwurf

Id	Name	Vorname	Projekte
1	Chaplin	Charlie	2024/018 Lagerhaltung Amazonas
2	Cluseau	Inspektor	2025/011 Patientenverwaltung 2024/032 Webauftritt Sparkasse Schotten, 2025/003 Data Warehouse Frankfurter Börse 2024/018 Lagerhaltung Amazonas
5	Einstein	Albert	2024/033 $E = m \cdot c^2$, 2024/037 Halteproblem
11	Schneider	Helge	
7	Sellers	Peter	2024/018 Lagerhaltung Amazonas

Id ist hier der Primärschlüssel. Welche Normalform ist hier verletzt?

- 8.2 Ich korrigiere meinen fehlerhaften Entwurf und bequeme mich zu einer Aufspaltung in drei Tabellen (Tabellen 8.2, 8.4 und 8.5). (Hier ist *Id* jeweils der Primärschlüssel.)

Tabelle 8.2: „Stamm“-Tabelle: MITARBEITER

MITARBEITER		
Id	Name	Vorname
1	Chaplin	Charlie
2	Cluseau	Inspektor
5	Einstein	Albert
11	Schneider	Helge
7	Sellers	Peter

Tabelle 8.3: „Stamm“-Tabelle: PROJEKT

PROJEKT		
Id	Nummer	Projektname
3	2024/018	Lagerhaltung Amazonas
5	2025/011	Patientenverwaltung
6	2024/032	Webauftritt Sparkasse Schotten
7	2025/003	Data Warehouse Frankfurter Börse
11	2024/033	E = m c c
12	2024/037	Halteproblem

Und eine Beziehungstabelle MITARBEITER_PROJEKT:

Tabelle 8.4: Eine Beziehungstabelle P_T

MaId	PrId	MaName	MaArt	Beginn	Ende
1	3	Chaplin	Software	01.01.2024	30.06.2024
2	3	Cluseau	Leitung	01.10.2023	17.07.2024
2	5	Cluseau	Dokumentation	07.04.2023	08.04.2023
2	6	Cluseau	Konzeption	15.03.2022	01.05.2022
2	7	Cluseau	Leitung	01.03.2022	30.08.2023
5	11	Einstein	Leitung	01.01.2010	31.12.2020
5	12	Einstein	Konzeption	01.01.2020	
7	3	Sellers	Hardware	01.01.2024	30.04.2024

Hier ist die Attributkombination (*MaId*, *PrId*) der Primärschlüssel (soll heißen: MitarbeiterId und ProjektId) . Welche Normalformen sind hier verletzt?

- 8.3 Ich korrigiere meinen fehlerhaften Entwurf der Tabelle MITARBEITER_PROJEKT (vergleiche Tabelle 8.5).

Tabelle 8.5: Ein Redesign der Beziehungstabelle MITARBEITER_PROJEKT


Id	MaId	PrId	MaName	MaArt	Beginn	Ende
12	1	3	Chaplin	Software	01.01.2024	30.06.2024
13	2	3	Cluseau	Leitung	01.10.2023	17.07.2024
16	2	5	Cluseau	Dokumentation	07.04.2023	08.04.2023
22	2	6	Cluseau	Konzeption	15.03.2022	01.05.2022
18	2	7	Cluseau	Leitung	01.03.2022	30.08.2023
19	5	11	Einstein	Leitung	01.01.2010	31.12.2020
6	5	12	Einstein	Konzeption	01.01.2020	
8	7	3	Sellers	Hardware	01.01.2024	30.04.2024

Hier ist die *Id* der Primärschlüssel. Welche Normalformen sind hier verletzt?

A. Lösungen der Aufgaben zur Selbstüberprüfung

- 1.1 Füge alle Sätze aus der Datei ADRESS, bei denen der Ort = Frankfurt ist, in die Datei ERGEBNIS ein und zeige die Sätze dieser Datei an.
- 1.2 Lies den ersten Satz der Datei ADRESS.
 - Wiederhole die folgende Verarbeitung, solange noch nicht das Ende der Datei erreicht wurde:
 - Falls der Satz die Daten einer Person enthält, bei der der Name = Connery und der Vorname = Sean ist, gib diesen Satz aus.
 - Lies den nächsten Satz in der Datei ADRESS.
- 1.3
 - (I) Ein Autor kann mehrere Bücher geschrieben haben, das führt dazu, dass der Name des Autors mehrfach in der Datei vorkommen kann und unter Umständen nicht einheitlich geschrieben wird. Ein typischer Redundanzfehler.
 - (II) Genauso kann ein Buch von mehreren Autoren geschrieben worden sein, d.h., der Titel des Buches kann mehrfach in der Datei vorkommen mit den unter (I) beschriebenen Konsequenzen.
 - (III) Noch dramatischer wird es mit dem Namen des Verlags: Da ein Verlag im Allgemeinen mehr als ein Buch veröffentlicht, erscheint der Name des Verlags hier viel zu oft. Da ja ein Buch von mehreren Autoren geschrieben werden kann, erscheint in solchen Fällen der Name des Verlags sogar mehrfach für ein und dasselbe Buch.
- 1.4 Man muss sowohl die Autoren als auch die Buchtitel als auch die Verlage in jeweils eigenen Tabellen abspeichern und nur die Verbindungen zwischen diesen Sätzen (ein Autor mit der Identifikation 4711 gehört zu einem Buch mit der Identifikation 1213, das in einem Verlag mit der Identifikation 815 erscheint) in gesonderten Verbindungstabellen speichern.

1.5 a)

Mitarbeiter A	Zeit	Mitarbeiter B
10 Stück Artikel X im Lager		10 Stück Artikel X im Lager
-----		-----
Satz <i>Artikel X</i> wird gelesen Info: es sind noch 10 Stück im Lager		-----
-----		Satz <i>Artikel X</i> wird gelesen Info: es sind noch 10 Stück im Lager
Die 10 Stück werden als „ent- nommen“ gebucht		-----
-----		Die 10 Stück werden noch mal als „entnommen“ gebucht

b) Der Satz des Artikels X muss von dem Zeitpunkt des Lesens bis zur Buchung für alle anderen Benutzer gesperrt werden.

2.1 a) Bei einer Datenbank über verschiedene Alphabete, bei der Eigenschaften einzelner Buchstaben wie z.B. ihre relative Häufigkeit gespeichert werden.

b) Jede Datenbank für ein Fremdsprachenlexikon.

c) Eine Datenbank einer Bibliothek.

2.2 Eine Tabelle über CDs könnte die folgenden Attribute enthalten:

Id	Titel	Musiker	Label	Lagerplatz
...

2.3 Die Abfrage widerspricht unserer Charakterisierung relationaler Datenbanksysteme, weil sie nicht in der Logik der Tabellensicht abgefasst ist. Eine bessere Formulierung wäre:

Bilde aus der Tabelle PERSON die Teiltabelle, die nur den Satz von Karl Engels enthält (das werden wir später eine Restriktion nennen) und erstelle aus dieser Tabelle eine „schmalere“ Tabelle, die nur noch aus dem Attribut „Telefonnummer“ besteht (das werden wir später eine Projektion nennen).

3.1 Hier gilt:

a) Es gibt zwei Kandidaten für einen Primärschlüssel, nämlich (Id) und die Kombination (KundeId, ArtikelId, LfdNr).

b) Beide Kandidaten sind möglich und vernünftig, für (Id) spricht lediglich die einfachere Struktur.

- 3.2 Die Constraints für BESTELLUNGEN könnten lauten:
- (I) Id ist eindeutig für jeden Satz und ungleich NULL.
 - (II) KundeId hat stets einen Wert, der einem Wert von Id in der Tabelle KUNDE entspricht.
 - (III) ArtikelId hat stets einen Wert, der einem Wert von Id in der Tabelle ARTIKEL entspricht.
 - (IV) Menge muss stets eine Zahl > 0 sein.
 - (V) Bestelldatum muss ein gültiges Datum sein.
- 3.3 Nur Möglichkeit a) ist korrekt. Nur so kann es Personen geben, die keine Kunden sind, während es hingegen unmöglich ist, dass es Kunden gibt, die keine Personen sind. Bei den Möglichkeiten b) und c) müsste jede Person auch ein Kunde sein.
- 4.1 Sei $A := \{1, 2, 3\}$, $B := \{\text{eins}, \text{zwei}\}$
- a) $A \times B = \{(1, \text{eins}), (1, \text{zwei}), (2, \text{eins}), (2, \text{zwei}), (3, \text{eins}), (3, \text{zwei})\}$
 - b) R ist eine Teilmenge von $A \times B$, daher ist R eine Relation.
 - c) Die (vollständige) Tabellendarstellung von R lautet:

Zahl	Name
1	eins
2	zwei

- 4.2 Solch eine Relation kann höchstens
 $| \text{Name} | \cdot | \text{Vorname} | \cdot | \text{Ort} | = 6 \cdot 4 \cdot 2 = 48$ Datensätze enthalten.
- 5.1 Gesucht waren die verschiedenen Beschreibungen für die Anzeige aller Artikel, die bestellt wurden.
- a) Umgangssprachlich:
 Zeige alle Artikel, deren Id bei mindestens einem Satz der Tabelle BESTELLUNGEN im Feld ArtikelId vorkommt.
 Formal:
 ARTIKEL
 WHERE ARTIKEL.Id ist enthalten in
 (PROJECT (BESTELLUNGEN) (ArtikelId))
 - b) PROJECT
 (
 ARTIKEL EQUIJOIN BESTELLUNGEN
 ON ARTIKEL.Id = BESTELLUNGEN.ArtikelId
)
 (Alle Attribute von ARTIKEL)

c) PROJECT
 (
 ARTIKEL × BESTELLUNGEN
 WHERE ARTIKEL.Id = BESTELLUNGEN.ArtikelId
)

(Alle Attribute von ARTIKEL)

5.2 Grad (ARTIKELGRUPPE × ARTIKEL) =

Grad (ARTIKELGRUPPE) + Grad (ARTIKEL) = 2 + 7 = 9

Kardinalität (ARTIKELGRUPPE × ARTIKEL) =

Kardinalität (ARTIKELGRUPPE) · Kardinalität
 (ARTIKEL) = 10 · 30 = 300

5.3 Der einzige sinnvolle Join zwischen ARTIKEL und ARTIKELGRUPPE zeigt zu jedem Artikel seine Artikelgruppe an, also

ARTIKEL NATURAL JOIN ARTIKELGRUPPE
 ON ARTIKEL.ArtikelgruppeId = ARTIKELGRUPPE.Id, bzw.
 PROJECT

(
 ARTIKEL × ARTIKELGRUPPE
 WHERE ARTIKEL.Id = ARTIKELGRUPPE.Id
)

(Alle Attribute von ARTIKEL und ARTIKELGRUPPE.Bezeichnung)

6.1 Die richtigen Antworten sind:

- a) Die Attributkombination (*Id, Autor*) ist kein Schlüsselkandidat, weil sie nicht irreduzibel ist: Man kann *Autor* daraus entfernen, ohne dass die Eindeutigkeit verloren geht.
- b) (Id) und (Autor, Titel, Verlag)
- c) Für Id, denn dieses Attribut enthält keine anwenderrelevanten Informationen.

6.2 Wie Ihnen die folgende Übersicht zeigt, sind die Tabellen ARTIKELGRUPPE, ARTIKEL, BESTELLUNGEN und ERLEDIGTEBESTELLUNGEN betroffen. Es werden insgesamt 16 Sätze aus der Datenbank gelöscht.

Nämlich zu 1 Satz mit ARTIKELGRUPPE.Id = 1 gibt es **fünf Sätze** in der ARTIKEL-Tabelle mit ArtikelgruppeId = 1, das sind die Sätze mit ARTIKEL.Id = 1, 12, 18, 20 und 27.

Zu ARTIKEL.Id = 1 gibt es 1 Satz in der Tabelle BESTELLUNGEN mit ArtikelId = 1, das ist der Satz mit BESTELLUNGEN.Id = 3.

Zu ARTIKEL.Id = 12 gibt es 2 **Sätze** in der Tabelle BESTELLUNGEN mit ArtikelId = 12, das sind die Sätze mit BESTELLUNGEN.Id = 10 und 16.

Zu ARTIKEL.Id = 18 gibt es 2 **Sätze** in der Tabelle BESTELLUNGEN mit ArtikelId = 18, das sind die Sätze mit BESTELLUNGEN.Id = 5 und 20.

Sowohl zu ARTIKEL.Id = 20 als auch 27 gibt es keine Bestellungen.

Zu BESTELLUNGEN.Id = 3, 10, 16, 5 und 20 gibt es je 1 **Satz** in der Tabelle ERLEDIGTEBESTELLUNGEN.

Das sind insgesamt 16 Sätze.

- 6.3 Jetzt wären nur der entsprechende Satz aus der Tabelle ARTIKELGRUPPE und die fünf Verweise in der Tabelle ARTIKEL betroffen.
- 6.4 Eine gute Möglichkeit besteht darin, die Delete-Option auf „Restricted“ und die Update-Version auf „Cascades“ zu setzen. Es gibt aber auch noch andere Varianten.
- 7.1 Das E/R-Modell könnte folgendermaßen aussehen:

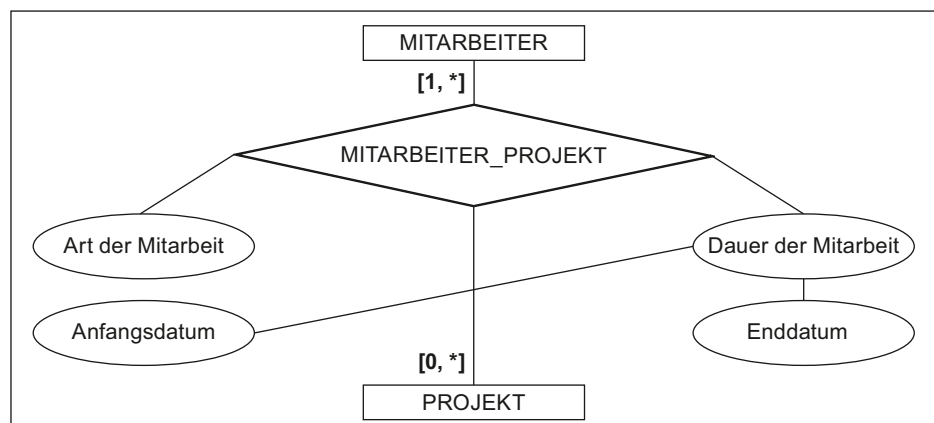


Abb. A.1: Aufgabe 7.1 Lösung

7.2 Der Lösungsausschnitt sieht folgendermaßen aus:

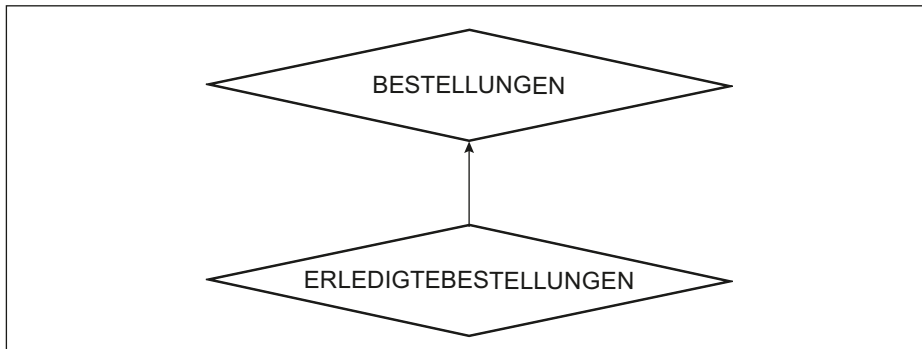


Abb. A.2: Aufgabe 7.2 Lösung

Argumente zur Verwendung einer abgeleiteten Klasse
ERLEDIGTEBESTELLUNGEN:

Dafür	Dagegen
Falls man ohne eine solche neue Tabelle für die erledigten Bestellungen arbeitet, steht im Feld <i>Commit-datum</i> immer eine Lüge (oder ein NULL-Wert). Beides macht Abfragen sehr viel schwerer und stets unlogisch.	Man braucht zur Anzeige der Daten der erledigten Bestellungen einen weiteren Join und verschlechtert somit die Performance seiner Anwendungen. Außerdem steigt der Komplexitätsgrad der Abfragen.

7.3 a) Es gilt:

- Jeder Mitarbeiter arbeitet in n Projekten mit, $n \geq 0$.
- Zu jedem Projekt gehören m Mitarbeiter, $m \geq 1$.
- Zu jedem Projekt gehört genau ein Kunde.
- Zu jedem Kunden gehören m Projekte, $m \geq 0$.

b) Pfeildiagramm



Abb. A.3: Pfeildiagramm

[min, max]-Darstellung



Abb. A.4: [min, max]-Darstellung

c) Das Tabellendesign

Mitarbeiter	Projekt														
<table><tr><th>Id</th><th>Name</th><th>Vorname</th></tr><tr><td>...</td><td>...</td><td>...</td></tr></table>	Id	Name	Vorname	<table><tr><th>Id</th><th>Bezeichnung</th><th>Anfangsdatum</th><th>Kundeld</th></tr><tr><td>...</td><td>...</td><td>...</td><td>...</td></tr></table>	Id	Bezeichnung	Anfangsdatum	Kundeld
Id	Name	Vorname													
...													
Id	Bezeichnung	Anfangsdatum	Kundeld												
...												
Kunde	MitarbeiterProjekt														
<table><tr><th>Id</th><th>Firmenbezeichnung</th></tr><tr><td>...</td><td>...</td></tr></table>	Id	Firmenbezeichnung	<table><tr><th>Id</th><th>MitarbeiterId</th><th>ProjektId</th><th>DatumEin</th><th>DatumAus</th></tr><tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr></table>	Id	MitarbeiterId	ProjektId	DatumEin	DatumAus
Id	Firmenbezeichnung														
...	...														
Id	MitarbeiterId	ProjektId	DatumEin	DatumAus											
...											

Abb. A.5: Tabellendesign

- 8.1 Hier ist die Erste Normalform verletzt.
- 8.2 Hier ist die Zweite Normalform verletzt, weil in der Beziehungstabelle MITARBEITER_PROJEKT das Attribut *MaName* (Name des Mitarbeiters) von der echten Teilmenge $\{MaId\}$ des Primärschlüssels funktional abhängig ist. Wie Sie sehen, erhalten wir durch diese Verletzung der Zweiten Normalform redundante Aufzählungen ein- und desselben Mitarbeiternamens.
- 8.3 Hier ist zunächst die Zweite Normalform verletzt, denn $(MaId, PrId)$ ist ein Schlüsselkandidat der Beziehungstabelle MITARBEITER_PROJEKT und wieder ist das Attribut *MaName* von der echten Teilmenge $\{MaId\}$ abhängig.

Außerdem haben wir die nicht-triviale transitive Abhängigkeit des Attributs *MaName*, das zu keinem Schlüsselkandidaten gehört, von *Id*. Es gilt:

$$Id \rightarrow MaId \rightarrow MaName$$

Damit ist die Dritte Normalform verletzt. Außerdem ist *MaId* eine nicht-triviale minimale Determinante für *MaName*. *MaId* ist aber kein Schlüsselkandidat. Damit ist auch die Boyce/Codd-Normalform verletzt.

B. Literaturverzeichnis

Chen, Peter Pi-Shan:

The Entity-Relationship Model: Towards a Unified View of Data.

ACM Transactions on Database System, Vol. 1, No. 1, March 1976

Date, Chris J.:

An Introduction to Database Systems.

Addison-Wesley, 2004

Date, Chris J.:

Database Design and Relational Theory: Normal Forms and All That Jazz.

O'Reilly Media, 2012

Geisler, Frank:

Datenbanken – Grundlagen und Design.

Verlagsgruppe Hüthig-Jehle-Rehm, 2009

Kemper, Alfons; Eickler, André:

Datenbanksysteme.

Oldenburger Wissenschaftsverlag, 2011

Saake, Gunter; Sattler, Kai-Uwe; Heuer, Andreas:

Datenbanken – Konzepte und Sprachen.

mitp, 2010

Schubert, Matthias:

Datenbanken: Theorie, Entwurf und Programmierung relationaler Datenbanken.

Vieweg + Teubner Verlag, 2007

Steiner, René:

Grundkurs relationale Datenbanken.

Vieweg + Teubner Verlag, 2009

C. Abbildungsverzeichnis

Abb. 3.1:	UML-Diagramm zur Vererbungshierarchie der Klassen PERSON, KUNDE und LIEFERANT	22
Abb. 3.2:	UML-Diagramm zur Vererbungshierarchie der Klassen „Bestellungen“ und „erledigteBestellungen“	24
Abb. 3.3:	Eine Übersicht über die Tabellen und Beziehungen der Datenbank „LieferBar“	25
Abb. 7.1:	Erster Schritt eines E/R-Diagramms für unser Warenhaus „LieferBar“ –Entitätstypen ohne Attribute und Beziehungen ...	74
Abb. 7.2:	Erster Schritt eines E/R-Diagramms für unser Warenhaus „LieferBar“. Entitätstypen ohne Attribute und Beziehungen	75
Abb. 7.3:	Einfaches E/R-Diagramm zur Darstellung einer 1:1-Beziehung	76
Abb. 7.4:	Einfaches E/R-Pfeildiagramm zur Darstellung einer 1:1-Beziehung.....	76
Abb. 7.5:	E/R-Diagramm in [min, max]-Notation zur Darstellung einer 1:1-Beziehung.....	76
Abb. 7.6:	Einfaches E/R-Diagramm zur Darstellung einer 1:1-Beziehung	77
Abb. 7.7:	Einfaches E/R-Pfeildiagramm zur Darstellung einer 1:1-Beziehung.....	77
Abb. 7.8:	E/R-Diagramm in [min, max]-Notation zur Darstellung einer 1:1-Beziehung.....	78
Abb. 7.9:	E/R-Diagramm in [min, max]-Notation zur Darstellung einer 1:1-Beziehung.....	78
Abb. 7.10:	Einfaches E/R-Diagramm zur Darstellung einer 1:1-Beziehung	79
Abb. 7.11:	Einfaches E/R-Pfeildiagramm zur Darstellung einer 1:1-Beziehung.....	79
Abb. 7.12:	E/R-Diagramm in [min, max]-Notation zur Darstellung einer 1:1-Beziehung.....	79
Abb. 7.13:	E/R-Diagramm in [min, max]-Notation zur Darstellung einer rekursiven 1:1-Beziehung.....	80
Abb. 7.14:	Einfaches E/R-Diagramm zur Darstellung einer n:1-Beziehung	81
Abb. 7.15:	E/R-Diagramm als Pfeildiagramm zur Darstellung einer n:1-Beziehung.....	81
Abb. 7.16:	E/R-Diagramm in [min, max]-Notation zur Darstellung einer n:1-Beziehung.....	81
Abb. 7.17:	Einfaches E/R-Diagramm zur Darstellung einer konditionalen n:1-Beziehung.....	82
Abb. 7.18:	E/R-Diagramm als Pfeildiagramm zur Darstellung einer konditionalen n:1-Beziehung.....	82
Abb. 7.19:	E/R-Diagramm in [min, max]-Notation zur Darstellung einer n:1-Beziehung.....	82

Abb. 7.20:	E/R-Diagramm in [min, max]-Notation zur Darstellung einer n:1-Beziehung	83
Abb. 7.21:	E/R-Diagramm in [min, max]-Notation zur Darstellung einer n:1-Beziehung	84
Abb. 7.22:	E/R-Diagramm in [min, max]-Notation zur Darstellung einer rekursiven 1:n-Beziehung	84
Abb. 7.23:	E/R-Diagramm in [min, max]-Notation zur Darstellung einer m:n-Beziehung mit Attributen der Beziehungsentitäten	85
Abb. 7.24:	Dritter Schritt eines E/R-Diagramms für unser Warenhaus „LieferBar“ – Entitäten mit (einigen) Attributen und mit Beziehungen	86
Abb. 7.25:	E/R-Diagramm.....	89
Abb. 8.1:	Zehn Sätze aus der Tabelle BESTELL_2_NF in einer nach der ArtikelId sortierten Ansicht.....	92
Abb. 8.2:	Die ersten zwölf Sätze aus der Tabelle BESTELL_3_NF – wieder in einer nach der ArtikelId sortierten Ansicht.....	93
Abb. A.1:	Aufgabe 7.1 Lösung	102
Abb. A.2:	Aufgabe 7.2 Lösung	103
Abb. A.3:	Pfeildiagramm	103
Abb. A.4:	[min, max]-Darstellung	103
Abb. A.5:	Tabellendesign.....	104

D. Tabellenverzeichnis

Tabelle 1.1:	Bibliotheksdaten.....	8
Tabelle 2.1:	Ein erster Entwurf einer Adresstabelle	10
Tabelle 3.1:	Einige Artikel, geordnet nach ihren zugehörigen Artikelgruppen	16
Tabelle 3.2:	Struktur der Tabelle ARTIKELGRUPPE.....	16
Tabelle 3.3:	Einige Sätze der Tabelle ARTIKELGRUPPE	16
Tabelle 3.4:	Struktur der Tabelle ARTIKEL.....	17
Tabelle 3.5:	Einige Sätze der Tabelle ARTIKEL	17
Tabelle 3.6:	Struktur der Tabelle PERSON.....	21
Tabelle 3.7:	Einige Sätze der Tabelle PERSON	22
Tabelle 3.8:	Struktur der Tabelle KUNDE.....	23
Tabelle 3.9:	Ein Satz aus der Tabelle KUNDE	23
Tabelle 3.10:	Struktur der Tabelle LIEFERANT	23
Tabelle 3.11:	Ein Satz aus der Tabelle LIEFERANT	23
Tabelle 3.12:	Struktur der Tabelle BESTELLUNGEN	24
Tabelle 3.13:	Einige Sätze der Tabelle BESTELLUNGEN.....	24
Tabelle 3.14:	Struktur der Tabelle ERLEDIGTEBESTELLUNGEN.....	25
Tabelle 3.15:	Ein Satz aus der Tabelle ERLEDIGTEBESTELLUNGEN...	25
Tabelle 4.1:	Elemente der Menge $M1 \times M2 \times M3$	30
Tabelle 4.2:	Elemente der Menge Name \times Vorname \times Alter	31
Tabelle 4.3:	Elemente der Relation $QUOTIENT \subseteq M1 \times M2 \times M3$	32
Tabelle 4.4:	Eine Tabellendarstellung der Relation $QUOTIENT \subseteq M1 \times M2 \times M3$	33
Tabelle 4.5:	Elemente der Menge Name \times Vorname \times Alter	34
Tabelle 4.6:	Struktur der Tabelle ARTIKEL.....	35
Tabelle 4.7:	Tabelle FREUNDE.....	37
Tabelle 4.8:	Werte von Alter aus der Tabelle FREUNDE ohne Unterdrückung gleicher Werte.....	37
Tabelle 4.9:	Werte von Alter aus der Tabelle FREUNDE mit Unterdrückung gleicher Werte	38
Tabelle 5.1:	$UNION(ARTIKEL \text{ mit } Id = 9, ARTIKEL \text{ mit } Preis > 50000)$:	42
Tabelle 5.2:	$INTERSECTION(KUNDE, LIEFERANT)$	43
Tabelle 5.3:	$DIFFERENCE(PERSON, LIEFERANT)$: Personen, die keine Lieferanten sind	44
Tabelle 5.4:	$PRODUKT(ARTIKEL, ARTIKELGRUPPE)$: Ein Ausschnitt von elf Sätzen	46

Tabelle 5.5:	RESTRICT(ARTIKEL) WHERE Bezeichnung beginnt mit D	47
Tabelle 5.6:	PRODUKT(ARTIKEL,ARTIKELGRUPPE) WHERE ARTIKEL.ArtikelgruppeId = ARTIKELGRUPPE.Id: Ein Ausschnitt von zwölf Sätzen	48
Tabelle 5.7:	PROJECT(ARTIKEL) (Bezeichnung, Bestandsmenge): Fünf Sätze.....	50
Tabelle 5.8:	PROJECT(PRODUKT(ARTIKEL,ARTIKELGRUPPE) WHERE ARTIKEL.ArtikelgruppeId = ARTIKELGRUPPE.Id) (alles außer ARTIKELGRUPPE.Id): Ein Ausschnitt von zwölf Sätzen	51
Tabelle 6.1:	Die ersten zehn Artikel bei alphabetischer Sortierung nach der Bezeichnung	61
Tabelle 7.1:	Schema für 1:1-Beziehung	76
Tabelle 7.2:	Schema für 1:1-Beziehung	77
Tabelle 7.3:	Schema für 1:1-Beziehung	78
Tabelle 7.4:	Schema für 1:1-Beziehung.....	79
Tabelle 7.5:	Schema für rekursive 1:1-Beziehung	80
Tabelle 7.6:	Schema für 1:n-Beziehung	81
Tabelle 7.7:	Schema für 1:n-Beziehung.....	82
Tabelle 7.8:	Schema für 1:n-Beziehung.....	83
Tabelle 7.9:	Schema für eine rekursive n:1-Beziehung.....	84
Tabelle 7.10:	Schema für 1:n-Beziehung.....	85
Tabelle 7.11:	Schema für 1:n-Beziehung	86
Tabelle 8.1:	Entwurf	95
Tabelle 8.2:	„Stamm“-Tabelle: MITARBEITER.....	96
Tabelle 8.3:	„Stamm“-Tabelle: PROJEKT	96
Tabelle 8.4:	Eine Beziehungstabelle P_T	96
Tabelle 8.5:	Ein Redesign der Beziehungstabelle MITARBEITER_PROJEKT	97
Tabelle F.1:	Personaldaten.....	113
Tabelle F.2:	Musikerdaten.....	115

E. Sachwortverzeichnis

A

Abhängigkeit	
funktionale.....	90
transitive	94
triviale	94
Aspekt	
manipulativer	14
struktureller.....	14
atomar	91
Attribut.....	11, 18, 70
Attributname	11
Attributtyp.....	18
Attributwert.....	11, 70

B

Beziehung	
1 zu 1-.....	75
1 zu n-.....	80
m zu n-	85
n zu 1-.....	80
Boyce/Codd-Normalform	94

C

Candidate Key	58
Cascades	64
Chen, P. P. S.....	70
Codd	14
constraints	20

D

Datenanalyse	
semantische	70
Datenbankentwurf.....	88
mit E/R-Modell.....	87
mit Normalisierung.....	88
Datenbankrelation	35
Datenmodell	
semantisches.....	69
Datensatz.....	11
Default-Wert.....	19
Determinante.....	90
funktionale.....	90
minimale.....	90

E

E/R Beziehungsanzeige	
einfache	76
E/R-Diagrammtechnik	
Attribut	74
Beziehung.....	75
Entitätsklasse	69
E/R-Modell	70
Entität	70
Entitätstyp.....	71, 73
Entity/Relationship-Modell	70
Entity-Analyse	70
Entity-Integrität	20, 65

F

foreign key.....	60
Fremdschlüssel	20, 60
funktional.....	75

G

Grad	
einer Relation	39

I

Integrität	20
referentielle	20, 64, 65
INTERSECTION	43

K

Kardinalität	39, 75
Kardinalitätendarstellung.....	86
konditional.....	77
Kreuzprodukt	29

L

LieferBar (Datenbank).....	21
----------------------------	----

M

Mehr-Benutzer-Betrieb.....	6
Modell	
relationales.....	12, 14, 39

N

normal.....	91
Normalform	
Dritte.....	94
Erste	91
Zweite.....	92
normalisiert	91
Notation	
[min, max]-Notation	76
NULL-Wert	19

O

Objekt.....	10
Operator.....	41
DIFFERENCE.....	43
DIVIDE	55
EQUIJOIN	54
JOIN	51
NATURAL JOIN.....	51
PRODUCT.....	44
PROJECT	49
RESTRICT	47
THETA-JOINS.....	54
UNION	42

P

persistent	3
Pfeilnotation.....	76
Primärschlüssel	14, 59
primary key	59
Produkt	
kartesisches	29

R

Redundanz	5
Relation	11, 32
Relationship-Analyse	70
restricted.....	64

S

Schlüsselkandidat	15
-------------------------	----

T

Tabelle	11
ARTIKEL	17
ARTIKELGRUPPE	16
BESTELL_3_NF	93
BESTELLUNGEN	23
ERLEDIGTEBESTELLUNGEN	
.....	25
KUNDE	23
LIEFERANT	23
PERSON	21
Tabellendarstellung	
einer Relation.....	32
Tabellenkopf	32
typ-kompatibel	42
U	
union-kompatibel	42

F. Einsendeaufgabe

Theoretische Grundlagen von DB-Systemen

Typ A

Name:		Vorname:	
Postleitzahl und Ort:		Straße:	
Matrikelnummer:		Studiengangs-Nr.:	
DBI11XX	1	Auflage: 0213 N01	

Einsendeaufgabencode: DBI11-XX1-N01
Tutor:
Datum:
Note:
Unterschrift:

Bitte reichen Sie mit Ihren Lösungen die Aufgabenstellungen ein!
 Füllen Sie das Adressfeld bitte sorgfältig aus!

1. In einer Firma wird eine Datenbank eingerichtet, mit der man die Mitarbeiter, ihre Zuordnung zu einer Abteilung und die jeweiligen Abteilungsleiter verwalten will. Sehen Sie in der Tabelle F.1 die Datei, die man dazu implementiert. In der Struktur dieser Datei finden sich mehrere Entwurfsfehler, die zu einer erheblichen Redundanz bei der Speicherung von Daten führen können. Beschreiben Sie diese Entwurfsfehler.

10 Pkt.

Tabelle F.1: Personaldaten

Id	Mitarbeiter name	Mitarbeiter Vorname	Abteilung	Abteilungs-leiterName	Abteilungs-leiter Vorname
22	Müller	Theodor	Einkauf	Mueller	Hedwig
24	Mann	Markus	Controlling	Fricke	Lydia
26	Wanka	Sabine	Finanzen	Smart	Maxwell
27	Meinart	Otto	Controlling	Fricke	Lydia
35	Meinart	Otto	Finanzen	Smart	Maxwell
86	Wollert	Kurt	Einkauf	Mueller	Hedwig
236	Diekmann	Carla	Finanzen	Smart	Maxwell
283	Löw	Hans	Personal	Heinz	Karl
312	Billmann	Helga	Forschung	Perel	Gregor
915	Niederlande	Frank	Einkauf	Mueller	Hedwig

2. Skizzieren Sie am Beispiel eines Online-Versandhandels ein Szenario, bei dem zu gleicher Zeit stattfindende Einlagerungs- und Bestellbuchungen bei fehlenden Sicherheitsvorkehrungen zu falschen Bestandszahlen in der Datenbank führen können.
- 10 Pkt.*
3. Nehmen Sie an, Sie hätten eine relationale Datenbank für Ihre Adresstabelle zur Verfügung. Wie müsste die Anweisung „Ändere Karl Engels in Friedrich Engels“ relational korrekt lauten?
- 8 Pkt.*
4. Sei A:= **Vornamen: varchar(20)**, B:= **Namen: varchar(20)**. A enthalte sechs Vornamen, B enthalte drei Namen.
- a) Wie viele Elemente enthält $A \times B$?
- 5 Pkt.*
- b) Es sei $R := \{(x, y) \in A \times B \mid \text{Es gibt einen Fussballnationalspieler mit Vornamen} = x \text{ und Namen} = y.\}$ Ist R eine Relation?
- 5 Pkt.*
5. In unserer Datenbank „LieferBar“ hatten wir in der Tabelle BESTELLUNGEN das Feld **KundeId**, das angab, welcher Kunde zu dieser Bestellung gehörte. Es referenzierte auf die entsprechenden Primärschlüssel **Id** sowohl in der Tabelle KUNDE als auch in der Tabelle PERSON. Sie wollen nun alle Attribute der Personen sehen, die irgendetwas bestellt haben. Stellen Sie diese Abfrage auf dreierlei Weisen dar (Sie müssen jeweils nur die Bedingung formulieren).
- a) In der Form
PERSON WHERE *Bedingung*
- 5 Pkt.*
- b) In der Form
PROJECT
(
PERSON EQUIJOIN BESTELLUNGEN
ON *Bedingung*
)
(Alle Attribute von PERSON)
- 5 Pkt.*
- c) In der Form
PROJECT
(
PERSON x BESTELLUNGEN
WHERE *Bedingung*
)
(Alle Attribute von PERSON)
- 5 Pkt.*

Theoretische Grundlagen von DB-Systemen

Grundlage für die folgenden Aufgaben ist der Entwurf für eine Musikdatenbank, wir wollen uns auf die Entitätstypen Musiker und Instrumente beschränken.

Es gilt: Ein Musiker spielt ein oder mehrere Instrumente, ein Instrument wird von 0, einem oder mehreren Musikern gespielt.

6. Erstellen Sie für diese Situation ein E/R-Modell. Stellen Sie dabei die Kardinalitäten auf die drei verschiedenen, im Heft besprochenen Arten dar (einfach, Pfeil und in der [min, max]-Notation).

12 Pkt.

7. Machen Sie nun für dieses Beispiel einen Tabellenentwurf.

15 Pkt.

8. Ein Kollege erstellt den folgenden Tabellenentwurf. (Tabelle F.2) Id ist hier der Primärschlüssel. Welche Normalform ist hier verletzt?

5 Pkt.

Tabelle F.2: Musikerdaten

Id	Name	Vorname	Instrumente
1	Davis	Miles	Trompete, Flügelhorn, Synthesizer
2	Collins	Phil	Gesang, Schlagzeug
5	Lang	Lang	Klavier
11	Schneider	Helge	Gesang, Klavier, Schlagzeug, Trompete
7	Simone	Nina	Gesang, Klavier

9. Ein anderer Kollege präsentiert den folgenden Datenbankentwurf:

MUSIKER			INSTRUMENT	
Id	Name	Vorname	Id	Instrument
1	Davis	Miles	300	Trompete
2	Collins	Phil	305	Flügelhorn
5	Lang	Lang	306	Synthesizer
11	Schneider	Helge	307	Gesang
7	Simone	Nina	311	Schlagzeug
			312	Klavier

Hier ist jeweils **Id** der Primärschlüssel

Theoretische Grundlagen von DB-Systemen

MusikerId	InstrumentId	PersonName
11	300	Schneider
1	305	Davis
11	307	Schneider
7	307	Simone
1	300	Davis
2	311	Collins
11	312	Schneider
2	307	Collins
7	312	Simone
11	311	Schneider
1	306	Davis
5	312	Lang

Hier ist die Attributkombination (**MusikerId,InstrumentId**) der Primärschlüssel.

Welche Normalformen sind hier verletzt?

5 Pkt.

10. Schließlich wird folgender Datenbankentwurf vorgestellt:

MUSIKER			INSTRUMENT	
Id	Name	Vorname	Id	Instrument
1	Davis	Miles	300	Trompete
2	Collins	Phil	305	Flügelhorn
5	Lang	Lang	306	Synthesizer
11	Schneider	Helge	307	Gesang
7	Simone	Nina	311	Schlagzeug
			312	Klavier

Theoretische Grundlagen von DB-Systemen

Id	MusikerId	InstrumentId	PersonName
6	11	300	Schneider
22	1	305	Davis
12	11	307	Schneider
24	7	307	Simone
14	1	300	Davis
16	2	311	Collins
20	11	312	Schneider
2	2	307	Collins
8	7	312	Simone
18	11	311	Schneider
10	1	306	Davis
4	5	312	Lang

Hier ist in allen drei Fällen **Id** der Primärschlüssel. Welche Normalformen sind hier verletzt?

10 Pkt.