

eXamen.press

eXamen.press ist eine Reihe, die Theorie und Praxis aus allen Bereichen der Informatik für die Hochschulausbildung vermittelt.

Andreas Meier

Relationale und postrelationale Datenbanken

6. überarbeitete und erweiterte Auflage
Mit 119 Abbildungen

 Springer

Andreas Meier

Departement für Informatik

Universität Fribourg

Bd. de Perolles 90

1700 Fribourg, Schweiz

andreas.meier@unifr.ch

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen

Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über

<http://dnb.d-nb.de> abrufbar.

Bis zur 5. Auflage erschienen im Springer-Verlag Berlin Heidelberg unter dem Titel
Relationale Datenbanken. Leitfaden für die Praxis.

ISSN 1614-5216

ISBN 978-3-540-46553-9 Springer Berlin Heidelberg New York

ISBN 978-3-540-00905-4 5.Auflage Springer Berlin Heidelberg New York

Dieses Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere die der Übersetzung, des Nachdrucks, des Vortrags, der Entnahme von Abbildungen und Tabellen, der Funksendung, der Mikroverfilmung oder der Vervielfältigung auf anderen Wegen und der Speicherung in Datenverarbeitungsanlagen, bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Eine Vervielfältigung dieses Werkes oder von Teilen dieses Werkes ist auch im Einzelfall nur in den Grenzen der gesetzlichen Bestimmungen des Urheberrechtsgesetzes der Bundesrepublik Deutschland vom 9. September 1965 in der jeweils geltenden Fassung zulässig. Sie ist grundsätzlich vergütungspflichtig. Zuwiderhandlungen unterliegen den Strafbestimmungen des Urheberrechtsgesetzes.

Springer ist ein Unternehmen von Springer Science+Business Media

springer.de

© Springer-Verlag Berlin Heidelberg 1991, 1994, 1997, 2001, 2004, 2007

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften. Text und Abbildungen wurden mit größter Sorgfalt erarbeitet. Verlag und Autor können jedoch für eventuell verbliebene fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Satz: Druckfertige Daten des Autors

Herstellung: LE-TEX, Jelonek, Schmidt & Vöckler GbR, Leipzig

Umschlaggestaltung: KünkelLopka Werbeagentur, Heidelberg

Gedruckt auf säurefreiem Papier 33/3142 YL – 5 4 3 2 1 0

Für Heiri, Ramani und Tina

Geleitwort

Dass ein Fachbuch der Informatik mehrere Auflagen in kurzer Zeit erlebt, ist eine Seltenheit. Die Frage, die zu stellen ist, lautet: Wie kommt diese Ausnahmesituation zustande?

Eine zutreffende, aber leider nur einfach aussehende Antwort wäre: Das Buch von Andreas Meier ist für Praktiker und Lernende verständlich geschrieben, wobei wir verständlich als praktisch nachvollziehbar deuten. Warum, so lautet eine bohrende Zusatzfrage, ist das Buch von Meier nachvollziehbar? Meine Antwort auf diese Frage ist etwas umfangreicher:

Das Buch ist für den Lernenden nachvollziehbar, weil es einsichtig, angemessen und richtig ist. «Einsichtig» soll in diesem Zusammenhang bedeuten, dass das Buch den Anfangswissensstand des Lesers berücksichtigt. Sprache und Bild harmonisieren in dem Buch. Höchste und präzise Abstraktionen sind für einen lernenden Anfänger nicht einsichtig. «Angemessenheit» verlangt, dass eine Darstellung für den Lernenden nach Umfang und Tiefe in den Kontext passt, in dem sie gefordert wird. Akademische Forschung und industrielle Anwendung sind in diesem Sinne zwei grundsätzlich verschiedene Kontexte. Meier hat die industrielle Anwendung im Sinn. «Richtig» ist für jeden Autor der heikelste und anstrengendste Punkt, wenn Richtigkeit nicht einfach als logische Wahrheit, sondern als Übereinstimmung mit der anerkannten Modellbildung verstanden wird, die sich ständig ändert.

In der Datenbankwelt, die aus den alten Dateisystemen der Betriebssysteme heraus geboren wurde, findet seit über dreißig Jahren eine rege Modell-Evolution statt. Über den CODASYL-Ansatz gelangten wir zum Relationen-Modell und stehen nun mit dem Internet vor dem größten vorstellbaren Datenbanksystem. Die Konsequenzen für die Zukunft sind noch nicht absehbar. Gesagt werden kann nur, dass kein ruhiges Fahrwasser vor uns liegt. Es ist deshalb wichtig, dass der Autor in der überarbeiteten Auflage der Integration heterogener Datenbanken im Web Gewicht beimisst und mit einem Kapitel über postrelationale Datenbanken die Zukunftsperspektiven aufzeigt.

Ich wünsche der neuen Auflage eine breite und interessierte Leserschaft.

Erlangen, im November 2006

Hartmut Wedekind

Vorwort zur sechsten Auflage

Wir kennen den sektoralen Strukturwandel, d.h. die längerfristige Verlagerung der Beschäftigung von der Landwirtschaft über den Produktionssektor hin zu Dienstleistungen und zur Informationsverarbeitung. Bereits heute dominieren die Informationsberufe gegenüber anderen Berufssparten deutlich. Immer mehr Personen beschäftigen sich mit der Produktion, Verarbeitung und Verteilung von Informationen. Die westlichen Länder befinden sich auf dem Weg zu einer Wissens- und Informationsgesellschaft.

Datenbanksysteme mit ihren Sprach- und Auswertungsfunktionen zählen neben der Kommunikationsinfrastruktur zu den Schlüsseltechnologien unserer Wirtschaft. Im elektronischen Geschäft benötigen wir webbasierte Produktkataloge für die Informationsphase. Zudem verwenden wir Kundendatenbanken in der Offertstellung, in der Vertragsverhandlung und bei der Kontraktabwicklung. Auch die Distribution der Produkte überwachen wir mit Online-Datenbanken, unabhängig davon, ob es sich um digitale oder materielle Güter handelt. Das kundenindividuelle Marketing ohne Datenbank oder Data Warehouse ist kaum mehr vorstellbar.

Das vorliegende Fachbuch zeigt das gesamte Anwendungsspektrum vom relationalen Datenbankentwurf bis hin zur Entwicklung von postrelationalen Datenbanksystemen im Überblick und praxisnah. Als *fundierte Einführung in das Gebiet der relationalen und postrelationalen Datenbanksysteme* ermöglicht es dem interessierten Praktiker,

- bei der Datenmodellierung und in der Datenbankentwurfsmethodik das Entitäten-Beziehungsmodell und das Relationenmodell anzuwenden,
- relationale Abfrage- und Manipulationssprachen kennenzulernen und einzusetzen,
- die automatischen Abläufe und Techniken innerhalb eines relationalen oder postrelationalen Datenbanksystems zu verstehen,
- Konzepte für die Integration von Datenbanken ins Web und für die Migration zu bewerten sowie

- die Stärken und Schwächen relationaler Technologie zu erkennen, in seine Überlegungen miteinzubeziehen und künftige Entwicklungen einigermaßen richtig abzuschätzen.

Das Fachbuch richtet sich an *Praktiker und Ausbildungsverantwortliche von Unternehmen, an Dozierende und Studierende von Universitäten, Fachhochschulen und Berufsakademien* sowie an alle, die eine praxisbezogene Einführung in die relationale und postrelationale Datenbanktechnik suchen. Im Zentrum stehen wichtige Begriffe, die zwar vielerorts gebraucht, jedoch selten richtig verstanden und angewendet werden. Illustriert wird das Fachbuch durch eine große Anzahl einfacher, meist selbsterklärender Abbildungen. Ein gezielt zusammengestelltes Literaturverzeichnis verweist den interessierten Leser auf weiterführende Publikationen, die einen vertieften Einblick in die angeschnittenen Themenkreise bieten.

Diese Einführung geht die relationale und postrelationale Datenbanktechnologie von verschiedenen Blickwinkeln an. Sie erläutert Entwurfsmethoden und Sprachaspekte ebenso wie wichtige Architekturkonzepte von Datenbanksystemen. Auf eine Beschreibung spezifischer Herstellerprodukte wird verzichtet, um den *grundlegenden Methoden und Techniken mehr Platz einzuräumen* und das Verständnis beim Benutzen relationaler und postrelationaler Datenbanken zu fördern. Somit schließt diese Veröffentlichung eine Lücke in der praxisbezogenen Fachliteratur.

Neu hinzugekommen sind in der sechsten, überarbeiteten und erweiterten Auflage ein Tutorium in SQL (Structured Query Language). Damit wird dem Wunsche vieler Leserinnen und Leser Rechnung getragen, der international standardisierten Sprache mehr Gewicht zu verleihen. Ein Repetitorium sowie ein Appendix mit einer Fallstudie für das Softwaresystem Access erlauben außerdem, die Kenntnisse in SQL konkret zu prüfen. Weitere Übungsaufgaben und Lösungsvorschläge finden sich außerdem auf der neu erstellten Website <http://www.RelationaleDatenbanken.ch>.

Alle Kapitel sind überarbeitet und teilweise erweitert worden. So findet sich neu in Kapitel 1 eine Einführung in das Gebiet der Informationssysteme: Hier wird argumentiert, dass Informationen spezifische Eigenschaften besitzen und als Produktionsfaktor besondere Herausforderungen an Unternehmen und Organisationen stellen. In Kapitel 2 wurde der Teil über die Normalformen vervollständigt, obwohl höhere Normalformen in der Praxis selten anzuwenden sind. Bei den Kapiteln 4 und 5 wurde der Tatsache Rechnung getragen, dass webbasierte Technologien und Lösungen immer mehr Fuß fassen. Schließlich konnte in Kapitel 6 bei der Diskussion von fuzzy Datenbanken ein attraktives Beispiel aus dem Kundenbeziehungsma-

nagement gewählt werden. Damit lassen sich Konflikte bei scharfen Kundensegmenten im Marketing beheben.

Das Fachbuch ist im Rahmen eines firmeninternen Ausbildungsprogramms im Bankenumfeld entstanden, ergänzt durch Diskussionsbeiträge aus den Vorlesungen «Praxis relationaler Datenbanken» an der ETH in Zürich (für Informatiker und Ingenieure) sowie «Informationssysteme und Datenbanken» an der Universität Fribourg (für Studierende der Wirtschaftswissenschaften). Viele Fachkolleginnen und -kollegen aus Praxis und Hochschule haben dazu beigetragen, den Text verständlicher und die Abbildungen anschaulicher zu gestalten. Mein Dank richtet sich an Urs Bebler, Eirik Danielson, Bernardin Denzel, Emmerich Fuchs, Peter Gasche, Caroline Grässle-Mutter, Michael Hofmann, Stefan Hüsemann, Günther Jakobitsch, Hans-Peter Joos, Klaus Küspert, Gitta Marchand, Michael Matousek, Thomas Myrach, Mikael Norlindh, Michel Patcas, Fabio Patocchi, Ernst-Rudolf Patzke, Thomas Rätz, Marco Savini, Werner Schaad, August Scherrer, Walter Schnider, Henrik Stormer, Max Vetter, Hartmut Wedekind und Gerhard Weikum. Ein besonderes Kompliment richte ich an Andreea Ionas, die unermüdlich Text und Zeichnungen redigiert hat. Dem Springer-Verlag, vor allem Clemens Heine, Agnes Herrmann und Michael Reinfarth, danke ich für die angenehme Zusammenarbeit.

Fribourg, im November 2006

Andreas Meier

Inhaltsverzeichnis

| | | |
|----------|---|----|
| 1 | Der Weg zum Datenmanagement | 1 |
| 1.1 | Der Untersuchungsgegenstand der Wirtschaftsinformatik | 1 |
| 1.2 | Grundbegriffe des Relationenmodells | 4 |
| 1.3 | Die international standardisierte Sprache SQL | 7 |
| 1.4 | Die Komponenten eines relationalen Datenbanksystems | 10 |
| 1.5 | Zur Organisation des Datenbankeinsatzes | 12 |
| 1.6 | Bemerkungen zur Literatur | 14 |
| 2 | Schritte zur Datenmodellierung | 17 |
| 2.1 | Von der Datenanalyse zur Datenbank | 17 |
| 2.2 | Das Entitäten-Beziehungsmodell | 20 |
| 2.2.1 | Entitäten und Beziehungen | 20 |
| 2.2.2 | Assoziationstypen | 22 |
| 2.2.3 | Generalisation und Aggregation | 24 |
| 2.3 | Das relationale Datenbankschema | 28 |
| 2.3.1 | Überführen des Entitäten-Beziehungsmodells | 28 |
| 2.3.2 | Abbildungsregeln für Beziehungsmengen | 30 |
| 2.3.3 | Abbildungsregeln für Generalisation und Aggregation | 35 |
| 2.4 | Abhängigkeiten und Normalformen | 38 |
| 2.4.1 | Sinn und Zweck von Normalformen | 38 |
| 2.4.2 | Funktionale Abhängigkeiten | 41 |
| 2.4.3 | Transitive Abhängigkeiten | 43 |
| 2.4.4 | Mehrwertige Abhängigkeiten | 45 |
| 2.4.5 | Verbundabhängigkeit | 48 |
| 2.5 | Strukturelle Integritätsbedingungen | 51 |
| 2.6 | Unternehmensweite Datenarchitektur | 53 |
| 2.7 | Rezept zum Datenbankentwurf | 57 |
| 2.8 | Bibliografische Angaben | 59 |
| 3 | Abfrage- und Manipulationssprachen | 61 |
| 3.1 | Benutzung einer Datenbank | 61 |
| 3.2 | Grundlagen der Relationenalgebra | 63 |
| 3.2.1 | Zusammenstellung der Operatoren | 63 |
| 3.2.2 | Die mengenorientierten Operatoren | 65 |
| 3.2.3 | Die relationenorientierten Operatoren | 68 |



| | | |
|----------|---|------------|
| 3.3 | Relational vollständige Sprachen | 73 |
| 3.4 | Übersicht über relationale Sprachen | 75 |
| 3.4.1 | SQL | 75 |
| 3.4.2 | QUEL | 78 |
| 3.4.3 | QBE | 79 |
| 3.5 | Eingebettete Sprachen | 82 |
| 3.6 | Behandlung von Nullwerten | 83 |
| 3.7 | Datenschutzaspekte | 85 |
| 3.8 | Formulierung von Integritätsbedingungen | 88 |
| 3.9 | Bibliografische Angaben | 91 |
| 4 | Elemente der Systemarchitektur | 93 |
| 4.1 | Wissenswertes über die Systemarchitektur | 93 |
| 4.2 | Übersetzung und Optimierung | 96 |
| 4.2.1 | Erstellen eines Anfragebaumes | 96 |
| 4.2.2 | Optimierung durch algebraische Umformung | 98 |
| 4.2.3 | Berechnen des Verbundoperators | 101 |
| 4.3 | Mehrbenutzerbetrieb | 104 |
| 4.3.1 | Der Begriff der Transaktion | 104 |
| 4.3.2 | Serialisierbarkeit | 106 |
| 4.3.3 | Pessimistische Verfahren | 109 |
| 4.3.4 | Optimistische Verfahren | 113 |
| 4.4 | Speicher- und Zugriffsstrukturen | 115 |
| 4.4.1 | Baumstrukturen | 115 |
| 4.4.2 | Hash-Verfahren | 118 |
| 4.4.3 | Mehrdimensionale Datenstrukturen | 121 |
| 4.5 | Fehlerbehandlung | 124 |
| 4.6 | Die Systemarchitektur im Detail | 126 |
| 4.7 | Bibliografische Angaben | 128 |
| 5 | Integration und Migration von Datenbanken | 131 |
| 5.1 | Zur Nutzung heterogener Datenbestände | 131 |
| 5.2 | Datenbanken im Web | 132 |
| 5.2.1 | Aufbau eines webbasierten Informationssystems | 133 |
| 5.2.2 | XML-Dokumente und XML-Schemas | 134 |
| 5.2.3 | Die Abfragesprache XQuery | 137 |
| 5.3 | Abbildungsregeln für Integration und Migration | 139 |
| 5.3.1 | Abbildungen für einfache Entitätsmengen und Wiederholungsgruppen | 139 |
| 5.3.2 | Abbildungen für abhängige Entitätsmengen | 142 |
| 5.3.3 | Indirekte Abbildungen für die Datenintegration und -migration | 145 |
| 5.4 | Migrationsvarianten für heterogene Datenbanken | 147 |
| 5.4.1 | Charakterisierung unterschiedlicher Migrationsvarianten | 148 |
| 5.4.2 | Systemkonforme Spiegelung von Datenbanken | 150 |
| 5.5 | Grundsätze der Integrations- und Migrationsplanung | 153 |

| | | |
|----------|--|------------|
| 5.6 | Bibliografische Angaben | 156 |
| 6 | Postrelationale Datenbanksysteme | 159 |
| 6.1 | Weiterentwicklung – weshalb und wohin? | 159 |
| 6.2 | Verteilte Datenbanken | 160 |
| 6.3 | Temporale Datenbanken | 165 |
| 6.4 | Objektrelationale Datenbanken | 169 |
| 6.5 | Multidimensionale Datenbanken | 173 |
| 6.6 | Fuzzy Datenbanken | 178 |
| 6.7 | Wissensbasierte Datenbanken | 184 |
| 6.8 | Literatur zur Forschung und Entwicklung | 188 |
| | Repetitorium | 191 |
| | Tutorium in SQL | 201 |
| | Eine Datenbank mit Access erstellen | 217 |
| | Glossar | 239 |
| | Fachbegriffe englisch/deutsch | 245 |
| | Literaturverzeichnis | 249 |
| | Stichwortverzeichnis | 259 |

1 Der Weg zum Datenmanagement

1.1

Der Untersuchungsgegenstand der Wirtschaftsinformatik

Der Wandel von der Industrie- zur Informations- und Wissensgesellschaft spiegelt sich in der Bewertung der Information als Produktionsfaktor. *Information* (engl. *information*) hat im Gegensatz zu materiellen Wirtschaftsgütern folgende Eigenschaften:

*Information als
Produktionsfaktor*

- Darstellung: Information wird durch Zeichen, Signale, Nachrichten oder Sprachelemente spezifiziert.
- Verarbeitung: Information kann mit der Hilfe von Algorithmen (Berechnungsvorschriften) übermittelt, gespeichert, klassifiziert, aufgefunden und in andere Darstellungsformen transformiert werden.
- Alter: Information unterliegt keinem physikalischen Alterungsprozess.
- Original: Information ist beliebig kopierbar und kennt keine Originale.
- Träger: Information benötigt keinen fixierten Träger, d.h., sie ist unabhängig vom Ort.

Diese Eigenschaften belegen, dass sich digitale Güter (Informationen, Software, Multimedia etc.) in der Handhabung sowie in der ökonomischen und rechtlichen Wertung von materiellen Gütern stark unterscheiden. Beispielsweise verlieren Produkte durch die Nutzung meistens an Wert, gegenseitige Nutzung von Informationen hingegen entspricht einem Wertzuwachs. Ein weiterer Unterschied besteht darin, dass materielle Güter mit mehr oder weniger hohen Kosten hergestellt werden, die Verfielfältigung von Informationen jedoch

*Unterschiede
zwischen
digitalen und
materiellen
Gütern*

einfach und kostengünstig ist (Rechenaufwand, Material des Informationsträgers). Dies wiederum führt dazu, dass die Eigentumsrechte und Besitzverhältnisse schwer zu bestimmen sind, obwohl man digitale Wasserzeichen und andere Datenschutz- und Sicherheitsmechanismen zur Verfügung hat.

Fasst man die *Information als Produktionsfaktor* im Unternehmen auf, so hat das wichtige Konsequenzen:

- Informationen bilden Entscheidungsgrundlagen und sind somit in allen Organisationsfunktionen von Bedeutung.
- Informationen können aus unterschiedlichen Quellen zugänglich gemacht werden; die Qualität der Information ist von der Verfügbarkeit, Korrektheit und Vollständigkeit abhängig.
- Durch das Sammeln, Speichern und Verarbeiten von Informationen fallen Aufwände und Kosten an.
- Aufgabengebiete jeder Organisation sind durch Informationsbeziehungen miteinander verknüpft, die Erfüllung ist damit von hohem Maße vom Integrationsgrad der Informationsfunktion abhängig.

Bedeutung des Informationsmanagements

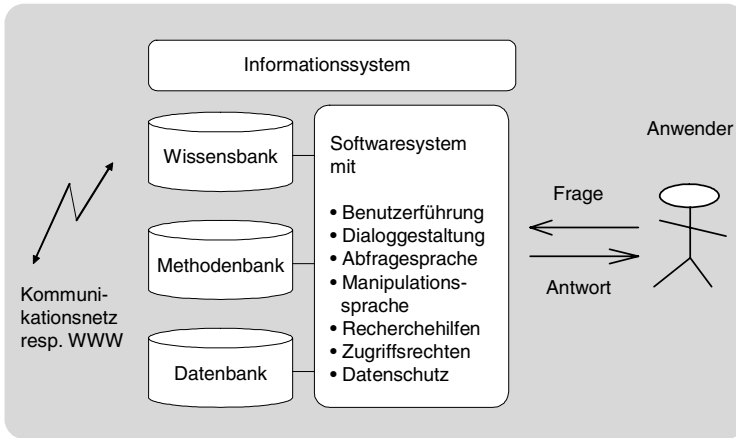
Ist man bereit, die Information als Produktionsfaktor zu betrachten, muss diese Ressource geplant, gesteuert, überwacht und kontrolliert werden. Damit ergibt sich die Notwendigkeit, das Informationsmanagement als Führungsaufgabe wahrzunehmen. Dies bedeutet einen grundlegenden Wechsel im Unternehmen: Neben einer technisch orientierten Funktion wie Betrieb der Informatikmittel (Produktion) muss die Planung und Gestaltung der Informationsfunktion ebenso wahrgenommen werden.

Untersuchungsgegenstand der Wirtschaftsinformatik

Der Untersuchungsgegenstand der *Wirtschaftsinformatik* (engl. *information management*) sind rechnergestützte Informationssysteme. Das Fachgebiet beschäftigt sich mit der *Konzeption, Entwicklung, Einführung und Nutzung betrieblicher Informations- und Kommunikationssysteme*.

Dialog mit dem Anwender

Ein rechnergestütztes *Informationssystem* (engl. *information system*) erlaubt dem Anwender gemäß Abbildung 1-1, Fragen zu stellen und Antworten zu erhalten. Je nach Art des Informationssystems sind hier Fragen zu einem begrenzten Anwendungsbereich zulässig. Darüber hinaus existieren offene Informationssysteme im World Wide Web, die beliebige Anfragen mit der Hilfe eines Browsers bearbeiten können (siehe dazu Abschnitt 5.2). In Abbildung 1-1 ist das rechnergestützte Informationssystem mit einem Kommunikationsnetz resp. mit dem Internet verbunden, um webbasierte Informationssysteme in die Recherchearbeiten einzubeziehen.



*Abb. 1-1
Architektur und
Komponenten
eines
Informationssystems*

Ein Informations- oder Datenbanksystem besteht prinzipiell aus einer Speicherkomponente und einer Softwarekomponente (vgl. Abschnitt 1.3). Die Speicherkomponente umfasst nicht nur Daten, sondern kann Verfahren (Methoden) betreffen. Bei bestimmten Typen von Informationssystemen ist es möglich, mit Hilfe spezifischer Verfahren (Inferenzmaschine) noch nicht bekannte Sachverhalte aus den Datensammlungen zu extrahieren. In einem solchen Anwendungsfall spricht man von einer Wissensbank resp. von einem wissensbasierten Informationssystem (vgl. Abschnitt 6.7).

*Zur
Speicherungs-
komponente*

Die Softwarekomponente eines Informationssystems enthält eine Abfrage- und Manipulationssprache, um die Daten und Informationen auswerten und verändern zu können. Dabei wird der Anwender mit einer Dialogkomponente geführt, die Hilfestellungen (Helpdienst) und Erklärungen anbietet. Die Softwarekomponente bedient nicht nur die Benutzerschnittstelle, sondern verwaltet auch Zugriffs- und Bearbeitungsrechte der Anwender.

*Die Sprach-
schnittstelle*

Ein relationales Datenbanksystem ist ein Informationssystem, bei dem die Daten und Datenbeziehungen ausschließlich in Tabellen abgelegt werden (Abschnitt 1.2). Zudem ist bei einem solchen System die Sprache eine relationenorientierte Abfrage- und Manipulationssprache, wie sie in Abschnitt 1.3 erläutert wird.

1.2 Grundbegriffe des Relationenmodells

*Tabellarische
Form der
Informations-
darstellung*

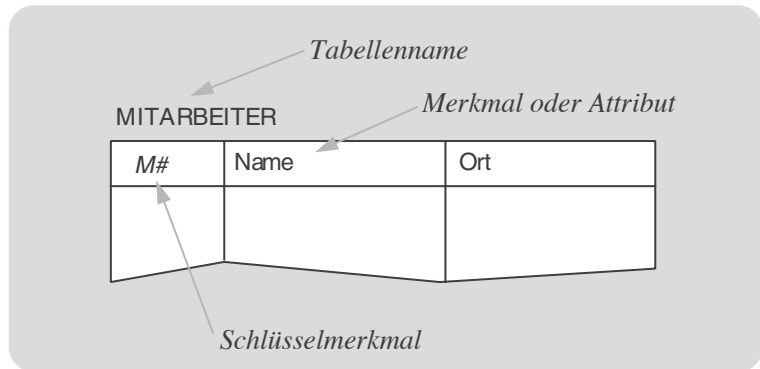
Eine einfache und anschauliche Form Daten oder Informationen zu sammeln oder darzustellen, ist die der Tabelle. Von jeher sind wir es gewohnt, tabellarische Datensammlungen ohne Interpretationshilfen zu lesen und zu verstehen.

Möchten wir Informationen über Mitarbeiter sammeln, so können wir ein Tabellengerüst gemäß Abb. 1-2 entwerfen. Der in Grossbuchstaben geschriebene Tabellename MITARBEITER bezeichnet die Tabelle selbst. Die einzelnen Tabellenspalten werden mit den gewünschten Merkmals- oder Attributnamen überschrieben; in unserem Beispiel sind dies die Mitarbeiternummer «M#», der Mitarbeitername «Name» und der Wohnort «Ort» des Mitarbeiters.

*Merkmale oder
Attribute sind
Eigenschaften*

Ein *Merkmal* oder *Attribut* (engl. *attribute*) ordnet jedem Eintrag in der Tabelle einen bestimmten Datenwert aus einem vordefinierten *Wertebereich* (engl. *domain*) als Eigenschaft zu. In der Tabelle MITARBEITER ermöglicht das Merkmal M# das eindeutige Identifizieren

Abb. 1-2
Tabellengerüst
für eine Tabelle
MITARBEITER



der Mitarbeiter. Aufgrund dieser Eigenschaft erklären wir die Mitarbeiternummer zum Schlüssel. Zur Verdeutlichung der Schlüsseleigenschaft werden die Schlüsselmerkmale im Folgenden kursiv im Tabellenkopf¹ angeschrieben. Mit dem Merkmal Ort werden die dazugehörigen Ortsnamen, mit dem Merkmal Name die entsprechenden Mitarbeiternamen bezeichnet.

*Eine Tabelle ist
eine Menge von
Datensätzen*

Ohne weiteres lassen sich nun die gewünschten Daten der Mitarbeiter in die Tabelle MITARBEITER zeilenweise eintragen (vgl. Abb. 1-3). Dabei können gewisse Datenwerte mehrfach in der Tabelle

¹ In einigen Standardwerken der Datenbankliteratur werden die Schlüsselmerkmale durch Unterstreichung kenntlich gemacht.

erscheinen. So bemerken wir in der Tabelle MITARBEITER, dass der Wohnort Liestal zweimal vorkommt. Dieser Sachverhalt ist wesentlich und sagt aus, dass sowohl der Mitarbeiter Becker als auch der Mitarbeiter Meier in Liestal wohnen. In der Tabelle MITARBEITER können nicht nur Ortsbezeichnungen mehrfach vorkommen, sondern auch Mitarbeiternamen. Aus diesen Gründen ist das bereits erwähnte Schlüsselmerkmal *M#* notwendig, das jeden Mitarbeiter in der Tabelle eindeutig bestimmt.

Ein *Identifikationsschlüssel* oder *Schlüssel* (engl. *identification key*) einer Tabelle ist ein Merkmal oder eine minimale Merkmalskombination, wobei innerhalb der Tabelle die Schlüsselwerte die Datensätze (genannt *Zeilen* oder *Tupel*) eindeutig identifizieren. Aus dieser Kurzdefinition lassen sich zwei wichtige *Schlüsseigenschaften* herleiten:

Was ist ein Identifikationsschlüssel?

MITARBEITER

| M# | Name | Ort |
|-----|-----------|-------------|
| M19 | Schweizer | Frenkendorf |
| M4 | Becker | Liestal |
| M1 | Meier | Liestal |
| M7 | Huber | Basel |

Abb.1-3
Tabelle
MITARBEITER
mit
Ausprägungen

- Jeder Schlüsselwert identifiziert eindeutig einen Datensatz innerhalb der Tabelle, d.h., verschiedene Tupel dürfen keine identischen Schlüssel aufweisen (*Eindeutigkeit*).
- Falls der Schlüssel eine echte Kombination von Merkmalen darstellt, muss diese minimal sein. Mit anderen Worten: Kein Merkmal der Kombination kann gestrichen werden, ohne dass die Eindeutigkeit der Identifikation verlorengeht (*Minimalität*).

Eindeutigkeit und Minimalität sind gefordert

Mit den beiden Forderungen nach Eindeutigkeit und Minimalität ist ein Schlüssel vollständig charakterisiert.

Vorsicht bei der
Festlegung eines
Schlüssels

Anstelle eines natürlichen Merkmals oder einer natürlichen Merkmalskombination kann ein Schlüssel als künstliches Merkmal eingeführt werden. Die Mitarbeiternummer M# aus unserem Beispiel ist künstlich, weil sie keine natürliche Eigenschaft der Mitarbeiter darstellt. Aus ideellen Gründen sträuben wir uns zwar oft dagegen, *künstliche Schlüssel* oder «Nummern» als identifizierende Merkmale vorzusehen, vor allem wenn es sich um personenbezogene Informationen handelt. Auf der anderen Seite führen natürliche Schlüssel oder Schlüsselkombinationen nicht selten zu Datenschutzproblemen. Als Beispiel sei die in der Schweiz übliche Alters- und Hinterbliebenen-Versicherungsnummer (AHV-Nummer) genannt, die unter anderem das Geburtsdatum jeder Person enthält. Es fragt sich, ob die auf vielen Ausweisen und Dokumenten abgedruckte AHV-Nummer mit ihren personenbezogenen Bestandteilen als Identifikationsmerkmal vom Datenschutz her vernünftig ist.

Künstliche
Schlüssel
bewähren sich

Ein künstlicher Schlüssel sollte aufgrund obiger Überlegungen *anwendungsneutral und ohne Semantik* (Aussagekraft, Bedeutung) definiert werden. Sobald aus den Datenwerten eines Schlüssels irgendwelche Sachverhalte abgeleitet werden können, besteht ein Interpretationsspielraum. Auch kann es vorkommen, dass sich die ursprünglich wohldefinierte Bedeutung eines Schlüsselwertes im Laufe der Zeit ändert oder verlorenght. So wird bei «sprechenden» Schlüsseln die Forderung nach Eindeutigkeit auf einmal verletzt.

Tabellendefinition

Zu den
Eigenschaften
einer Relation

Zusammenfassend verstehen wir unter einer *Tabelle* oder *Relation* (engl. *table*, *relation*) eine Menge von Tupeln, die tabellenförmig dargestellt werden und folgende Anforderungen erfüllen:

- Eine Tabelle besitzt einen eindeutigen Tabellennamen.
- Innerhalb der Tabelle ist jeder Merkmalsname eindeutig und bezeichnet eine bestimmte Spalte mit der gewünschten Eigenschaft.
- Die Anzahl der Merkmale ist beliebig, die Ordnung der Spalten innerhalb der Tabelle ist bedeutungslos.
- Eines der Merkmale oder eine Merkmalskombination identifiziert eindeutig die Tupel innerhalb der Tabelle und wird als Primärschlüssel bezeichnet.
- Die Anzahl der Tupel einer Tabelle ist beliebig, die Ordnung der Tupel innerhalb der Tabelle ist bedeutungslos.

Beim *Relationenmodell* (engl. *relational model*) wird gemäß obiger Definition jede Tabelle als *Menge ungeordneter Tupel* aufgefasst. Zu

beachten ist, dass aufgrund dieses Mengenbegriffs in einer Tabelle ein und dasselbe Tupel nur einmal vorkommen darf.

1.3

Die international standardisierte Sprache SQL

Wie wir gesehen haben, stellt das Relationenmodell Informationen in Form von Tabellen dar. Dabei entspricht jede Tabelle einer Menge von Tupeln oder Datensätzen desselben Typs. Dieses Mengenkonzept erlaubt grundsätzlich, *Abfrage- und Manipulationsmöglichkeiten mengenorientiert* anzubieten. Das Resultat einer Selektionsoperation ist eine Menge, d.h., jedes Ergebnis eines Suchvorgangs wird vom Datenbanksystem als Tabelle zurückgegeben. Falls keine Tupel der durchsuchten Tabelle die geforderten Eigenschaften erfüllen, erhält der Anwender eine leere Resultattabelle. Änderungsoperationen wirken ebenfalls auf eine Tabelle oder auf einzelne Tabellenbereiche.

Die wichtigste Abfrage- und Manipulationssprache für Tabellen heißt *Structured Query Language* oder abgekürzt SQL (vgl. Abb. 1-4). Diese Sprache wurde durch das ANSI (American National Standards Institute) und durch die ISO (International Organization for Standardization) genormt.²

Die Sprache SQL genügt einem allgemeinen Grundmuster, das wir anhand der in Abb. 1-4 aufgeführten Abfrage illustrieren:

«Selektiere (SELECT) das Merkmal Name
aus (FROM) der Tabelle MITARBEITER,
wobei (WHERE) der Wohnort Liestal ist!»

Der Ausdruck SELECT-FROM-WHERE wirkt auf eine oder mehrere Tabellen und erzeugt als Resultat immer eine Tabelle. Auf unser Beispiel bezogen erhalten wir für obige Abfrage eine Resultattabelle mit den gewünschten Namen Becker und Meier.

Diese mengenorientierte Arbeitsweise unterscheidet SQL prinzipiell von nicht-relationalen Datenbanksprachen. Hier liegt denn auch ein wesentlicher Vorteil für den Anwender, da eine einzige SQL-Abfrage eine ganze Reihe von Aktionen im Datenbanksystem auslösen kann. So ist es nicht notwendig, dass der Anwender die Suchvorgänge selbst «ausprogrammiert». Ein relationales Datenbanksystem nimmt ihm diese Arbeit ab.

Mengen-orientierte Datenbanksprache

Die Sprache SQL ist zertifiziert

Jede Abfrage generiert eine Resultattabelle

SQL ist mengenorientiert

² Das ANSI ist der nationale Normenausschuss der USA und entspricht dem DIN (Deutsches Institut für Normung) in Deutschland. Der ISO gehören die nationalen Normenausschüsse an.

Abb.1-4
Formulierung
einer Abfrage mit
SQL

MITARBEITER

| M# | Name | Ort |
|-----|-----------|-------------|
| M19 | Schweizer | Frenkendorf |
| M4 | Becker | Liestal |
| M1 | Meier | Liestal |
| M7 | Huber | Basel |

Beispiel einer Abfrage

«Selektiere die Namen der Mitarbeiter, die in Liestal wohnen»

Formulierung in SQL

SELECT Name
FROM MITARBEITER
WHERE Ort = 'Liestal'

Resultattabelle

| Name |
|--------|
| Becker |
| Meier |

Das WAS zählt,
nicht das WIE

Relationale Abfrage- und Manipulationssprachen sind deskriptiv, also beschreibend. Allein durch das Festlegen der gesuchten Eigenschaften erhält der Anwender die gewünschten Informationen. Eine Anleitung zur Berechnung der resultierenden Datensätze muss von ihm nicht spezifiziert werden. Das Datenbanksystem übernimmt diese Aufgabe, bearbeitet die Abfrage oder die Manipulation mit eigenen Such- und Zugriffsmethoden und erstellt die gewünschte Resultattabelle.

Im Gegensatz zu den deskriptiven Abfrage- oder Manipulations-sprachen müssen bei den herkömmlichen *prozeduralen Datenbank-sprachen* durch den Anwender selbst die Abläufe zur Bereitstellung der gesuchten Informationen ausprogrammiert werden. Dabei ist das Ergebnis jeder Abfrageoperation ein einzelner Datensatz und nicht eine Menge von Tupeln.

Das Navigieren in
Datenbeständen
übernimmt das
System

Bei der deskriptiven Formulierung einer Abfrage beschränkt sich SQL auf die Angabe der gewünschten Selektionsbedingung in der WHERE-Klausel, bei den prozeduralen Sprachen hingegen muss ein Algorithmus zum Auffinden der einzelnen Datensätze vom Anwen-der spezifiziert werden. Lehnen wir uns an bekannte Datenbankspra-chen von hierarchischen Datenbanksystemen, so suchen wir gemäß Abb. 1-5 zuerst mit GET_FIRST einen ersten Datensatz, der das gewünschte Suchkriterium erfüllt. Anschließend lesen wir sämtliche

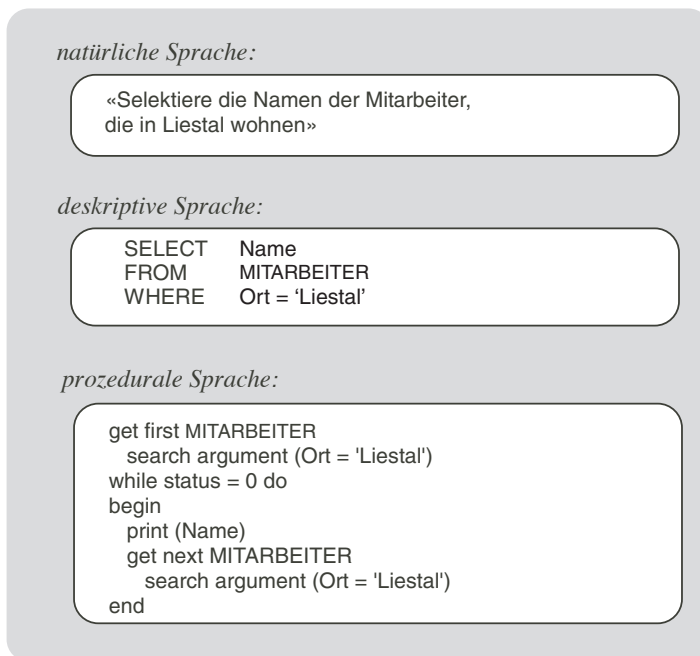
Datensätze durch GET_NEXT-Befehle, bis wir das Ende der Datei oder eine nächste Hierarchiestufe innerhalb der Datenbank erreichen.

Bei den prozeduralen Datenbanksprachen stellen wir zusammenfassend fest, dass sie satzorienteerte oder navigierende Befehle für das Bearbeiten von Datensammlungen verlangen. Dieser Sachverhalt setzt vom Anwendungsentwickler einigen Sachverstand und Kenntnis der inneren Struktur der Datenbank voraus. Zudem kann ein gelegentlicher Benutzer eine Datenbank nicht selbstständig auswerten. Im Gegensatz zu den prozeduralen Sprachen müssen bei relationalen Abfrage- und Manipulationssprachen keine Zugriffspfade, Verarbeitungsabläufe oder Navigationswege spezifiziert werden. Dadurch wird der Entwicklungsaufwand für Datenbankauswertungen wesentlich reduziert.

Möchte man Datenbankabfragen und -auswertungen von den Fachabteilungen oder von den Endbenutzern selbst durchführen lassen, so kommt dem deskriptiven Ansatz eine große Bedeutung zu. Untersuchungen deskriptiver Datenbankschnittstellen haben offengelegt, dass auch ein *gelegentlicher Benutzer eine echte Chance* hat, mit Hilfe deskriptiver Sprachelemente seine gewünschten Auswertungen selbstständig durchführen zu können. Aus Abb. 1-5 ist zudem ersichtlich, dass die Sprache SQL der natürlichen Sprache nahesteht. So existieren heute relationale Datenbanksysteme, die über einen natürlich-sprachlichen Zugang verfügen.

*Die physische
Datenstruktur
bleibt verborgen*

*Der
gelegentliche
Anwender
bekommt eine
Chance*



*Abb. 1-5
Unterschied
zwischen
deskriptiven und
prozeduralen
Sprachen*

1.4

Die Komponenten eines relationalen Datenbanksystems

Ted Codd ist der Begründer des Relationenmodells

Das Relationenmodell wurde Anfang der siebziger Jahre durch die Arbeiten von Edgar Frank Codd begründet. Daraufhin entstanden in Forschungslabors erste relationale Datenbanksysteme, die SQL oder ähnliche Datenbanksprachen unterstützten. Ausgereifere Produkte haben inzwischen die Praxis erobert.

Was ist ein relationales DBMS?

Ein *relationales Datenbankmanagementsystem* (engl. *relational database management system*), abgekürzt RDBMS und oft vereinfachend relationales Datenbanksystem genannt, ist gemäß Abb. 1-6 ein integriertes System zur einheitlichen Verwaltung relationaler Datenbanken. Neben Dienstfunktionen stellt ein RDBMS eine deskriptive Sprache für Datenbeschreibungen und Datenmanipulationen zur Verfügung.

Aufgaben der Speicherungs- und Verwaltungskomponenten

Jedes relationale Datenbanksystem besteht aus einer Speicherungs- und einer Verwaltungskomponente (vgl. Abb. 1-6): Die *Speicherungskomponente* dient dazu, sowohl Daten als auch Beziehungen zwischen ihnen lückenlos in Tabellen abzulegen. Neben Tabellen mit Benutzerdaten aus unterschiedlichen Anwendungen existieren vordefinierte Systemtabellen, die beim Betrieb der Datenbanken benötigt werden. Diese enthalten Beschreibungsinformationen und lassen sich vom Anwender jederzeit abfragen, nicht aber verändern. Die *Verwaltungskomponente* enthält als wichtigsten Bestandteil eine relationale Datendefinitions- und Datenmanipulationssprache. Daneben umfasst diese Sprache auch Dienstfunktionen für die Wiederherstellung von Datenbeständen nach einem Fehlerfall, zum Datenschutz und zur Datensicherung.

Die Eigenschaften eines relationalen Datenbanksystems lassen sich wie folgt zusammenfassen:

Relationale DBMS sind formal abgesichert

- Ein RDBMS dient der strukturierten Datenorganisation und hat eine *klare formale Grundlage*. Alle Informationen werden in Tabellen abgespeichert, wobei Abhängigkeiten zwischen den Merkmalswerten einer Tabelle oder mehrfach vorkommende Sachverhalte aufgedeckt werden können. Diese formalen Instrumente ermöglichen einen widerspruchsfreien Datenbankentwurf und garantieren saubere Datenstrukturen.

Relationale Sprachen sind mengenorientiert

- Ein RDBMS unterstützt eine *mengenorientierte Sprachschnittstelle* zur Datendefinition, -selektion und -manipulation. Die Sprachkomponente ist deskriptiv und entlastet den Anwender bei Auswertungen oder bei Programmierertätigkeiten. Bei dieser deskriptiven Sprachschnittstelle gibt der Anwender aufgrund

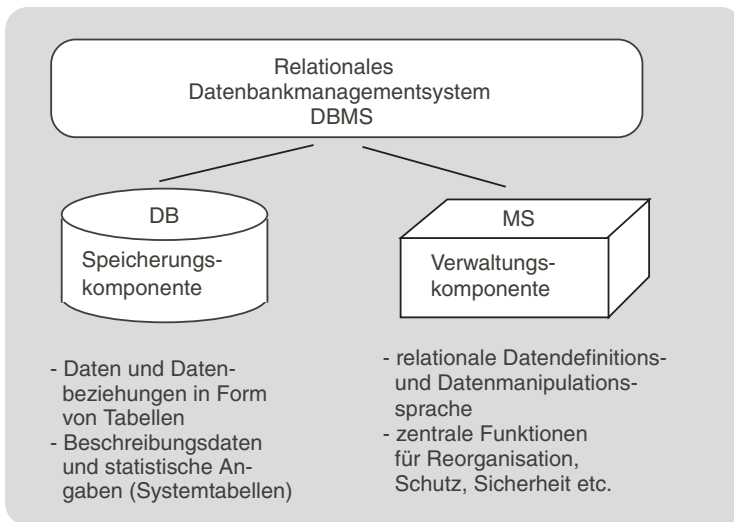


Abb.1-6
Die zwei
Komponenten
eines
relationalen
Datenbank-
systems

seiner Vorstellungen eine Selektionsbedingung an. Das Durchsuchen der Datenbank und das Bereitstellen der Resultattabelle hingegen übernimmt das Datenbanksystem.

- Ein RDBMS gewährleistet eine große *Datenunabhängigkeit*, d.h., Daten und Anwendungsprogramme bleiben weitgehend voneinander getrennt. Diese Unabhängigkeit ergibt sich aus der Tatsache, dass die eigentliche Speicherkomponente eines RDBMS von der Anwenderseite durch eine Verwaltungskomponente entkoppelt ist. Im Idealfall können physische Änderungen in den relationalen Datenbanken vorgenommen werden, ohne dass die entsprechenden Anwendungsprogramme anzupassen sind.
- Ein RDBMS ermöglicht und unterstützt den *Mehrbenutzerbetrieb*, d.h., es können mehrere Benutzer gleichzeitig ein und dieselbe Datenbank abfragen oder bearbeiten. Das relationale Datenbanksystem muss also dafür sorgen, dass parallel ablaufende Aktionen auf einer Datenbank sich nicht gegenseitig behindern oder gar die Korrektheit der Daten beeinträchtigen.
- Ein RDBMS stellt Hilfsmittel zur *Gewährleistung der Datenintegrität* bereit. Unter Datenintegrität versteht man die fehlerfreie und korrekte Speicherung der Daten sowie ihren Schutz vor Zerstörung, vor Verlust, vor unbefugtem Zugriff und Missbrauch.

*Unabhängigkeit
zwischen Daten-
organisation und
Anwendungs-
programmen*

*Mehrere
Benutzer können
gleichzeitig
arbeiten*

*Datenkonsistenz
und -integrität
bleiben gewahrt*

*Relationale
DBMS
dominieren den
Markt*

Nicht-relationale Datenbanksysteme erfüllen solche Eigenschaften nur teilweise. Aus diesem Grunde haben die relationalen Datenbanksysteme in den letzten Jahren einen Durchbruch auf dem Markt erzielt; ein Ende dieser erfolgreichen Entwicklung ist zur Zeit nicht abzusehen. Relationale Datenbanksysteme legen auch auf dem Gebiet der *Leistungsfähigkeit oder Performanz* (engl. *performance*) von Jahr zu Jahr zu, obwohl die mengenorientierte Verarbeitung ihren Preis hat. Erfahrungen aus der Betriebspraxis mit relationalen Datenbanksystemen befruchten wiederum neuere Entwicklungen auf dem Gebiet verteilter oder wissensbasierter Daten- und Methodenbanken. Überdies wurzeln viele Forschungs- und Entwicklungsarbeiten zu Datenbanken in der relationalen Datenbanktheorie (vgl. Kapitel 6).

1.5 Zur Organisation des Datenbankeinsatzes

*Zum Produktions-
faktor Information*

Viele Firmen und Institutionen betrachten ihre Datenbestände als *unentbehrliche Ressource*. Sie pflegen und unterhalten zu Geschäftszwecken nicht nur ihre eigenen Daten, sondern schließen sich mehr und mehr an öffentlich zugängliche Datensammlungen an. Die weltweite Zunahme und das stetige Wachstum der Informationsanbieter mit ihren Dienstleistungen rund um die Uhr illustriert den Stellenwert webbasierter Datenbestände.

*Ein betriebliches
Daten-
management ist
gefordert*

Die Bedeutung aktueller und realitätsbezogener Information hat einen direkten Einfluss auf die Ausgestaltung des Informatikbereiches. So sind vielerorts Stellen des Datenmanagements entstanden, um die datenbezogenen Aufgaben und Pflichten bewusster angehen zu können. Ein zukunftsgerichtetes Datenmanagement befasst sich sowohl strategisch mit der Informationsbeschaffung und -bewirtschaftung als auch operativ mit der effizienten Bereitstellung und Auswertung von aktuellen und konsistenten Daten.

*Zur Langlebigkeit
der Daten-
bestände*

Aufbau und Betrieb eines Datenmanagements verursachen beträchtliche Kosten mit anfänglich nur schwer messbarem Nutzen. Es ist nämlich nicht einfach, klar strukturierte Datenmodelle, widerspruchsfreie und für jedermann verständliche Datenbeschreibungen, saubere und konsistente Datenbestände, griffige Sicherheitskonzepte, aktuelle Auskunftsbereitschaft und anderes mehr eindeutig zu bewerten und somit aussagekräftig in Wirtschaftlichkeitsüberlegungen einzubeziehen. Erst ein allmähliches Bewusstwerden von Bedeutung und Langlebigkeit der Daten relativiert für das Unternehmen die notwendigen Investitionen.

Um den Begriff *Datenmanagement* (engl. *data management*) besser fassen zu können, sollte das Datenmanagement zunächst in seine Aufgabenbereiche Datenarchitektur, Datenadministration, Datentechnik und Datennutzung aufgegliedert werden. Die Abb. 1-7 charakterisiert diese vier Teilgebiete des Datenmanagements mit ihren Zielen und Instrumenten.

Aufgaben und Pflichten des Datenmanagements

| | Ziele | Werkzeuge |
|---------------------|---|---|
| Datenarchitektur | Formulieren und Pflegen des unternehmensweiten Datenmodells, Unterstützen der Anwendungsentwicklung bei der Datenmodellierung | Datenanalyse und Entwurfsmethodik, Werkzeuge für die rechnergestützte Datenmodellierung |
| Datenadministration | Verwalten von Daten und Funktionen anhand von Standardisierungsrichtlinien und internationalen Normen, Beraten von Entwicklern und Endbenutzern | Data-Dictionary-Systeme, Werkzeuge für den Verwendungsnachweis |
| Datentechnik | Installieren, Reorganisieren und Sicherstellen von Datenbanken, Durchführen von Datenbankrestaurierungen nach einem Fehlerfall | Datenbankverwaltungssysteme, Hilfsmittel zur Wiederherstellung von Datenbanken und zur Leistungsoptimierung |
| Datennutzung | Bereitstellen von Auswertungs- und Reportfunktionen unter Berücksichtigung des Datenschutzes resp. der Dateneignerschaft | Sprache für Datenbankabfragen und -manipulationen, Reportgeneratoren |

*Abb. 1-7
Die vier Eckpfeiler des Datenmanagements*

Die *Datenarchitektur* analysiert, klassifiziert und strukturiert mit ausgefeilter Methodik die Unternehmensdaten. Neben der eigentlichen Analyse der Daten- und Informationsbedürfnisse müssen die wichtigsten Datenklassen und ihre gegenseitigen Beziehungen untereinander in Datenmodellen unterschiedlichster Detaillierung festgehalten werden. Diese aus der Abstraktion der realen Gegebenheiten entstandenen und aufeinander abgestimmten Datenmodelle bilden die Basis der Datenarchitektur.

Die Datenarchitektur ist grundlegend und strategisch

Die *Datenadministration* bezweckt, die Datenbeschreibungen und die Datenformate sowie deren Verantwortlichkeiten einheitlich zu erfassen und zu kontrollieren, um eine anwendungsübergreifende Nutzung der langlebigen Unternehmensdaten zu gewährleisten. Beim heutigen Trend zu einer dezentralen Datenhaltung auf intelligenten Arbeitsplatzrechnern oder auf verteilten Abteilungsrechnern

Begriffe und Formate müssen administriert werden

kommt der Datenadministration eine immer größere Verantwortung bei der Pflege der Daten und bei der Vergabe von Berechtigungen zu.

*Die technische
Herausforderung
des Daten-
managements*

Die Spezialisten der *Datentechnik* installieren, überwachen und reorganisieren Datenbanken und stellen diese in einem mehrstufigen Verfahren sicher. Dieser Fachbereich, oft auch Datenbanktechnik oder Datenbankadministration genannt, ist zudem für das so genannte Technologiemanagement verantwortlich, da Erweiterungen in der Datenbanktechnologie immer wieder berücksichtigt und bestehende Methoden und Werkzeuge laufend verbessert werden müssen.

*Das Information
Center
organisiert die
Datennutzung*

Der vierte Eckpfeiler des Datenmanagements, die *Datennutzung*, ermöglicht die eigentliche Bewirtschaftung der Unternehmensdaten. Mit einem besonderen Benutzerservice, den unter Umständen auch so genannte Information Centers anbieten, werden die Fachabteilungen gezielt angeleitet, ihre eigenen Datenbestände selbstständig zu pflegen, nachzuführen und auszuwerten.

Somit ergibt sich nun von der Charakterisierung der datenbezogenen Aufgaben und Pflichten her gesehen für das Datenmanagement folgende Definition:

*Definition des
Daten-
managements*

Unter dem Datenmanagement fasst man alle betrieblichen und technischen Funktionen der Datenarchitektur, der Datenadministration und der Datentechnik zusammen, die der unternehmensweiten Datenhaltung, Datenpflege und Datennutzung dienen.

*Kein
Informations-
management
ohne Daten-
management*

Die hier vorgeschlagene Begriffsbildung umfasst technische wie betriebliche Funktionen. Dies bedeutet allerdings nicht zwangsläufig, dass in der Aufbauorganisation eines Unternehmens die Funktionen der Datenarchitektur, der Datenadministration, der Datentechnik und des Benutzerservice in einer einzigen Organisationseinheit zusammengezogen werden müssen.

1.6 Bemerkungen zur Literatur

*Standardwerke
zu Datenbank-
systemen*

Es gibt eine große Anzahl von Standardwerken zu dem Gebiet der Datenbanken, woran sich auch die Bedeutung dieses Informatikbereiches ablesen lässt. Einige Lehrbücher behandeln nicht nur relationale, sondern auch die heute noch gebräuchlichen hierarchischen und netzwerkartigen Datenbanksysteme. Sehr bekannt sind die Werke von Connolly und Begg (2004), Hoffer et al. (2005) sowie von Date (2004), eher theoretisch ist das Textbuch von Ullman (1982), aufschlussreich dasjenige von Silberschatz et al. (2005) und umfassend das Standardwerk von Elmasri und Navathe (2006). Gardarin und Valduries (1989) geben eine Einführung in die relationale Datenbanktechnologie sowie in das Gebiet der Wissensbanken.

Als deutschsprachige Lehrbücher im Datenbankbereich sind Schlageter und Stucky (1983), Vossen (2000), Wedekind (1981) und Zehnder (2002) zu erwähnen. Die jüngsten Werke von Heuer und Saake (2000), Kemper und Eickler (2006) sowie von Lang und Lockemann (1995) gehen auf Grundlagen und Erweiterungen von Datenbanksystemen ein.

Das eigentliche Fachbuch über betriebliche Aspekte des Datenmanagements stammt von Dippold et al. (2005). Biethahn et al. (2000) widmen in ihrem Band über das Daten- und Entwicklungsmanagement der Datenarchitektur und der Datenadministration mehrere Kapitel, die Einführung von Martin (1986) konzentriert sich auf die Datenbanktechnik, schließt aber Managementüberlegungen mit ein. Das Handbuch der Wirtschaftsinformatik von Kurbel und Strunz (1990) enthält ein Kapitel über das Datenmanagement. Scheer (1991) und Vetter (1998) beschränken sich in ihren Werken auf die Datenarchitektur und erläutern das methodische Werkzeug zur unternehmensweiten Datenmodellierung. Heinrich und Lehner (2005), Martiny und Klotz (1989) sowie Österle et al. (1991) streifen in ihren Werken über das Informationsmanagement auch Themen des Datenmanagements. Außer den erwähnten Fachbüchern existieren einige Artikel; zu erwähnen ist der Aufsatz von Gemünden und Schmitt (1991), dem eine empirische Untersuchung in deutschen Großunternehmen angefügt ist. Meier (1994) charakterisiert in seinem Beitrag die Ziele und Aufgaben des Datenmanagements aus der Sicht des Praktikers. Fragen der Datenadministration werden von Meier und Johnner (1991) sowie von Ortner et al. (1990) aufgegriffen.

Deutschsprachige Lehrbücher im Datenbankbereich

Werke und Forschungsliteratur zum betrieblichen Datenmanagement

2 Schritte zur Datenmodellierung

2.1

Von der Datenanalyse zur Datenbank

Ein *Datenmodell* (engl. *data model*) beschreibt auf strukturierte und formale Weise die für ein Informationssystem notwendigen Daten und Datenbeziehungen. Benötigt man für die Bearbeitung von Informatikprojekten gemäß Abb. 2-1 Informationen über Mitarbeiter, Detailangaben über Projektvorhaben und Auskunft über einzelne Firmenabteilungen, so können in einem entsprechenden Datenmodell die dazu notwendigen Datenklassen (Datenkategorien) bestimmt und in Beziehung zueinander gebracht werden. Das Festlegen von Datenklassen, im Fachjargon Entitätsmengen genannt, und das Bestimmen von Beziehungsmengen geschieht vorläufig noch unabhängig davon, auf welchem Rechner oder Datenbanksystem die Informationen später erfasst, gespeichert und nachgeführt werden. Damit möchte man erreichen, dass Daten und Datenbeziehungen beim Ausbau von Computersystemen oder bei Software-Erweiterungen *vom Anwender aus gesehen stabil* bleiben.

Zur Beschreibung eines Ausschnittes aus der realen Welt bis hin zur Festlegung der eigentlichen Datenbank sind drei wesentliche Schritte notwendig, nämlich die Datenanalyse, der Entwurf eines Entitäten-Beziehungsmodells und dessen Überführung in ein relationales Datenbankschema.

Bei der Datenanalyse geht es darum, zusammen mit dem Benutzer die für das Informationssystem notwendigen Daten und deren Beziehungen samt Mengengerüst zu ermitteln. Nur so lassen sich überhaupt die Systemgrenzen frühzeitig festlegen. Mit Hilfe von Interviews, Bedarfsanalysen, Fragebogenaktionen, Formularsammlungen etc. muss durch ein iteratives Vorgehen eine aussagekräftige Dokumentation zusammengestellt werden. Diese umfasst mindestens sowohl eine verbale Beschreibung des Auftrages mit einer klaren Zielsetzung als auch eine *Liste der relevanten Informationssachver-*

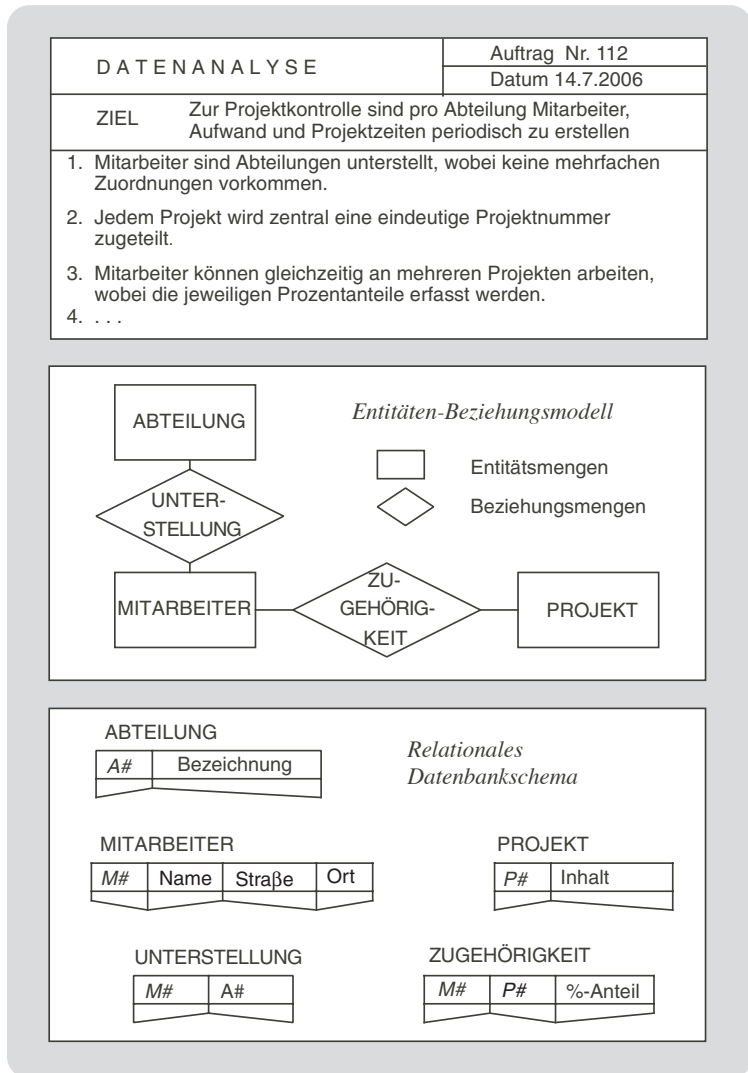
*Ein Datenmodell
abstrahiert die
Wirklichkeit*

*Die drei Schritte
der Daten-
modellierung*

*Erster Schritt:
Analyse der
Daten und
Beziehungen*

halte (vgl. Beispiel in Abb. 2-1). Die verbale Beschreibung der Datenzusammenhänge kann bei der Datenanalyse ergänzt werden durch grafische Darstellungen oder durch ein zusammenfassendes Beispiel. Wichtig ist, dass die Datenanalyse die für den späteren Aufbau einer Datenbank notwendigen Fakten in der Sprache des Anwenders formuliert.

Abb. 2-1
Notwendige
Schritte bei der
Daten-
modellierung



Im nächsten Abstraktionsschritt wird das so genannte *Entitäten-Beziehungsmodell* (engl. *entity relationship model*) entworfen, das neben den Entitätsmengen auch die dazugehörigen Beziehungsmengen angibt. Dieses Modell stellt die Entitätsmengen grafisch durch Rechtecke und die Beziehungsmengen durch Rhomben dar. Aufgrund der Datenanalyse aus Abb. 2-1 erhält man ABTEILUNG, MITARBEITER und PROJEKT¹ als wesentliche Entitätsmengen. Um festzuhalten, in welchen Abteilungen die Mitarbeiter tätig sind und an welchen Projekten sie arbeiten, werden die beiden Beziehungsmengen UNTERSTELLUNG und ZUGEHÖRIGKEIT definiert und grafisch mit den entsprechenden Entitätsmengen verknüpft. Das Entitäten-Beziehungsmodell erlaubt somit, die in der Datenanalyse zusammengestellten Fakten zu strukturieren und anschaulich darzustellen. Dabei darf nicht verschwiegen werden, dass das Erkennen von Entitäts- und Beziehungsmengen sowie das Festlegen der zugehörigen Merkmale nicht immer einfach und eindeutig erfolgen kann. Vielmehr verlangt dieser Entwurfsschritt vom Datenarchitekten einige Übung und praktische Erfahrung.

Das Entitäten-Beziehungsmodell wird nun in ein *relationales Datenbankschema* (engl. *relational database schema*) überführt. Unter einem Datenbankschema versteht man die formale Beschreibung der Datenbankobjekte. Da ein relationales Datenbanksystem als Objekte nur Tabellen zulässt, müssen *sowohl die Entitätsmengen als auch die Beziehungsmengen in Tabellenform* ausgedrückt werden. In Abb. 2-1 ergibt sich deshalb für die Entitätsmengen ABTEILUNG, MITARBEITER und PROJEKT je eine Entitätsmengentabelle. Um die Beziehungen ebenfalls tabellarisch darstellen zu können, definiert man für jede Beziehungsmenge eine eigenständige Tabelle. Solche Beziehungsmengentabellen enthalten immer die Schlüssel der in die Beziehung eingehenden Entitätsmengen als so genannte Fremdschlüssel (vgl. Abschnitt 2.3.1) und allenfalls weitere Beziehungsmerkmale. In unserem Beispiel erscheinen in der Tabelle UNTERSTELLUNG nur die beiden Fremdschlüssel Mitarbeiter- und Abteilungsnummer, in der Tabelle ZUGEHÖRIGKEIT die Fremdschlüssel Mitarbeiter- und Projektnummer nebst dem Zeitaufwand in Prozenten.

Das Durchführen einer Datenanalyse, das Entwickeln eines Entitäten-Beziehungsmodells und das Definieren eines relationalen Datenbankschemas sind hier nur grob umrissen. Wesentlich ist die Erkenntnis, dass ein Datenbankentwurf sinnvollerweise anhand eines

*Zweiter Schritt:
Freilegen von
Entitäts- und
Beziehungsmengen*

*Dritter Schritt:
Entwurf des
relationalen
Datenbankschemas*

*Analyse und
Entwurf erfolgen
unabhängig von
technischen
Details*

¹ In Analogie zur Bezeichnung von Tabellen verwenden wir für die Entitätsmengen und die entsprechenden Beziehungsmengen ebenfalls Großbuchstaben.

Entitäten-Beziehungsmodells entwickelt werden sollte. Dies erlaubt, losgelöst von einem bestimmten Datenbanksystem, mit dem Anwender der Datenmodellierungsaspekte unabhängig festzuhalten und zu diskutieren. Erst in einem weiteren Entwurfsschritt wird das zweckmäßige Datenbankschema definiert, wobei im Falle des Relationenmodells klare Abbildungsregeln existieren (vgl. Abschnitt 2.3).

2.2 Das Entitäten-Beziehungsmodell

2.2.1 Entitäten und Beziehungen

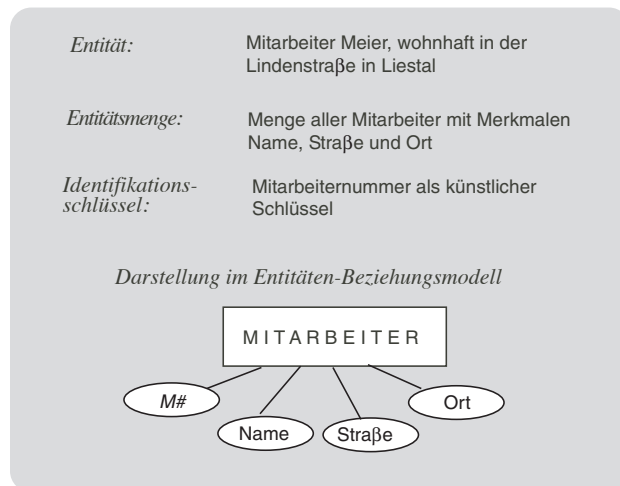
Entitäten sind wohlunterscheidbare Objekte

Unter *Entität* (engl. *entity*) versteht man ein bestimmtes, d.h. von anderen wohlunterscheidbares Objekt der realen Welt oder unserer Vorstellung. Dabei kann es sich um ein Individuum, um einen Gegenstand, um einen abstrakten Begriff oder um ein Ereignis handeln. Entitäten des gleichen Typs werden zu *Entitätsmengen* zusammengefasst und durch Merkmale weiter charakterisiert. Solche sind Eigenschaftskategorien der Entität bzw. der Entitätsmenge wie z.B. Größe, Bezeichnung, Gewicht.

Auszeichnen von Identifikationsschlüsseln

Für jede Entitätsmenge ist ein Identifikationsschlüssel, d.h. ein Merkmal oder eine Merkmalskombination zu bestimmen, der die Entitäten innerhalb der Entitätsmenge eindeutig festlegt. Neben der Forderung der Eindeutigkeit gilt auch die Forderung nach minimaler Merkmalskombination, wie wir sie in Abschnitt 1.2 für Tabellenschlüssel diskutiert haben.

Abb. 2-2
Beispiel:
Entitätsmenge
MITARBEITER



Die Abb. 2-2 charakterisiert einen bestimmten Mitarbeiter durch seine konkreten Merkmale als Entität. Möchte man für die betriebsinterne Projektkontrolle sämtliche Mitarbeiter mit ihren Namensangaben und Adressdaten erfassen, so legt man eine Entitätsmenge MITARBEITER fest. Neben den Merkmalen Name, Straße und Ort erlaubt eine künstliche Mitarbeiternummer, die einzelnen Mitarbeiter (Entitäten) innerhalb der Belegschaft (Entitätsmenge) eindeutig zu identifizieren.

Neben den Entitätsmengen selbst sind *Beziehungen* (engl. *relationships*) zwischen ihnen von Bedeutung. Diese bilden wiederum eine Menge. Beziehungsmengen können, ebenso wie Entitätsmengen, durch eigene Merkmale näher charakterisiert werden.

In Abb. 2-3 wird die Aussage «Mitarbeiter Meier arbeitet zu 70% am Projekt P17» als konkretes Beispiel einer Mitarbeiter-Projektbeziehung verstanden. Die entsprechende Beziehungsmenge ZUGEHÖRIGKEIT soll sämtliche Projektzugehörigkeiten unter den Mitarbeitenden aufzählen. Sie enthält als zusammengesetzten Schlüssel die Fremdschlüssel Mitarbeiter- und Projektnummer. Mit dieser Merkmalskombination lässt sich jede Mitarbeiter-Projektzugehörigkeit eindeutig festhalten. Neben diesem zusammengesetzten Schlüssel wird ein eigenständiges Beziehungsmerkmal mit der Bezeichnung «%-Anteil» beigefügt. Dieses nennt den prozentualen Anteil der Arbeitszeit, die ein Projektmitarbeiter seinen zugewiesenen Projekten widmet.

Entwurf einer Entitätsmenge für Mitarbeitende

Zur Definition von Beziehungsmengen

Beziehungsmengen können eigenständige Merkmale aufweisen

| | |
|----------------------------------|--|
| <i>Beziehung:</i> | Mitarbeiter Meier arbeitet zu 70% am Projekt P17 |
| <i>Beziehungsmenge:</i> | Menge aller Mitarbeiter-Projektzugehörigkeiten mit Merkmalen Mitarbeiternummer, Projektnummer und %-Anteil |
| <i>Identifikationsschlüssel:</i> | zusammengesetzter Schlüssel aus Mitarbeiter- und Projektnummer |

Darstellung im Entitäten-Beziehungsmodell:

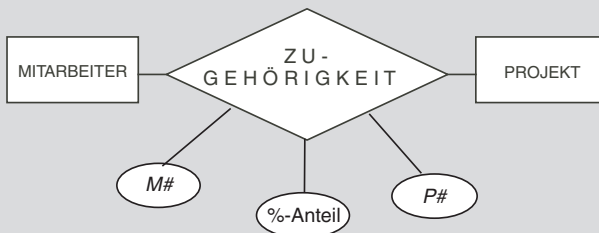


Abb. 2-3
Beziehung ZUGEHÖRIGKEIT zwischen Mitarbeitenden und Projekten

Beziehungen
bestehen aus
zwei
Assoziationen

Im Allgemeinen lassen sich Beziehungen immer in zwei Richtungen als so genannte Assoziationen deuten. Die Beziehungsmenge ZUGEHÖRIGKEIT kann aus der Sicht der Entitätsmenge MITARBEITER wie folgt interpretiert werden: Ein Mitarbeiter kann an mehreren Projekten mitwirken. Von der Entitätsmenge PROJEKT aus betrachtet lässt sich die Beziehung so verstehen, dass ein Projekt von mehreren Mitarbeitenden bearbeitet wird.

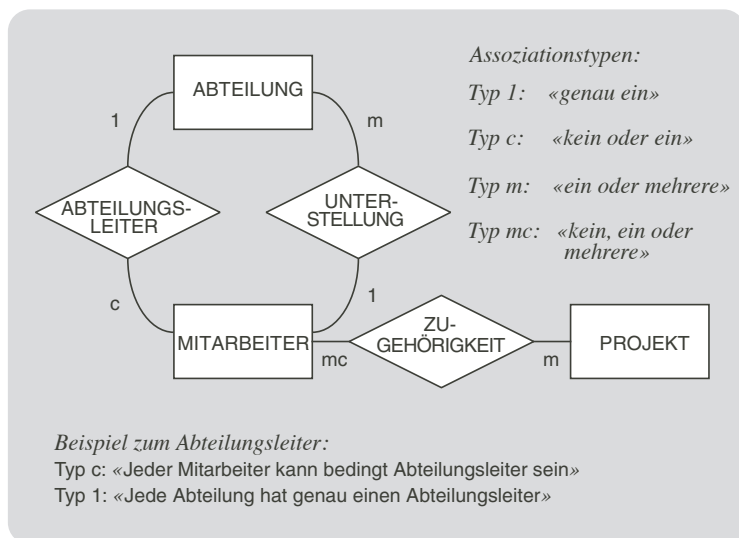
2.2.2 Assoziationstypen

Die Bedeutung
der Beziehung
wird durch
Assoziationen
ausgedrückt

Unter *Assoziation* (engl. *association*) einer Entitätsmenge EM_1 nach einer zweiten Entitätsmenge EM_2 versteht man die Bedeutung der Beziehung in dieser Richtung. Betrachten wir dazu die Beziehung ABTEILUNGSLEITER in Abb. 2-4, so erkennen wir für diese Beziehungsmenge zwei Assoziationen. Einerseits gehört zu jeder Abteilung ein Mitarbeiter in der Funktion des Abteilungsleiters, andererseits können gewisse Mitarbeiter die Funktion des Abteilungsleiters für eine bestimmte Abteilung ausüben.

Jede Assoziation einer Entitätsmenge EM_1 nach einer Entitätsmenge EM_2 kann mit einem Assoziationstyp gewichtet werden. Der Assoziationstyp von EM_1 nach EM_2 gibt an, wie viele Entitäten aus der assoziierten Entitätsmenge EM_2 einer bestimmten Entität aus EM_1 zugeordnet werden können. Im Wesentlichen werden einfache, konditionelle, mehrfache und mehrfach-konditionelle Assoziationstypen unterschieden:

Abb. 2-4
Entitäten-
Beziehungs-
modell
mit Assoziations-
typen



Einfache Assoziation (Typ 1)

Bei einer einfachen Assoziation oder einer Assoziation vom Typ 1 ist jeder Entität aus der Entitätsmenge EM_1 «genau eine» Entität aus der Entitätsmenge EM_2 zugeordnet. Beispielsweise ist aufgrund unserer Datenanalyse jeder Mitarbeiter genau einer Abteilung unterstellt; es wird also keine «Matrixorganisation» zugelassen. Die Assoziation UNTERSTELLUNG aus Abb. 2-4 von Mitarbeitern zu Abteilungen ist somit einfach oder vom Typ 1.

*Eindeutige
Assoziation vom
Typ 1*

Konditionelle Assoziation (Typ c)

Jeder Entität aus der Entitätsmenge EM_1 ist «keine oder eine», also höchstens eine Entität aus der Entitätsmenge EM_2 zugeordnet. Der Assoziationstyp ist *bedingt* oder *konditionell* (engl. *conditional*). Eine bedingte Beziehung tritt beispielsweise beim ABTEILUNGSLEITER auf (vgl. Abb. 2-4), da nicht jeder Mitarbeiter die Funktion eines Abteilungsleiters ausüben kann.

*Bedingte
Assoziation vom
Typ c*

Mehrfache Assoziation (Typ m)

Bei einer mehrfachen Assoziation oder einer Assoziation vom Typ m sind jeder Entität aus der Entitätsmenge EM_1 «eine oder mehrere» Entitäten in der Entitätsmenge EM_2 zugeordnet. Dieser und der nächste Assoziationstyp werden oft als *komplex* bezeichnet, da eine Entität aus EM_1 mit beliebig vielen in EM_2 in Beziehung stehen kann. Ein Beispiel einer mehrfachen Assoziation ist in Abb. 2-4 die Beziehung ZUGEHÖRIGKEIT von Projekten zu Mitarbeitern: Jedes Projekt kann von mehreren, muss jedoch von mindestens einem Mitarbeiter bearbeitet werden.

*Komplexe
Assoziation vom
Typ m*

Mehrfach-konditionelle Assoziation (Typ mc)

Jeder Entität aus der Entitätsmenge EM_1 sind «keine, eine oder mehrere» Entitäten aus der Entitätsmenge EM_2 zugeordnet. Der *mehrfach-konditionelle Assoziationstyp* hebt sich von dem mehrfachen dadurch ab, dass hier nicht zu jeder Entität aus EM_1 eine Beziehung zu denen von EM_2 bestehen muss. Als Beispiel können wir nochmals die ZUGEHÖRIGKEIT aus Abb. 2-4 betrachten, diesmal aus Sicht der Mitarbeiter: Nicht jeder Mitarbeiter braucht Projektarbeit zu leisten; es gibt andererseits Mitarbeiter, die an mehreren Projekten mitwirken.

*Bedingt-
komplexe
Assoziation vom
Typ mc*

Die Assoziationstypen machen eine Aussage über die Mächtigkeit der Beziehung. Wie wir gesehen haben, umfasst jede Beziehung zwei Assoziationstypen. Die *Mächtigkeit einer Beziehung* zwischen den Entitätsmengen EM_1 und EM_2 ergibt sich somit durch ein *Paar von Assoziationstypen* der Form:

*Zur Mächtigkeit
einer Beziehung*

Mächtigkeit := (Typ von EM_1 nach EM_2, Typ von EM_2 nach EM_1).²

Beispielsweise sagt das Paar (mc,m) von Assoziationstypen zwischen MITARBEITER und PROJEKT aus, dass die Beziehung ZUGEHÖRIGKEIT (bedingt-komplex, komplex) ist.

Zur Angabe von
Beziehungs-
schränken

Anstelle der Assoziationstypen können auch *Minimal- und Maximalschranken* angegeben werden, falls dies sinnvoll erscheint. So könnte man für den mehrfachen Assoziationstyp von Projekten zu Mitarbeitern anstelle von «m» eine Angabe (MIN,MAX) := (3,8) festlegen. Die untere Schranke sagt aus, dass an einem Projekt definitionsgemäß mindestens drei Mitarbeiter beteiligt sein müssen. Umgekehrt verlangt die obere Schranke die Begrenzung eines Projektes auf höchstens acht Mitarbeiter.

2.2.3

Generalisation und Aggregation

Entitäten lassen
sich
spezialisieren
und
verallgemeinern

Unter *Generalisation* (engl. *generalization*) versteht man das Verallgemeinern von Entitäten und somit auch von Entitätsmengen. Die Generalisation ist ein Abstraktionsvorgang, bei dem einzelne Entitätsmengen zu einer übergeordneten Entitätsmenge verallgemeinert werden. Umgekehrt lassen sich die in einer Generalisationshierarchie abhängigen Entitätsmengen oder Subentitätsmengen als *Spezialisierung* interpretieren. Bei der Generalisation von Entitätsmengen treten die folgenden Fälle auf:

Überlappende
Entitäts-
mengen

- Die Subentitätsmengen der Spezialisierung *überlappen sich gegenseitig*. Betrachten wir als Beispiel die Entitätsmenge MITARBEITER mit zwei Subentitätsmengen FOTOCLUB und SPORTCLUB, so können wir die Clubmitglieder als Mitarbeiter auffassen. Umgekehrt kann ein Mitarbeiter sowohl im firmeninternen Fotoclub wie auch im Sportclub aktiv mitwirken, d.h., die Subentitätsmengen FOTOCLUB und SPORTCLUB überlappen sich.

Überlappend-
vollständige
Entitäts-
mengen

- Die Subentitätsmengen der Spezialisierung *überlappen sich gegenseitig und überdecken vollständig* die verallgemeinerte Entitätsmenge der Generalisation. Ergänzen wir die beiden obigen Subentitätsmengen SPORTCLUB und FOTOCLUB durch einen SCHACHCLUB und nehmen wir an, dass jeder Mitarbeiter beim Eintritt in die Firma mindestens einem dieser drei Clubs beitrifft, so liegt eine «überlappend-vollständige» Konstellation vor. Ein

² Die Zeichenkombination «:=» bedeutet «ist definiert durch ...».

Mitarbeiter ist also mindestens in einem der drei Clubs, kann aber auch in zwei oder in allen drei Clubs mitwirken.

- Die Subentitätsmengen der Spezialisierung sind *gegenseitig disjunkt*, d.h., sie schließen sich gegenseitig aus. Als Beispiel betrachten wir die Entitätsmenge MITARBEITER mit den Spezialisierungen FÜHRUNGSKRAFT und FACHSPEZIALIST. Da jeder Mitarbeiter nicht gleichzeitig eine Kaderposition bekleiden und eine Fachkarriere verfolgen kann, fassen wir die beiden Spezialisierungen FÜHRUNGSKRAFT und FACHSPEZIALIST als gegenseitig disjunkt auf.
- Die Subentitätsmengen der Spezialisierung sind *gegenseitig disjunkt und überdecken vollständig* die verallgemeinerte Entitätsmenge der Generalisation. In diesem Fall muss also zu jeder Entität aus der übergeordneten Entitätsmenge eine Subentität in der Spezialisierung bestehen und umgekehrt. Als Beispiel nehmen wir wiederum die Entitätsmenge MITARBEITER, wobei wir neben den Spezialisierungen FÜHRUNGSKRAFT und FACHSPEZIALIST eine weitere Spezialisierung LEHRLING betrachten. Dabei soll gelten, dass jeder Mitarbeiter entweder als Führungskraft, Fachspezialist oder Lehrling tätig ist.

*Disjunkte
Entitätsmengen*

*Disjunkt-
vollständige
Entitäts-
mengen*

Jede Generalisationshierarchie wird durch ein spezielles grafisches Gabelungssymbol dargestellt, ergänzt durch die Charakterisierung «überlappend-unvollständig», «überlappend-vollständig», «disjunkt-unvollständig» oder «disjunkt-vollständig».

*Grafisches
Symbol zeichnet
Generalisations-
hierarchie aus*

In Abb. 2-5 ist die Entitätsmenge MITARBEITER als disjunkte und vollständige Verallgemeinerung von FÜHRUNGSKRAFT, FACHSPEZIALIST und LEHRLING aufgeführt. Jede abhängige Entität wie Gruppenchef oder Abteilungsleiter der Entitätsmenge FÜHRUNGSKRAFT

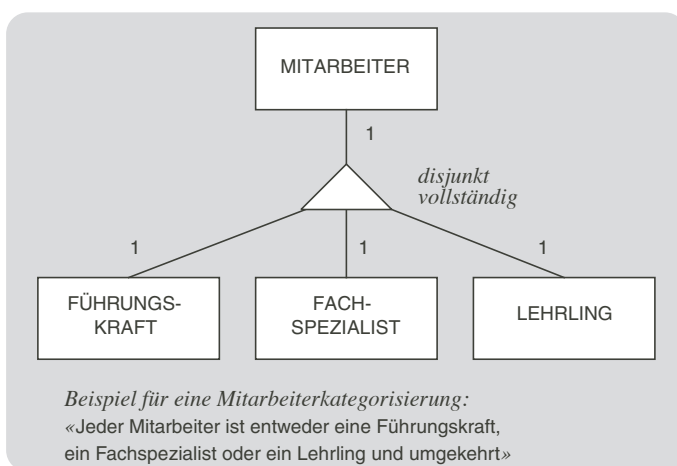


Abb. 2-5
*Generalisation
am Beispiel
MITARBEITER*

*Generalisation
als IS-A-Struktur*

ist vom Typ MITARBEITER, da der entsprechende Assoziationstyp 1 ist. Aus diesem Grund wird die Generalisation oft in Anlehnung ans Englische als *IS-A-Struktur* bezeichnet: Ein Gruppenchef «ist ein» (engl. *is a*) Mitarbeiter, und ein Abteilungsleiter ist ebenfalls als Mitarbeiter angestellt. Die umgekehrte Assoziation ist bei einer disjunkten und vollständigen Generalisationshierarchie wiederum vom Typ 1; z.B. gehört jeder Mitarbeiter eindeutig einer Subentitätsmenge an.

*Zur Bildung von
Aggregations-
strukturen*

Neben der Generalisation spielt eine zweite Beziehungsstruktur eine wichtige Rolle. Unter dem Begriff *Aggregation* (engl. *aggregation*) versteht man das Zusammenfügen von Entitäten zu einem übergeordneten Ganzen. Dabei werden die Struktureigenschaften der Aggregation in einer Beziehungsmenge erfasst.

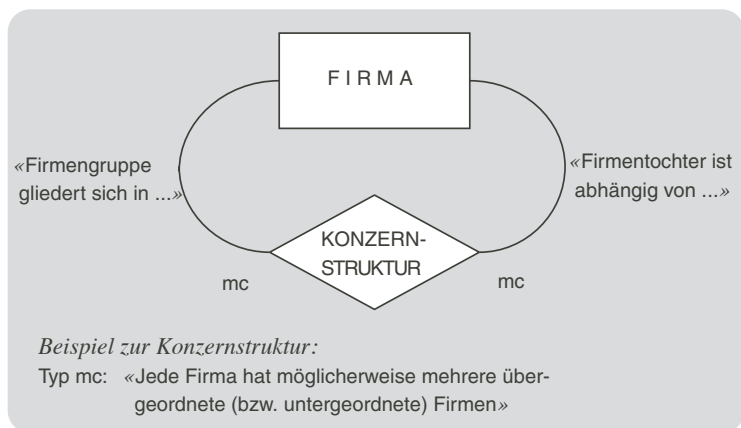
*Beispiel einer
Konzernstruktur*

Soll für einen Konzern die Beteiligungsstruktur modelliert werden, wie in Abb. 2-6 dargestellt, so geschieht das mit einer Beziehungsmenge KONZERNSTRUKTUR. Diese beschreibt ein Beziehungsnetz der Entitätsmenge FIRMA auf sich selbst. Die Firmennummer aus der Entitätsmenge FIRMA wird zweimal in der KONZERNSTRUKTUR als Fremdschlüssel verwendet, nämlich einmal als Nummer der übergeordneten und einmal als Nummer der untergeordneten Firmengesellschaften (vgl. auch Abb. 2-14). Daneben können in der KONZERNSTRUKTUR weitere Beziehungsmerkmale wie z.B. das der Beteiligungen geführt werden.

*Aggregation als
PART-OF-
Struktur*

Allgemein beschreibt eine Aggregation das strukturierte Zusammenfügen von Entitäten, was wir in Anlehnung ans Englische als *PART-OF-Struktur* bezeichnen. So kann bei der KONZERNSTRUKTUR jede Firma eine «Tochter von» (engl. *part of*) einem bestimmten Konzern sein. Da in unserem Beispiel die KONZERNSTRUKTUR netzwerkartig definiert ist, müssen beide Assoziationstypen der übergeordneten wie der untergeordneten Teile komplex sein.

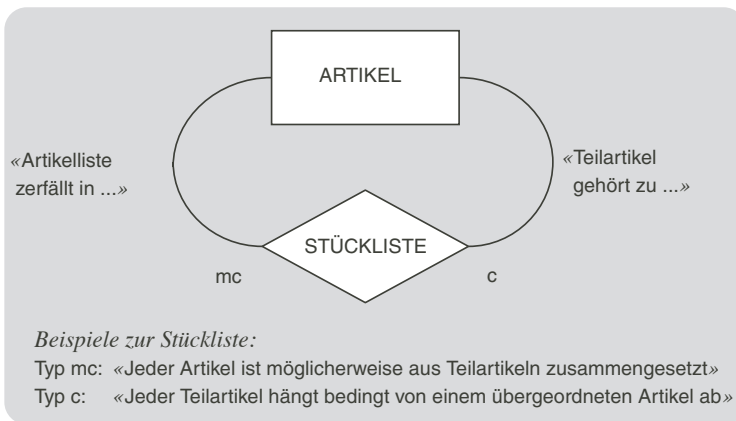
*Abb. 2-6
Netzwerkartige
Aggregation am
Beispiel
KONZERN-
STRUKTUR*



Die beiden Abstraktionskonzepte Generalisation und Aggregation sind wichtige Strukturierungselemente³ bei der Datenmodellierung. Im Entitäten-Beziehungsmodell können sie durch eigene grafische Symbole charakterisiert oder als Spezialkästchen ausgedrückt werden. So könnte man obige Aggregation aus Abb. 2-6 durch eine verallgemeinerte Entitätsmenge KONZERN darstellen, die implizit die Entitätsmenge FIRMA und die Beziehungsmenge KONZERNSTRUKTUR einschließen würde.

Neben netzwerkartigen PART-OF-Strukturen gibt es auch hierarchische. In Abb. 2-7 ist dazu eine STÜCKLISTE illustriert: Jeder Artikel kann aus mehreren Teilartikeln zusammengesetzt sein. Umgekehrt zeigt jeder Teilartikel mit Ausnahme des obersten Artikels genau auf einen übergeordneten Artikel (vgl. Abb. 2-15).

Generalisation und Aggregation sind wichtige Strukturierungskonzepte



*Abb. 2-7
Hierarchische Aggregation am Beispiel STÜCKLISTE*

Beim Einsatz von rechnergestützten Werkzeugen zur Datenmodellierung spielt das Entitäten-Beziehungsmodell eine besondere Rolle, wird es doch von vielen CASE-Werkzeugen (CASE = Computer Aided Software Engineering) mehr oder weniger unterstützt. Je nach Güte dieser Werkzeuge können neben Entitätsmengen und Beziehungsmengen auch Generalisation und Aggregation in separaten Konstruktionsschritten beschrieben werden. Danach erst lässt sich das *Entitäten-Beziehungsmodell in ein Datenbankschema teilweise automatisch überführen*. Da dieser Abbildungsprozess nicht immer eindeutig ist, liegt es am Datenarchitekten, die richtigen Entscheidungen zu treffen. Aus diesem Grund werden in den nächsten Abschnitten einfache Abbildungsregeln erläutert, die ein Entitäten-Beziehungsmodell konfliktfrei in ein relationales Datenbankschema transformieren.

Rechnergestützte Werkzeuge zur Datenmodellierung

³ Erweiterte Datenbanksysteme unterstützen Generalisation und Aggregation als Strukturierungskonzepte (vgl. Abschnitt 6.4).

2.3

Das relationale Datenbankschema

2.3.1

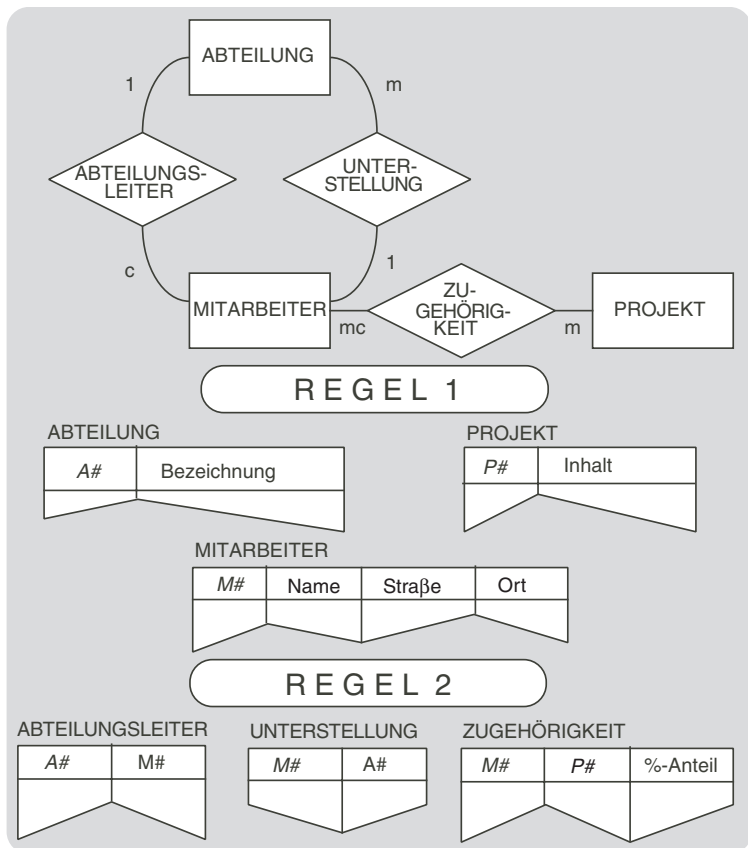
Überführen des Entitäten-Beziehungsmodells

In diesem und den folgenden Abschnitten behandeln wir die Abbildung des Entitäten-Beziehungsmodells auf ein relationales Datenbankschema. Im Wesentlichen müssen wir dabei festhalten, auf welche Weise sich *Entitäts- und Beziehungsmengen* durch Tabellen darstellen lassen.

Was versteht man unter einem Datenbankschema?

Unter einem *Datenbankschema* (engl. *database schema*) versteht man eine Datenbankbeschreibung, d.h. die Spezifikation von Datenstrukturen mitsamt ihren zugehörigen Integritätsbedingungen. Ein relationales Datenbankschema enthält die Definition der Tabellen, der Merkmale und der Primärschlüssel. Integritätsbedingungen legen

Abb. 2-8
Abbildung von
Entitäts- und
Beziehungsmengen
in Tabellen



Einschränkungen für die Wertebereiche, für die Abhängigkeiten zwischen verschiedenen Tabellen (referentielle Integrität gemäß Abschnitt 2.5) sowie für die eigentlichen Datenvorkommen fest.

Beim Überführen eines Entitäten-Beziehungsmodells auf ein relationales Datenbankschema sind die Abbildungsregeln 1 und 2 von Bedeutung (vgl. Abb. 2-8).

Regel 1 (Entitätsmengen)

Jede *Entitätsmenge* muss als *eigenständige Tabelle* mit einem eindeutigen Primärschlüssel definiert werden. Als Primärschlüssel der Tabelle dient entweder der entsprechende Schlüssel der Entitätsmenge oder ein Schlüsselkandidat. Die übrigen Merkmale der Entitätsmengen gehen in die korrespondierenden Attribute der Tabellen über.

*Die Muss-Regel
für Entitäts-
mengen*

Die Definition einer Tabelle (Abschnitt 1.2) verlangt einen eindeutigen Primärschlüssel. Nun kann es sein, dass in einer Tabelle mehrere *Schlüsselkandidaten* (engl. *candidate keys*) vorliegen, die allesamt die Forderung nach Eindeutigkeit und Minimalität erfüllen. Dann entscheidet der Datenarchitekt, welchen der Schlüsselkandidaten er als Primärschlüssel auszeichnen möchte.

*Schlüssel-
kandidat
auswählen*

Regel 2 (Beziehungsmengen)

Jede *Beziehungsmenge* kann als *eigenständige Tabelle* definiert werden, wobei die Identifikationsschlüssel der zugehörigen Entitätsmengen als *Fremdschlüssel* in dieser Tabelle auftreten müssen. Der Primärschlüssel der Beziehungsmengentabelle kann der aus den Fremdschlüsseln zusammengesetzte Identifikationsschlüssel sein oder ein anderer Schlüsselkandidat, z.B. in Form eines künstlichen Schlüssels. Weitere Merkmale der Beziehungsmenge erscheinen als zusätzliche Attribute in der Tabelle.

*Die Kann-Regel
für Beziehungs-
mengen*

Als *Fremdschlüssel* (engl. *foreign key*) einer Tabelle wird ein Merkmal bezeichnet, das in derselben oder in einer anderen Tabelle als Identifikationsschlüssel auftritt. Somit dürfen Identifikationsschlüssel in weiteren Tabellen wiederverwendet werden, um die gewünschten Beziehungen zwischen den Tabellen herstellen zu können.

*Fremdschlüssel
decken
Beziehungen
zwischen
Tabellen auf*

In Abb. 2-8 zeigen wir die Anwendung der Regeln 1 und 2 anhand unseres konkreten Beispiels. Jede Entitätsmenge ABTEILUNG, MITARBEITER und PROJEKT wird in ihre entsprechende Tabelle ABTEILUNG, MITARBEITER und PROJEKT überführt. Auf analoge Art definieren wir zu jeder Beziehungsmenge ABTEILUNGSLEITER, UNTERSTELLUNG und ZUGEHÖRIGKEIT eine entsprechende Tabelle. Die Tabellen ABTEILUNGSLEITER und UNTERSTELLUNG verwenden die Abteilungsnummer und die Mitarbeiternummer als Fremdschlüssel. Die Tabelle ZUGEHÖRIGKEIT entlehnt die Identifikationsschlüs-

sel den beiden Tabellen MITARBEITER und PROJEKT und führt das Merkmal «%-Anteil» als weiteres Beziehungsmerkmal auf.

*Schlüssel von
Beziehungs-
mengentabellen*

Da zu jeder Abteilung genau ein Abteilungsleiter gehört, genügt die Abteilungsnummer A# in der Tabelle ABTEILUNGSLEITER als Identifikationsschlüssel. Auf analoge Art erklären wir die Mitarbeiternummer M# als Identifikationsschlüssel in der Tabelle UNTERSTELLUNG. Die Mitarbeiternummer genügt hier, da jeder Mitarbeitende genau einer Abteilung unterstellt ist.

Im Gegensatz zu den Tabellen ABTEILUNGSLEITER und UNTERSTELLUNG müssen bei der Tabelle ZUGEHÖRIGKEIT die beiden Fremdschlüssel der Mitarbeiter- und Projektnummer als zusammengesetzter Identifikationsschlüssel definiert werden. Der Grund liegt in der Tatsache, dass ein Mitarbeiter mehrere Projekte bearbeiten kann und dass umgekehrt in jedes Projekt eventuell mehrere Mitarbeitende einbezogen sind.

*Wann müssen
Beziehungs-
mengen als
eigenständige
Tabellen definiert
werden?*

Die Anwendung der Regeln 1 und 2 führt nicht in jedem Fall zu einem optimalen relationalen Datenbankschema. Störend wirkt, dass bei diesem Vorgehen unter Umständen eine ansehnliche Anzahl von Tabellen entsteht. Es fragt sich z.B., ob es sinnvoll ist, gemäß Abb. 2-8 für die Funktion des Abteilungsleiters eine eigene Tabelle zu verlangen. Wie im nächsten Abschnitt erläutert, könnten wir aufgrund der Abbildungsregel 5 tatsächlich auf die Tabelle ABTEILUNGSLEITER verzichten. Die Funktion «Abteilungsleiter» würde in der Tabelle ABTEILUNG lediglich als zusätzliche Merkmalskategorie stehen und für jede Abteilung die Mitarbeiternummer des jeweiligen Abteilungschefs anführen.

2.3.2

Abbildungsregeln für Beziehungsmengen

Dieser Abschnitt illustriert, wie Beziehungsmengen auf differenzierte Art in Tabellen überführt werden können. In Abhängigkeit von den Beziehungstypen werden die weiteren Abbildungsregeln 3 bis 5 aufgestellt.

*Einfach-einfache
Beziehungen*

Bereits in Abschnitt 2.2.2 wurde die Mächtigkeit einer Beziehung durch ein Assoziationstypenpaar definiert. Unter *einfach-einfachen Beziehungen* versteht man nun Beziehungen, bei welchen die beiden Assoziationstypen entweder einfach oder konditionell sind. Gemäß Abb. 2-9 erhalten wir vier Möglichkeiten einer einfach-einfachen Beziehung. Diese sind durch die Mächtigkeiten (1,1), (1,c), (c,1) und (c,c) charakterisiert.

*Einfach-
komplexe
Beziehungen*

Auf analoge Art definieren wir *einfach-komplexe Beziehungen* als Beziehungen, bei denen die Mächtigkeit aus einem einfachen (Typ 1 oder c) und einem komplexen (Typ m oder mc) Assoziationstyp

$B_j := (A1, A2)$ Grad der Beziehung mit Assoziationstypen A1 und A2

| A1 \ A2 | 1 | c | m | mc |
|---------|--------|--------|--------|---------|
| 1 | (1,1) | (1,c) | (1,m) | (1,mc) |
| c | (c,1) | (c,c) | (c,m) | (c,mc) |
| m | (m,1) | (m,c) | (m,m) | (m,mc) |
| mc | (mc,1) | (mc,c) | (mc,m) | (mc,mc) |

B1 einfach-einfache Beziehungen
B2 einfach-komplexe Beziehungen
B3 komplex-komplexe Beziehungen

Abb. 2-9
Übersicht über
die
Mächtigkeiten
von Beziehungen

besteht. Dazu ergeben sich acht Möglichkeiten für eine einfach-komplexe Beziehung.

Mit *komplex-komplexen Beziehungen* bezeichnen wir diejenigen, die als Mächtigkeit zweimal einen komplexen Assoziationstyp enthalten. Konkret erhalten wir die Fälle (m,m), (m,mc), (mc,m) und (mc,mc).

Aufgrund der Mächtigkeit von Beziehungen lassen sich drei Abbildungsregeln formulieren, die die Beziehungsmengen des Entitäten-Beziehungsmodells im entsprechenden relationalen Datenbankschema tabellarisch ausdrücken. Um die Anzahl der Tabellen nicht unnötig zu erhöhen, beschränkt man sich mit Hilfe der Regel 3 vorerst auf diejenigen Beziehungsmengen, die auf jeden Fall eine eigene Tabelle verlangen:

Regel 3 (komplex-komplexe Beziehungen)

Jede *komplex-komplexe Beziehungsmenge* muss als *eigenständige Tabelle* definiert werden. Dabei treten mindestens die Identifikationsschlüssel der zugehörigen Entitätsmengen als Fremdschlüssel auf. Der Primärschlüssel der Beziehungsmengentabelle ist entweder der aus den Fremdschlüsseln zusammengesetzte Identifikationsschlüssel oder ein anderer Schlüsselkandidat. Die weiteren Merkmale der Beziehungsmenge gehen in Attribute der Tabelle über.

Die Regel 3 schreibt vor, dass z.B. die Beziehungsmenge ZUGEHÖRIGKEIT aus Abb. 2-10 als eigenständige Tabelle auftreten muss, versehen mit einem Primärschlüssel. Als Identifikationsschlüssel der Tabelle ZUGEHÖRIGKEIT wird hier der zusammengesetzte Schlüssel ausgewählt, der die Fremdschlüsselbeziehungen zu den Tabellen

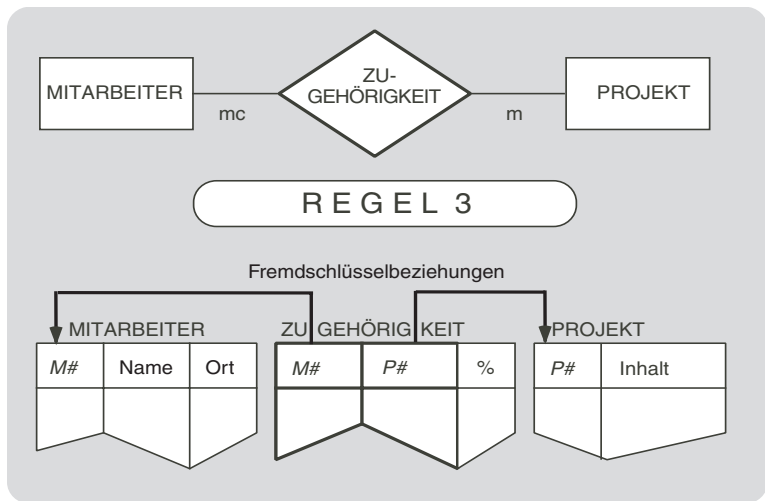
Komplex-komplexe Beziehungen

Jede Beziehungs-kategorie hat ihre Abbildungsregel

Die Muss-Regel für komplex-komplexe Beziehungsmengen

Wahl des zusammengesetzten Schlüssels

Abb. 2-10
Abbildungsregel
für komplex-
komplexe
Beziehungen



MITARBEITER und PROJEKT ausdrückt. Das Merkmal «%-Anteil» beschreibt das prozentuale Ausmaß dieser Zugehörigkeit.

*Haben wir
eine Matrix-
organisation
oder nicht?*

Wie wir gesehen haben, könnten wir gemäß Regel 2 für die Beziehungsmenge UNTERSTELLUNG eine eigenständige Tabelle definieren, und zwar mit den beiden Fremdschlüsseln Abteilungs- und Mitarbeiternummer. Dies wäre dann sinnvoll, wenn wir eine Matrixorganisation unterstützen und die eindeutige Unterstellungseigenschaft mit dem Assoziationstyp 1 in nächster Zukunft aufgeben möchten; zwischen ABTEILUNG und MITARBEITER wäre in diesem Fall eine komplex-komplexe Beziehung festgelegt. Sind wir hingegen überzeugt, dass keine Matrixorganisation vorgesehen ist, so können wir aufgrund der einfach-komplexen Beziehung die Regel 4 anwenden:

*Die Kann-Regel
für einfach-
komplexe
Beziehungs-
mengen*

Regel 4 (einfach-komplexe Beziehungen)

Eine einfach-komplexe Beziehungsmenge kann ohne eine eigenständige Beziehungsmengentabelle durch die beiden Tabellen der zugeordneten Entitätsmengen ausgedrückt werden. Dazu wird in der Tabelle mit der einfachen Assoziation (d.h. mit Assoziationstyp 1 oder c) ein Fremdschlüssel auf die referenzierte Tabelle mit eventuell weiteren Merkmalen der Beziehungsmenge geführt.

*Zur Verwendung
von Rollennamen*

Beim Verzicht auf eine eigenständige Tabelle für eine Beziehung kann diese gemäß Regel 4 durch eine bereits bestehende Tabelle ausgedrückt werden, indem wir eine der beiden Tabellen durch den Fremdschlüssel aus der anderen Tabelle ergänzen. Zur klaren Festlegung der Beziehung empfiehlt es sich, den Namen des entsprechenden Identifikationsschlüssels durch einen *Rollennamen* zu ergänzen. Ein Rollename drückt aus, welche Bedeutung ein bestimmter Schlüsselbereich in einer fremden Tabelle spielt.

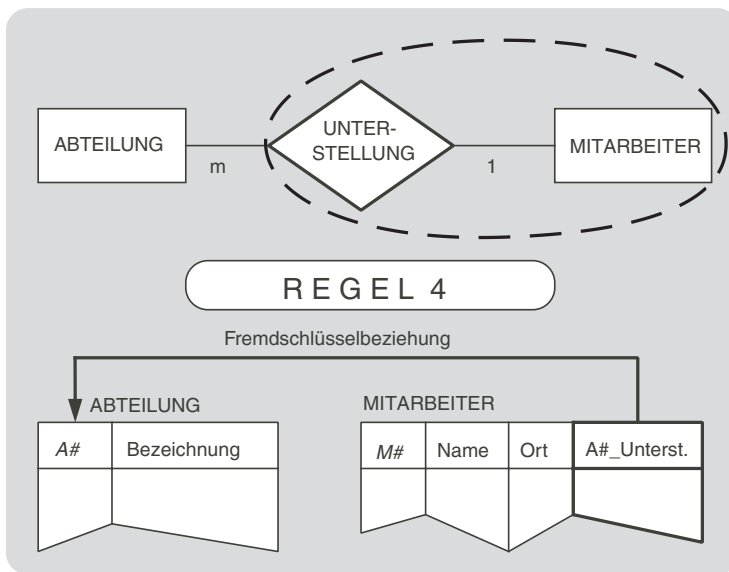


Abb. 2-11
Abbildungsregel
für
einfach-komplexe
Beziehungen

In Abb. 2-11 verzichten wir gemäß Regel 4 auf eine eigenständige Tabelle UNTERSTELLUNG. Anstelle einer zusätzlichen Beziehungsmengentabelle erweitern wir die Tabelle MITARBEITER um den Fremdschlüssel «A#_Unterstellung», der die Abteilungsnummer der Unterstellungsbeziehung pro Mitarbeiter angibt. Die Fremdschlüsselbeziehung wird durch ein Merkmal gegeben, das sich aus dem entliehenen Identifikationsschlüssel A# und dem Rollennamen «Unterstellung» zusammensetzt.

*Ausgeliehene
Schlüssel
erhalten
eine Rolle*

Liegen «einfach-komplexe» Beziehungsmengen vor, so ist das Entleihen des Fremdschlüssels auf eindeutige Art möglich. In Abb. 2-11 führen wir gemäß Regel 4 die Abteilungsnummer als Fremdschlüssel in der Tabelle MITARBEITER. Würden wir umgekehrt die Mitarbeiternummer in der Tabelle ABTEILUNG vorsehen, müssten wir für jeden Mitarbeiter einer Abteilung die Abteilungsbezeichnung wiederholen. Solche überflüssige oder redundante Informationen sind unerwünscht; sie werden im nächsten Abschnitt anhand der Normalisierungstheorie untersagt.

*Die
MITARBEITER-
Tabelle wird um
eine Beziehungs-
eigenschaft
erweitert*

Regel 5 (einfach-einfache Beziehungen)

Eine *einfach-einfache* Beziehungsmenge kann ohne eine eigenständige Tabelle durch die beiden Tabellen der zugeordneten Entitätsmengen ausgedrückt werden, indem einer der Identifikationsschlüssel der referenzierten Tabelle als Fremdschlüssel in die andere Tabelle eingebracht wird.

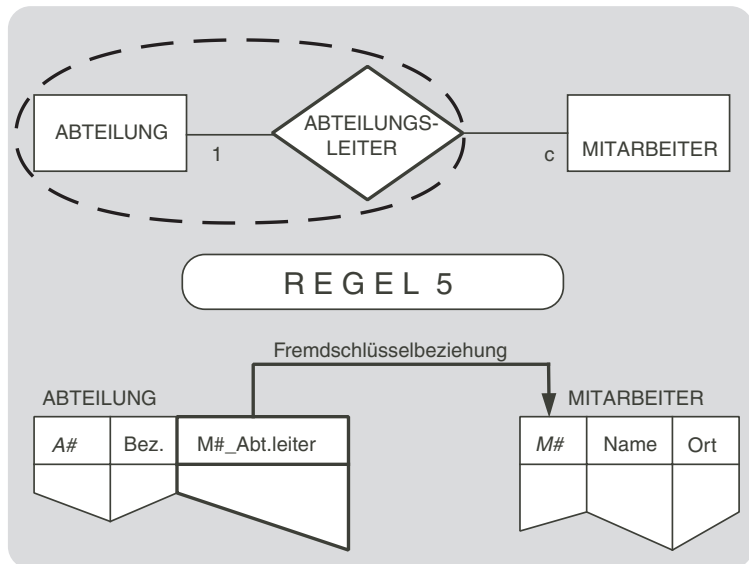
*Die Kann-Regel
für einfach-
einfache
Beziehungs-
mengen*

Auch hier ist nicht unbedeutend, aus welcher der beiden Tabellen solche Fremdschlüsselkandidaten entliehen werden. Normalerweise fügen wir in die Tabelle mit dem Assoziationstyp 1 den Fremdschlüssel der referenzierten Tabelle ein.

Vermeidung von
Nullwerten

In Abb. 2-12 ergänzen wir die Tabelle ABTEILUNG um die Mitarbeiternummer des Abteilungsleiters. Die Beziehungsmenge ABTEILUNGSLEITER wird damit durch das Merkmal «M#_Abteilungsleiter» ausgedrückt. Jeder Eintrag in diesem fremdbezogenen Merkmal mit der Rolle «Abteilungsleiter» zeigt, wer der jeweilige Abteilungsleiter ist. Würden wir anstelle dieses Vorgehens die Abteilungsnummer in der Tabelle MITARBEITER führen, so müssten wir für die meisten Mitarbeiter Nullwerte (vgl. dazu Abschnitt 3.6) auflisten. Lediglich bei denjenigen Mitarbeitenden, die eine Abteilung leiten, könnten wir die entsprechende Abteilungsnummer einsetzen. Aus diesem Grund führen wir die Rolle des Abteilungsleiters in der Tabelle ABTEILUNG.

Abb. 2-12
Abbildungsregel
für
einfach-einfache
Beziehungen



2.3.3

Abbildungsregeln für Generalisation und Aggregation

Treten in einem Entitäten-Beziehungsmodell Generalisationshierarchien oder Aggregationsstrukturen auf, so müssen diese ebenfalls in ein relationales Datenbankschema überführt werden. Obwohl die bereits bekannten Assoziationstypen bei diesen speziellen Arten von Beziehungen erscheinen, unterscheiden sich die entsprechenden Abbildungsregeln von den bisherigen.

Regel 6 (Generalisation)

Jede *Entitätsmenge einer Generalisationshierarchie* verlangt eine *eigenständige Tabelle*, wobei der Primärschlüssel der übergeordneten Tabelle auch Primärschlüssel der untergeordneten Tabellen wird.

Da das Relationenmodell die Beziehungsstruktur einer Generalisation nicht direkt unterstützt, müssen die Eigenschaften dieser Beziehungshierarchie indirekt nachgebildet werden. Bei einer *überlappend-unvollständigen, überlappend-vollständigen, disjunkt-unvollständigen oder disjunkt-vollständigen Generalisation* müssen die Identifikationsschlüssel der Spezialisierungen immer mit denjenigen der übergeordneten Tabelle übereinstimmen. Die Eigenschaft von überlappenden Spezialisierungen verlangt keine speziellen Prüfregele, hingegen muss die Disjunktheit im Relationenmodell nachvollzogen werden. Eine Möglichkeit besteht darin, in der übergeordneten Tabelle ein Merkmal «Kategorie» mitzuführen. Dieses Merkmal entspricht einer Klassenbildung und drückt aus, um welche Spezialisierung es sich handelt. Darüber hinaus muss bei einer disjunkten und vollständigen Generalisation garantiert werden, dass pro Eintrag in der übergeordneten Tabelle genau ein Eintrag in einer Tabelle der Spezialisierung vorliegt und umgekehrt.

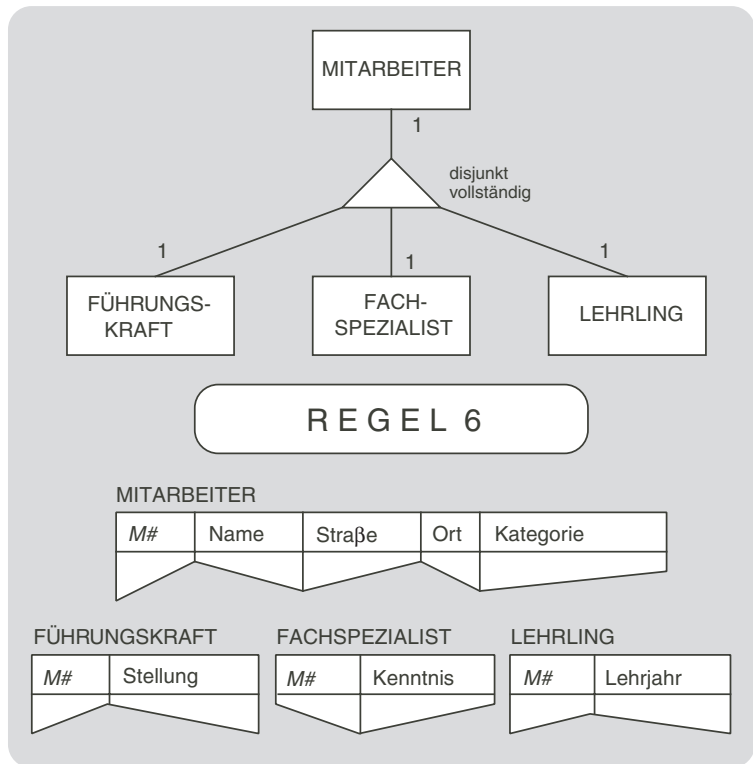
In Abb. 2-13 zeigen wir die Mitarbeiterinformation als Generalisation. Die Regel 6 führt zu den Tabellen MITARBEITER, FÜHRUNGSKRAFT, FACHSPEZIALIST und LEHRLING. In den von der Tabelle MITARBEITER abhängigen Tabellen muss derselbe Identifikationsschlüssel M# verwendet werden. Damit ein bestimmter Mitarbeiter nicht mehreren Kategorien gleichzeitig angehört, führen wir das Merkmal «Kategorie» ein. Dieses kann die Werte «Führungskraft», «Fachspezialist» oder «Lehrling» annehmen. Es garantiert somit die Eigenschaft einer disjunkten Generalisationshierarchie (gemäß Abschnitt 2.2.3), wonach die einzelnen Entitätsmengen der Spezialisierung sich nicht gegenseitig überlappen dürfen. Die Eigenschaft der

Abbildungsregel für eine IS-A-Struktur

Zur Unterscheidung der Generalisationshierarchien

Beispiel einer disjunkt-vollständigen Generalisation

Abb. 2-13
Generalisation in
Tabellenform



Vollständigkeit kann nicht explizit im Datenbankschema ausgedrückt werden und bedarf einer speziellen Integritätsbedingung.

Regel 7 (Aggregation)

Die Abbildungs-
regel für eine
PART-OF-
Struktur

Bei einer Aggregation müssen sowohl die Entitätsmenge als auch die Beziehungsmenge je als *eigenständige Tabelle* definiert werden, falls der Beziehungstyp *komplex-komplex* ist. Die Tabelle der Beziehungsmenge enthält in diesem Fall zweimal den Schlüssel aus der Tabelle der zugehörigen Entitätsmenge als zusammengesetzten Identifikationsschlüssel, mit entsprechenden Rollennamen. Im Falle einer *einfach-komplexen* Beziehung (hierarchische Struktur) kann die Entitätsmenge mit der Beziehungsmenge zu einer *einzigsten Tabelle* kombiniert werden.

Im Beispiel der Abb. 2-14 weist die Konzernstruktur die Mächtigkeit (mc,mc) auf. Gemäß Regel 7 müssen zwei Tabellen FIRMA und KONZERNSTRUKTUR festgelegt werden. Die Beziehungsmengentabelle KONZERNSTRUKTUR zeigt durch Identifikationsschlüssel in jedem Eintrag an, welche die direkt abhängigen Teile einer Firmengruppe bzw. welche die direkt übergeordneten Gruppen eines bestimmten

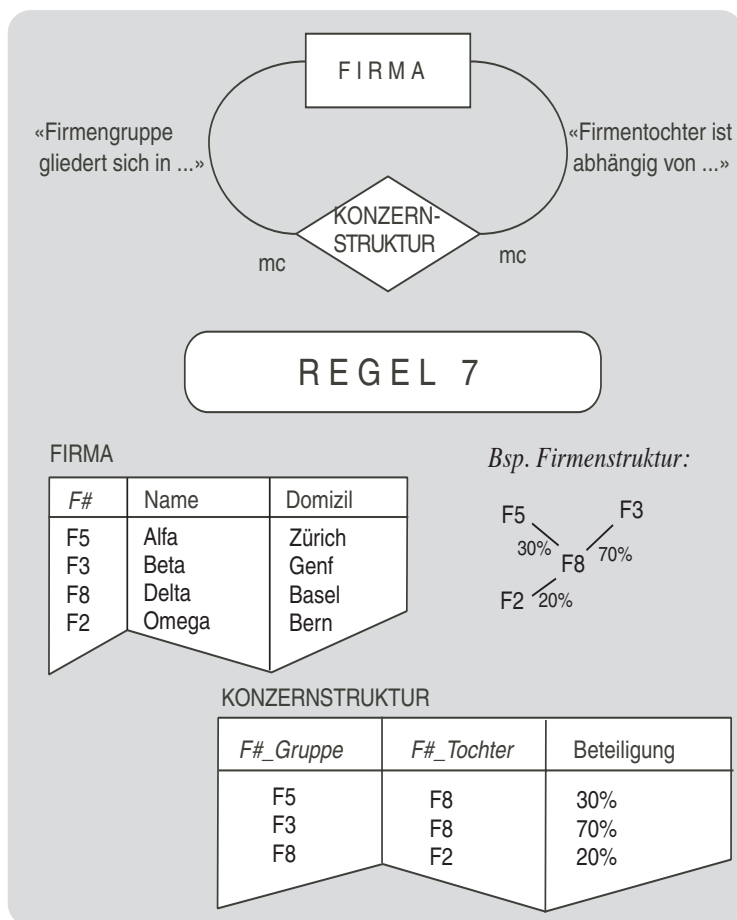


Abb. 2-14
Netzwerkartige
Firmenstruktur in
Tabellenform

Firmenteils sind. Neben den beiden Fremdschlüsselbeziehungen vervollständigt ein Beteiligungsmerkmal die Tabelle KONZERNSTRUKTUR.

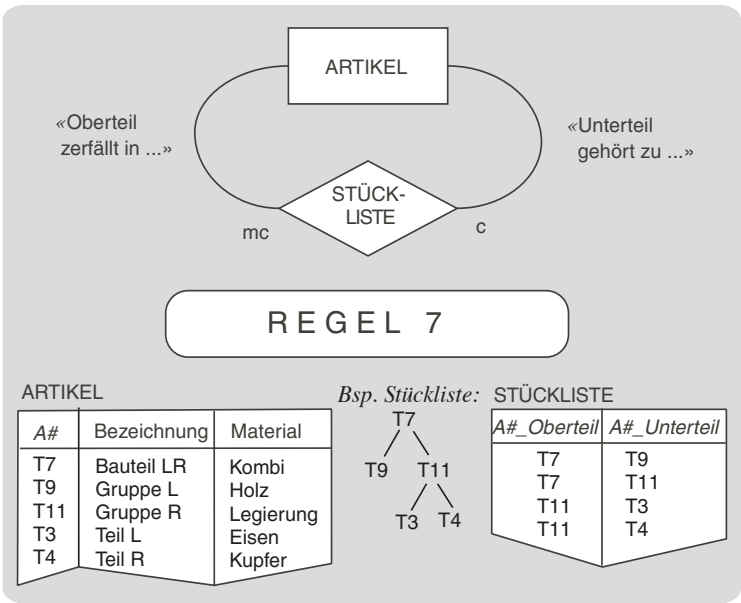
Eine hierarchische Aggregationsstruktur ist in Abb. 2-15 illustriert. In der Tabelle ARTIKEL werden die einzelnen Komponenten mit ihren Materialeigenschaften aufgeführt. Die Tabelle STÜCKLISTE definiert die hierarchische Beziehungsstruktur der jeweiligen Baugruppen. So ist der Artikel T7 aus zwei Teilgruppen zusammengesetzt, nämlich aus T9 und T11. Die Teilgruppe T11 wiederum setzt sich aus den Teilen T3 und T4 zusammen.

Die beiden Tabellen ARTIKEL und STÜCKLISTE könnten in einer einzigen Tabelle ARTIKELSTRUKTUR zusammengefasst werden. Dabei würde man zu den Arteikeleigenschaften je die Artikelnummer des eindeutig übergeordneten Artikels aufführen.

Die
hierarchische
Struktur der
Stückliste

Stückliste
zusammen-
gefasst in einer
Tabelle möglich

Abb. 2-15
Hierarchische
Artikelstruktur in
Tabellenform



2.4 Abhängigkeiten und Normalformen

2.4.1 Sinn und Zweck von Normalformen

*Zur Theorie der
Normalformen*

Die Untersuchung des Relationenmodells hat eine eigentliche Datenbanktheorie hervorgebracht, bei der die formalen Aspekte, teils losgelöst von realen Gegebenheiten, präzise beschrieben werden. Ein bedeutendes Teilgebiet dieser Datenbanktheorie bilden die so genannten *Normalformen* (engl. *normal forms*). Mit diesen werden innerhalb von Tabellen Abhängigkeiten studiert und aufgezeigt, oft zur Vermeidung redundanter Information und damit zusammenhängender Anomalien.

Zur Redundanz eines Merkmals

*Wann ist eine
Tabelle
redundant?*

Ein Merkmal einer Tabelle ist redundant, wenn einzelne Werte dieses Merkmals innerhalb der Tabelle *ohne Informationsverlust weggelassen* werden können.

Betrachten wir dazu als Beispiel eine Tabelle ABTEILUNGSMITARBEITER, die neben Mitarbeiternummer, Name, Straße und Ort für je-

ABTEILUNGSMITARBEITER

| M# | Name | Straße | Ort | A# | Bezeichnung |
|-----|-----------|--------------|-------------|----|-------------|
| M19 | Schweizer | Hauptstraße | Frenkendorf | A6 | Finanz |
| M1 | Meier | Lindenstraße | Liestal | A3 | Informatik |
| M7 | Huber | Mattenweg | Basel | A5 | Personal |
| M4 | Becker | Wasserweg | Liestal | A6 | Finanz |

Abb. 2-16
Redundante und
anomalien-
trächtige Tabelle

den Mitarbeitenden noch seine Abteilungsnummer samt Abteilungsbezeichnung enthält.

In Abb. 2-16 ist für jeden Mitarbeiter aus der Abteilung A6 der Abteilungsname Finanz aufgelistet. Entsprechendes gilt für die anderen Abteilungen, da unserer Annahme gemäß in einer Abteilung mehrere Mitarbeiter tätig sind. Das Merkmal Bezeichnung ist also redundant, da mehrmals ein und dieselbe Abteilungsbezeichnung in der Tabelle vorkommt. Es würde genügen, sich die Bezeichnung der Abteilungen in einer separaten Tabelle ein für allemal zu merken, anstatt diesen Sachverhalt für jeden Mitarbeitenden redundant mitzuführen.

Abteilungs-
bezeichnungen
sind redundant

Bei Tabellen mit redundanter Information können so genannte *Mutationsanomalien* auftreten. Möchten wir aus organisatorischen Überlegungen in der Tabelle ABTEILUNGSMITARBEITER von Abb. 2-16 eine neue Abteilung A9 mit der Bezeichnung Marketing definieren, so können wir diesen Sachverhalt nicht ohne weiteres in unserer Tabelle festhalten. Es liegt eine *Einfügeanomalie* vor, weil wir keine neue Tabellenzeile ohne eine eindeutige Mitarbeiternummer einbringen können.

Redundante
Tabellen führen
zu Anomalien

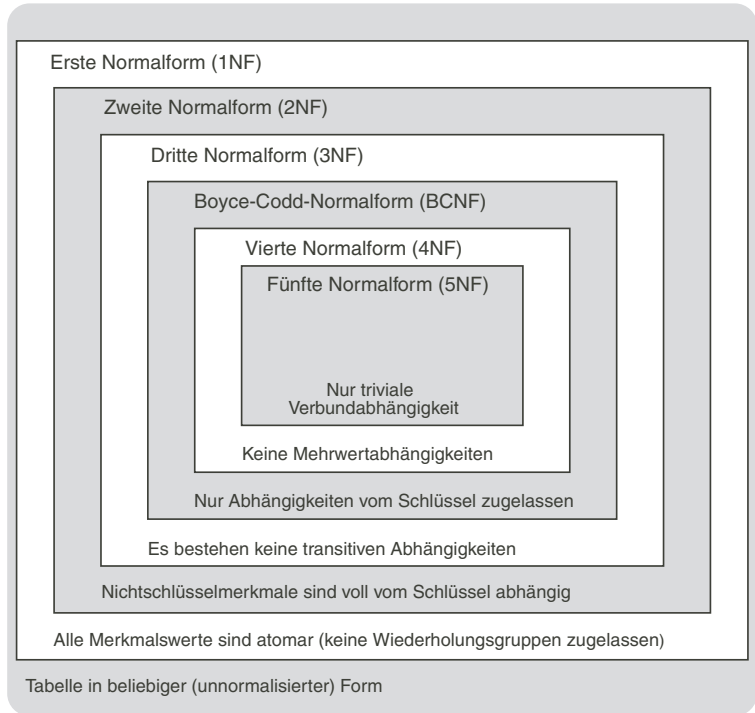
Von einer *Löschanomalie* spricht man, wenn ein Sachverhalt ungewollt verloren geht. Falls wir in unserer Tabelle ABTEILUNGSMITARBEITER sämtliche Mitarbeiter löschen, verlieren wir gleichzeitig die Abteilungsnummern mit ihren Bezeichnungen.

Schließlich gibt es auch *Änderungsanomalien*: Soll die Bezeichnung der Abteilung A3 von Informatik auf DV-Abteilung abgeändert werden, so muss bei sämtlichen Mitarbeitenden der Informatikabteilung dieser Namenswechsel vollzogen werden. Anders ausgedrückt: Obwohl nur ein einziger Sachverhalt eine Veränderung erfährt, muss die Tabelle ABTEILUNGSMITARBEITER an verschiedenen Stellen angepasst werden. Dieser Nachteil wird als Änderungsanomalie bezeichnet.

Probleme beim
Nachführen
redundanter
Tabellen

In den folgenden Ausführungen werden Normalformen diskutiert, die Redundanzen und Anomalien vermeiden helfen. Die Abb. 2-17 gibt

Abb. 2-17
Übersicht über
die Normal-
formen und ihre
Charakterisie-
rung



Abhängigkeiten
können zu
Konflikten führen

eine Übersicht über die Normalformen und ihre Bedeutungen. Sie illustriert insbesondere, dass mit Hilfe der Normalformen verschiedene Abhängigkeiten zu Konfliktsituationen führen können. Diese Abhängigkeiten sollen in den nächsten Abschnitten mit Beispielen eingehender studiert werden.

Wie Abb. 2-17 zeigt, schränken die Normalformen die Menge der zugelassenen Tabellen mehr und mehr ein. So muss beispielsweise eine Tabelle oder ein ganzes Datenbankschema in dritter Normalform die erste und die zweite Normalform erfüllen, und zusätzlich dürfen keine so genannten transitiven Abhängigkeiten unter den Nichtschlüsselmerkmalen auftreten.

Zur praktischen
Relevanz der
dritten
Normalform

Beim Studium der Normalformen ist wesentlich, dass nicht alle Normalformen dieselbe Bedeutung haben. *Meistens beschränkt man sich in der Praxis auf die ersten drei Normalformen*, da Mehrwert- oder Verbundabhängigkeiten kaum in Erscheinung treten; entsprechende Anwendungsbeispiele sind selten. Die vierte und die fünfte Normalform werden deshalb nur kurz diskutiert.

Das Verständnis für die Normalformen hilft, die in den vorigen Abschnitten erläuterten Abbildungsregeln zu untermauern. Tatsächlich bleiben, wie wir sehen werden, bei einer sauberen Definition

eines Entitäten-Beziehungsmodells und einer konsequenten Anwendung der diskutierten Abbildungsregeln die Normalformen jederzeit erfüllt. Mit anderen Worten, wir können *auf das Prüfen der Normalformen bei jedem einzelnen Entwurfsschritt weitgehend verzichten*, falls ein Entitäten-Beziehungsmodell erstellt und anhand der Regeln 1 bis 7 auf das zugehörige Datenbankschema abgebildet wird.

2.4.2

Funktionale Abhängigkeiten

Die erste Normalform bildet die Ausgangsbasis für die übrigen Normalformen und lautet wie folgt:

Erste Normalform (1NF)

Eine Tabelle ist in erster Normalform, falls die *Wertebereiche der Merkmale* atomar sind. Die erste Normalform verlangt, dass jedes Merkmal Werte aus einem unstrukturierten Wertebereich bezieht. Somit dürfen keine Mengen, Aufzählungen oder Wiederholungsgruppen in den einzelnen Merkmalen vorkommen.

*1NF verlangt
atomare
Merkmalswerte*

Die Tabelle PROJEKTMITARBEITER aus Abb. 2-18 ist vorerst nicht normalisiert, enthält sie doch pro Mitarbeitertupel mehrere Nummern von Projekten, welche vom jeweiligen Mitarbeiter zu bearbeiten sind. Die unnormalisierte Tabelle lässt sich auf einfache Art in die erste Normalform bringen, indem für jedes Projektengagement ein eigenes Tupel erzeugt wird. Dabei fällt auf, dass der Schlüssel der Tabelle PROJEKTMITARBEITER beim Überführen in die erste Normalform erweitert werden muss, da wir für das eindeutige Identifizieren der Tupel sowohl die Mitarbeiter- als auch die Projektnummer benötigen. Es ist üblich (aber nicht zwingend), bei zusammengesetzten Schlüsseln beide Schlüsselteile am Tabellenanfang direkt hintereinander zu zeigen (vgl. Abb. 2-18).

*Überführen in die
erste Normalform*

Paradoxerweise erhalten wir bei der Anwendung der ersten Normalform eine Tabelle mit Redundanz. In Abb. 2-18 sind sowohl Namen wie Adressangaben der Mitarbeiter redundant, da sie bei jedem Projektengagement wiederholt werden. Die zweite Normalform schafft hier Abhilfe:

Zweite Normalform (2NF)

Eine Tabelle ist in zweiter Normalform, wenn sie in erster Normalform ist und wenn jedes Nichtschlüsselmerkmal von jedem Schlüssel *voll funktional abhängig* bleibt.

*2NF fordert volle
funktionale
Abhängigkeit*

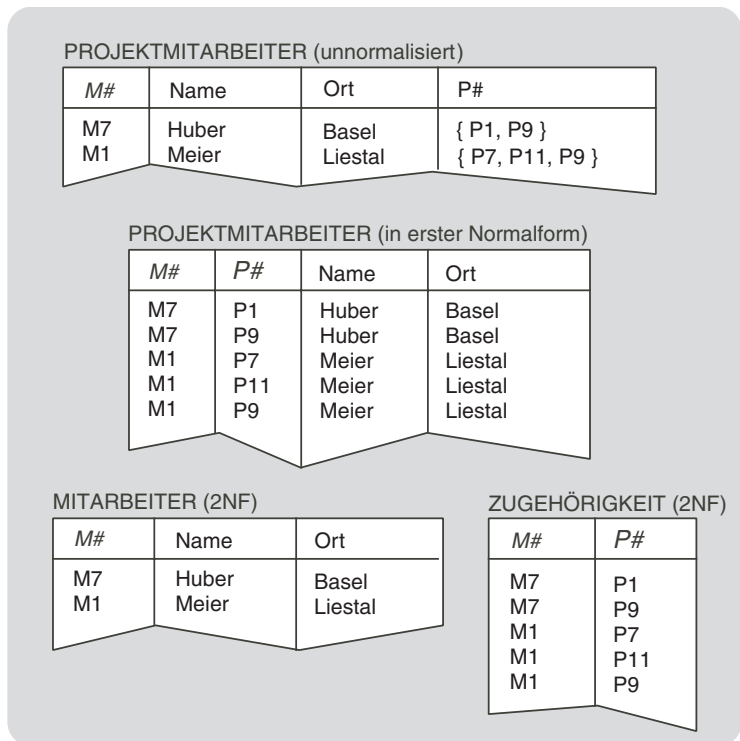
*Funktionale
Abhängigkeit
zwischen
Merkmalen*

Ein Merkmal B ist *funktional abhängig* vom Merkmal A, falls zu jedem Wert von A genau ein Wert aus B existiert (abgekürzt durch die Schreibweise $A \rightarrow B$). Die *funktionale Abhängigkeit* (engl. *functional dependency*) B von A verlangt somit, dass jeder Wert von A auf eindeutige Art einen Wert von B bestimmt. Bekanntlich haben alle Identifikationsschlüssel die Eigenschaft, dass die Nichtschlüsselmerkmale eindeutig vom Schlüssel abhängig sind. Es gilt also allgemein für einen Identifikationsschlüssel S und für ein beliebiges Merkmal B einer bestimmten Tabelle die funktionale Abhängigkeit $S \rightarrow B$.

*Voll funktionale
Abhängigkeit bei
zusammen-
gesetzten
Schlüsseln*

Bei zusammengesetzten Schlüsseln müssen wir den Begriff der funktionalen Abhängigkeit zum Begriff der vollen funktionalen Abhängigkeit wie folgt erweitern: Ein Merkmal B ist *voll funktional abhängig* von einem aus S1 und S2 zusammengesetzten Schlüssel (abgekürzt durch die Schreibweise $(S1, S2) \Rightarrow B$), falls B funktional abhängig vom Gesamtschlüssel, nicht jedoch von seinen Teilschlüsseln ist. Volle funktionale Abhängigkeit besteht also dann, wenn der zusammengesetzte Schlüssel allein in seiner Gesamtheit die übrigen Nichtschlüsselattribute eindeutig bestimmt. Es muss die funktionale Abhängigkeit $(S1, S2) \rightarrow B$ gelten, hingegen dürfen weder $S1 \rightarrow B$ noch

Abb. 2-18
Tabellen in erster
und zweiter
Normalform



S2->B auftreten. Die *volle funktionale Abhängigkeit* (engl. *full functional dependency*) eines Merkmals vom Schlüssel verbietet also, dass es von seinen Teilen funktional abhängig ist.

Betrachten wir die in die erste Normalform überführte Tabelle PROJEKTMITARBEITER in Abb. 2-18. Sie enthält den zusammengesetzten Schlüssel (M#,P#) und muss somit auf ihre volle funktionale Abhängigkeit hin überprüft werden. Für den Namen sowie den Wohnort der Projektmitarbeiter gelten die funktionalen Abhängigkeiten (M#,P#)->Name sowie (M#,P#)->Ort. Jede Kombination der Mitarbeiternummer mit der Projektnummer bestimmt nämlich eindeutig einen Mitarbeiternamen oder einen Ort. Hingegen wissen wir, dass sowohl der Name als auch der Wohnort des Mitarbeiters nichts mit den Projektnummern zu tun haben. Also müssen die beiden Merkmale Name und Ort von einem Teil des Schlüssels funktional abhängig sein, d.h., es gilt $M\# \rightarrow \text{Name}$ und $M\# \rightarrow \text{Ort}$. Dies ist ein Widerspruch zur Definition der vollen funktionalen Abhängigkeit. Somit ist die Tabelle PROJEKTMITARBEITER nicht in zweiter Normalform.

Teilabhängigkeiten sind unerwünscht

Ist eine Tabelle mit einem zusammengesetzten Schlüssel nicht in zweiter Normalform, so muss sie in Teiltabellen zerlegt werden. Dabei fasst man die Merkmale, die von einem Teilschlüssel abhängig sind, und diesen Teilschlüssel in einer eigenständigen Tabelle zusammen. Die Resttabelle mit dem zusammengesetzten Schlüssel und eventuell weiteren Beziehungsattributen belässt man als übrigbleibende Tabelle.

Redundante Tabellen müssen zerlegt werden

Im Beispiel aus Abb. 2-18 erhält man die beiden Tabellen MITARBEITER und ZUGEHÖRIGKEIT. Beide Tabellen sind in erster und zweiter Normalform. In der Tabelle MITARBEITER tritt kein zusammengesetzter Schlüssel mehr auf, und die zweite Normalform ist offensichtlich erfüllt. Die Tabelle ZUGEHÖRIGKEIT hat keine Nichtschlüsselattribute, so dass sich eine Überprüfung der zweiten Normalform auch hier erübrigt.

2.4.3 Transitive Abhängigkeiten

In Abb. 2-19 zeigen wir unsere anfangs diskutierte Tabelle ABTEILUNGSMITARBEITER, die neben Mitarbeiterangaben auch Abteilungsinformationen enthält. Wir erkennen sofort, dass sich die Tabelle in erster und zweiter Normalform befindet. Da kein zusammengesetzter Schlüssel auftritt, müssen wir die Eigenschaft der vollen funktionalen Abhängigkeit nicht überprüfen. Doch offensichtlich tritt das Merkmal Bezeichnung redundant auf. Dieser Missstand ist

Redundante Tabelle trotz zweiter Normalform

mit der dritten Normalform zu beheben, die wir in diesem Abschnitt näher erläutern.

*3NF verbietet
transitive
Abhängigkeiten*

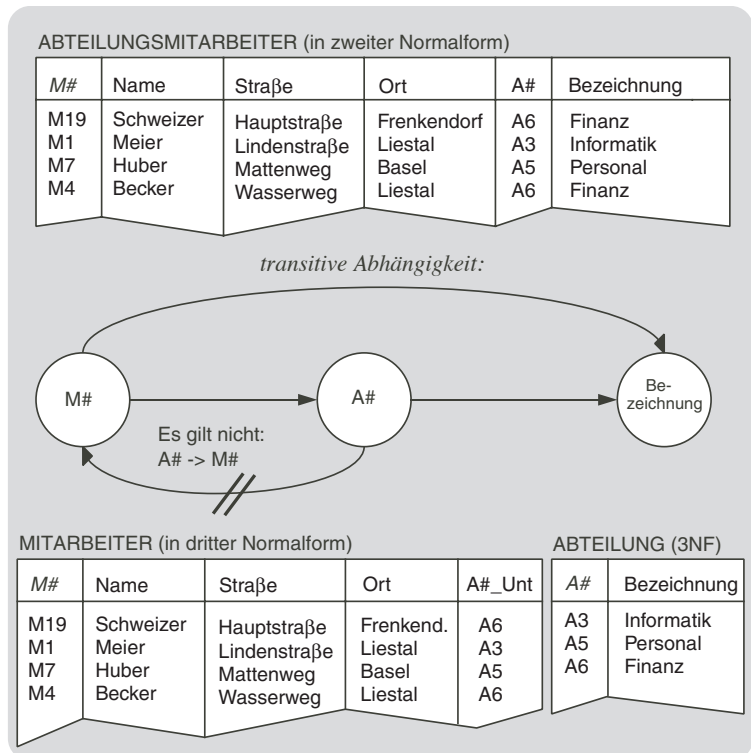
Dritte Normalform (3NF)

Eine Tabelle ist in dritter Normalform, wenn sie in zweiter Normalform ist und *kein Nichtschlüsselmerkmal von irgendeinem Schlüssel transitiv abhängig* ist.

*Bedeutung
transitiver
Abhängigkeit*

Einmal mehr definieren wir eine Normalform durch ein Abhängigkeitskriterium: Transitiv abhängig bedeutet, *über Umwege funktional abhängig* zu sein. Beispielsweise ist unser Merkmal Bezeichnung über den Umweg Abteilungsnummer von der Mitarbeiternummer funktional abhängig. Wir erkennen nämlich zwischen der Mitarbeiternummer und der Abteilungsnummer eine funktionale Abhängigkeit sowie eine weitere zwischen Abteilungsnummer und Bezeichnung. Die beiden funktionalen Abhängigkeiten $M\# \rightarrow A\#$ und $A\# \rightarrow \text{Bezeichnung}$ lassen sich also über die Abteilungsnummer zur transitiven Abhängigkeit $M\# \rightarrow \text{Bezeichnung}$ zusammenfügen.

Abb. 2-19
Transitive
Abhängigkeit und
dritte Normalform



Allgemein ist bei zwei funktionalen Abhängigkeiten $A \rightarrow B$ und $B \rightarrow C$ mit einem gemeinsamen Merkmal B die zusammengesetzte Abhängigkeit $A \rightarrow C$ ebenfalls funktional. Falls nämlich A die Werte in B eindeutig bestimmt und B diejenigen in C, so vererbt sich diese Eigenschaft von A auf C. Die Abhängigkeit $A \rightarrow C$ ist somit sicher funktional, und sie wird zusätzlich transitiv genannt, wenn neben den beiden funktionalen Abhängigkeiten $A \rightarrow B$ und $B \rightarrow C$ nicht gleichzeitig A funktional von B abhängig ist.⁴ Somit gilt für die *transitive Abhängigkeit* (engl. *transitive dependency*) die Definition: Das Merkmal C ist transitiv abhängig von A, falls B funktional abhängig von A, C funktional abhängig von B und nicht gleichzeitig A funktional abhängig von B ist.

Wie wird eine transitive Abhängigkeit definiert?

Aus dem Beispiel ABTEILUNGSMITARBEITER in Abb. 2-19 geht hervor, dass das Merkmal Bezeichnung transitiv vom Merkmal M# abhängig ist. Deshalb ist definitionsgemäß die Tabelle ABTEILUNGSMITARBEITER nicht in dritter Normalform. Durch Zerlegung befreien wir sie von der transitiven Abhängigkeit, indem wir das redundante Merkmal Bezeichnung zusammen mit der Abteilungsnummer als eigenständige Tabelle ABTEILUNG führen. Die Abteilungsnummer bleibt als Fremdschlüssel mit dem Merkmalsnamen «A#_Unterstellung» in der Resttabelle MITARBEITER. Die Beziehung zwischen Mitarbeiter und Abteilung ist dadurch nach wie vor gewährleistet.

Transitive Abhängigkeiten durch Teiltabellen eliminieren

2.4.4 Mehrwertige Abhängigkeiten

Mit der zweiten und dritten Normalform gelingt es, Redundanzen bei den Nichtschlüsselattributen zu eliminieren. Das Aufspüren redundanter Informationen darf aber nicht bei den Nichtschlüsselmerkmalen stehen bleiben, da auch zusammengesetzte Schlüssel redundant auftreten können.

Eine allenfalls erforderliche Erweiterung der dritten Normalform wird aufgrund der Arbeiten von Boyce und Codd als «*Boyce-Codd-Normalform*» oder BCNF bezeichnet. Eine solche kommt zum Tragen, wenn mehrere Schlüsselkandidaten in ein und derselben Tabelle auftreten. So existieren Tabellen mit sich gegenseitig überlappenden Schlüsseln, die zwar in dritter Normalform sind, aber die Boyce-Codd-Normalform verletzen. In einem solchen Fall muss die Tabelle

Zur Boyce-Codd-Normalform

⁴ Schlüsselkandidaten sind immer vom Identifikationsschlüssel abhängig und umgekehrt, vgl. als Beispiel die Tabelle MITARBEITER mit je einer Mitarbeiter- und AHV-Nummer.

aufgrund der Schlüsselkandidaten zerlegt werden. Ein Hinweis auf die Literatur in Abschnitt 2.8 soll genügen.

*Beispiel einer
mehrwertigen
Abhängigkeit*

Eine weitere Normalform ergibt sich beim Studium von *mehrwertigen Abhängigkeiten* (engl. *multi-valued dependency*) zwischen einzelnen Merkmalen des Schlüssels. Obwohl in der Praxis mehrwertige Abhängigkeiten eine untergeordnete Rolle spielen, zeigt Abb. 2-20 diese Normalform kurz an einem einfachen Beispiel. Die Ausgangstabelle METHODE liegt als unnormalisierte Tabelle vor, da neben der Angabe der Methode eventuell mehrere Autoren und mehrere Begriffe auftreten. So enthält die Methode Struktogramm die Autoren Nassi und Shneiderman. Gleichzeitig ist sie durch die drei Sprachelemente Sequenz, Iteration und Verzweigung ebenfalls mit mehreren Wertangaben charakterisiert.

Wir überführen die unnormalisierte Tabelle in die erste Normalform, indem die Mengen {Nassi, Shneiderman} und {Sequenz, Iteration, Verzweigung} aufgelöst werden. Dabei fällt auf, dass die neue Tabelle aus lauter Schlüsselmerkmalen besteht. Diese Tabelle ist nicht nur in erster Normalform, sondern auch in zweiter und in dritter. Hingegen zeigt die Tabelle redundante Informationen, obwohl sie

*Abb. 2-20
Tabelle mit
mehrwertigen
Abhängigkeiten*

| METHODE (unnormalisiert) | | |
|--------------------------|----------------------|-----------------------------------|
| Methode | Autor | Begriff |
| Struktogramm | {Nassi, Shneiderman} | {Sequenz, Iteration, Verzweigung} |
| Datenmodell | {Chen} | {Entitätsmenge, Beziehungsmenge} |

| METHODE (in dritter Normalform) | | |
|---------------------------------|-------------|-----------------|
| Methode | Autor | Begriff |
| Struktogramm | Nassi | Sequenz |
| Struktogramm | Nassi | Iteration |
| Struktogramm | Nassi | Verzweigung |
| Struktogramm | Shneiderman | Sequenz |
| Struktogramm | Shneiderman | Iteration |
| Struktogramm | Shneiderman | Verzweigung |
| Datenmodell | Chen | Entitätsmenge |
| Datenmodell | Chen | Beziehungsmenge |

| METHODIKER (4NF) | |
|------------------|-------------|
| Methode | Autor |
| Struktogramm | Nassi |
| Struktogramm | Shneiderman |
| Datenmodell | Chen |

| METHODIK (4NF) | |
|----------------|-----------------|
| Methode | Begriff |
| Struktogramm | Sequenz |
| Struktogramm | Iteration |
| Struktogramm | Verzweigung |
| Datenmodell | Entitätsmenge |
| Datenmodell | Beziehungsmenge |

in dritter Normalform ist. Beispielsweise merken wir uns pro Autor des Struktogramms, dass die drei Sprachelemente (Begriffe) Sequenz, Iteration und Verzweigung Anwendung finden. Umgekehrt merken wir uns pro Begriff des Struktogramms die beiden Autoren Nassi und Shneiderman. Mit anderen Worten haben wir es hier mit paarweise mehrwertigen Abhängigkeiten zu tun, die eliminiert werden müssen.

Für eine Tabelle mit den Merkmalen A, B und C gilt die folgende Definition für mehrwertige Abhängigkeiten: Ein Merkmal C ist *mehrwertig abhängig* vom Merkmal A (ausgedrückt durch die Schreibweise $A \twoheadrightarrow C$), falls zu jeder Kombination eines bestimmten Wertes aus A mit einem beliebigen Wert aus B eine identische Menge von Werten aus C erscheint.

Auf unser Beispiel von Abb. 2-20 bezogen können wir folgern, dass das Merkmal «Begriff» mehrwertig abhängig vom Merkmal «Methode» ist; es gilt somit $\text{«Methode»} \twoheadrightarrow \text{«Begriff»}$. Kombinieren wir das Struktogramm mit dem Autor Nassi, so erhalten wir dieselbe Menge {Sequenz, Iteration, Verzweigung} wie bei der Kombination des Struktogramms mit dem Autor Shneiderman. Zusätzlich gilt die mehrwertige Abhängigkeit zwischen den Merkmalen Autor und Methode in der Form $\text{«Methode»} \twoheadrightarrow \text{«Autor»}$. Kombinieren wir hier eine bestimmte Methode wie Struktogramm mit einem beliebigen Begriff wie Sequenz, so erhalten wir die Autoren Nassi und Shneiderman. Dieselbe Autorenschaft tritt zu Tage durch die Kombination von Struktogramm mit den Begriffen Iteration oder Verzweigung.

Treten mehrwertige Abhängigkeiten in einer Tabelle auf, so können Redundanzen und Anomalien entstehen. Um diese zu tilgen, muss eine weitere Normalform berücksichtigt werden:

Vierte Normalform (4NF)

Die *vierte Normalform* lässt es nicht zu, dass in ein und derselben Tabelle *zwei echte und voneinander verschiedene mehrwertige Abhängigkeiten* vorliegen.

Auf unser Beispiel bezogen muss die Tabelle METHODE also in die zwei Teiltabellen METHODIKER und METHODIK aufgeteilt werden. Erstere verbindet Methoden mit Autoren, letztere Methoden mit Begriffen. Offensichtlich sind beide Tabellen redundanzfrei und in vierter Normalform.

*Definition
mehrwertig
abhängiger
Merkmale*

*Beispiel zweier
mehrwertiger
Abhängigkeiten*

*4NF untersagt
paarweise
mehrwertige
Abhängigkeiten*

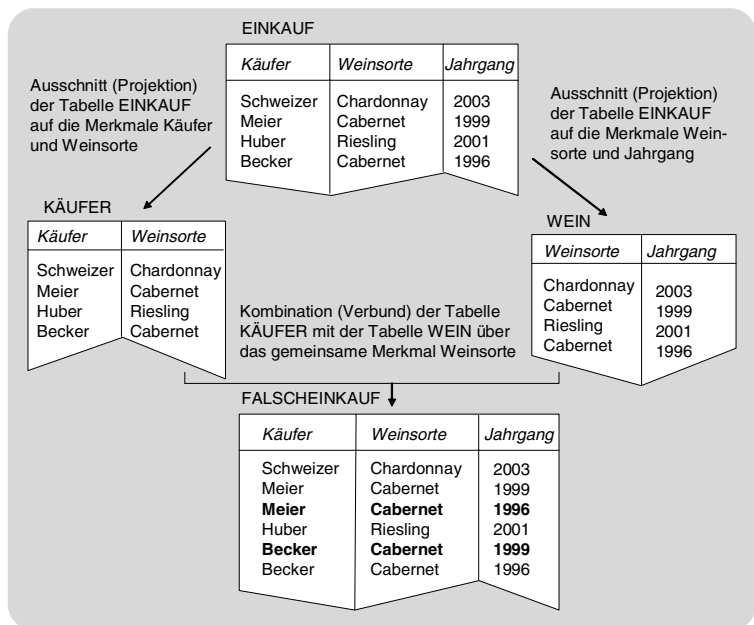
2.4.5 Verbundabhängigkeit

Verlustfreie
Kombination von
Tabellen

Es ist gar nicht selbstverständlich, dass bei einer Zerlegung einer Tabelle in Teiltabellen nicht bestimmte Sachverhalte verloren gehen. Aus diesem Grunde hat man Kriterien gesucht, die eine *Aufteilung von Tabellen ohne Informationsverlust* garantieren, d.h. so, dass die ursprünglichen Sachverhalte über die Teiltabellen erhalten bleiben.

In Abb. 2-21 ist eine solch unerlaubte Zerlegung der Tabelle EINKAUF in die beiden Teiltabellen KÄUFER und WEIN dargestellt: Die Tabelle EINKAUF enthält die getätigten Weinkäufe von Schweizer, Meier, Huber und Becker. Kombiniert man die davon abgeleiteten Teiltabellen KÄUFER und WEIN über das gemeinsame Merkmal Weinsorte, so erhält man die Tabelle FALSCH EINKAUF. Diese Tabelle zeigt gegenüber der Ursprungstabelle EINKAUF einen Informationsverlust auf, da hier suggeriert wird, dass Meier und Becker dieselben Weinpräferenzen besitzen und je den Cabernet 1996 wie 1999 eingekauft haben (die fälschlichen Tabelleneinträge für Meier und Becker sind in Abb. 2-21 durch fette Schreibweise hervorgehoben).

Abb. 2-21
Unerlaubte
Zerlegung
der Tabelle
EINKAUF



Das Zerlegen der Tabelle EINKAUF in die Tabelle KÄUFER geschieht, indem man die Einkaufstabelle auf die beiden Merkmale Käufer und Weinsorte reduziert. Der dazu notwendige Projektionsoperator ist ein Filteroperator, der Tabellen vertikal in Teiltabellen zerlegt (vgl. Abb. 3-3 resp. Abschnitt 3.2.3). Entsprechend erhält man die Teiltabelle WEIN durch die Projektion der Tabelle EINKAUF auf die beiden Merkmale Weinsorte und Jahrgang. Neben dem Zerlegen von Tabellen in Teiltabellen (Projektion) können Teiltabellen auch zu Tabellen kombiniert werden (Verbundoperator). Ein Beispiel ist in Abb. 2-21 aufgezeigt: Die beiden Tabellen KÄUFER und WEIN können dank dem gemeinsamen Merkmal Weinsorte kombiniert werden, indem alle Tupeleinträge aus der Tabelle KÄUFER mit den entsprechenden Einträgen aus der Tabelle WEIN ergänzt werden. Als Resultat bekommt man die Verbundtabelle FALSCH EINKAUF (zum Verbund oder Join siehe Abb. 3-3 resp. Abschnitt 3.2.3). Die Tabelle listet alle Einkäufe der Personen Schweizer, Meier, Huber und Becker. Allerdings enthält sie auch die beiden Einkäufe (Meier, Cabernet, 1996) und (Becker, Cabernet, 1999), die gar nie getätigt wurden. Abhilfe für solche Konfliktfälle schafft die fünfte Normalform resp. das Studium der Verbundabhängigkeit.

*Beispiel für
Projektion und
Verbund*

Fünfte Normalform (5NF)

Eine Tabelle ist in fünfter Normalform, wenn sie *keine Verbundabhängigkeit* aufweist.

Die fünfte Normalform (oft Projektion-Verbund-Normalform genannt) gibt an, unter welchen Umständen eine Tabelle problemlos in Teiltabellen zerlegt und gegebenenfalls ohne Einschränkung (Informationsverlust) wieder rekonstruiert werden kann. Die Zerlegung wird mit Hilfe des Projektionsoperators durchgeführt, die Rekonstruktion mit dem Verbundoperator.

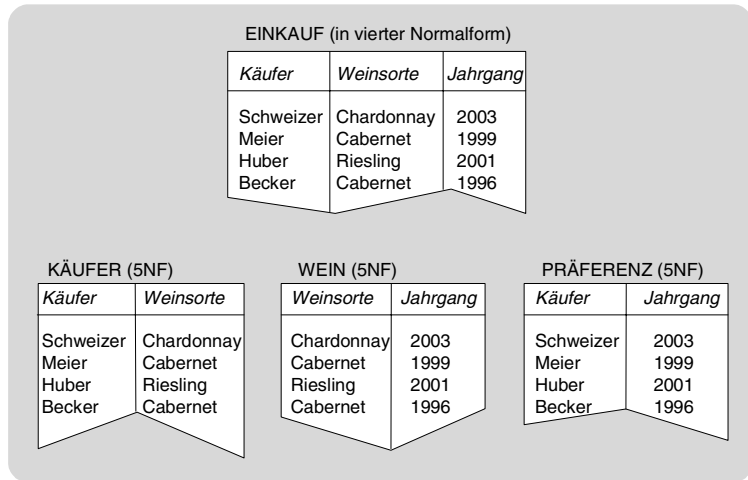
*5NF fordert
verlustfreien
Verbund*

Um bei der Rekonstruktion keine falschen Sachverhalte zu erhalten, muss auf Verbundabhängigkeit geprüft werden: Eine Tabelle R mit den Merkmalen A, B und C erfüllt die *Verbundabhängigkeit* (engl. *join dependency*), falls die projizierten Teiltabellen R1(A,B) und R2(B,C) beim Verbund über das gemeinsame Merkmal B die Ursprungstabelle R ergeben. Man spricht in diesem Zusammenhang auch von einem *verlustlosen Verbund* (engl. *lossless join*). Wie bereits diskutiert, ist die Tabelle EINKAUF aus Abb. 2-21 nicht verbundabhängig und somit nicht in fünfter Normalform.

*Wie ist Verbund-
abhängigkeit
definiert?*

In Abb. 2-22 ist die Tabelle EINKAUF gegeben, die in vierter Normalform ist. Da sie das Kriterium der Verbundabhängigkeit verletzt, muss sie in die fünfte Normalform überführt werden. Dies ist möglich, falls die drei Teiltabellen KÄUFER, WEIN und PRÄFERENZ

Abb. 2-22
Beispiel von
Tabellen in fünfter
Normalform



durch entsprechende Projektionen aus der Tabelle EINKAUF gewonnen werden. Eine Nachprüfung bestätigt, dass ein Verbund der drei Tabellen tatsächlich die Ursprungstabelle EINKAUF ergibt. Dazu müssen die beiden Tabellen KÄUFER und WEIN über das gemeinsame Merkmal Weinsorte kombiniert werden. Die Resultatstabelle kann dann mit einem weiteren Verbund über das Merkmal Jahrgang mit der Tabelle PRÄFERENZ zur Einkaufstabelle kombiniert werden.

*Analytische
versus
synthetische
Datenanalyse*

Allgemein ist die Frage von Interesse, ob man beim Datenbankentwurf anstelle von Zerlegungsregeln oder Normalformen (analytisches Vorgehen) umgekehrt auch synthetisch vorgehen kann. Tatsächlich gibt es Algorithmen, die das Zusammensetzen von Teiltabellen zu Tabellen in dritter oder höherer Normalform erlauben. Solche Zusammensetzungsregeln erlauben, ausgehend von einem Satz von Abhängigkeiten entsprechende Datenbankschemas aufbauen zu können (synthetisches Vorgehen). Sie begründen formell, dass der Entwurf relationaler Datenbankschemas sowohl top down (analytisch) als auch bottom up (synthetisch) erfolgreich vorgenommen werden kann. Leider unterstützen nur wenige CASE-Werkzeuge beide Vorgehensweisen; der Datenbankarchitekt tut deshalb gut daran, die Korrektheit seines Datenbankentwurfs manuell zu überprüfen.

2.5

Strukturelle Integritätsbedingungen

Unter dem Begriff *Integrität* oder *Konsistenz* (engl. *integrity, consistency*) versteht man die Widerspruchsfreiheit von Datenbeständen. Eine Datenbank ist integer oder konsistent, wenn die gespeicherten Daten fehlerfrei erfasst sind und den gewünschten Informationsgehalt korrekt wiedergeben. Die Datenintegrität ist dagegen verletzt, wenn Mehrdeutigkeiten oder widersprüchliche Sachverhalte zu Tage treten. Bei einer konsistenten Tabelle MITARBEITER setzen wir beispielsweise voraus, dass Mitarbeiternamen, Straßenbezeichnungen, Ortsangaben etc. korrekt sind und real auch existieren.

Strukturelle Integritätsbedingungen zur Gewährleistung der Integrität sind solche Regeln, die durch das Datenbankschema selbst ausgedrückt werden können. Bei relationalen Datenbanken zählen die folgenden Integritätsbedingungen zu den strukturellen:

- Die *Eindeutigkeitsbedingung*: Jede Tabelle besitzt einen Identifikationsschlüssel (Merkmal oder Merkmalskombination), der jedes Tupel in der Tabelle auf eindeutige Art bestimmt.
- Die *Wertebereichsbedingung*: Die Merkmale einer Tabelle können nur Datenwerte aus einem vordefinierten Wertebereich annehmen.
- Die *referenzielle Integritätsbedingung*: Jeder Wert eines Fremdschlüssels muss effektiv als Schlüsselwert in der referenzierten Tabelle existieren.

Die *Eindeutigkeitsbedingung* verlangt für jede Tabelle einen festgelegten Schlüssel. Natürlich können innerhalb derselben Tabelle mehrere Schlüsselkandidaten vorkommen. In diesem Fall muss aufgrund der Eindeutigkeitsbedingung ein Schlüssel als Primärschlüssel deklariert werden. Das Prüfen auf Eindeutigkeit von Primärschlüsseln selbst wird vom Datenbanksystem vorgenommen.

Die *Wertebereichsbedingung* kann hingegen nicht vollständig vom Datenbanksystem gewährleistet werden. Zwar können die Wertebereiche einzelner Tabellenspalten spezifiziert werden, doch decken solche Angaben nur einen geringen Teil der Validierungsregeln ab. Bei der Prüfung von Ortsbezeichnungen oder Straßenangaben auf Korrektheit ist es mit der Definition eines Wertebereiches nicht getan. So sagt beispielsweise die Spezifikation «CHARACTER (20)» für eine Zeichenkette nichts über sinnvolle Orts- oder Straßennamen aus. Wie weit man bei einer Validierung von Tabelleninhalten gehen möchte, bleibt größtenteils dem Anwender überlassen.

Integrität garantiert Widerspruchsfreiheit

Was sind strukturelle Integritätsbedingungen?

Eindeutigkeit muss vom Datenbanksystem garantiert bleiben

Wie lassen sich Wertebereichsbedingungen überprüfen?

| | |
|---|---|
| <p>Zur Bedeutung von Wertebereichsbedingungen</p> | <p>Eine bedeutende Unterstützung für die Wertebereichsbedingung bieten <i>Aufzählungstypen</i>. Dabei werden sämtliche Datenwerte, die das Merkmal annehmen kann, in einer Liste angegeben. Beispiele von Aufzählungstypen sind durch die Merkmalskategorien (Attribute) BERUF = {Programmierer, Analytiker, Organisator} oder JAHRGANG = {1950..1990} gegeben. Die meisten heutigen Datenbanksysteme unterstützen solche Validierungsregeln.</p> |
| <p>Was ist referenzielle Integrität?</p> | <p>Eine wichtige Klasse von Prüfregeln wird unter dem Begriff <i>referenzielle Integrität</i> (engl. <i>referential integrity</i>) zusammengefasst. Eine relationale Datenbank erfüllt die referenzielle Integritätsbedingung, wenn jeder Wert eines Fremdschlüssels als Wert beim zugehörigen Primärschlüssel vorkommt. Die Abb. 2-23 macht dies deutlich: Die Tabelle ABTEILUNG hat die Abteilungsnummer A# als Primärschlüssel. Dieser wird in der Tabelle MITARBEITER als Fremdschlüssel mit dem Merkmal «A#_Unterstellung» verwendet, um die Abteilungszugehörigkeit eines Mitarbeiters zu fixieren. Die Fremd-Primärschlüssel-Beziehung erfüllt die Regel der referenziellen Integrität, falls alle Abteilungsnummern des Fremdschlüssels aus der Tabelle MITARBEITER in der Tabelle ABTEILUNG als Primärschlüsselwerte aufgeführt sind. In unserem Beispiel verletzt also keine Unterstellung die Regel der referenziellen Integrität.</p> |
| <p>Verletzung der Integrität</p> | <p>Nehmen wir an, wir möchten in die Tabelle MITARBEITER, wie sie Abb. 2-23 zeigt, ein neues Tupel «M20, Müller, Riesweg, Olten, A7» einfügen. Unsere Einfügeoperation wird abgewiesen, falls das Datenbanksystem die referenzielle Integrität unterstützt. Der Wert A7 wird nämlich als ungültig erklärt, da er in der referenzierten Tabelle ABTEILUNG nicht vorkommt.</p> |
| | <p>Die Gewährleistung der referenziellen Integrität hat neben der Einfügeproblematik natürlich Einfluss auf die übrigen Datenbankoperationen. Wird ein Tupel in einer Tabelle gelöscht, das von anderen Tupeln aus einer Fremdtabelle referenziert wird, so sind mehrere Varianten von Systemreaktionen möglich:</p> |
| <p>Restriktive Löschung</p> | <p>Bei der <i>restriktiven Löschung</i> (engl. <i>restricted deletion</i>) wird eine Löschoperation nicht ausgeführt, solange das zu löschende Tupel auch nur von einem Tupel einer anderen Tabelle referenziert wird. Möchten wir z.B. das Tupel «A6, Finanz» in Abb. 2-22 entfernen, so wird nach der restriktiven Löschregel unsere Operation verweigert, da die beiden Mitarbeiter Schweizer und Becker in der Abteilung A6 tätig sind.</p> |
| <p>Fortgesetzte Löschung</p> | <p>Anstelle der restriktiven Löschung kann für die Löschoperation eine <i>fortgesetzte Löschregel</i> (engl. <i>cascaded deletion</i>) bei der Spezifikation der beiden Tabellen MITARBEITER und ABTEILUNG verlangt werden. Eine fortgesetzte Löschung bewirkt, dass bei der Löschung eines Tupels <i>sämtliche abhängigen Tupel entfernt</i> werden. In unserem Beispiel aus Abb. 2-23 bedeutet eine fortgesetzte Löschregel bei der</p> |

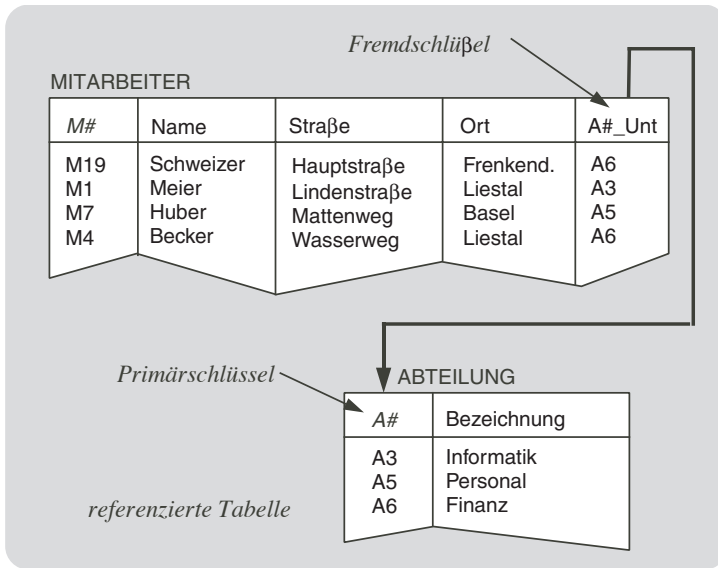


Abb. 2-23
Gewährleistung
der
referenziellen
Integrität

Eliminierung des Tupels «A6, Finanz», dass gleichzeitig die beiden Tupel «M19, Schweizer, Hauptstraße, Frenkendorf, A6» und «M4, Becker, Wasserweg, Liestal, A6» entfernt werden.

Eine weitere Löschregelvariante unter Berücksichtigung der referenziellen Integrität macht es möglich, dass referenzierte Fremdschlüssel bei Löschvorgängen auf den Wert «unbekannt» gesetzt werden. Diese dritte Löschregel wird nach der Behandlung der so genannten Nullwerte in Abschnitt 3.8 näher erläutert. Schließlich können an Änderungsoperationen einschränkende Bedingungen geknüpft werden, die die referenzielle Integrität jederzeit garantieren.

Löschen durch
Nullsetzen

2.6 Unternehmensweite Datenarchitektur

Verschiedene Untersuchungen haben offengelegt, dass während der Definitions- und Aufbauphase eines Informationssystems die künftigen Anwender zwar komplexe Funktionen und ausgeklügelte Verfahren priorisiert verlangen, beim Gebrauch der Anwendungssysteme hingegen der *Aussagekraft der Daten* (Abschnitt 1.5) größeres Gewicht beimessen. Die Datenarchitekten tun deshalb gut daran, wenn sie sich gleich zu Beginn den folgenden Fragen stellen: Welche Daten sollen in Eigenregie gesammelt, welche von externen Datenlieferanten bezogen werden? Wie lassen sich unter Berücksichtigung nationaler oder internationaler Gegebenheiten die Datenbestände

Zur Langlebigkeit
der Daten

klassifizieren und strukturieren? Wer ist verantwortlich für Unterhalt und Pflege der geografisch verstreuten Daten? Welche Aufgaben bestehen im internationalen Umfeld bezüglich Datenschutz und -sicherheit? Welche Rechte und Pflichten gelten für Datenaustausch und -weitergabe? Dies sind genau die Fragestellungen, die die Bedeutung der Datenarchitektur für das Unternehmen untermauern und die entsprechenden Datenmodelle in den Brennpunkt der Betrachtung rücken.

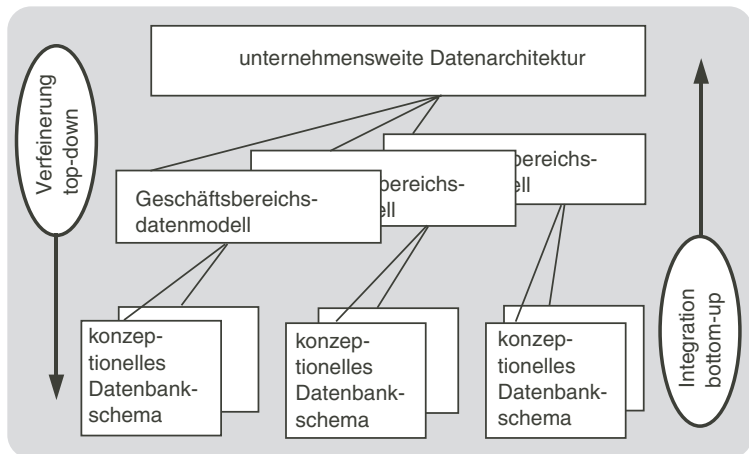
Wie begegnet man dem Datenchaos?

Aufgrund überbordender Anforderungen von der Anwenderseite her werden heute die Analyse- und Entwurfsarbeiten meistens nur für einzelne Zusatzfunktionen oder bestenfalls für bestimmte Anwendungsgebiete vorangetrieben. Dieses Vorgehen birgt gerade beim Einsatz von relationalen Datenbanken die Gefahr in sich, dass eine Fülle von Tabellen lediglich ad hoc oder nur aus einem lokalen Anwendungsbedarf heraus definiert werden. Eine *unkontrollierte Inflation von Tabellen* mit sich gegenseitig überlappenden oder mehrdeutigen Tabelleninhalten ist die Folge und führt zwangsläufig zu einem Datenchaos. Anwendungsübergreifende Auswertungen können kaum oder nur mit großem Aufwand vorgenommen werden.

Was verlangt eine unternehmensweite Datenarchitektur?

Eine *unternehmensweite Datenarchitektur* (engl. *corporate-wide data architecture*) schafft hier Abhilfe, da sie die wichtigsten Entitätsmengen und Beziehungen aus langfristiger und globaler Sicht des Unternehmens beschreibt. Sie soll einzelnen Geschäftsfeldern und lokalen Datensichten übergeordnet bleiben und das Verständnis für die Gesamtzusammenhänge des Unternehmens fördern. Diese Datenarchitektur und davon abgeleitete Datenmodelle bilden die Grundlage für eine abgestimmte Entwicklung von Informationssystemen.

Abb. 2-24
Verschiedene Abstraktionsebenen der unternehmensweiten Datenarchitektur



Die Abb. 2-24 stellt den Zusammenhang zwischen den bereichsübergreifenden und den anwendungsspezifischen Datenmodellen schematisch dar. Die unternehmensweite Datenarchitektur beschreibt die für das Unternehmen lebensnotwendigen Datenklassen und ihre Beziehungen. Daraus abgeleitet werden geschäftsbereichsspezifische Datenmodelle entwickelt, die pro Anwendung zu konzeptionellen Datenbankschemas führen. Solche Verfeinerungsschritte lassen sich in der Praxis natürlich nicht stur nach der top-down-Methode vollziehen, da die hierfür aufzubringende Zeit fehlt. Bei Änderungen bestehender Informationssysteme oder bei der Entwicklung neuer Anwendungen werden die konzeptionellen Datenbankschemas vielmehr nach der bottom-up-Methode mit den teils lückenhaft vorhandenen Geschäftsbereichsdatenmodellen und mit der unternehmensweiten Datenarchitektur abgestimmt und dadurch schrittweise auf die längerfristige Unternehmensentwicklung hin ausgerichtet.

Die unterschiedlichen Ebenen einer Datenarchitektur

Neben den in Eigenregie zu entwickelnden Geschäftsbereichsdatenmodellen existieren *Branchendatenmodelle*, die auf dem Softwaremarkt käuflich erworben werden können. Bei der Nutzung solcher Standardisierungsbemühungen reduziert sich der Aufwand für die Integration käuflich erworbener Anwendungssoftware. Gleichzeitig vereinfachen Branchenmodelle über Konzernbereiche oder Firmen hinweg den Austausch von Informationen.

Nutzung von Branchendatenmodellen

Wir beschreiben exemplarisch und lückenhaft eine unternehmensweite Datenarchitektur, indem wir uns auf die Entitätsmengen PARTNER, ROHSTOFF, PRODUKT, KONTRAKT und GESCHÄFTSFALL beschränken (vgl. Abb. 2-25):

Hauptkomponenten

- Unter PARTNER versteht man alle natürlichen und juristischen Personen, an denen das Unternehmen Interesse zeigt und über die für die Abwicklung der Geschäfte Informationen benötigt werden. Insbesondere zählen Kunden, Mitarbeiter, Lieferanten, Aktionäre, öffentlich-rechtliche Körperschaften, Institutionen und Firmen zur Entitätsmenge PARTNER.
- Mit ROHSTOFF werden Rohwaren, Metalle, Devisen, Wertschriften oder Immobilien bezeichnet, die der Markt anbietet und die im Unternehmen eingekauft, gehandelt oder veredelt werden. Allgemein können sich solche Güter auf materielle wie auf immaterielle Werte beziehen. Beispielsweise könnte sich eine Beratungsfirma mit bestimmten Techniken und entsprechendem Know-how eindecken.
- Mit PRODUKT soll die Produkt- oder Dienstleistungspalette des Unternehmens definiert werden. Auch hier können je nach Branche die produzierten Artikel materiell oder immateriell sein. Der Unterschied zur Entitätsmenge ROHSTOFF liegt darin, dass mit

Kunden und Lieferanten sind PARTNER

ROHSTOFF umfasst materielle und digitale Artikel

PRODUKT definiert das Leistungsangebot

PRODUKT die unternehmensspezifische Entwicklung und Herstellung von Waren oder Dienstleistungen charakterisiert wird.

Vereinbarungen
werden im
KONTRAKT
abgelegt

- Unter einem KONTRAKT wird eine rechtlich verbindliche Übereinkunft verstanden. Zu dieser Entitätsmenge zählen sowohl Versicherungs-, Verwaltungs- und Finanzierungsvereinbarungen wie auch Handels-, Beratungs-, Lizenz- und Verkaufsverträge.

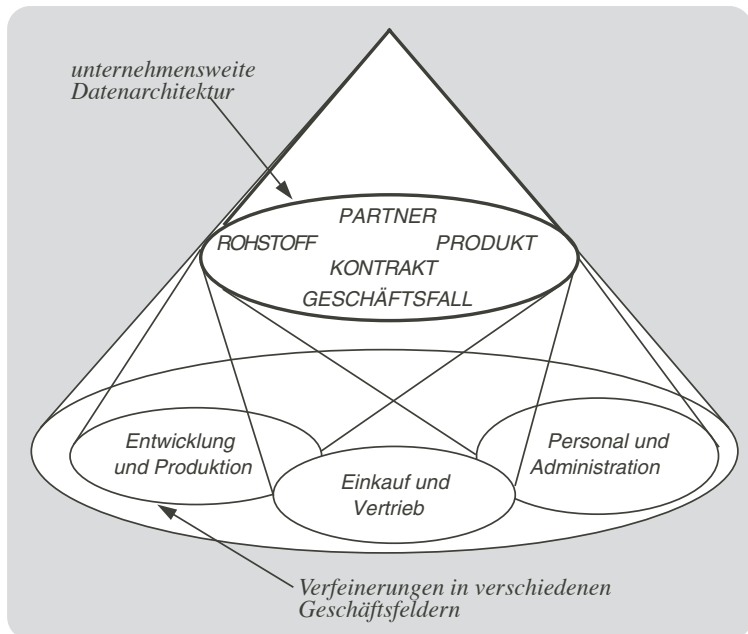
GESCHÄFTSFALL
dient der
Prozess-
abwicklung

- Ein GESCHÄFTSFALL bedeutet innerhalb einer Kontraktabwicklung einen einzelnen geschäftsrelevanten Schritt, ein Vorkommnis. Das kann zum Beispiel eine einzelne Zahlung, eine Buchung, eine Faktur oder eine Lieferung sein. Die Entitätsmenge GESCHÄFTSFALL hält die Bewegungen auf obigen Entitätsmengen fest.

Wichtige
Beziehungen
offenlegen

Neben den verallgemeinerten Entitätsmengen PARTNER, ROHSTOFF, PRODUKT, KONTRAKT und GESCHÄFTSFALL müssen *die wichtigsten Beziehungen aus der Sicht des Unternehmens* charakterisiert werden. So lässt sich beispielsweise festlegen, welche Rohwaren über welche Partner bezogen werden (vgl. *supply chain management*), von wem welche firmeneigenen Produkte produziert werden oder welche Konditionen in einem Kontrakt bezüglich Partner oder Waren gelten sollen.

Abb. 2-25
Die
datenorientierte
Sicht der
Geschäftsfelder



Selbstverständlich können mit einem solch groben Ansatz einer unternehmensweiten Datenarchitektur noch keine Informationssysteme aufgebaut oder entwickelt werden. Vielmehr müssen die Entitätsmengen und ihre Beziehungen schrittweise verfeinert werden, und zwar anhand einzelner Geschäftsfelder oder bestimmter Anwendungsbereiche. Bei diesem datenorientierten Vorgehen ist wesentlich, dass jede Konkretisierung eines *anwendungsspezifischen Datenmodells in Abstimmung mit der unternehmensweiten Datenarchitektur* vorgenommen wird. Nur so kann die Entwicklung von Informationssystemen kontrolliert und im Sinne der längerfristigen Ziele des Unternehmens realisiert werden.

Abstimmung mit der unternehmensweiten Datenarchitektur

2.7

Rezept zum Datenbankentwurf

In diesem Abschnitt fassen wir unsere Erkenntnisse zur Datenmodellierung in einem Vorgehensplan rezeptartig zusammen. Die Abb. 2-26 zeigt die zehn Entwurfsschritte, die je nach Phase der Projektentwicklung unterschiedlich durchlaufen werden. Praktische Erfahrungen zeigen, dass in jeder *Vorstudie* die Datenanalyse mit einem groben Entitäten-Beziehungsmodell erarbeitet werden sollte. Im *Grob- oder Detailkonzept* werden dann die Analyseschritte verfeinert, ein relationales Datenbankschema erstellt und auf Konsistenz und Implementierungsaspekte hin untersucht. Die zehn Entwurfsschritte lassen sich wie folgt charakterisieren:

Von der Vorstudie zum Detailkonzept

Zuerst müssen bei der Datenanalyse die *relevanten Informations-sachverhalte* schriftlich in einer Liste festgehalten werden. In den weiteren Entwurfsschritten kann die Liste in Abstimmung mit dem künftigen Anwender ergänzt und verfeinert werden, da das Entwurfsverfahren ein iterativer Prozess ist. Im zweiten Schritt werden die *Entitäts- und Beziehungsmengen* bestimmt, ihre Identifikationsschlüssel und Merkmalskategorien festgelegt. Das Entitäten-Beziehungsmodell wird vervollständigt, indem die verschiedenen Assoziationstypen eingetragen werden. Nun können auch Generalisationshierarchien und Aggregationsstrukturen speziell im dritten Schritt hervorgehoben werden. Im vierten Schritt wird das Entitäten-Beziehungsmodell mit der unternehmensweiten Datenarchitektur verglichen und abgestimmt, damit die Weiterentwicklung von Informationssystemen koordiniert und den längerfristigen Zielen entsprechend vorangetrieben werden kann.

Die vier Schritte für die Vorstudie

Abb. 2-26
 Vom Groben zum
 Detail in zehn
 Entwurfsschritten

| Rezeptschritte zum Datenbankentwurf | Vorstudie | Grobkonzept | Detailkonzept |
|--|-----------|-------------|---------------|
| 1. Datenanalyse 2. Entitäts- und Beziehungsmengen 3. Generalisation und Aggregation 4. Abstimmung mit der unternehmensweiten Datenarchitektur | X | X | X |
| 5. relationales Datenbankschema 6. Normalisierung 7. referenzielle Integrität 8. Konsistenzbedingungen | | X | X |
| 9. Zugriffspfade 10. physische Datenstruktur | | | X |

Das Grob-
konzept definiert
das logische
Datenmodell

Mit dem fünften Schritt wird das Entitäten-Beziehungsmodell in ein *relationales Datenbankschema* überführt. Dabei werden die erläuterten Regeln für Entitätsmengen, Beziehungsmengen, Generalisation und Aggregation benutzt. Das Datenbankschema wird im sechsten Schritt mit Hilfe der *Normalformen* kontrolliert. Beim detaillierten Betrachten der verschiedenen Abhängigkeiten können eventuelle Unstimmigkeiten entdeckt und behoben werden. Die Schritte sieben und acht befassen sich mit den *Integritätsbedingungen*. Vorerst werden die Fremd-Primärschlüssel-Beziehungen freigelegt und auf die entsprechenden Manipulationsregeln der referenziellen Integrität hin überprüft. Anschließend werden die übrigen Konsistenzbedingungen definiert, auch wenn nicht alle vom eingesetzten Datenbanksystem in vollem Umfang abgedeckt werden. Das Festlegen der Konsistenzbedingungen soll erlauben, systemnah eventuell eigene Validierungsregeln realisieren zu können, damit nicht jeder Anwender individuell die Integrität überprüfen muss.

Das Detail-
konzept
beschreibt den
physischen
Datenentwurf

Im neunten Schritt werden die *Zugriffspfade* für die wichtigsten applikatorischen Funktionen festgehalten. Die häufigsten Merkmale für künftige Datenbankzugriffe müssen analysiert und in einer Zugriffsmatrix aufgezeigt werden. Für sämtliche Tabellen des relationalen Datenbankschemas erläutert diese Matrix, welche Merkmale oder Merkmalskombinationen wie oft durch Einfüge-, Veränderungs-

oder Löschoperationen beansprucht werden. Das Bestimmen eines eigentlichen Mengengerüsts sowie das *Definieren der physischen Datenstruktur* geschieht im zehnten Schritt. Dabei werden die physischen Zugriffspfade und eventuelles Zurückstufen der Normalisierungsschritte (Denormalisierung) vorgenommen, um die Performanz der künftigen Anwendungen optimieren zu können (vgl. Kapitel 4).

Das in Abb. 2-26 gezeigte Rezept beschränkt sich im Wesentlichen auf die Datenaspekte. Neben den Daten spielen natürlich auch Funktionen beim Aufbau von Informationssystemen eine große Rolle. So sind in den letzten Jahren CASE-Werkzeuge entstanden, die nicht nur den Datenbankentwurf, sondern auch den Funktionsentwurf unterstützen. Wer sich für die Methodik der Anwendungsentwicklung interessiert, findet im nächsten Abschnitt weiterführende Literatur.

*Daten durch
Funktionsentwurf
ergänzen*

2.8 Bibliografische Angaben

Das Entitäten-Beziehungsmodell wurde durch die Arbeiten von Senko und Chen bekannt (vgl. Chen 1976). Seit 1979 gibt es regelmäßig internationale Konferenzen, bei denen Erweiterungen und Verfeinerungen des Entitäten-Beziehungsmodells vorgeschlagen und gemeinsam diskutiert werden.

Viele heutige CASE-Tools verwenden zur Datenmodellierung das Entitäten-Beziehungsmodell, wobei unterschiedliche grafische Symbole für Entitätsmengen, Beziehungsmengen oder Assoziationstypen verwendet werden; vgl. z.B. die Untersuchungen von Balzert (1993), Olle et al. (1988) und Martin (1990). Eine Übersicht über weitere logische Datenmodelle geben Tsichritzis und Lochovsky (1982).

*Geburtsstunde
des Entitäten-
Beziehungs-
modells*

*Vorgehen bei der
Daten-
modellierung*

Blaha und Rumbaugh (2004), Booch (2006) sowie Coad und Yourdon (1991) behandeln den objektorientierten Entwurf. Ferstl und Sinz (1991) zählen zu den deutschsprachigen Autoren, die das objektorientierte Vorgehen für die Entwicklung von Informationssystemen vorschlagen. Balzert (2004) kombiniert Methodenansätze von Coad, Booch und Rumbaugh für die objektorientierte Analyse. Einen Vergleich objektorientierter Analysemethoden gibt Stein (1994). Eine Einführung in die Unified Modelling Language (UML) primär für die Softwareentwicklung stammt von Hitz et al. (2005).

*Objektorientierte
Analyse und
Entwurf*

Smith und Smith (1977) haben die Konzepte der Generalisation und Aggregation für den Datenbankbereich eingeführt. Diese Strukturkonzepte sind vor allem auf dem Gebiet der wissensbasierten Systeme seit langem bekannt, beispielsweise bei der Beschreibung semantischer Netze, vgl. dazu Findler (1979).

*Generalisation
und Aggregation*

*Arbeiten zu den
Normalformen*

Das Studium der Normalformen hat dem Datenbankbereich eine eigentliche Datenbanktheorie beschert (Fagin 1979). Zu den Standardwerken mit eher theoretischem Anstrich zählen die Werke Maier (1983), Ullman (1982, 1988) und Paredaens et al. (1989). Dutka und Hanson (1989) geben kompakt und anschaulich eine zusammenfassende Darstellung der Normalformen. Die Standardwerke von Date (2004), von Elmasri und Navathe (2006), von Kemper und Eickler (2006) oder von Silberschatz et al. (2005) widmen einen wesentlichen Teil der Normalisierung.

*Unternehmens-
weite Daten-
architektur*

Fragen der unternehmensweiten Datenarchitektur werden in Dippold et al. (2005), Meier et al. (1991), Scheer (1991) und Silverston (2001) diskutiert. Aufgaben und Pflichten der Datenmodellierung und -administration umreißen Meier und Johner (1991) sowie Ortner et al. (1990).

*Daten-
modellierung*

Im deutschsprachigen Raum behandeln die Werke von Dürr und Radermacher (1990), Schlageter und Stucky (1983), Simsion und Witt (2005), Vossen (2000) und Vetter (1998) ausführlich Datenmodellierungsaspekte; letztere Studie eignet sich als Lehrgang für die Praxis.

3 Abfrage- und Manipulationssprachen

3.1

Benutzung einer Datenbank

Zum erfolgreichen Betreiben einer Datenbank ist eine Datenbanksprache notwendig, mit der die unterschiedlichen Anforderungen der Benutzer abgedeckt werden können. Relationale Abfrage- und Manipulationssprachen haben den Vorteil (vgl. Abb. 3-1), dass man mit ein und derselben Sprache Datenbanken erstellen, Benutzerrechte vergeben oder Tabelleninhalte verändern und auswerten kann.

Der *Datenadministrator* verwaltet mit einer relationalen Datenbanksprache die für das Unternehmen gültigen Datenbeschreibungen von Tabellen und Merkmalen. Sinnvollerweise wird er dabei durch ein Data-Dictionary-System unterstützt. Zusammen mit dem *Datenarchitekten* sorgt er dafür, dass die Beschreibungsdaten im Sinne der unternehmensweiten Datenarchitektur einheitlich und konsistent verwaltet und nachgeführt werden, eventuell unter Zuhilfenahme eines geeigneten CASE-Werkzeuges. Neben der Kontrolle der Datenformate legt er Berechtigungen fest, durch die sich einerseits die Datenbenutzung auf einzelne Tabellen oder sogar einzelne Merkmale, andererseits eine bestimmte Operation wie das Löschen oder Verändern einer Tabelle auf einen Benutzerkreis einschränken lässt.

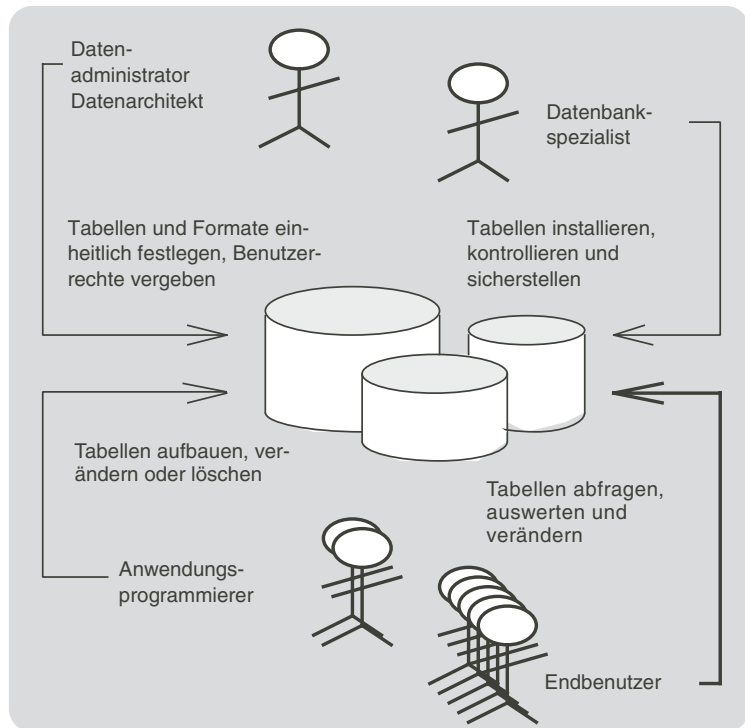
Der *Datenbankspezialist* verwendet nach Möglichkeit dieselbe Datenbanksprache. Er definiert, installiert und überwacht die Datenbanken mittels dafür vorgesehener Systemtabellen. Diese bilden den Systemkatalog, der zur Betriebszeit des Datenbanksystems alle notwendigen Datenbankbeschreibungen und statistischen Angaben umfasst. Mit den Systeminformationen kann sich der Datenbankspezialist durch vorgegebene Abfragen ein aktuelles Bild über sämtliche Datenbanken machen, ohne sich um die eigentlichen Tabellen mit den Benutzerdaten im Einzelnen zu kümmern. Aus Datenschutzgrün-

Notwendigkeit einer Datenbanksprache

Datenadministratoren und -architekten legen Tabellen und Formate fest

Datenbankspezialisten installieren und überwachen Datenbanken

Abb. 3-1
Verwendung
einer Datenbank-
sprache für
verschiedene
Zwecke



den sollte ihm der Zugriff auf Datenbanken im Betrieb nur in Ausnahmesituationen erlaubt sein, z.B. zur Behebung von Fehlern.

Anwendungs-
programmierer
entwickeln
Informations-
systeme

Der *Anwendungsprogrammierer* benutzt die Datenbanksprache, um neben Auswertungen auch Veränderungen auf den Datenbanken vornehmen zu können. Da eine relationale Abfrage- und Manipulationssprache mengenorientiert ist, benötigt der Programmierer ein *Cursorkonzept* (vgl. Abschnitt 3.5). Dieses erlaubt ihm, eine Menge von Tupeln satzweise in einem Programm abzuarbeiten. Für die eigentlichen Tests seiner Anwendungen steht ihm die relationale Sprache ebenfalls zur Verfügung. Er kann damit auf einfache Art seine Testdatenbanken kontrollieren und diese sogleich an den zukünftigen Anwender in Form von Prototypen weitergeben.

Endbenutzer
werten Daten-
bestände aus

Schließlich verwendet der *Endbenutzer* relationale Datenbanksprachen für seine *täglichen Auswertungsbedürfnisse*. Als Endbenutzer sind Anwender verschiedener Fachabteilungen gemeint, die bezüglich Informatik beschränkte Kenntnisse besitzen und bei Sachfragen auf spontane Art eigene Auswertungen durchführen möchten.

Wie wir gesehen haben, können verschiedene Benutzergruppen ihre Bedürfnisse mit einer relationalen Datenbanksprache abdecken. Dabei verwendet der Endbenutzer dieselbe Sprache wie der Anwendungsentwickler, der Datenbankspezialist oder auch der Datenadministrator. Datenbankanwendungen oder -auswertungen wie auch technische Arbeiten im Bereich Sicherstellen und Reorganisieren von Datenbanken können also in einer einheitlichen Sprache vorgenommen werden. Dadurch wird der Ausbildungsaufwand reduziert. Überdies lassen sich Erfahrungen zwischen den verschiedenen Benutzergruppen besser austauschen.

Eine Datenbanksprache für unterschiedliche Bedürfnisse

3.2 Grundlagen der Relationenalgebra

3.2.1 Zusammenstellung der Operatoren

Die *Relationenalgebra* (engl. *relational algebra*) bildet den *formalen Rahmen für die relationalen Datenbanksprachen*. Sie definiert einen Satz von algebraischen Operatoren, die immer auf Tabellen wirken. Zwar verwenden die meisten der heutigen relationalen Datenbanksprachen diese Operatoren nicht direkt, sie können aber nur dann im Sinne des Relationenmodells als relational vollständige Sprachen

Die Relationenalgebra bildet das Fundament

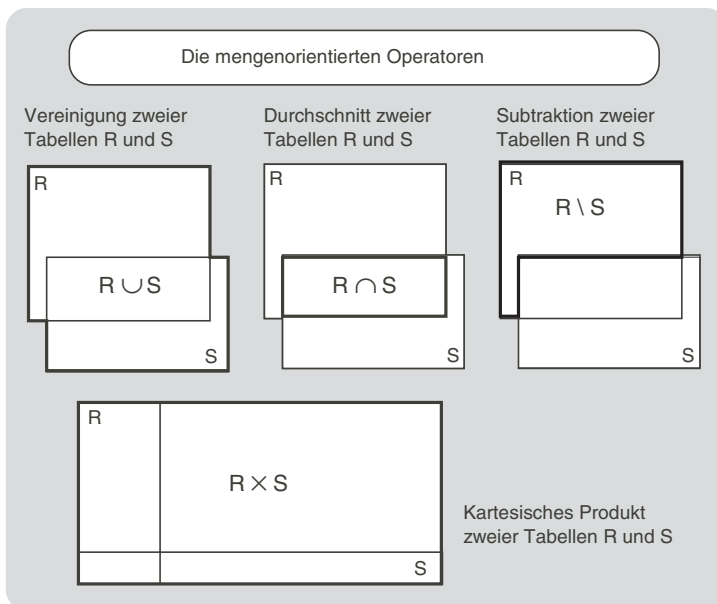


Abb. 3-2
Vereinigung,
Durchschnitt,
Differenz und kar-
tesisches Produkt
von Tabellen

bezeichnet werden, wenn das ursprüngliche Potenzial der Relationalenalgebra erhalten bleibt.

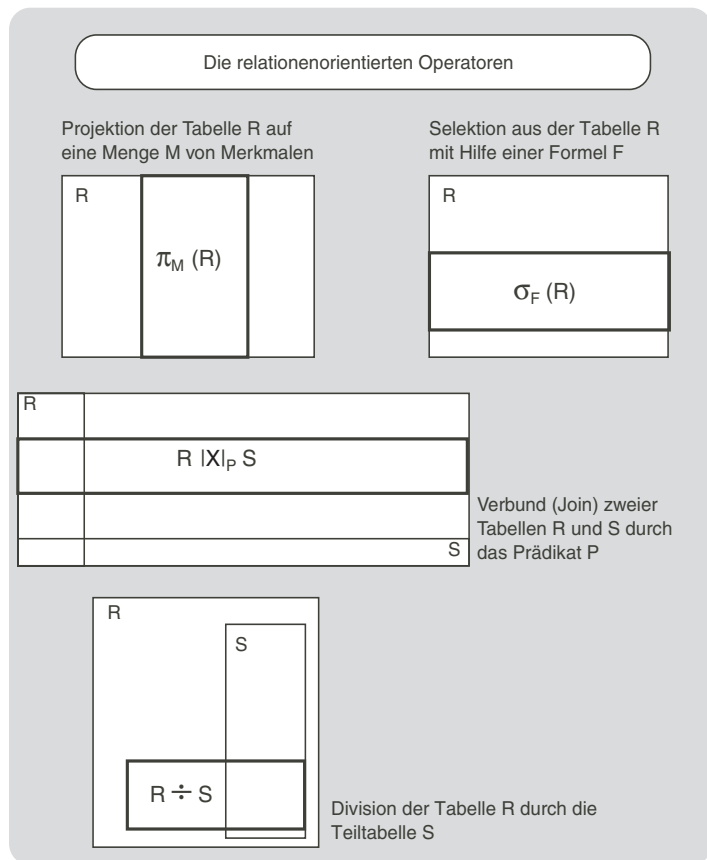
*Operatoren
lassen sich
kombinieren*

Im Folgenden wird für zwei beliebige Tabellen R und S eine Übersicht über die Operatoren der Relationalenalgebra gegeben, aufgeteilt nach *mengenorientierten* und *relationenorientierten* Operatoren. Sämtliche Operatoren verarbeiten entweder eine Tabelle oder zwei Tabellen und erzeugen wieder eine Tabelle. Diese Einheitlichkeit (algebraische Eigenschaft) macht es möglich, mehrere Operatoren miteinander zu kombinieren und auf Tabellen wirken zu lassen.

*Tabellen
addieren,
subtrahieren und
multiplizieren*

Die *mengenorientierten Operatoren* entsprechen den bekannten Mengenoperationen (vgl. Abb. 3-2 und nächsten Abschnitt 3.2.2). Zu diesen zählen die Vereinigung, symbolisiert durch das Spezialzeichen « \cup », der Durchschnitt « \cap », die Subtraktion « \setminus » und das kartesische Produkt « \times ». Aufgrund eines Verträglichkeitskriteriums können zwei Tabellen R und S miteinander vereinigt ($R \cup S$) oder geschnitten ($R \cap S$) oder voneinander subtrahiert ($R \setminus S$) werden. Zusätzlich lassen

*Abb. 3-3
Projektion,
Selektion,
Verbund und
Division von
Tabellen*



sich je zwei beliebige Tabellen R und S bedingungslos miteinander multiplizieren ($R \times S$). Das Resultat solcher Mengenoperationen ist wiederum eine Menge von Tupeln, also eine Tabelle.

Die *relationenorientierten Operatoren* sind gemäß Abb. 3-3 speziell für Tabellen festgelegt und werden ausführlich in Abschnitt 3.2.3 erläutert. Mit Hilfe des Projektionsoperators, abgekürzt durch den Buchstaben π (griechisch Pi), lassen sich Tabellen auf Teiltabellen reduzieren. So bildet der Ausdruck $\pi_M(R)$ eine Teiltabelle der Tabelle R anhand einer Menge M von Merkmalen. Der Selektionsoperator $\sigma_F(R)$, symbolisiert durch σ (griechisch Sigma), trifft eine Auswahl von Tupeln aus der Tabelle R anhand eines Selektionskriteriums oder einer so genannten Formel F. Der Verbundoperator, gegeben durch das Spezialzeichen \bowtie , kombiniert zwei Tabellen zu einer neuen. So lassen sich die beiden Tabellen R und S durch die Operation $R \bowtie_P S$ verbinden, wobei P die entsprechende Verbundsbedingung (Verbundsprädikat) angibt. Schließlich berechnet die Divisionsoperation $R \div S$ eine Teiltabelle, indem sie die Tabelle R durch die Tabelle S dividiert. Der Divisionsoperator wird durch das Spezialzeichen \div dargestellt.

In den nächsten beiden Abschnitten erläutern wir die mengen- und relationenorientierten Operatoren der Relationenalgebra anhand anschaulicher Beispiele.

*Tabellen
reduzieren oder
kombinieren*

3.2.2 Die mengenorientierten Operatoren

Da jede Tabelle eine Menge von Datensätzen (Tupeln) darstellt, können verschiedene Tabellen mengentheoretisch verknüpft werden. Von zwei Tabellen lässt sich jedoch nur dann eine Vereinigung, ein Durchschnitt oder eine Differenz berechnen, wenn diese *vereinigungsverträglich* sind.

Zwei Tabellen sind vereinigungsverträglich, wenn folgende zwei Bedingungen erfüllt sind: Beide Tabellen weisen die gleiche Anzahl Merkmale auf und die Datenformate der korrespondierenden Merkmalskategorien sind identisch.

Betrachten wir dazu das in Abb. 3-4 dargestellte Beispiel: Aus einer Mitarbeiterdatei ist für die beiden firmeneigenen Clubs je eine Tabelle definiert worden, die neben der Mitarbeiternummer den Namen und die Adresse enthält. Obwohl die Merkmalsnamen teilweise unterschiedlich lauten, sind die beiden Tabellen SPORTCLUB und FOTOCLUB vereinigungsverträglich. Sie weisen nämlich dieselbe Anzahl Merkmale auf, wobei die Merkmalswerte aus ein und derselben Mitarbeiterdatei stammen und somit auch über gleiche Wertebereiche definiert sind.

*Zur Vereinigungs-
verträglichkeit*

*Beispiel
vereinigungs-
verträglicher
Tabellen*

Abb. 3-4
Vereinigungs-
verträgliche
Tabellen SPORT-
und FOTOCLUB

SPORTCLUB

| M# | Name | Straße | Wohnort |
|-------------------------|-----------------------------|--|---------------------------------|
| M1 M7 M19 | Meier Huber Schweizer | Lindenstraße Mattenweg Hauptstraße | Liestal Basel Frenkendorf |

FOTOCLUB

| M# | Mitglied | Straße | Wohnort |
|----------|-----------------|------------------------|------------------|
| M4 M7 | Becker Huber | Wasserweg Mattenweg | Liestal Basel |

Der
Vereinigungs-
operator

Allgemein werden zwei vereinigungsverträgliche Tabellen R und S mengentheoretisch durch die *Vereinigung* (engl. *union*) $R \cup S$ kombiniert, indem *sämtliche Einträge aus R und sämtliche Einträge aus S in die Resultattabelle eingefügt* werden. Gleichzeitig werden identische Datensätze eliminiert; diese sind in der Resultatmenge $R \cup S$ aufgrund der Merkmalsausprägungen nicht mehr unterscheidbar.

Die Tabelle CLUBMITGLIEDER (Abb. 3-5) ist eine Vereinigung der Tabellen SPORTCLUB und FOTOCLUB. Jedes Resultattupel kommt entweder in der Tabelle SPORTCLUB oder FOTOCLUB oder in beiden gleichzeitig vor. Das Clubmitglied Huber erscheint nur einmal, da identische Einträge in der Vereinigungstabelle nicht zugelassen sind.

Die übrigen mengenorientierten Operatoren sind auf analoge Art definiert: Zum *Durchschnitt* (engl. *intersection*) $R \cap S$ zweier vereinigungsverträglicher Tabellen R und S zählen nur diejenigen Einträge, die *sowohl in R als auch in S vorhanden* sind. In unserem Tabellenauszug ist nur Huber im SPORT- wie im FOTOCLUB Aktivmitglied.

Abb. 3-5
Vereinigung der
beiden Tabellen
SPORTCLUB und
FOTOCLUB

CLUBMITGLIEDER = SPORTCLUB \cup FOTOCLUB

| M# | Name | Straße | Ort |
|-----------------------|---------------------------------------|---|--|
| M1 M7 M19 M4 | Meier Huber Schweizer Becker | Lindenstraße Mattenweg Hauptstraße Wasserweg | Liestal Basel Frenkendorf Liestal |

Die erhaltene Resultattabelle $\text{SPORTCLUB} \cap \text{FOTOCLUB}$ ist somit einelementig, da genau ein Mitarbeiter eine Doppelmitgliedschaft aufweist.

Schließlich können vereinigungsverträgliche Tabellen voneinander subtrahiert werden. Die *Differenz* (engl. *difference*) $R \setminus S$ erhält man dadurch, dass man in R sämtliche Einträge entfernt, die in S enthalten sind. Auf unser Beispiel angewendet, bedeutet die Subtraktion $\text{SPORTCLUB} \setminus \text{FOTOCLUB}$, dass wir in der Resultatrelation nur die beiden Mitglieder Meier und Schweizer finden. Das Mitglied Huber wird eliminiert, da es ja zusätzlich Mitglied im FOTOCLUB ist. Somit erlaubt der Differenzoperator die Bestimmung derjenigen Sportclubmitglieder, die nicht gleichzeitig dem Fotoclub angehören.

Allgemein besteht zwischen dem Durchschnitts- und dem Differenzoperator zweier vereinigungsverträglicher Tabellen die folgende Beziehung:

$$R \cap S = R \setminus (R \setminus S).$$

Somit kann die *Durchschnittsbildung auf die Differenzbildung von Tabellen zurückgeführt* werden. Diese Beziehung lässt sich leicht an unserem Beispiel der Sport- und Fotoclubmitglieder veranschaulichen.

Bei den mengenorientierten Operatoren fehlt noch das kartesische Produkt zweier beliebiger Tabellen R und S , die keineswegs vereinigungsverträglich sein müssen. Unter dem *kartesischen Produkt* (engl. *Cartesian product*) $R \times S$ zweier Tabellen R und S versteht man die Menge aller möglichen Kombinationen von Tupeln aus R mit Tupeln aus S .

Als Beispiel betrachten wir in Abb. 3-6 die Tabelle WETTKAMPF, die eine Mitgliederkombination $(\text{SPORTCLUB} \setminus \text{FOTOCLUB}) \times \text{FOTOCLUB}$ darstellt. Diese Tabelle enthält also alle möglichen Kombinationen von Sportclubmitgliedern (die nicht zugleich im Fotoclub sind) mit den Fotoclubmitgliedern. Sie drückt eine typische Wettkampfszusammenstellung der beiden Clubs aus, wobei das Doppelmitglied Huber natürlich nicht gegen sich selbst antreten kann und aufgrund der Subtraktion $\text{SPORTCLUB} \setminus \text{FOTOCLUB}$ auf der Seite der Fotoclubmitglieder kämpft.

Der Subtraktionsoperator

Zusammenhang zwischen Durchschnitts- und Differenzoperator

Zum kartesischen Produkt zweier Tabellen

Beispiel einer Wettkampftabelle

WETTKAMPF = $(\text{SPORTCLUB} \setminus \text{FOTOCLUB}) \times \text{FOTOCLUB}$

| M# | Name | Straße | Wohnort | M# | Mitgl. | Straße | Ort |
|-----|-----------|--------------|-------------|----|--------|-----------|---------|
| M1 | Meier | Lindenstraße | Liestal | M4 | Becker | Wasserweg | Liestal |
| M1 | Meier | Lindenstraße | Liestal | M7 | Huber | Mattenweg | Basel |
| M19 | Schweizer | Hauptstraße | Frenkendorf | M4 | Becker | Wasserweg | Liestal |
| M19 | Schweizer | Hauptstraße | Frenkendorf | M7 | Huber | Mattenweg | Basel |

*Abb. 3-6
Tabelle
WETTKAMPF
als Beispiel eines
kartesischen
Produktes*

Zum
Multiplizieren
von Tabellen

Diese Operation heißt (kartesisches) Produkt, da die jeweiligen Einträge der Ausgangstabellen miteinander multipliziert werden. Haben allgemein zwei beliebige Tabellen R und S m beziehungsweise n Einträge, so hat das kartesische Produkt $R \times S$ insgesamt m mal n Tupelinträge.

3.2.3

Die relationenorientierten Operatoren

Der Projektions-
operator

Die relationenorientierten Operatoren ergänzen die mengenorientierten, wobei hier die jeweiligen Tabellen – wie schon beim kartesischen Produkt – nicht vereinigungsverträglich sein müssen. Der *Projektionsoperator* (engl. *projection operator*) $\pi_M(R)$ bildet eine *Teiltabelle* aus der Tabelle R aufgrund der durch M definierten Merkmalsnamen. Beispielsweise bedeutet für eine Tabelle R mit den Merkmalen (A,B,C,D) die Schreibweise $\pi_{A,C}(R)$, dass R auf die Spalten A und C reduziert wird. Es ist erlaubt, in einer Projektion die Spaltennamen in einer beliebigen Reihenfolge aufzulisten. So bedeutet $R' := \pi_{C,A}(R)$ die Projektion der Tabelle R = (A,B,C,D) auf $R' = (C,A)$.

Beispiele von
Projektionen

Das erste Beispiel $\pi_{\text{Ort}}(\text{MITARBEITER})$ in Abb. 3-7 listet alle Orte aus der Mitarbeitertabelle wiederholungsfrei in eine einspaltige Tabelle. Im zweiten Beispiel $\pi_{\text{Unt,Name}}(\text{MITARBEITER})$ erhalten wir eine Teiltabelle mit sämtlichen Abteilungsnummern und Namen der zugehörigen Mitarbeiter.

Abb. 3-7
Projektions-
operatoren am
Beispiel
MITARBEITER

| MITARBEITER | | | | |
|-------------|-----------|--------------|-------------|-----|
| M# | Name | Straße | Ort | Unt |
| M19 | Schweizer | Hauptstraße | Frenkendorf | A6 |
| M1 | Meier | Lindenstraße | Liestal | A3 |
| M7 | Huber | Mattenweg | Basel | A5 |
| M4 | Becker | Wasserweg | Liestal | A6 |

$\pi_{\text{Ort}}(\text{MITARBEITER})$

| Ort |
|-------------|
| Frenkendorf |
| Liestal |
| Basel |

$\pi_{\text{Unt,Name}}(\text{MITARBEITER})$

| Unt | Name |
|-----|-----------|
| A6 | Schweizer |
| A3 | Meier |
| A5 | Huber |
| A6 | Becker |

Ein weiterer wichtiger Operator $\sigma_F(R)$ dient der *Selektion* (engl. *selection*) von Tupeln aus der Tabelle R anhand der Formel F . Eine Formel F besitzt eine bestimmte Anzahl von Merkmalsnamen oder konstanten Werten, die durch Vergleichsoperatoren wie $\langle<\rangle$, $\langle>\rangle$ oder $\langle=\rangle$ sowie durch logische Operatoren wie AND, OR und NOT miteinander kombiniert werden können. Unter $\sigma_F(R)$ versteht man somit alle Tupel aus R , die die Selektionsbedingung F erfüllen.

Betrachten wir dazu die Beispiele der Abb. 3-8 zur Selektion von Tupeln aus der Tabelle MITARBEITER: Im ersten Beispiel werden alle Mitarbeitenden bestimmt, die die Bedingung $\langle\text{Ort}=\text{Liestal}\rangle$ erfüllen und somit in Liestal wohnen. Das zweite Beispiel mit der Bedingung $\langle\text{Unt}=\text{A6}\rangle$ selektiert nur die in der Abteilung A6 tätigen Mitarbeitenden. Schließlich kombiniert das dritte Beispiel die beiden ersten Selektionsbedingungen durch eine logische Verknüpfung anhand der Formel $\langle\text{Ort}=\text{Liestal AND Unt}=\text{A6}\rangle$. Dabei erhalten wir als Resultattabelle eine einelementige Tabelle, denn nur Mitarbeiter Becker stammt aus Liestal und arbeitet in der Abteilung mit der Nummer A6.

Natürlich lassen sich die bereits besprochenen Operatoren der Relationenalgebra auch miteinander kombinieren. Wenden wir zum Beispiel nach einer Selektion der Mitarbeitenden der Abteilung A6 durch $\sigma_{\text{Unt}=\text{A6}}(\text{MITARBEITER})$ anschließend eine Projektion auf das Merkmal Ort durch den Operator $\pi_{\text{Ort}}(\sigma_{\text{Unt}=\text{A6}}(\text{MITARBEITER}))$ an, so erhalten wir als Resultattabelle die beiden Ortschaften Frenkendorf und Liestal.

Der Selektionsoperator

Beispiele zur Selektion

Das Kombinieren von Operatoren ist zulässig

Abb. 3-8
Beispiele von Selektionsoperatoren

$\sigma_{\text{Ort}=\text{Liestal}}(\text{MITARBEITER})$

| M# | Name | Straße | Ort | Unt |
|----|--------|--------------|---------|-----|
| M1 | Meier | Lindenstraße | Liestal | A3 |
| M4 | Becker | Wasserweg | Liestal | A6 |

$\sigma_{\text{Unt}=\text{A6}}(\text{MITARBEITER})$

| M# | Name | Straße | Ort | Unt |
|-----|-----------|-------------|-------------|-----|
| M19 | Schweizer | Hauptstraße | Frenkendorf | A6 |
| M4 | Becker | Wasserweg | Liestal | A6 |

$\sigma_{\text{Ort}=\text{Liestal AND Unt}=\text{A6}}(\text{MITARBEITER})$

| M# | Name | Straße | Ort | Unt |
|----|--------|-----------|---------|-----|
| M4 | Becker | Wasserweg | Liestal | A6 |

Der Verbundoperator
(*equi-join*)

Betrachten wir jetzt den *Verbundoperator* (engl. *join operator*), durch den sich zwei Tabellen zu einer einzigen zusammenfügen lassen. Der Verbund $R \bowtie_P S$ der beiden Tabellen R und S über das Prädikat P ist eine *Kombination aller Tupel aus R mit allen Tupeln aus S , die jeweils das Verbundprädikat P erfüllen*. Das Verbundprädikat enthält ein Merkmal aus der Tabelle R und eines aus S . Diese beiden Merkmale werden durch Vergleichsoperatoren $\langle \langle \rangle \rangle$, $\langle \rangle \rangle$ oder $\langle = \rangle$ in Beziehung gesetzt, damit die Tabellen R und S kombiniert werden können. Enthält das Verbundprädikat P den Vergleichsoperator $\langle = \rangle$, so spricht man von einem *Gleichheitsverbund* (engl. *equi-join*).

Der Verbundoperator stößt oft auf Verständnisschwierigkeiten und kann dadurch zu falschen oder ungewollten Resultaten führen. Der Grund liegt meistens darin, dass das Prädikat für die Kombination zweier Tabellen vergessen oder falsch definiert wird.

Vorsicht beim
Spezifizieren
eines Verbunds

Betrachten wir in Abb. 3-9 als Beispiel zwei Verbundoperatoren mit und ohne Angabe des Verbundprädikates. Mit der Spezifikation $\text{MITARBEITER} \bowtie_{\text{Unt}=\text{A\#}} \text{ABTEILUNG}$ verknüpfen wir die beiden Tabellen MITARBEITER und ABTEILUNG , indem wir die Angaben der Mitarbeitenden mit den Angaben ihrer zugehörigen Abteilungen ergänzen. Vergessen wir im Beispiel aus Abb. 3-9 die Angabe eines Verbundprädikates P und spezifizieren $\text{MITARBEITER} \times \text{ABTEILUNG}$, so erhalten wir das kartesische Produkt der beiden Tabellen MITARBEITER und ABTEILUNG . Dieses Beispiel illustriert eine nicht sehr sinnvolle Kombination der beiden Tabellen, da sämtliche Mitarbeitenden sämtlichen Abteilungen gegenübergestellt werden. Wir finden also in der Resultattabelle auch die Kombination von Mitarbeitern mit Abteilungen, denen die Mitarbeiter organisatorisch gar nicht zugeteilt sind (vgl. dazu die Tabelle WETTKAMPF in Abb. 3-6).

Verbund als
eingeschränktes
kartesisches
Produkt

Wie die Beispiele in Abb. 3-9 zeigen, ist der Verbundoperator \bowtie mit dem Verbundprädikat P nichts anderes als eine Einschränkung des kartesischen Produktes. Allgemein drückt ein Verbund zweier Tabellen R und S ohne die Spezifikation des Verbundprädikates ein kartesisches Produkt der beiden Tabellen R und S aus, d.h., es gilt mit dem leeren Prädikat $P=\{\}$

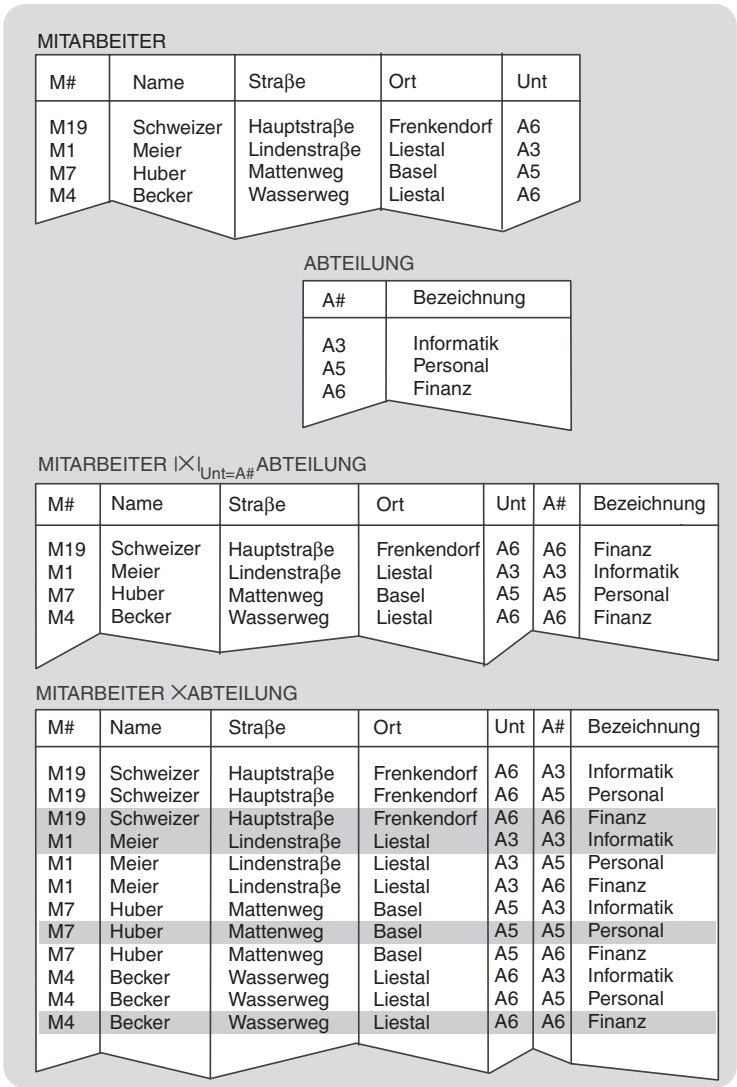
$$R \bowtie_{P=\{\}} S = R \times S.$$

Verwenden wir bei der Selektion als Selektionsprädikat ein Verbundprädikat, so erhalten wir:

$$R \bowtie_P S = \sigma_P (R \times S).$$

Diese allgemeine Formel drückt aus, dass jeder Verbund in einem ersten Schritt durch ein kartesisches Produkt und in einem zweiten Schritt durch eine Selektion ausgedrückt werden kann.

Abb. 3-9
Verbund zweier
Tabellen mit und
ohne
Verbundprädikat



Auf das Beispiel aus Abb. 3-9 bezogen können wir unseren gewünschten Ausdruck des Verbundes $\text{MITARBEITER} \bowtie_{\text{Unt}=\text{A\#}} \text{ABTEILUNG}$ durch die folgenden zwei Schritte berechnen: Zuerst bilden wir das kartesische Produkt der beiden Tabellen MITARBEITER und ABTEILUNG. In der Tabelle dieses Zwischenresultats werden nun diejenigen Einträge durch die Selektion $\sigma_{\text{Unt}=\text{A\#}}(\text{MITARBEITER} \times \text{ABTEILUNG})$ bestimmt, bei denen das Verbundprädikat « $\text{Unt}=\text{A\#}$ » erfüllt ist. Damit erhalten wir dieselben Tupel wie bei der direkten Berechnung des Ver-

Die zwei Schritte
zur Berechnung
des Verbunds

bundes MITARBEITER $\bowtie_{\text{Unt}=\text{A}\#} \text{ABTEILUNG}$ (vgl. dazu die grau hinterlegten Tupel­einträge in Abb. 3-9).

Der Divisionsoperator

Eine *Division* (engl. *division*) der Tabelle R durch die Tabelle S kann nur durchgeführt werden, falls S als Teiltabelle in R enthalten ist. Der Divisionsoperator $R \div S$ berechnet eine Teiltabelle R' aus R mit der Eigenschaft, dass die Kombinationen aller Tupel r' aus R' mit den Tupeln s aus S in der Tabelle R liegen. Es muss also das kartesische Produkt $R' \times S$ in der Tabelle R enthalten sein.

Beispiel einer Division

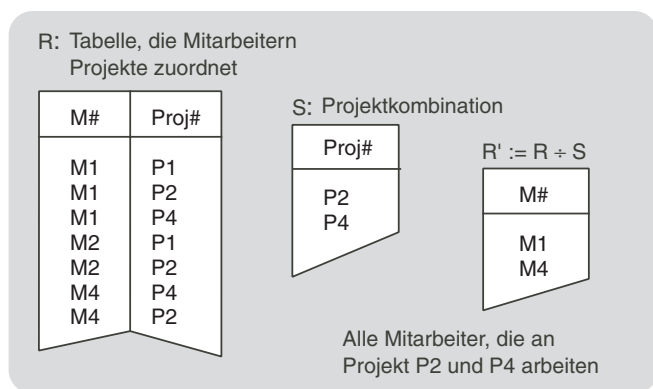
Die in Abb. 3-10 dargestellte Tabelle R zeigt, welche Mitarbeiter an welchen Projekten arbeiten. Wir interessieren uns nun für alle Mitarbeiter, die gleichzeitig an den Projekten P2 und P4 arbeiten. Dazu definieren wir die Tabelle S mit den Projektnummern P2 und P4. Es ist offensichtlich, dass S in R enthalten ist, und wir können somit die Division $R' := R \div S$ berechnen. Die Division liefert als Resultat in der Tabelle R' die beiden Mitarbeiter M1 und M4. Eine kleine Kontrollrechnung bestätigt, dass sowohl M1 als auch M4 gleichzeitig an den Projekten P2 und P4 arbeiten, da die Tupel (M1,P2), (M1,P4) bzw. (M4,P2) und (M4,P4) in der Tabelle R auftreten.

Man kann einen Divisionsoperator durch Projektions- und Differenzoperatoren sowie durch ein kartesisches Produkt ausdrücken. Damit zählt der Divisionsoperator neben dem Durchschnitts- und dem Verbundoperator zu den ersetzbaren Operatoren der Relationenalgebra.

Minimale Anzahl von Operatoren

Zusammenfassend bilden Vereinigung, Differenz, kartesisches Produkt, Projektions- und Selektionsoperatoren die kleinste Menge von Operatoren, die die Relationenalgebra voll funktionsfähig macht: Der Durchschnitts-, der Verbund- sowie der Divisionsoperator lassen sich – wenn auch manchmal umständlich – durch diese fünf Operatoren der Relationenalgebra jederzeit herleiten.

Abb. 3-10
Beispiel eines Divisionsoperators



Die Operatoren der Relationenalgebra sind nicht nur aus theoretischer Sicht interessant, sondern haben auch ihre praktische Bedeutung. So werden sie für die Sprachschnittstelle relationaler Datenbanksysteme verwendet, um Optimierungen vorzunehmen (vgl. Abschnitt 4.2.2). Darüber hinaus gelangen sie beim Bau von Datenbankrechnern zur Anwendung: Die Operatoren der Relationenalgebra oder abgewandelte Formen davon müssen nicht softwaremäßig realisiert werden, sondern lassen sich direkt in Hardwarekomponenten implementieren.

Zur Entwicklung von Datenbankrechnern

3.3

Relational vollständige Sprachen

Relational vollständige Sprachen sind Sprachen, die zumindest der Relationenalgebra oder dem so genannten Relationenkalkül ebenbürtig sind. Es lässt sich nämlich zeigen, dass der Relationenkalkül gleich mächtig ist wie die im vorigen Abschnitt diskutierte Relationenalgebra.

Was heißt relational vollständig?

Der *Relationenkalkül* (engl. *relational calculus*) orientiert sich an der Aussagenlogik. Er untersucht Ausdrücke, die eine vom Benutzer festgelegte Selektionsbedingung (Prädikat genannt) erfüllen. Die Prädikate beziehen sich auf eine Tupelvariable einer bestimmten Tabelle und lassen sich durch die logischen Verknüpfungen AND, OR und NOT zusammensetzen. Zusätzlich können Quantoren wie «für alle ...» (All-Quantor) oder «es existiert ...» (Existenz-Quantor) angegeben werden. Beim All-Quantor müssen alle Tupel der Tabelle das entsprechende Prädikat erfüllen. Der Existenz-Quantor überprüft die Tabelle auf das Vorkommen mindestens eines Tupels, das das Prädikat erfüllt.

Zum Relationenkalkül

Die in der Praxis gebräuchlichen Sprachen relationaler Datenbanksysteme orientieren sich an der Relationenalgebra oder am Relationenkalkül:

So kann die bereits erwähnte Sprache SQL, die Structured Query Language, als eine Kombination von Relationenalgebra und Relationenkalkül angesehen werden (vgl. Abschnitt 3.4.1).

Standard SQL

Die Sprache QUEL, die Query Language, illustriert den Ansatz des Relationenkalküls, mit Tupelvariablen zu arbeiten (Abschnitt 3.4.2).

Sprache QUEL

Unter QBE, Query by Example, versteht man eine Sprache, mit der man anhand von Hilfsgrafiken die eigentlichen Abfragen und Manipulationen durchführen kann (Abschnitt 3.4.3). QBE lehnt sich ebenfalls an den Relationenkalkül an, unterstützt aber gleichzeitig das benutzerfreundliche Arbeiten mit Tabellen durch grafische Elemente.

Grafische Sprache QBE

SQL, QUEL und QBE sind gleich mächtig wie die Relationenalgebra bzw. der Relationenkalkül und gelten deshalb als relational vollständige Sprachen.

Die *relationale Vollständigkeit einer Datenbanksprache* bedeutet, dass die Operatoren der Relationenalgebra (oder äquivalent dazu diejenigen des Relationenkalküls) in ihr darstellbar sind.

Relational vollständige Sprachen *Kriterium der Vollständigkeit*
Eine Datenbankabfragesprache heißt *relational vollständig* im Sinne der Relationenalgebra, wenn sie *mindestens die mengenorientierten Operatoren* Vereinigung, Subtraktion und kartesisches Produkt sowie *die relationenorientierten Operatoren* Projektion und Selektion ermöglicht.

Qualitätsanspruch an relationale Sprachen
Das Kriterium der Vollständigkeit ist das wichtigste Kriterium beim Überprüfen einer Datenbanksprache auf ihre relationale Tauglichkeit. Nicht jede Sprache, die mit Tabellen arbeitet, ist relational vollständig. Fehlt z.B. die Möglichkeit, verschiedene Tabellen über gemeinsame Merkmale zu kombinieren, so ist diese Sprache nicht mit der Relationenalgebra oder dem Relationenkalkül äquivalent. Daher verdient sie auch nicht die Auszeichnung als relational vollständige Datenbanksprache.

Erweiterung der Grundsprache
Die Relationenalgebra sowie der Relationenkalkül bilden die Grundlage primär für den Abfrageteil einer relationalen Datenbanksprache. Natürlich möchte man neben Auswertungsoperationen auch einzelne Tabellen oder Teile davon manipulieren können. Zu den Manipulationsoperationen zählen beispielsweise das Einfügen, Löschen oder Verändern von Tupelmengen. Aus diesem Grunde müssen relational vollständige Sprachen um die folgenden Funktionen erweitert werden, um sie für die Praxis tauglich zu machen:

- Zusatzforderungen für die Tauglichkeit relationaler Sprachen*
- Es müssen Tabellen und Merkmale definiert werden können.
 - Es müssen Einfüge-, Veränderungs- und Löschoperationen vorgesehen werden.
 - Die Sprache sollte Aggregatsfunktionen wie Summenbildung, Maximumbestimmung, Berechnung des Minimal- oder des Durchschnittswertes einer einzelnen Spalte der Tabelle enthalten.
 - Tabellen sollten formatiert und nach verschiedenen Kriterien ausgedruckt werden können. So interessieren Sortierreihenfolgen oder Gruppenbrüche zum Darstellen von Tabellen.
 - Sprachen für relationale Datenbanken müssen zwingend Sprach-elemente für die Vergabe von Benutzerberechtigungen und für den Schutz der Datenbanken anbieten (vgl. Abschnitt 3.7).

- Es wäre von Vorteil, wenn relationale Datenbanksprachen arithmetische Ausdrücke oder Berechnungen unterstützen würden.
- Relationale Datenbanksprachen müssen dem Mehrbenutzeraspekt Rechnung tragen (Transaktionsprinzip) und Befehle für die Datensicherheit bereitstellen (vgl. Kapitel 4).

Durch die Beschreibung der Relationenalgebra bzw. des Relationenkalküls haben wir den formalen Rahmen relationaler Datenbanksprachen abgesteckt. In der Praxis ist es nun so, dass diese formalen Sprachen nicht direkt eingesetzt werden. Vielmehr wurde schon früh versucht, relationale *Datenbanksprachen möglichst benutzerfreundlich* zu gestalten. Da die algebraischen Operatoren dem Anwender eines Datenbanksystems in reiner Form nicht zumutbar sind, werden die Operatoren durch aussagekräftigere Sprachelemente dargestellt. Die drei Sprachen SQL, QUEL und QBE sollen dies in den folgenden Abschnitten beispielhaft demonstrieren.

Zum Stellenwert der Benutzerschnittstelle

3.4 Übersicht über relationale Sprachen

3.4.1 SQL

Die Sprache SQL wurde Mitte der siebziger Jahre für «System R» geschaffen; dieses Testsystem war eines der ersten lauffähigen relationalen Datenbanksysteme. In der Zwischenzeit ist die Sprache durch die ISO normiert worden und gilt als führende Sprache auf diesem Gebiet.

SQL als ISO-Standard

Die Grundstruktur der Sprache SQL sieht gemäß Abschnitt 1.3 wie folgt aus:

| | |
|--------|--------------------|
| SELECT | Merkmale |
| FROM | Tabellen |
| WHERE | Selektionsprädikat |

Grundstruktur von SQL

Die SELECT-Klausel entspricht dem Projektionsoperator der Relationenalgebra, indem sie eine Liste von Merkmalen angibt. Aus der Abfrage $\pi_{\text{Unt, Name}}(\text{MITARBEITER})$ in Form eines Projektionsoperators der Relationenalgebra, wie in Abb. 3-7 dargestellt, macht SQL ganz einfach:

| | |
|--------|-------------|
| SELECT | Unt, Name |
| FROM | MITARBEITER |

Projektion

In der FROM-Klausel werden alle benötigten Tabellen aufgeführt. Beispielsweise wird das kartesische Produkt zwischen MITARBEITER und ABTEILUNG wie folgt in SQL dargestellt:

| | | |
|-----------------------------|--------|---|
| <i>Kartesisches Produkt</i> | SELECT | M#, Name, Straße, Ort, Unt, A#, Bezeichnung |
| | FROM | MITARBEITER, ABTEILUNG |

Dieser Befehl erzeugt die Resultattabelle der früheren Abb. 3-9, wie die gleichwertigen Operatoren $\text{MITARBEITER} \bowtie_{P=\{ \}} \text{ABTEILUNG}$ oder $\text{MITARBEITER} \times \text{ABTEILUNG}$.

Formuliert man in der WHERE-Klausel das Verbundprädikat «Unt=A#», so erhält man den Gleichheitsverbund zwischen den beiden Tabellen MITARBEITER und ABTEILUNG in SQL-Notation:

| | | |
|----------------|--------|--|
| <i>Verbund</i> | SELECT | M#, Name, Strasse, Ort, Unt, A#, Bezeichnung |
| | FROM | MITARBEITER, ABTEILUNG |
| | WHERE | Unt = A# |

Qualifizierte Selektionen lassen sich beschreiben, indem in der WHERE-Klausel verschiedene Aussagen durch die logischen Operatoren AND und OR verknüpft werden. Die früher vorgenommene Selektion der Mitarbeiter, nämlich $\sigma_{\text{Ort=Liestal AND Unt=A6}}(\text{MITARBEITER})$, dargestellt in Abb. 3-8, lautet in SQL:

| | | |
|------------------|--------|--------------------------------|
| <i>Selektion</i> | SELECT | * |
| | FROM | MITARBEITER |
| | WHERE | Ort = 'Liestal' AND Unt = 'A6' |

Ein «*» in der SELECT-Klausel bedeutet, dass alle Merkmale der entsprechenden Tabellen selektiert werden; man bekommt also eine Resultattabelle mit den Merkmalen M#, Name, Straße, Ort und Unt (Unterstellung). Die WHERE-Klausel enthält das gewünschte Selektionsprädikat. Die Ausführung der obigen Abfrage durch das Datenbanksystem führt somit zum Mitarbeiter Becker aus Liestal, der in der Abteilung A6 tätig ist.

Eingebaute SQL-Funktionen Neben den üblichen Operatoren der Relationenalgebra und des Relationenkalküls existieren bei SQL so genannte *eingebaute Funktionen* (engl. *built-in functions*), die in der SELECT-Klausel verwendet werden. Zu diesen Funktionen gehören für jeweils eine bestimmte Tabellenspalte COUNT für eine Zählung, SUM für eine Summenbildung, AVG für die Berechnung des Mittels (engl. *average*), MAX zur Bestimmung des Maximalwertes und MIN für die Feststellung des Minimalwertes.

Beispielsweise können alle Mitarbeitenden gezählt werden, die in der Abteilung A6 arbeiten. In SQL lautet diese Aufforderung wie folgt:

| | |
|--------|-------------|
| SELECT | COUNT (M#) |
| FROM | MITARBEITER |

WHERE Unt = 'A6'

Als Resultat erhält man eine einelementige Tabelle mit einem einzigen Wert 2, der gemäß Tabellenauszug für die beiden Mitarbeiter Schweizer und Becker steht.

Zur Definition einer Tabelle in SQL steht ein CREATE-Befehl zur Verfügung. Die Tabelle MITARBEITER wird wie folgt spezifiziert:

Tabellendefinition

```
CREATE TABLE    MITARBEITER
                 (M#                CHAR(6) NOT NULL,
                 Name               CHAR(20),
                 ... )
```

Mit dem entgegengesetzten Befehl DROP TABLE können Tabellendefinitionen gelöscht werden. Dabei ist zu beachten, dass dieser Befehl auch sämtliche Tabelleninhalte und dazugehörige Benutzungsrechte eliminiert (vgl. Abschnitt 3.7).

*Vorsicht beim
Löschen*

Ist die Tabelle MITARBEITER definiert, so können durch die Anweisung

```
INSERT INTO MITARBEITER
             VALUES ('M20', 'Müller', 'Riesweg', 'Olten', 'A6')
```

neue Tupel, Tabellenzeilen, eingefügt werden.

Eine Manipulation der Tabelle MITARBEITER ist durch eine UPDATE-Anweisung möglich:

*Verändern von
Datenwerten*

```
UPDATE        MITARBEITER
SET            Ort = 'Basilea'
WHERE         Ort = 'Basel'
```

Diese Beispiel-Anweisung ersetzt in allen Tupeln der Tabelle MITARBEITER sämtliche Wohnorte mit dem Wert Basel durch den neuen Ortsnamen Basilea. Die Veränderungsoperation UPDATE ist ebenfalls mengenorientiert und verändert gegebenenfalls eine mehrelementige Menge von Tupeln.

Schließlich können ganze Tabellen oder Teile davon durch eine DELETE-Anweisung gelöscht werden:

Löschoperation

```
DELETE
FROM          MITARBEITER
WHERE         Ort = 'Basilea'
```

Die Löschanweisung betrifft normalerweise eine Menge von Tupeln, falls das Selektionsprädikat auf mehrere Tabelleneinträge zutrifft. Im Falle referenzieller Integrität (vgl. Abschnitt 3.8) kann sich eine solche Löschoperation auch auf abhängige Tabellen auswirken.

Ein Tutorial für die Sprache SQL ist im Anhang aufgeführt.

3.4.2 QUEL

QUEL verlangt eine Tupelvariable Die Sprache QUEL basiert auf dem tupelorientierten Relationenkalkül und wurde Mitte der siebziger Jahre für das Datenbanksystem «Ingres» definiert. Jede Abfrage erfüllt das folgende Grundschemata:

```
RANGE OF Tupelvariable IS Tabelle
RETRIEVE (Merkmal, ...)
WHERE Selektionsprädikat
```

Verarbeitung einer QUEL-Abfrage Eine Tupelvariable, im Folgenden dargestellt durch einen Kleinbuchstaben, wird einzeln durch eine Klausel RANGE OF spezifiziert, wobei die zugehörige Tabelle durch ihren Namen angegeben werden muss. Umgekehrt muss für jede zu selektierende Tabelle eine Tupelvariable verwendet werden. In der RETRIEVE-Klausel werden innerhalb eines Klammerspaares von den relevanten Tabellen die Merkmale angegeben, die selektiert werden sollen. In der WHERE-Klausel schließlich bestimmt das Selektionsprädikat, unter welchen Bedingungen die Daten gesucht werden.

Eine Projektion der Tabelle MITARBEITER auf die Merkmale Abteilungsnummer (Unt) und Name lautet in QUEL:

Projektion

```
RANGE OF m IS MITARBEITER
RETRIEVE (m.Unt, m.Name)
```

Die Tupelvariable m durchläuft die Tabelle MITARBEITER, und mit «m.Unt» und «m.Name» wird die Variable an die beiden Merkmale gebunden. Die Tupelvariable m kann in diesem Fall nur Werte der Merkmale Abteilungsnummer und Mitarbeitername ausgeben.

Möchte man Mitarbeitende auflisten, die in Liestal wohnen und in der Abteilung A6 arbeiten, so kommt die WHERE-Klausel zum Zug:

Selektion

```
RANGE OF m IS MITARBEITER
RETRIEVE (m.M#, m.Name, m.Straße, m.Ort, m.Unt)
WHERE m.Ort = 'Liestal' AND m.Unt = 'A6'
```

Um einen Verbund zwischen den beiden Tabellen MITARBEITER und ABTEILUNG zu berechnen, müssen zwei Tupelvariablen (Variablen m und a) festgelegt und das Verbundprädikat in der WHERE-Klausel mit den richtigen Tupelvariablen angegeben werden:

Verbund zweier Tabellen

```
RANGE OF m IS MITARBEITER
RANGE OF a IS ABTEILUNG
RETRIEVE (m.M#, m.Name, a.A#, a.Bezeichnung)
WHERE m.Unt = a.A#
```

Interessiert man sich lediglich für die Anzahl der Mitarbeiter, die in der Abteilung A6 tätig sind, so kann dies mit Hilfe der Funktion COUNT ermittelt werden:

```
RANGE OF m IS MITARBEITER
RETRIEVE COUNT (m.M#)
WHERE      m.Unt = 'A6'
```

*Eingebaute
Funktion*

Analog lassen sich die übrigen Funktionen SUM, AVG, MAX und MIN für Summe, Durchschnitt, Maximum und Minimum angeben.

Auf einfache Art können Zwischentabellen berechnet werden, indem aus einer bestehenden Tabelle die gewünschten Werte in eine neue Tabelle eingelesen werden:

```
RANGE OF m IS MITARBEITER
RETRIEVE INTO TEMP_TABELLE (m.M#, m.Name)
WHERE      m.Unt = 'A6'
```

*Temporäre
Tabellen*

Analog zur RETRIEVE-Klausel gibt es eine APPEND-Klausel, die Tupel zu einer entsprechenden Tabelle hinzufügt. Möchte man im obigen Beispiel mit der Tabelle TEMP_TABELLE noch diejenigen Mitarbeiter anfügen, die in Basel wohnen, so ergibt sich die folgende Abfrage:

```
RANGE OF m IS MITARBEITER
APPEND TO TEMP_TABELLE (m.M#, m.Name)
WHERE      m.Ort = 'Basel'
```

*Erweitern von
Tabellen*

Mit den beiden Varianten RETRIEVE und APPEND lassen sich die mengenorientierten Operatoren der Relationenalgebra realisieren.

3.4.3 QBE

QBE ist eine Datenbanksprache, bei der der Benutzer seine Auswertungsvorstellungen durch Beispiele entwerfen und ausführen kann. Als erstes verlangt er ein allgemeines Skelett einer bestimmten Tabelle durch den Befehl:

*Intuitive
Auswertungen*

```
DRAW MITARBEITER
```

Damit erhält er eine grafische Darstellung der Tabelle MITARBEITER mit den dazugehörenden Merkmalen:

```
MITARBEITER  | M# | Name | Straße | Ort | Unt
-----
              |   |      |         |    |    |
```

Dieses Skelett kann nun für Selektionen verwendet werden, indem der Benutzer Anzeigebefehle (abgekürzt durch «P.»), Variablen oder

Konstanten einfügt. Möchte er beispielsweise die Namen der Mitarbeiter und die Abteilungsnummern auflisten, so könnte er das Skelett wie folgt ergänzen:

Projektion

| MITARBEITER | M# | Name | Straße | Ort | Unt |
|-------------|----|------|--------|-----|-----|
| ----- | | | | | |
| | | P. | | | P. |

Der Befehl «P.» (für *print*) entspricht der Forderung nach Darstellung sämtlicher Datenwerte der entsprechenden Tabellenspalte. Damit können Projektionen auf einzelne Merkmale der Tabelle veranlasst werden.

Möchte der Anwender alle Merkmale aus der Tabelle selektiert haben, so kann er den Anzeigebefehl direkt unter den Tabellennamen schreiben:

Selektion

| MITARBEITER | M# | Name | Straße | Ort | Unt |
|-------------|----|------|--------|-----|-----|
| ----- | | | | | |
| P. | | | | | |

Durch Selektionsbedingungen erweiterte Abfragen auf der Tabelle MITARBEITER können ebenfalls formuliert werden: Beispielsweise interessiert sich der Benutzer für alle Namen von Mitarbeitern, die in Liestal wohnen und in der Abteilung A6 arbeiten. Er fügt zu diesem Zweck in der Spalte Ort die Konstante Liestal und in der Spalte Unt die Abteilungsnummer A6 ein:

Qualifizierte Selektion

| MITARBEITER | M# | Name | Straße | Ort | Unt |
|-------------|----|------|--------|-----------|------|
| ----- | | | | | |
| | | P. | | 'Liestal' | 'A6' |

Werden Selektionsbedingungen auf der gleichen Zeile eingegeben, so entsprechen diese immer einer AND-Verknüpfung. Für eine OR-Verknüpfung werden zwei Zeilen für die entsprechenden Bedingungen gebraucht wie in folgendem Beispiel:

| MITARBEITER | M# | Name | Straße | Ort | Unt |
|-------------|----|------|--------|-----------|------|
| ----- | | | | | |
| | | P. | | 'Liestal' | |
| | | P. | | | 'A6' |

Die Abfrage entspricht einer Selektion sämtlicher Namen von Mitarbeitern, die entweder in Liestal wohnen oder in der Abteilung A6 arbeiten. Die Resultattabelle kann also mit dem äquivalenten Ausdruck der Relationenalgebra

$$\pi_{\text{Name}} (\sigma_{\text{Ort=Liestal OR Unt=A6}} (\text{MITARBEITER}))$$

bestimmt werden.

Bei Abfragen mit QBE können neben Konstanten auch Variablen verwendet werden. Diese beginnen immer mit einem Underscore-Zeichen «_», gefolgt von einer beliebigen Zeichenkette. Variablen werden beispielsweise für Verbundoperatoren benötigt. Wir nehmen an, die beiden Tabellen MITARBEITER und ABTEILUNG seien bereits durch den DRAW-Befehl verlangt worden. Interessiert sich der Anwender danach für die Namen und Adressen der Mitarbeiter, die in der Abteilung Informatik arbeiten, so lautet die Abfrage in QBE:

Variablen für den Verbundoperator

| | | | | | | | | | | |
|-------------|--|----|--|--------------|--|--------|--|-----|--|-----|
| MITARBEITER | | M# | | Name | | Straße | | Ort | | Unt |
| ----- | | | | | | | | | | |
| | | | | P. | | P. | | P. | | _A |
| ABTEILUNG | | A# | | Bezeichnung | | | | | | |
| ----- | | | | | | | | | | |
| | | _A | | 'Informatik' | | | | | | |

Verbund zweier Tabellen

Der Verbund der beiden Tabellen MITARBEITER und ABTEILUNG wird durch die Variable «_A» gebildet, die das bekannte Verbundprädikat «Unt=A#» darstellt.

Um einen neuen Mitarbeiter (M20, Müller, Riesweg, Olten, A6) in die Tabelle MITARBEITER einzufügen, schreibt der Benutzer den Einfügebefehl «I.» (für *insert*) in die Spalte des Tabellennamens und füllt die Tabellenzeile mit den Eingabedaten aus:

| | | | | | | | | | | |
|-------------|--|-------|--|----------|--|-----------|--|---------|--|------|
| MITARBEITER | | M# | | Name | | Straße | | Ort | | Unt |
| ----- | | | | | | | | | | |
| I. | | 'M20' | | 'Müller' | | 'Riesweg' | | 'Olten' | | 'A6' |

Einfügeoperator

Der Anwender kann eine Menge von Tupeln verändern oder löschen, indem er den Befehl «U.» (für *update*) oder «D.» (für *delete*) in die entsprechenden Spalten bzw. Tabellen schreibt.

Die in Liestal wohnenden Mitarbeiter aus der Tabelle MITARBEITER kann der Benutzer wie folgt entfernen:

| | | | | | | | | | | |
|-------------|--|----|--|------|--|--------|--|-----------|--|-----|
| MITARBEITER | | M# | | Name | | Straße | | Ort | | Unt |
| ----- | | | | | | | | | | |
| D. | | | | | | | | 'Liestal' | | |

Löschoperator

Wie der Name «Query By Example» andeutet, können ganz im Gegensatz zu SQL oder QUEL mit QBE keine Tabellen definiert oder Berechtigungen für die Benutzer vergeben und verwaltet werden. QBE beschränkt sich lediglich auf den Abfrage- und Manipulationsteil, ist aber diesbezüglich wie SQL und QUEL relational vollständig.

Beim Einsatz von QBE in der Praxis hat es sich gezeigt, dass für kompliziertere Abfragen QBE-Formulierungen weniger gut lesbar sind als die entsprechenden Anweisungen in SQL. Deshalb ist es

nicht verwunderlich, dass vor allem Gelegenheitsbenutzer für schwierigere Abfragen die Sprache SQL bevorzugen.

3.5 Eingebettete Sprachen

*Eingebettetes
SQL*

Die relationalen Abfrage- und Manipulationssprachen können nicht nur als selbstständige Sprachen interaktiv verwendet werden, sondern auch eingebettet in einer eigentlichen Programmiersprache (Wirtssprache). Für das Einbetten in eine Programmierumgebung müssen allerdings einige Vorkehrungen getroffen werden, auf die nun näher einzugehen ist.

*Das Cursor-
Konzept*

Das Konzept der eingebetteten Sprachen soll am Beispiel von SQL erläutert werden. Damit ein Programm durch ein SELECT-Statement eine Tabelle einlesen kann, muss es *von einem Tupel auf das nächste* zugreifen können, wozu ein CURSOR-Konzept benötigt wird.

Ein CURSOR *ist ein Zeiger*, der in einer vom Datenbanksystem vorgegebenen Reihenfolge eine Menge von Tupeln durchlaufen kann. Da ein konventionelles Programm eine ganze Tabelle nicht auf einen Schlag verarbeiten kann, erlaubt das CURSOR-Konzept ein tabellenzeilenweises Vorgehen.

Für die Selektion einer Tabelle muss ein CURSOR wie folgt im Programm definiert werden:

*Deklaration eines
Cursors*

```
DECLARE cursor-name CURSOR FOR  
SELECT-statement
```

Entsprechend wird mit einer CURSOR-Deklaration eine Veränderungsabsicht kundgetan:

```
DECLARE cursor-name CURSOR FOR  
SELECT-statement FOR  
UPDATE OF field [,field]
```

Auf diese Art wird eine Tabelle satzweise, d.h. Tupel um Tupel, abgearbeitet. Falls erforderlich, lassen sich gleichzeitig einige oder alle Datenwerte des jeweiligen Tupels verändern. Muss die Tabelle in einer bestimmten Sortierreihenfolge verarbeitet werden, so ist der obigen Spezifikation eine ORDER-BY-Klausel anzufügen.

*Mehrere Cursor
sind zulässig*

In einem Programm können zu Navigierungszwecken mehrere CURSORS verwendet werden. Diese müssen deklariert und anschließend durch OPEN- und CLOSE-Befehle aktiviert und wieder außer Kraft gesetzt werden. Der eigentliche Zugriff auf eine Tabelle und die Übertragung der Datenwerte in die entsprechenden Programmvariablen erfolgt durch einen FETCH-Befehl. Die Variab-

len, die in der Programmiersprache angesprochen werden, müssen typenkonform zu den Feldformaten der Tabellen sein. Der FETCH-Befehl lautet:

FETCH cursor-name INTO host-variable [,host-variable]

Fetch-Befehl

Jeder FETCH-Befehl positioniert den CURSOR um ein Tupel weiter, entweder aufgrund der physischen Reihenfolge der zugrunde liegenden Tabelle oder anhand einer ORDER-BY-Klausel. Wird kein Tupel mehr gefunden, so wird dem Programm ein entsprechender Statuscode geliefert.

Das CURSOR-Konzept erlaubt es, *eine mengenorientierte Abfrage- und Manipulationssprache in eine prozedurale Wirtssprache einzubetten*. So können bei SQL dieselben Sprachkonstrukte sowohl interaktiv (ad hoc) wie eingebettet (programmiersprachlich) verwendet werden. Beim Testen von eingebetteten Programmteilen zeigen sich ebenfalls Vorteile, da die Testtabellen jederzeit mit der interaktiven SQL-Sprache ausgewertet und überprüft werden können. Fairerweise muss erwähnt werden, dass das Einbetten einer relationalen Abfrage- und Manipulationssprache in eine beliebige Wirtssprache generell weniger der Übersichtlichkeit und dem strukturierten Programmierstil entgegenkommt.

Ad hoc versus eingebettet

Bevor ein Programm übersetzt und ausgeführt werden kann, muss ein Precompiler die SQL-Syntax kontrollieren. Dieser weist beispielsweise SELECT-Anweisungen, bei denen die Anzahl der ausgewählten Merkmale nicht mit der Anzahl der Variablen der INTO-Klausel übereinstimmen, als Fehler aus. Ist die Übersetzung fehlerfrei, werden Datenbankabfrage- und Zugriffsmodule generiert. Die optimale Verarbeitungsstrategie wird in einem Anwendungsplan in den Systemtabellen festgehalten. Dort wird sie während der Programmausführung wieder entnommen (vgl. Kapitel 4).

Übersetzen und Optimieren

3.6 Behandlung von Nullwerten

Beim Arbeiten mit Datenbanken kommt es immer wieder vor, dass einzelne Datenwerte für eine Tabelle nicht oder noch nicht bekannt sind. Beispielsweise möchte man einen Mitarbeitenden in die Tabelle MITARBEITER einfügen, dessen Adresse nicht vollständig vorliegt. In solchen Fällen ist es sinnvoll, anstelle wenig aussagekräftiger oder sogar falscher Werte so genannte Nullwerte (engl. *null values*) zu verwenden.

Zur Motivation von Nullwerten

Nullwerte

Ein Nullwert steht für einen Datenwert einer Tabellenspalte, der (noch) nicht bekannt ist.

Ein Nullwert ist
ein Platzhalter

Ein Nullwert – symbolisch durch das Fragezeichen «?» ausgedrückt – ist nicht mit der Ziffer «Null» oder dem Wert «Blank» (Space) zu verwechseln. Diese beiden Werte drücken bei relationalen Datenbanken einen bestimmten Sachverhalt aus, wogegen der Nullwert einen *Platzhalter* (engl. *dummy*) darstellt.

In Abb. 3-11 zeigen wir die Tabelle MITARBEITER mit Nullwerten bei den Merkmalen Straße und Ort. Natürlich dürfen nicht alle Merkmalskategorien Nullwerte enthalten, sonst sind Konflikte vorprogrammiert. Primärschlüssel dürfen definitionsgemäß keine Nullwerte aufweisen; im Beispiel gilt dies für die Mitarbeiternummer. Beim Fremdschlüssel «Unt» liegt die Wahl im Ermessen des Datenbankarchitekten. Maßgebend sind dessen Realitätsbeobachtungen.

Zur Löschregel
mit Nullsetzen

Normalerweise sind Nullwerte bei Fremdschlüsseln unerwünscht. Als Ausnahme gelten jedoch Fremdschlüssel, die unter eine spezielle Regelung der referenziellen Integrität fallen. So kann man beispielsweise bei der Löschregel für die referenzierte Tabelle ABTEILUNG angeben, ob eventuelle Fremdschlüsselverweise auf Null gesetzt werden sollen oder nicht. Die referenzielle Integritätsregel «Nullsetzen» sagt aus, dass *Fremdschlüsselwerte beim Löschen der referenzierten Tupel auf Null wechseln*. Löschen wir z.B. das Tupel (A6,

Abb. 3-11
Problematisches
Arbeiten mit
Nullwerten

| MITARBEITER | | | | |
|-------------|-----------|-------------|-------------|-----|
| M# | Name | Straße | Ort | Unt |
| M19 | Schweizer | Hauptstraße | Frenkendorf | A6 |
| M1 | Meier | ? | ? | A3 |
| M7 | Huber | Mattenweg | Basel | A5 |
| M4 | Becker | ? | ? | A6 |


```

SELECT *
FROM MITARBEITER
WHERE Ort = 'Liestal'
UNION
SELECT *
FROM MITARBEITER
WHERE NOT Ort = 'Liestal'

```


| RESULTATTABELLE | | | | |
|-----------------|-----------|-------------|-------------|-----|
| M# | Name | Straße | Ort | Unt |
| M19 | Schweizer | Hauptstraße | Frenkendorf | A6 |
| M7 | Huber | Mattenweg | Basel | A5 |

Finanz) in der Tabelle ABTEILUNG mit der Integritätsregel «Nullsetzen», so werden die beiden Fremdschlüssel der Mitarbeiter Schweizer und Becker in Abb. 3-11 auf Null gesetzt. Diese Regel ergänzt die besprochenen Regeln der restriktiven und fortgesetzten Löschung aus Abschnitt 2.5.

Das Arbeiten mit Nullwerten ist nicht ganz unproblematisch. Der Nullwert bildet nämlich neben den Werten TRUE und FALSE den neuen Informationsgehalt UNKNOWN. Damit verlassen wir die Logik, dass jede Aussage entweder falsch oder wahr ist.

Mit der Abfrage (Abb. 3-11), die sämtliche Mitarbeiter aus der Tabelle MITARBEITER selektiert, die entweder in Liestal oder außerhalb von Liestal wohnen, erhalten wir in der Resultattabelle nur eine Mitarbeiterteilmenge der ursprünglichen Tabelle, da einige Wohnorte von Mitarbeitern unbekannt sind. Dies widerspricht klar der herkömmlichen Logik, dass eine Vereinigung der Teilmenge «Mitarbeiter wohnhaft in Liestal» mit deren Komplement «Mitarbeiter NICHT wohnhaft in Liestal» die Menge aller Mitarbeitenden ergibt.

Die Aussagelogik mit den Werten TRUE, FALSE und UNKNOWN wird oft als Dreiwertlogik bezeichnet, da jede Aussage entweder «wahr» oder «falsch» oder «unbekannt» sein kann. Diese Logik ist weniger geläufig und stellt an die Anwender relationaler Datenbanken besondere Anforderungen, da Auswertungen von Tabellen mit Nullwerten schwieriger zu interpretieren sind. Deshalb wird in der Praxis entweder auf die Anwendung von Nullwerten verzichtet oder es werden anstelle von Nullwerten Default-Werte verwendet. In unserem Beispiel der Mitarbeitertabelle könnte die Geschäftsadresse als Default-Wert die noch unbekannte Privatadresse ersetzen.

Vorsicht bei Nullwerten

Nullwerte führen zur Dreiwertlogik

3.7

Datenschutzaspekte

Unter *Datenschutz* (engl. *data protection*) versteht man den Schutz der Daten vor unbefugtem Zugriff und Gebrauch. Schutzmaßnahmen sind Verfahren zur eindeutigen Identifizierung von Personen, zum Erteilen von Benutzerberechtigungen für bestimmte Datenzugriffe, aber auch kryptografische Methoden zur diskreten Speicherung oder Weitergabe von Informationen.

Im Gegensatz zum Datenschutz fallen unter den Begriff *Datensicherung* oder *Datensicherheit* (engl. *data security*) technische und softwaregestützte Maßnahmen zum Schutze der Daten vor Verfälschung, Zerstörung oder Verlust. Datensicherungsverfahren beim Sicherstellen oder Wiederherstellen von Datenbanken werden in Kapitel 4 behandelt.

Schutz vor unbefugtem Gebrauch

Schutz vor Zerstörung und Verlust

Das
View-Konzept
schränkt
Tabellen ein

Ein wesentlicher Datenschutzmechanismus besteht darin, den berechtigten Benutzern lediglich die für ihre Tätigkeit notwendigen Tabellen und Tabellenausschnitte zur Verfügung zu stellen. Dies wird durch *Sichten* (engl. *views*) auf Tabellen ermöglicht. Jede solche Sicht basiert entweder auf einer Tabelle oder auf mehreren physischen Tabellen und wird aufgrund einer SELECT-Anweisung definiert:

Viewdefinition

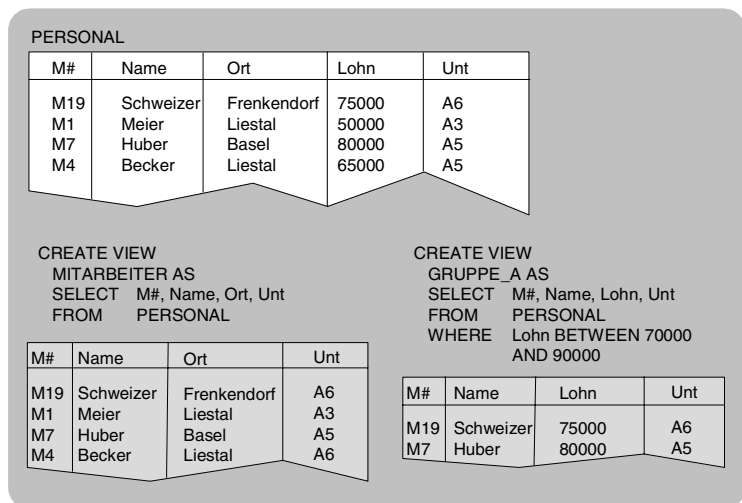
CREATE VIEW view-name AS SELECT-statement

In Abb. 3-12 werden zwei Beispiele von Sichten dargestellt, anhand der Basistabelle PERSONAL. Die Sicht MITARBEITER zeigt alle Merkmale außer den Lohnangaben. Bei der Sicht GRUPPE_A werden lediglich diejenigen Mitarbeiter mit ihren Lohnangaben gezeigt, die zwischen 70000 und 90000 Schweizer Franken im Jahr verdienen. Auf analoge Art können weitere Sichten festgelegt werden, um beispielsweise den Personalverantwortlichen pro Lohngruppe die vertraulichen Angaben zugänglich zu machen. Die beiden Beispiele in Abb. 3-12 illustrieren einen bedeutenden Schutzmechanismus. Einerseits können Tabellen durch Projektionen nur ganz bestimmter Merkmalskategorien auf Benutzergruppen eingeschränkt werden. Andererseits ist eine wertabhängige Zugriffskontrolle z.B. auf Lohnbereiche möglich, indem die Sichten in der WHERE-Klausel entsprechend spezifiziert werden.

Änderungen auf
Sichten sind
problematisch

Auf Sichten können analog zu den Tabellen Abfragen formuliert werden. Manipulationen auf Sichten lassen sich nicht in jedem Fall auf eindeutige Art durchführen. Falls eine Sicht z.B. als Verbund aus mehreren Tabellen festgelegt wurde, wird eine Änderungsoperation vom Datenbanksystem in bestimmten Fällen abgewiesen.

Abb. 3-12
Definition von
Sichten im
Umfeld des
Datenschutzes



Wichtig ist, dass verschiedene Sichten einer bestimmten Tabelle nicht mit den jeweiligen Daten redundant zur Basistabelle verwaltet werden. Vielmehr werden nur *die Definitionen der Sichten* festgehalten. Erst bei einer Abfrage auf die Sicht durch eine SELECT-Anweisung werden die entsprechenden Resultattabellen aus den Basistabellen der Sicht mit den erlaubten Datenwerten aufgebaut. Aus diesem Grunde zeigen wir in Abb. 3-12 die beiden Sichten MITARBEITER und GRUPPE_A in punktschraffierter Darstellung.

Sichten sind nicht materialisiert

Ein wirksamer Datenschutz indes verlangt mehr, als nur Tabellen durch Sichten einzuschränken. Auch die Funktionen für die Tabellen sollten benutzerspezifisch festgelegt werden können. Die SQL-Befehle GRANT und REVOKE dienen einer solchen Verwaltung von Benutzungsrechten.

Zur Weitergabe und Rücknahme von Rechten

Was mit GRANT vergeben wird, kann mit REVOKE zurückgenommen werden:

```
GRANT privilege ON relation TO user
REVOKE privilege ON relation FROM user
```

Vergabe und Rücknahme von Rechten

Der Befehl GRANT *verändert die Privilegienliste* derart, dass der begünstigte Benutzer Lese-, Einfüge- oder Löschoperationen auf bestimmten Tabellen oder Sichten durchführen darf. Entsprechend kann mit dem Befehl REVOKE ein vergebenes Recht einem Benutzer wieder weggenommen werden.

Auf der Sicht MITARBEITER in Abb. 3-12 sollen beispielsweise nur Leserechte vergeben werden:

```
GRANT SELECT ON MITARBEITER TO PUBLIC
```

Allgemeines Leserecht

Anstelle einer Benutzerliste wird in diesem Beispiel das Leserecht mit PUBLIC bedingungslos allen Benutzern vergeben, so dass diese das Recht haben, die mit der Sicht MITARBEITER eingeschränkte Tabelle anzusehen.

Möchte man die Rechte selektiv vergeben, so könnte man z.B. Beispiel für die Sicht GRUPPE_A in Abb. 3-12 nur für einen bestimmten Personalverantwortlichen mit der Benutzeridentifikation ID37289 eine Veränderungsoperation definieren:

```
GRANT UPDATE ON GRUPPE_A TO ID37289
WITH GRANT OPTION
```

Eingeschränktes Schreiberecht

Der Benutzer ID37289 kann die Sicht GRUPPE_A verändern und hat außerdem aufgrund der GRANT OPTION die Möglichkeit, dieses Recht oder ein eingeschränkteres Leserecht in Eigenkompetenz weiterzugeben und später auch wieder zurückzunehmen. Mit diesem Konzept lassen sich Abhängigkeitsbeziehungen von Rechten festlegen und verwalten.

Zur Weitergabe von Rechten

Bei der Freigabe einer relationalen Abfrage- und Manipulationssprache für den Endbenutzer darf der administrative Aufwand zur Vergabe und Rücknahme von Rechten nicht unterschätzt werden, obwohl dem Datenadministrator die Befehle GRANT und REVOKE zur Verfügung stehen. In der Praxis zeigt es sich nämlich, dass beim täglichen Änderungsdienst und bei der Überwachung der Benutzerrechte weitere Kontrollinstrumente benötigt werden. Zusätzlich verlangen auch interne und externe Kontroll- oder Aufsichtsorgane besondere Vorkehrungen, um den rechtmäßigen Umgang mit schützenswerten Daten jederzeit gewährleisten zu können (vgl. dazu die Pflichten des Datenschutzbeauftragten im Datenschutzgesetz).

3.8

Formulierung von Integritätsbedingungen

Datenbank-
system muss
Integrität
bewahren

Die Integrität einer Datenbank ist eine grundlegende Eigenschaft, die das Datenbankmanagementsystem unterstützen muss. Die Vorschriften, die bei Einfüge- oder Änderungsoperationen jederzeit gelten, werden *Integritätsregeln* (engl. *integrity constraints*) genannt. Solche Regeln werden sinnvollerweise nicht in jedem Programm einzeln spezifiziert, sondern einmal umfassend für einen Satz von Datenbanken. Bei diesen Integritätsregeln kann man deklarative und prozedurale unterscheiden.

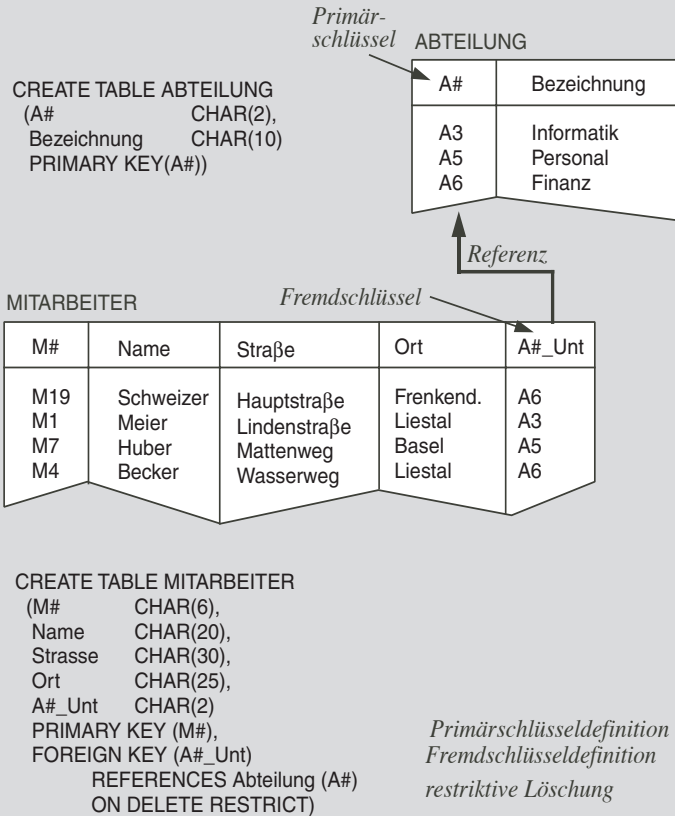
Deklarative
Integritätsregeln

Die *deklarativen Integritätsregeln* (engl. *declarative integrity constraints*) werden mit Hilfe der Datendefinitionssprache festgelegt. Im Beispiel der Abb. 3-13 wird bei der Tabelle ABTEILUNG der Primärschlüssel in Form einer Integritätsregel mit PRIMARY KEY spezifiziert. Analoges gilt für die Primär- und Fremdschlüsseldefinition in der Tabelle MITARBEITER.

Es gibt nun verschiedene Arten von deklarativen Integritätsregeln, nämlich

- *Primärschlüsseldefinition:* Mit PRIMARY KEY wird ein eindeutiger Primärschlüssel für eine Tabelle definiert. Ein Primärschlüssel darf definitionsgemäß nie Nullwerte enthalten.
- *Fremdschlüsseldefinition:* Mit FOREIGN KEY kann ein Fremdschlüssel spezifiziert werden, der durch die Angabe REFERENCES auf die zugehörige Tabelle verweist.
- *Eindeutigkeit:* Die Eindeutigkeit eines Attributes kann durch die Angabe UNIQUE festgehalten werden. Im Gegensatz zu Primärschlüsseln können eindeutige Attributswerte auch Nullwerte enthalten.

Abb. 3-13
Definition von
referenziellen
Integritätsregeln



- **Keine Nullwerte:** Mit der Angabe NOT NULL wird verhindert, dass ein Attribut Nullwerte besitzen kann.
- **Prüfregel:** Eine solche Regel kann durch den CHECK-Befehl deklariert werden. Jedes Tupel in der Tabelle muss die Prüfregel erfüllen. Beispielsweise wird mit der Angabe

CHECK Lohn > 25000

in der Tabelle PERSONAL aus Abb. 3-12 garantiert, dass das Jahreseinkommen für jeden Angestellten mindestens 25000 Schweizer Franken beträgt.

- **Löschen mit Nullsetzen:** Mit der Angabe ON DELETE SET NULL wird bei einer abhängigen Tabelle deklariert, dass beim Löschen des Tupels aus der Referenztable der Fremdschlüssel-

wert beim abhängigen Tupel auf Null gesetzt wird (vgl. Abschnitt 2.5).

- *Restriktives Löschen:* Mit ON DELETE RESTRICT können Referenztuple nicht gelöscht werden, solange sie noch abhängige Tuple besitzen (vgl. Abschnitt 2.5).
- *Fortgesetztes Löschen:* Mit der Angabe ON DELETE CASCADE kann die Löschung eines Referenztupels auf die abhängigen Tuple ausgeweitet werden (vgl. Abschnitt 2.5).

In Abb. 3-13 ist eine restriktive Löschregel spezifiziert, nämlich für die beiden Tabellen ABTEILUNG und MITARBEITER. Diese Regel garantiert, dass eine bestimmte Abteilung nur dann gelöscht werden kann, wenn sie keine abhängigen Mitarbeitertuple mehr aufweist.

Verletzung der
referenziellen
Integrität

Der Befehl

DELETE FROM Abteilung WHERE A# = 'A6'

würde somit eine Fehleranzeige ergeben, da die beiden Mitarbeiter Schweizer und Becker in der Finanzabteilung registriert sind.

Die deklarativen Integritätsregeln können neben den Löschoperationen auch bei den Einfüge- und Änderungsoperationen zur Geltung kommen. So ergibt die Einfügeoperation

INSERT INTO MITARBEITER
VALUES ('M20','Müller','Riesweg','Olten','A7')

eine Fehlermeldung. Die Abteilung A7 wird in der referenzierten Tabelle ABTEILUNG noch nicht geführt.

Prozedurale
Integritätsregeln

Bei den *prozeduralen Integritätsregeln* (engl. *procedural integrity constraints*) haben die so genannten Auslöserregeln (engl. *trigger*) an Bedeutung gewonnen. Die Auslöser oder Trigger stellen eine Alternative zu den deklarativen Integritätsregeln dar, da sie die Ausführung einer Gruppe von Anweisungen veranlassen. Ein Trigger ist im Wesentlichen durch die Angabe eines Auslösernamens, einer Datenbankoperation und einer Menge von Folgeaktionen definiert:

Deklaration von
Triggern

CREATE TRIGGER Personalbestand
ON INSERTION OF Mitarbeiter:
UPDATE Abteilung
SET Bestand = Bestand + 1
WHERE A# = A#_Unt

Auslöser
Datenbankoperation
Folgeaktion

Arbeitsweise
eines Triggers

Im obigen Beispiel nehmen wir an, dass die Tabelle ABTEILUNG ein Merkmal Bestand besitzt, das die Summe der Mitarbeiter in der jeweiligen Abteilung auflistet. Dieser Personalbestand wird durch den Trigger bei jeder Einfügeoperation in die Tabelle MITARBEITER entsprechend nachgeführt.

Das Arbeiten mit Auslöserregeln ist nicht ganz einfach, da im allgemeinen Fall einzelne Trigger weitere Trigger aufrufen können. Dabei stellt sich die Frage der Terminierung sämtlicher Folgeaktionen. Meistens ist in kommerziellen Datenbanksystemen das gleichzeitige Aktivieren von Auslösern unterbunden, damit eine eindeutige Aktionsreihenfolge garantiert bleibt und ein Trigger wohlgeordnet terminieren kann.

*Datenbank-
system
garantiert
Konfliktfreiheit*

3.9

Bibliografische Angaben

Die frühen Arbeiten von Codd (1970) beschreiben sowohl das Relationenmodell als auch die Relationenalgebra. Weitere Erläuterungen über die Relationenalgebra und den Relationenkalkül finden sich in den Standardwerken von Date (2004), Elmasri und Navathe (2006) sowie Maier (1983). Ullman (1982) zeigt die Äquivalenz von Relationenalgebra und Relationenkalkül auf.

*Grundlagen zur
Relationen-
algebra und zum
-kalkül*

Die Sprache SQL ist aus den Forschungsarbeiten für das relationale Datenbanksystem «System R» von Astrahan et al. (1976) entstanden. QUEL von Stonebraker (1986) ist die Abfrage- und Manipulationssprache des relationalen Datenbanksystems «Ingres», das neben «System R» zu den frühen relationalen Systemen zu zählen ist. QBE ist ebenfalls Mitte der siebziger Jahre durch Zloof (1977) entwickelt worden.

*Ursprung
relationaler
Sprachen*

Eine Übersicht über die verschiedenen Abfrage- und Manipulationssprachen findet sich im Datenbankhandbuch von Lockemann und Schmidt (1993). Sprachaspekte werden in den deutschsprachigen Werken von Heuer und Saake (2000), Kemper und Eickler (2006) sowie von Lang und Lockemann (1995) behandelt.

*Literatur zu
Sprachaspekten*

Über SQL gibt es mehrere Lehrbücher, z.B. Beaulieu (2006), Kuhlmann und Müllmerstadt (2004), Panny und Taudes (2000) oder Sauer (2002).

*Lehrbücher zu
SQL*

Darwen und Date (1997) widmen ihr Werk dem Standard von SQL. Pistor (1993) beschreibt in seinem Artikel die objektorientierten Konzepte des SQL-Standards. Das umfangreiche Werk von Melten und Simon (2002) beschreibt SQL:1999. Das Fachbuch Türker (2003) widmet sich dem Standard SQL:1999 und SQL:2003.

*Standard von
SQL*

4 Elemente der Systemarchitektur

4.1

Wissenswertes über die Systemarchitektur

Bei den Stärken eines relationalen Datenbanksystems haben wir insbesondere betont, dass es an der Sprach- und Benutzerschnittstelle keine Kenntnisse der inneren Struktur des Datenbanksystems voraussetzt. Weshalb widmen wir nun trotzdem ein ganzes Kapitel der Architektur relationaler Datenbanksysteme?

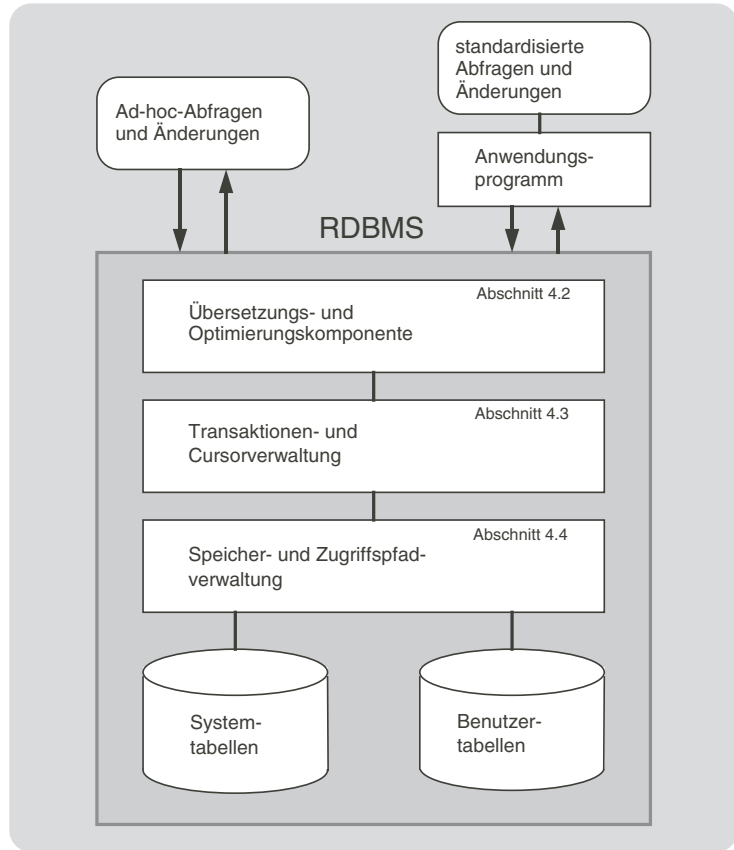
Relationale Abfrage- und Manipulationssprachen arbeiten deskriptiv im aussagekräftigen Befehlsjargon wie: «Gib mir alle Mitarbeitenden an, die in Liestal wohnen!» Es ist dabei interessant, wie eine *mengenorientierte Abfrage vom Datenbanksystem schrittweise übersetzt und verarbeitet* wird. Zu Beginn der relationalen Datenbanktheorie war nämlich umstritten, ob – und gegebenenfalls wie – ein mengenorientiertes Datenbanksystem überhaupt funktioniert. So sind beim Arbeiten mit relationalen Datenbanken z.B. die Verbundoperatoren aufwändig und zeitintensiv. Wie wir bereits wissen, drücken diese ein kartesisches Produkt in Verbindung mit einer Selektion aus. Das kartesische Produkt als Zwischenresultat zweier Tabellen kann aber recht umfangreich werden. Dieser Tatsache wird sich auch der gelegentliche Benutzer einer relationalen Datenbank spätestens dann bewusst, wenn er *längere Zeit auf die Antwort seiner Abfrage warten* muss. Zudem wird er oft überrascht, wie groß eine Resultattabelle aufgrund einer Selektionsbedingung ausfallen kann. Aus diesen Gründen ist es vorteilhaft, *Sinn und Zweck der Anfrageoptimierung* zu erfassen und die wichtigsten Strategien auf dem Weg zu einem Verbund von Tabellen zu kennen.

Meistens hat ein Benutzer eine Datenbank nicht für sich allein, sondern muss sie mit mehreren Benutzern gleichzeitig teilen. Natürlich sollte das Datenbanksystem selbst für den reibungslosen Betrieb

Durchbruch des mengenorientierten Ansatzes

Garantie des Mehrbenutzerbetriebes

Abb. 4-1
Grobübersicht
über die System-
architektur



mit konkurrierenden Benutzern verantwortlich sein. Dies führt uns zum zentralen Begriff der *Transaktion*. Darunter verstehen wir ein Programm, das *Lese- und Schreiboperationen auf einer Datenbank nach dem «Alles-oder-Nichts-Prinzip»* durchführt. Mit anderen Worten, eine Transaktion wird vom Datenbanksystem entweder vollständig oder gar nicht bearbeitet. Das Prinzip «Alles-oder-Nichts» garantiert Datenkonsistenz: Es ist ausgeschlossen, dass Transaktionen nur teilweise oder Änderungen auf der Datenbank nur lückenhaft ausgeführt werden. Die Kenntnis des Alles-oder-Nichts-Prinzips und weiterführender Konzepte der Transaktionsverwaltung fördert das Verständnis und das richtige Verhalten beim Anwendungsentwickler wie beim Datenbankbenutzer.

Erhöhung der
Effizienz durch
Parallelität

Zusätzlich stellt sich die Frage nach der *Parallelisierung von Transaktionen*, da aus Performancegründen die einzelnen Transaktionen nicht immer sequenziell nacheinander abgearbeitet werden können. Zwar braucht sich der Anwender im Einzelnen nicht um diese

Frage zu kümmern, dennoch soll gezeigt werden, wie das Datenbanksystem mit Hilfe von Protokollen oder Prüfverfahren «konkurrierende» oder «nebenläufige» (engl. *concurrent*) Transaktionen fehlerfrei abwickelt. Es wird erläutert, in welchen Fällen sich Transaktionen gegenseitig behindern können und wie solche Konflikte wiederum durch das System entschärft werden.

Schließlich ist es für den Datenbankadministrator und den Datenbankspezialisten wichtig zu wissen, wie *Tabellen intern verwaltet und nachgeführt* werden. Insbesondere bei der Implementierung größerer Datenbanken müssen die internen Parameter zur Indexierung, Speicherorganisation, Pufferdimensionierung etc. richtig gewählt werden, da diese das Leistungsvermögen (Durchsatz, Antwortzeit, Ablaufsicherung etc.) direkt beeinflussen. Als *Speicher- ebenso wie als Zugriffsstrukturen* eignen sich neben Adressberechnungsverfahren (Hashing) vor allem hierarchische Datenstrukturen (Bäume) und neuerdings hinzukommende mehrdimensionale Datenstrukturen. Verfahren zur Adressberechnung ermöglichen gestreute Speicherungsformen, Baumstrukturen verwalten Datensätze oder Zugriffsschlüssel hierarchisch, und bei den mehrdimensionalen Datenstrukturen wird der Datenraum untergliedert, so dass mit mehreren Attributen effizient zugegriffen werden kann.

In der Übersichtsdarstellung der Systemarchitektur von Abb. 4-1 sind die einzelnen Systemkomponenten wie folgt zusammengefasst:

- Die Anfrageübersetzung überführt eine deskriptive Abfrage in einen algebraischen Ausdruck der Relationenalgebra. Dabei werden optimale Umformungen und Verbundstrategien eingesetzt (Abschnitt 4.2).
- Die Transaktionenverwaltung garantiert auf konfliktfreie Art den Mehrbenutzerbetrieb: Mehrere Benutzer können gleichzeitig ein und dieselbe Datenbank bearbeiten, ohne die Integrität der Daten zu verletzen (Abschnitt 4.3).
- Mit Hilfe von Speicher- und Zugriffsstrukturen können die Tabellen durch das Datenbanksystem effizient verwaltet werden. Die Datenstrukturen müssen das dynamische Verändern von Tabelleneinträgen unterstützen (Abschnitt 4.4).
- Falls Unterbrechungen oder Fehler bei der Transaktionsverarbeitung auftreten, garantiert das Datenbanksystem den ursprünglichen Zustand der unvollständig nachgeführten Datenbanken (Abschnitt 4.5).

Gemäß Abb. 4-1 bietet ein relationales Datenbanksystem verschiedene Benutzerschnittstellen an. So können gelegentliche Benutzer ihre Ad-hoc-Abfragen starten und Datenbankauswertungen durch-

Interne Daten- und Zugriffsstrukturen

Ein Datenbanksystem ist in Schichten organisiert

Vielfältige Benutzerschnittstellen

führen. Für umfangreichere Anwendungen lassen sich standardisierte Abfragen sowie Datenbankmanipulationen ausprogrammieren.

4.2 Übersetzung und Optimierung

4.2.1 Erstellen eines Anfragebaumes

*Mengen-
orientierte
Benutzer-
schnittstelle*

Die Benutzerschnittstelle eines relationalen Datenbanksystems ist mengenorientiert, da dem Benutzer ganze Tabellen oder Sichten zur Verfügung gestellt werden. Beim Einsatz einer relationalen Abfrage- und Manipulationssprache muss vom Datenbanksystem deshalb eine Übersetzung und Optimierung der entsprechenden Anweisungen vorgenommen werden. Dabei ist wesentlich, dass sowohl das Berechnen als auch das Optimieren des so genannten Anfragebaumes ohne Benutzerintervention erfolgt.

Anfragebaum

*Ein Anfrage-
baum abstrahiert
einen SQL-
Ausdruck*

Ein Anfragebaum (engl. *query tree*) visualisiert grafisch eine relationale Abfrage durch den *äquivalenten Ausdruck der Relationenalgebra*. Die Blätter des Anfragebaumes entsprechen den für die Abfrage verwendeten Tabellen, der Wurzel- und die Stammknoten bezeichnen die algebraischen Operatoren.

Anfragebeispiel

Als Beispiel zur Illustration eines Anfragebaumes verwenden wir SQL und die bereits bekannten Tabellen MITARBEITER und ABTEILUNG (Abb. 4-2). Wir interessieren uns für eine Liste derjenigen Ortschaften, in denen die Mitarbeiter der Abteilung Informatik ihren Wohnsitz haben:

```
SELECT      Ort
FROM        MITARBEITER, ABTEILUNG
WHERE       Unt=A# AND Bezeichnung='Informatik'
```

Algebraisch können wir dieselbe Abfrage durch eine Sequenz von Operatoren ausdrücken:

```
TABELLE :=  $\pi_{Ort}$ 
           ( $\sigma_{Bezeichnung=Informatik}$ 
            (MITARBEITER  $\bowtie_{Unt=A\#}$  ABTEILUNG))
```

*Ein Anfrage-
baum wird von
unten nach oben
abgearbeitet*

Der Ausdruck berechnet zuerst einen Verbund der Tabelle MITARBEITER mit der Tabelle ABTEILUNG über die gemeinsame Abteilungsnummer. Anschließend werden im Zwischenresultat diejenigen Mitarbeiter selektiert, die in der Abteilung mit der Bezeichnung

Informatik arbeiten. Zuletzt werden durch eine Projektion die gesuchten Ortschaften zusammengestellt. Abbildung 4-2 zeigt den Ausdruck dieser algebraischen Operatoren in Form des zugehörigen Anfragebaumes.

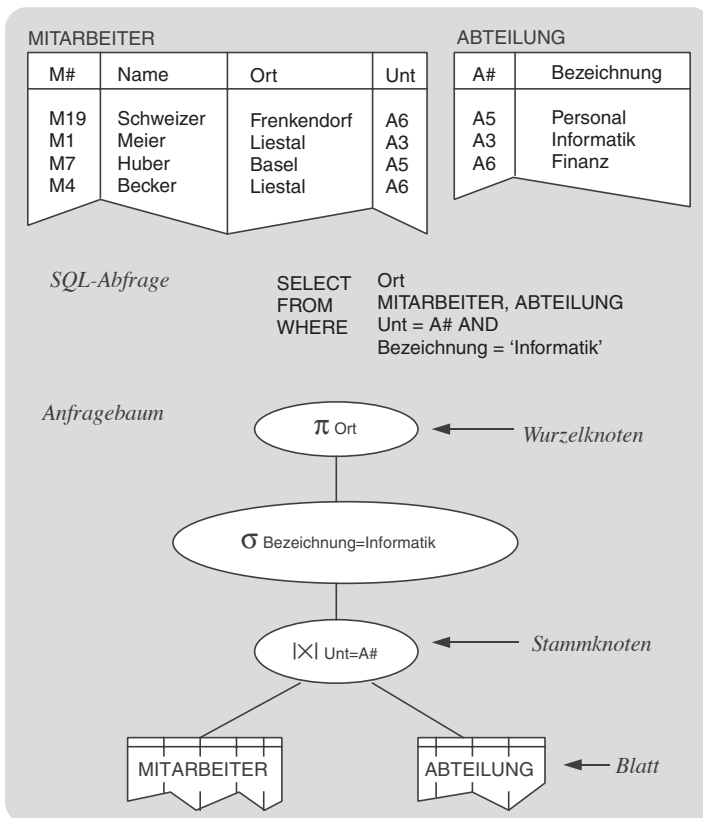
Der Anfragebaum in Abb. 4-2 kann wie folgt interpretiert werden: Die Blattknoten bilden die beiden Tabellen MITARBEITER und ABTEILUNG der entsprechenden Abfrage. Diese werden in einem ersten Stammknoten (Verbundoperator) zusammengehalten und anschließend durch einen weiteren Stammknoten (Selektionsoperator) auf diejenigen Einträge reduziert, die die Bezeichnung Informatik tragen. Der Wurzelknoten entspricht der Projektion, die die Resultattabelle mit den gesuchten Ortschaften erzeugt.

Wurzel- und Stammknoten eines Anfragebaumes verweisen entweder auf zwei Teilbäume oder auf einen Teilbaum. Je nachdem, ob die die Knoten bildenden Operatoren auf zwei Zwischentabellen (Teilbäume) oder auf eine einwirken, spricht man von binären oder unären Operatoren. *Binäre Operatoren* mit zwei Tabellen als Operanden sind die Vereinigung, der Durchschnitt, die Subtraktion, das kar-

*Interpretation des
Anfragebaumes*

*Binäre und unäre
Operatoren*

*Abb. 4-2
Anfragebaum
einer
qualifizierten
Abfrage über
zwei Tabellen*



tesische Produkt, der Verbund und die Division. *Unäre Operatoren*, die nur auf eine Tabelle wirken, sind der Projektions- und der Selektionsoperator (vgl. die früheren Abb. 3-2 und 3-3).

Systemtabellen
unterstützen die
Syntaxprüfung

Der Aufbau eines Anfragebaumes ist bei der Übersetzung und Ausführung einer relationalen Datenbankabfrage der erste Schritt. Die vom Benutzer angegebenen Tabellen- und Merkmalsnamen müssen in den Systemtabellen auffindbar sein, bevor weitere Verarbeitungsschritte stattfinden. Der Anfragebaum dient also der Überprüfung sowohl der Syntax der Abfrage als auch der Zugriffsberechtigung des Benutzers. Weitere Sicherheitsanforderungen können erst zur Laufzeit überprüft werden, wie beispielsweise wertabhängiger Datenschutz.

Von der
Optimierung zur
Codegenerierung

Nach dieser Zugriffs- und Integritätskontrolle erfolgt in einem nächsten Schritt die Wahl der Zugriffspfade und deren Optimierung, noch bevor in einem dritten Schritt die eigentliche Codegenerierung oder eine interpretative Ausführung der Abfrage stattfindet. Bei der Codegenerierung wird ein Zugriffsmodul zur späteren Verwendung in einer Modulbibliothek abgelegt; alternativ dazu übernimmt der Interpreter die dynamische Kontrolle zur Ausführung der Anweisung.

4.2.2 Optimierung durch algebraische Umformung

Vorteil
äquivalenter
algebraischer
Ausdrücke

Wie wir in Kapitel 3 gesehen haben, können die Operatoren der Relationalalgebra zusammengesetzt werden. Man spricht von *äquivalenten Ausdrücken*, wenn die algebraischen Ausdrücke trotz unterschiedlicher Operatorenreihenfolge dasselbe Resultat ergeben. Äquivalente Ausdrücke sind insofern interessant, als die Datenbankabfragen durch algebraische Umformungen optimiert werden können, ohne dass das Resultat verändert wird. Sie erlauben also, den Berechnungsaufwand zu reduzieren, und bilden damit einen wichtigen Teil der Optimierungskomponente eines relationalen Datenbanksystems.

Reihenfolge der
Operatoren
bestimmt
Effizienz

Welch großen Einfluss die Reihenfolge von Operatoren auf den Berechnungsaufwand haben kann, soll anhand der früher diskutierten Beispielabfrage illustriert werden: Den Ausdruck

$$\text{TABELLE} := \pi_{\text{Ort}} \left(\sigma_{\text{Bezeichnung} = \text{Informatik}} \left(\text{MITARBEITER} \bowtie_{\text{Unt}=\text{A\#}} \text{ABTEILUNG} \right) \right)$$

können wir – wie in Abb. 4-3 dargestellt – durch folgenden äquivalenten Ausdruck ersetzen:

$$\begin{aligned} \text{TABELLE} &:= \pi_{\text{Ort}} \\ &\quad (\pi_{\text{Unt,Ort}} (\text{MITARBEITER}) \\ &\quad \quad | \times |_{\text{Unt}=\text{A\#}} \\ &\quad \quad \pi_{\text{A\#}} \\ &\quad (\sigma_{\text{Bezeichnung}=\text{Informatik}} (\text{ABTEILUNG}))) \end{aligned}$$

Dabei wird zuerst die Selektion ($\sigma_{\text{Bezeichnung}=\text{Informatik}}$) auf die Tabelle ABTEILUNG durchgeführt, da nur die Abteilung mit der Bezeichnung Informatik relevant ist. Anschließend werden zwei Projektionsoperationen berechnet, eine auf die Mitarbeitertabelle ($\pi_{\text{Unt,Ort}}$) und eine auf die bereits erzeugte Zwischentabelle ($\pi_{\text{A\#}}$) der Abteilung Informatik. Jetzt erst wird die Verbundoperation ($| \times |_{\text{Unt}=\text{A\#}}$) über die Abteilungsnummer gebildet, anschließend auf die Ortschaften projiziert (π_{Ort}). Obwohl wir dasselbe Resultat erhalten, ist der Berechnungsaufwand auf diese Weise wesentlich geringer.

Allgemein lohnt es sich immer, *Projektions- und Selektionsoperatoren im Anfragebaum möglichst in die Nähe der «Blätter» zu bringen*. Dadurch können Zwischenresultate klein gehalten werden, bevor die zeitaufwändigen und deshalb teuren Verbundoperatoren

*Verbundoperator
spät berechnen*

*Projektionen und
Selektionen
möglichst früh
ausführen*

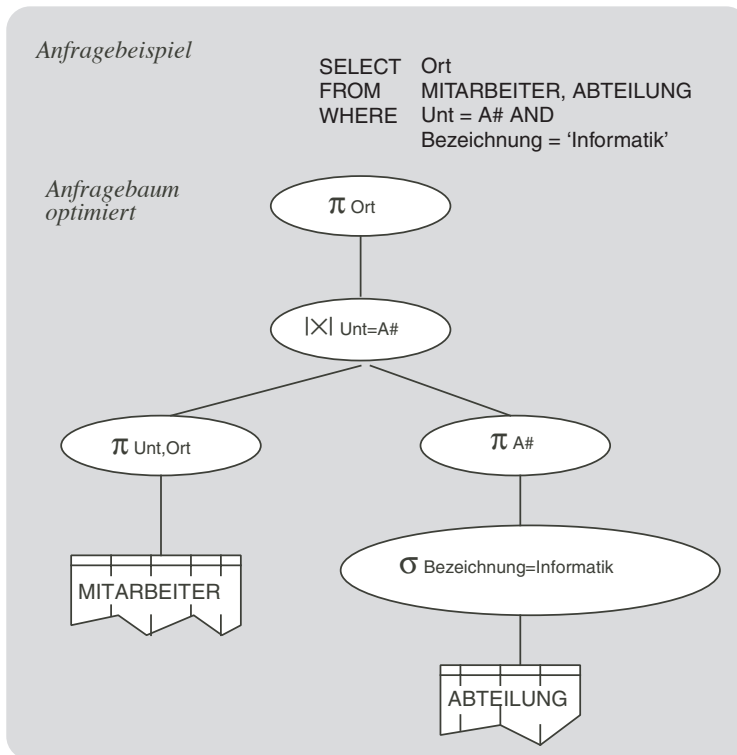


Abb. 4-3
Algebraisch
optimierter
Anfragebaum

kalkuliert werden. Gelingt eine Umformung eines Anfragebaumes dank einer solchen Berechnungsstrategie, so spricht man von einer *algebraischen Optimierung*; für sie gelten die folgenden Prinzipien:

*Die wichtigsten
Optimierungs-
grundsätze*

- Mehrere Selektionen auf ein und dieselbe Tabelle lassen sich zu einer einzigen verschmelzen, so dass das Selektionsprädikat nur einmal geprüft werden muss.
- Selektionen sind so früh wie möglich auszuführen, damit die Zwischenresultattabellen klein bleiben. Dies wird erreicht, wenn die Selektionsoperatoren so nahe wie möglich beim «Blattwerk» (bzw. bei den Ursprungstabellen) des Anfragebaumes angesiedelt werden.
- Projektionen sind ebenfalls so früh wie möglich durchzuführen, jedoch nie vor Selektionen. Sie reduzieren die Anzahl der Spalten und meistens auch die Anzahl der Tupel.
- Verbundoperatoren sind möglichst im Wurzelbereich des Anfragebaumes zu berechnen, da sie kostenaufwändig sind.

*Ausnutzen von
Daten- und
Zugriffsstrukturen*

Neben der algebraischen Optimierung können durch den Einsatz effizienter Speicher- und Zugriffsstrukturen (vgl. Abschnitt 4.4) bei der Bearbeitung einer relationalen Abfrage wesentliche Gewinne erzielt werden. So optimiert ein Datenbanksystem die einzelnen Selektions- und Verbundoperatoren aufgrund der Größe der Tabellen, der Sortierreihenfolgen, der Indexstrukturen etc. Gleichzeitig ist ein vernünftiges Modell zur Berechnung der Zugriffskosten unabdingbar, da oft mehrere Abarbeitungsvarianten in Frage kommen.

*Kostenformeln
bewerten
Verarbeitungs-
strategien*

Für das sequenzielle Suchen innerhalb einer Tabelle, das Suchen über Indexstrukturen, das Sortieren von Tabellen oder Teiltabellen, das Ausnutzen von Indexstrukturen bezüglich der Verbundmerkmale und das Berechnen von Gleichheitsverbundoperatoren über mehrere Tabellen benötigt man Kostenformeln, um den Berechnungsaufwand einer Datenbankabfrage kalkulieren zu können. Eine solche Kostenformel berücksichtigt die Anzahl der Zugriffe auf die *physischen Seiten* (engl. *physical pages*) und bildet ein gewichtetes Maß für die Ein- und Ausgabeoperationen sowie für die CPU-Belastung (CPU = Central Processing Unit). Je nach Rechnerkonfiguration beeinflussen die Zugriffszeit von externen Speichermedien und von Puffer- oder Hauptspeichern sowie die interne Rechenleistung die Kostenformel wesentlich.

4.2.3

Berechnen des Verbundoperators

Ein relationales Datenbanksystem muss über verschiedene Algorithmen verfügen, die die Operationen der Relationenalgebra bzw. des Relationenkalküls ausführen können. Gegenüber der Selektion von Tupeln aus einer einzigen Tabelle ist die Selektion aus mehreren Tabellen sehr kostenaufwändig. Deshalb gehen wir in diesem Abschnitt auf die verschiedenen Verbundstrategien näher ein, auch wenn der gelegentliche Benutzer die Berechnungsvarianten kaum beeinflussen kann.

*Unterschiedliche
Verbund-
strategien*

Die Implementierung der Verbundoperation auf zwei Tabellen zielt darauf ab, jedes Tupel der einen Tabelle bezüglich des Verbundprädikats mit jedem Tupel der anderen Tabelle zu vergleichen und gegebenenfalls als zusammengesetztes Tupel in die Resultattabelle zu stellen. Konzentrieren wir uns auf die Berechnung eines Gleichheitsverbundes, so lassen sich im Wesentlichen zwei Verbundstrategien unterscheiden, nämlich die des geschachtelten Verbundes und die des Sortier-Verschmelzungsverbundes.

*Verbund-
berechnung*

Geschachtelter Verbund

Bei einem geschachtelten Verbund (engl. *nested join*) zwischen der Tabelle R mit Merkmal A und der Tabelle S mit Merkmal B vergleichen wir *jedes Tupel in R mit jedem Tupel in S* darauf hin, ob das Verbundprädikat « $R.A=S.B$ » erfüllt ist oder nicht. Weisen die beiden Tabellen R und S je n und m Tupel auf, so sind n mal m Vergleiche aufzuwenden.

*Funktionsweise
des Nested Join*

Der Algorithmus des geschachtelten Verbundes berechnet das kartesische Produkt und prüft dabei die Erfüllung des Verbundprädikats. Da wir in einer äußeren Schleife OUTER_LOOP alle Tupel aus R mit allen Tupeln aus S der inneren Schleife INNER_LOOP vergleichen, ist der Aufwand quadratisch.

*Geschachtelter
Verbund besteht
aus zwei
Schleifen*

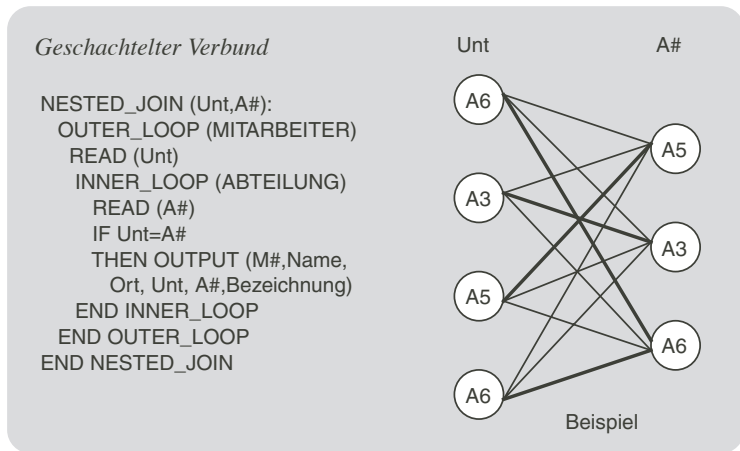
Abbildung 4-4 zeigt anhand der Abfrage über Mitarbeiter- und Abteilungsinformationen einen stark vereinfachten Algorithmus des geschachtelten Verbundes. Klar erscheinen die beiden Schleifen OUTER_LOOP und INNER_LOOP und lassen erkennen, dass dieser Algorithmus alle Tupel aus der Tabelle MITARBEITER mit allen Tupeln aus der Tabelle ABTEILUNG vergleicht.

*Beispiel eines
Verbundes*

Falls für das Merkmal A oder B ein so genannter Index existiert, können wir den Aufwand für den geschachtelten Verbund reduzieren. Unter einem *Index* (engl. *index*) eines Merkmals verstehen wir eine Zugriffsstruktur, die in einer bestimmten Reihenfolge für jeden Merkmalswert die internen Adressen der Datensätze liefert. Ein Index entspricht dem Stichwortverzeichnis eines

*Zur Nützlichkeit
eines Indexscan*

Abb. 4-4
Verbund-
berechnung
durch
Schachtelung



Buches: Auf jedes Stichwort – in alphabetischer Reihenfolge aufgeführt – folgen die Zahlen der Seiten, auf denen es im Text vorkommt.

Für
Zugriffsschlüssel
werden Indexe
aufgebaut

Für die Tabelle MITARBEITER wünschen wir beispielsweise einen Index über das Merkmal Name. Das Datenbanksystem baut nun eine Indexstruktur auf (vgl. Abschnitt 4.4), die dem Benutzer allerdings verborgen bleibt. Zu jedem Namen der Tabelle MITARBEITER wird in alphabetischer Reihenfolge in der Indexstruktur entweder der Identifikationsschlüssel M# oder die interne Adresse der Mitarbeitertupel festgehalten. Das Datenbanksystem nutzt bei einer entsprechenden Abfrage oder bei einem Verbund diesen Index der Mitarbeiternamen. Das Merkmal Name wird in diesem Fall als *Zugriffsschlüssel* bezeichnet.

Indexstrukturen
erhöhen
die Effizienz

Bei der Verbundberechnung in Abb. 4-4 besteht für das Merkmal A# ein Index, da diese Nummer der Primärschlüssel¹ der Tabelle ABTEILUNG ist. Das Datenbanksystem nutzt die Indexstruktur der Abteilungsnummer aus, indem bei der inneren Schleife nicht jedesmal sequenziell von Tupel zu Tupel die gesamte Tabelle ABTEILUNG abgesucht, sondern direkt über den Index gezielt zugegriffen wird. Im besten Fall besteht auch für das Merkmal «Unt» (Unterstellung) der Tabelle MITARBEITER ein Index, den das Datenbanksystem zu Optimierungszwecken verwenden kann. Dieses Beispiel zeigt, welche wichtige Rolle der geeigneten Auswahl von Indexstrukturen durch den Datenbankadministrator zukommt.

¹ Das Datenbanksystem baut für jeden Primärschlüssel automatisch eine Indexstruktur auf; bei zusammengesetzten Schlüsseln werden erweiterte Indexstrukturen verwendet.

Ein effizienterer Algorithmus als derjenige des geschachtelten Verbundes ist dann möglich, wenn die Tupel aus den Tabellen R und S bezüglich der beiden Merkmale A und B des Verbundprädikats physisch aufsteigend oder absteigend sortiert vorliegen. Dazu muss vor der Berechnung des eigentlichen Verbundes eventuell eine interne Sortierung vorgenommen werden, um die Tabelle R oder die Tabelle S oder beide zusammen zu ordnen. Zur Berechnung des Verbundes genügt es dann, die Tabellen nach auf- oder absteigenden Merkmalswerten des Verbundprädikats zu durchlaufen und gleichzeitig Wertvergleiche zwischen A und B vorzunehmen. Diese Strategie wird wie folgt charakterisiert:

Sortier-Verschmelzungsverbund

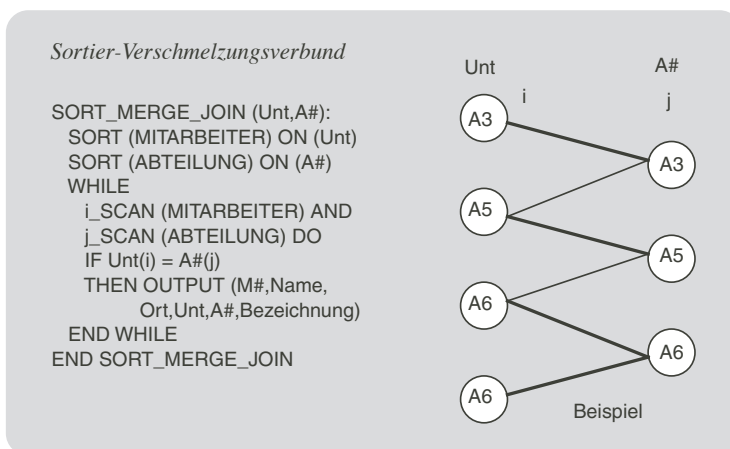
Der Sortier-Verschmelzungsverbund (engl. *sort-merge join*) setzt voraus, dass die beiden Tabellen R und S mit dem Verbundprädikat $R.A=S.B$ bezüglich der Merkmalswerte A aus R und B aus S sortiert vorliegen. Der Algorithmus berechnet den Verbund, indem er die *Vergleiche in der sortierten Reihenfolge* vornimmt. Sind die Merkmale A und B eindeutig definiert (z.B. als Primär- und als Fremdschlüssel), so ist der Aufwand linear.

Die Abb. 4-5 zeigt einen allgemeinen Algorithmus zum Sortier-Verschmelzungsverbund. Zuerst werden die beiden Tabellen anhand der im Verbundprädikat vorkommenden Merkmale sortiert. Danach werden beide Tabellen in der Sortierreihenfolge durchlaufen und die Vergleiche $R.A=S.B$ ausgeführt. Im allgemeinen Fall können die beiden Merkmale A und B in komplexer Beziehung zueinander stehen, d.h., ein und derselbe Merkmalswert kann sowohl in der Spalte R.A als auch in S.B mehrfach vorkommen. Dann gibt es bekanntlich zu einem bestimmten Tupel aus R mehrere verbundverträgliche Tupel aus S und umgekehrt. Somit müssen für solche Merkmalswerte die

*Vorteil sortierter
Merkmale*

*Funktionsweise
des Sort-Merge
Join*

*Beispiel eines
Sort-Merge Join*



*Abb. 4-5
Durchlaufen der
Tabellen in
Sortier-
reihenfolge*

entsprechenden Tupel aus R und S durch einen geschachtelten Teilverbund miteinander verknüpft werden.

*Linearer
Aufwand beim
Sortier-
Verschmelzungs-
verbund*

In unserer Abfrage in den Tabellen MITARBEITER und ABTEILUNG stellen wir fest, dass der Sortier-Verschmelzungsschritt wegen des Schlüsselmerkmals A# linear von den Vorkommen der Tupel abhängt. Die beiden Tabellen MITARBEITER und ABTEILUNG müssen zur Berechnung des Verbundes lediglich einmal durchlaufen werden.

*Nachführen der
Systemtabellen
ist notwendig*

Grundsätzlich kann die Auswahl einer geeigneten Verbundstrategie – wie überhaupt jeder Zugriffsstrategie – durch das Datenbanksystem nicht a priori getroffen werden. Im Gegensatz zur algebraischen Optimierung hängt sie nämlich vom aktuellen inhaltlichen Zustand der Datenbank ab. Aus diesem Grunde ist es wesentlich, dass die in den Systemtabellen enthaltenen statistischen Angaben entweder periodisch oder durch den Datenbankspezialisten ausgelöst regelmäßig aktualisiert werden.

4.3 Mehrbenutzerbetrieb

4.3.1 Der Begriff der Transaktion

*Konfliktfreies
Arbeiten ist durch
Mehrbenutzer-
betrieb garantiert*

Die Integrität der Daten zu gewährleisten ist eine der wichtigsten Forderungen aus der Sicht der Datenbankanwender. Die *Transaktionsverwaltung* eines Datenbanksystems dient dazu, *mehreren Benutzern ein konfliktfreies Arbeiten zu ermöglichen*. Dabei dürfen Änderungen in der Datenbank nach außen erst sichtbar werden, wenn die von den Benutzern definierten Integritätsbedingungen alle respektiert sind.

*Definition einer
Transaktion*

Unter dem Begriff der *Transaktion* (engl. *transaction*) versteht man an Integritätsregeln gebundene Datenbankoperationen, die Datenbankzustände konsistenzerhaltend nachführen. Präziser ausgedrückt, eine Transaktion ist eine Folge von Operationen, die atomar, konsistent, isoliert und dauerhaft sein muss:

*Atomarität
gehört dem
Alles-oder-
Nichts-Prinzip*

Atomarität

Eine Transaktion wird entweder komplett durchgeführt, oder sie hinterlässt keine Spuren ihrer Wirkung auf der Datenbank. Die von einzelnen Operationen erzeugten Zwischenzustände einer bestimmten Transaktion sind für die übrigen konkurrierenden Transaktionen nicht spürbar. In diesem Sinne bildet die Transaktion eine *Einheit für die Rücksetzbarkeit* nicht abgeschlossener Transaktionen (vgl. dazu Abschnitt 4.5).

Konsistenz

Während der Transaktion mögen zwar einzelne Konsistenzbedingungen zeitweise verletzt werden, bei Transaktionsende müssen jedoch alle wieder erfüllt sein. Eine Transaktion bewirkt also die Überführung einer Datenbank von einem konsistenten Zustand in einen anderen und garantiert die Widerspruchsfreiheit der Daten. Sie wird als *Einheit zur Konsistenzhaltung* aufgefasst.

Konsistenz bedeutet Widerspruchsfreiheit der Daten

Isolation

Das Prinzip der Isolation verlangt, dass gleichzeitig ablaufende Transaktionen dieselben Resultate wie im Falle einer Einbenutzerumgebung erzeugen müssen. Falls einzelne Transaktionen isoliert von parallel ablaufenden erscheinen, bleiben diese vor ungewollten Seiteneffekten geschützt. Die Transaktion gilt damit als *Einheit für die Serialisierbarkeit* (vgl. Abschnitt 4.3.2).

Isolation schützt vor Seiteneffekten

Dauerhaftigkeit

Datenbankzustände müssen so lange gültig und erhalten bleiben, bis sie von Transaktionen verändert werden. Bei Programmfehlern, Systemabbrüchen oder Fehlern auf externen Speichermedien garantiert die Dauerhaftigkeit die Wirkung einer korrekt abgeschlossenen Transaktion. Von den Wiederanlauf- und Wiederherstellungsverfahren von Datenbanken her gesehen kann jede Transaktion als *Recovery-Einheit* aufgefasst werden (vgl. Abschnitt 4.5).

Dauerhaftigkeit setzt Rekonstruierbarkeit voraus

Die vier Begriffe *Atomarität* (Atomicity), *Konsistenz* (Consistency), *Isolation* (Isolation) und *Dauerhaftigkeit* (Durability) beschreiben das so genannte *ACID-Prinzip einer Transaktion*. Dieses ist für Datenbanksysteme grundlegend und garantiert jedem Anwender, konsistente Datenbankzustände in ebensolche überführen zu können. Zwischenzeitlich inkonsistente Zustände bleiben nach außen unsichtbar und werden im Fehlerfall rückgängig gemacht.

Zum ACID-Prinzip

Um eine Folge von Operationen als Transaktion zu deklarieren, muss der Anwender die Datenbankoperationen durch ein `BEGIN_OF_TRANSACTION` und durch ein `END_OF_TRANSACTION` kennzeichnen. Start und Ende einer Transaktion signalisieren dem Datenbanksystem, welche Operationen eine Einheit bilden und durch das ACID-Prinzip geschützt werden müssen.

Deklaration von Transaktionen

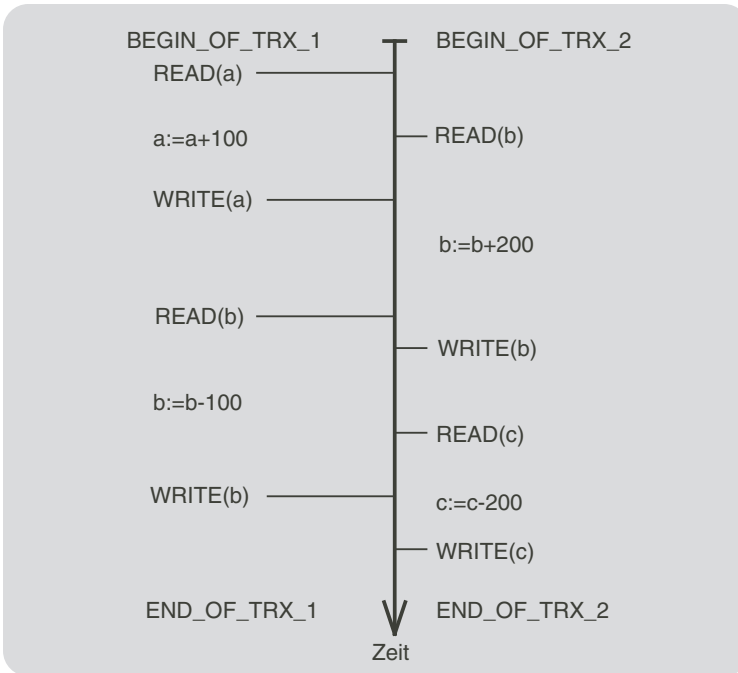
4.3.2

Serialisierbarkeit

| | |
|--|---|
| Zur Synchronisation konkurrierender Prozesse | Bei der Beschreibung von Betriebssystemen und Programmiersprachen kommt der Koordination (Synchronisation) aktiver Prozesse und dem wechselseitigen Ausschluss konkurrierender Prozesse eine große Bedeutung zu. Auch bei Datenbanksystemen müssen konkurrierende Zugriffe auf dieselben Datenobjekte serialisiert werden, da die einzelnen Datenbankanwender unabhängig voneinander arbeiten möchten. |
| Korrekte Synchronisation | <i>Prinzip der Serialisierbarkeit</i> Ein System gleichzeitig ablaufender Transaktionen heißt <i>korrekt synchronisiert</i> , wenn es eine serielle Ausführung gibt, die denselben Datenbankzustand erzeugt. |
| Analyse von Lese- und Schreiboperationen | Bei parallel ablaufenden Transaktionen garantiert das Prinzip der Serialisierbarkeit, dass die Resultate auf den Datenbanken identisch sind, gleichgültig ob die Transaktionen streng nacheinander ausgeführt worden sind oder nicht. Um Bedingungen zur Serialisierbarkeit festlegen zu können, gilt bei den einzelnen Transaktionen unser Augenmerk den READ- und WRITE-Operationen, die das <i>Lesen und Schreiben von Tupeln oder Datensätzen auf der Datenbank</i> bewerkstelligen. |
| Buchungstransaktionen müssen Konsistenz erhalten | Das klassische Beispiel zur Illustration konkurrierender Transaktionen stammt aus dem Bankbereich. Bei Buchungstransaktionen lautet die grundlegende Integritätsbedingung, dass Kontobelastungen und -gutschriften sich gegenseitig ausgleichen müssen. Abbildung 4-6 zeigt zwei parallel ablaufende Buchungstransaktionen mit ihren READ- und WRITE-Operationen in zeitlicher Abfolge. Jede Buchungstransaktion verändert für sich betrachtet die Gesamtsumme der Bestände der Konten a, b und c nicht. So schreibt die Transaktion TRX_1 dem Konto a 100 Währungseinheiten gut und belastet gleichzeitig das Gegenkonto b mit 100 Währungseinheiten. Entsprechendes gilt für die Buchungstransaktion TRX_2 mit dem Konto b und dem Gegenkonto c für den Betrag von 200 Währungseinheiten. Beide Buchungstransaktionen erfüllen somit die Integritätsbedingung der Buchführung, da sich die Salden zu Null aufheben. |
| Konfliktpotenzial wächst bei paralleler Ausführung | Bei der gleichzeitigen Ausführung der beiden Buchungstransaktionen hingegen entsteht ein <i>Konflikt</i> : Die Transaktion TRX_1 übersieht die von TRX_2 vorgenommene Gutschrift ² $b:=b+200$, da diese Wertveränderung nicht sofort zurückgeschrieben wird, und liest im |

² Unter der Zuweisung $b:=b+200$ versteht man, dass der aktuelle Bestand des Kontos b um 200 Währungseinheiten erhöht wird.

Abb. 4-6
Konfliktträchtige
Buchungs-
transaktionen



Konto b einen «falschen» Wert. Nach erfolgreichem Abschluss der beiden Buchungstransaktionen enthält das Konto a den ursprünglichen Wert plus 100 Einheiten ($a+100$), b hat sich um 100 Einheiten verringert ($b-100$) und c ist um 200 Einheiten gekürzt worden ($c-200$). Die Summe von Belastungen und Gutschriften ist nicht konstant geblieben, und die Integritätsbedingung ist verletzt, da im Konto b der Wert $b+200$ von der Transaktion TRX_1 übersehen statt verrechnet worden ist.

Wie sind nun Konfliktsituationen zu erkennen? Aus der Menge aller Transaktionen führt der Weg dieser Untersuchung jeweils über diejenigen READ- und WRITE-Operationen, die sich auf ein bestimmtes Objekt, d.h. einen einzelnen Datenwert, einen Datensatz, eine Tabelle oder im Extremfall sogar eine ganze Datenbank beziehen. Von der *Granularität* (der relativen Größe) dieser Objekte hängt es nämlich ab, wie gut die herausgepflückten Transaktionen parallelisiert werden können. Je gröber die Granularität des Objektes gewählt wird, desto kleiner wird der Grad der Parallelisierung von Transaktionen und umgekehrt. Die objektwirksamen READ- und WRITE-Operationen aus unterschiedlichen Transaktionen werden deshalb im so genannten *Logbuch* (engl. *log*) des Objektes x, im $\text{LOG}(x)$, festgehalten. Das Logbuch $\text{LOG}(x)$ eines bestimmten Objektes x listet in

Das Logbuch
protokolliert
Schreibe- und
Leseoperationen

zeitlicher Abfolge alle READ- und WRITE-Operationen auf, die auf das Objekt x zugreifen.

Nachführen des Logbuches

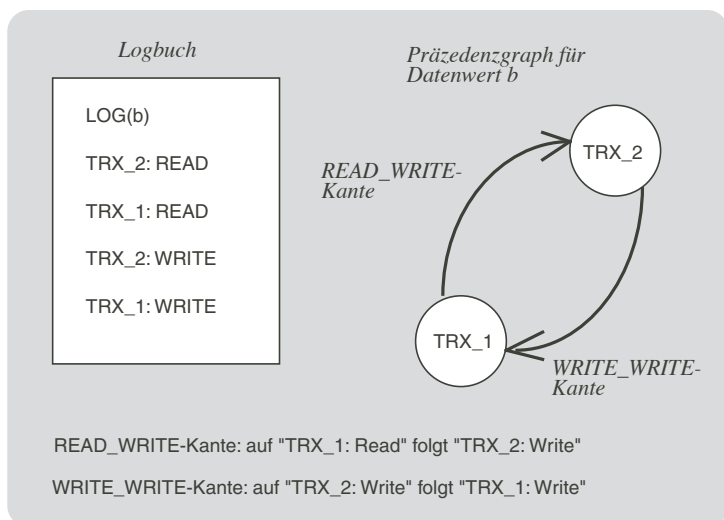
Als Beispiel für die parallelen Buchungstransaktionen TRX_1 und TRX_2 wählen wir die einzelnen Konten a, b und c als Objektgrößen. Wie in Abb. 4-7 dargestellt, erhält das Logbuch für das Objekt b beispielsweise vier Einträge (vgl. dazu auch Abb. 4-6). Zuerst liest Transaktion TRX_2 den Datenwert b, anschließend liest TRX_1 denselben Wert, noch bevor die Transaktion TRX_2 den veränderten Datenwert b zurückschreibt. Den letzten Eintrag ins Logbuch verursacht die Transaktion TRX_1, die mit ihrem veränderten Wert b jenen der Transaktion TRX_2 in der Datenbank überschreibt.

Visualisierung durch den Präzedenzgraphen

Eine Auswertung der Logbücher erlaubt uns nun auf einfache Weise, Konflikte bei konkurrierenden Transaktionen zu analysieren. Der so genannte *Präzedenzgraph* (engl. *precedence graph*) stellt die Transaktionen als Knoten und die möglichen READ_WRITE- oder WRITE_WRITE-Konflikte durch gerichtete Kanten (gebogene Pfeile) dar. Bezogen auf ein bestimmtes Objekt kann nämlich ein auf ein READ oder WRITE folgendes WRITE zu einem Konflikt führen. Hingegen gilt allgemein, dass mehrmaliges Lesen nicht konfliktträchtig ist. Aus diesem Grunde hält der Präzedenzgraph keine READ_READ-Kanten fest.

Abbildung 4-7 zeigt für die beiden Buchungstransaktionen TRX_1 und TRX_2 neben dem Logbuch des Objektes b auch den zugehörigen Präzedenzgraphen. Gehen wir vom Knoten TRX_1 aus, so folgt auf ein READ des Objektes b ein WRITE desselben durch Transaktion TRX_2, dargestellt durch eine gerichtete Kante vom Knoten TRX_1 zum Knoten TRX_2. Vom Knoten TRX_2 aus erhalten wir

Abb. 4-7
Auswertung des Logbuches durch den Präzedenzgraphen



gemäß Logbuch eine WRITE_WRITE-Kante zum Knoten TRX_1, da auf ein WRITE von TRX_2 ein weiteres WRITE desselben Objektes b von TRX_1 folgt. Der Präzedenzgraph ist also zyklisch oder kreisförmig, da von einem beliebigen Knoten ausgehend ein gerichteter Weg existiert, der zum Ursprung zurückführt. Diese zyklische Abhängigkeit zwischen den beiden Transaktionen TRX_1 und TRX_2 zeigt klar, dass sie nicht serialisierbar sind.

Serialisierbarkeitskriterium

Eine Menge von Transaktionen ist *serialisierbar*, wenn die zugehörigen Präzedenzgraphen *keine* Zyklen aufweisen.

Das Serialisierbarkeitskriterium besagt, dass die verschiedenen Transaktionen in einer Mehrbenutzerumgebung dieselben Resultate liefern wie in einer Einbenutzerumgebung. Zur Gewährleistung der Serialisierbarkeit verhindern *pessimistische Verfahren* von vornherein, dass überhaupt Konflikte bei parallel ablaufenden Transaktionen entstehen können. *Optimistische Verfahren* nehmen Konflikte in Kauf, beheben diese jedoch durch Zurücksetzen der konfliktträchtigen Transaktionen im Nachhinein.

READ_WRITE- und WRITE_WRITE-Kanten können zu Konflikten führen

Garantie der Serialisierbarkeit

Pessimistische und optimistische Verfahren

4.3.3 Pessimistische Verfahren

Eine Transaktion kann sich gegenüber anderen absichern, indem sie durch Sperren die zu lesenden oder zu verändernden Objekte vor weiteren Zugriffen schützt. *Exklusive Sperren* (engl. *exclusive locks*) sind solche, die ein bestimmtes Objekt ausschließlich von einer Transaktion bearbeiten und die übrigen konkurrierenden Transaktionen abweisen oder warten lassen. Sind solche Sperren gesetzt, müssen die übrigen Transaktionen warten, bis die entsprechenden Objekte wieder freigegeben sind.

Einführung exklusiver Sperren

In einem *Sperrprotokoll* (engl. *locking protocol*) wird festgehalten, auf welche Art und Weise Sperren verhängt bzw. aufgehoben werden. Falls Sperren zu früh oder leichtsinnig zurückgegeben werden, können nichtserialisierbare Abläufe entstehen. Auch muss verhindert werden, dass mehrere Transaktionen sich gegenseitig blockieren und eine so genannte Verklemmung oder Blockierung (engl. *deadlock*) heraufbeschwören.

Zur Verwendung eines Sperrprotokolls

Für das exklusive Sperren von Objekten sind die beiden Operationen LOCK und UNLOCK notwendig. Grundsätzlich muss jedes Objekt gesperrt werden, bevor eine Transaktion darauf zugreift. Falls ein Objekt x durch eine Sperre LOCK(x) geschützt ist, kann dieses von keiner anderen Transaktion gelesen oder verändert werden. Erst

LOCK und UNLOCK von Werten

nach Aufheben der Sperre für Objekt x durch UNLOCK(x) kann eine andere Transaktion erneut ein Sperre erwirken.

Normalerweise unterliegen Sperren einem wohldefinierten Protokoll und können nicht beliebig angefordert und zurückgegeben werden:

Regeln für das
Setzen von
Sperren

Zweiphasen- Sperrprotokoll

Das Zweiphasen-Sperrprotokoll (engl. *two-phase locking protocol*) untersagt einer Transaktion, nach dem ersten UNLOCK (Entsperren) ein weiteres LOCK (Sperren) zu verlangen.

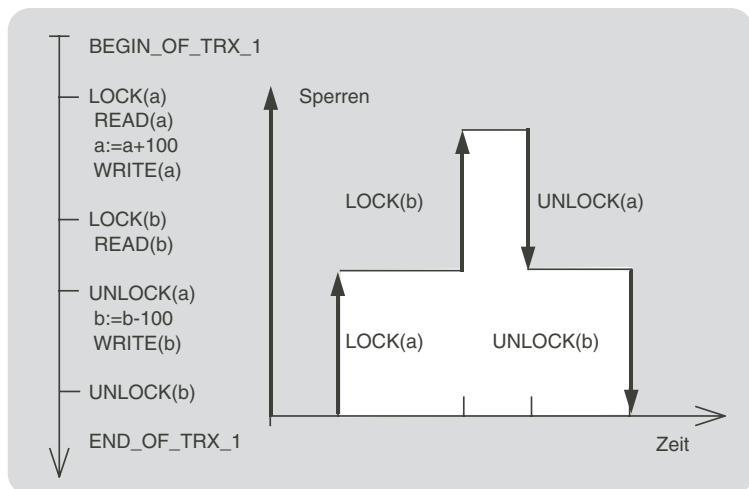
Zur Wachstums-
und
Schrumpfungs-
phase

Mit Hilfe dieses Sperrprotokolls läuft eine Transaktion immer in zwei Phasen ab: In der *Wachstumsphase* werden sämtliche Sperren angefordert und errichtet, in der *Schrumpfungsphase* werden die Sperren sukzessive wieder freigegeben. Bei einer Transaktion mit Zweiphasen-Sperrprotokoll dürfen also innerhalb der Wachstumsphase nur LOCKs nach und nach oder alle auf einmal gesetzt, jedoch nie freigegeben werden. Erst in der Schrumpfungsphase können UNLOCKS stufenweise oder insgesamt am Ende der Transaktion wieder ausgegeben werden. Das Zweiphasen-Sperrprotokoll verbietet somit ein Durchmischen von Errichten und Freigeben von Sperren.

Buchungs-
transaktion mit
Sperren

Abbildung 4-8 illustriert für die Buchungstransaktion TRX_1 ein mögliches Zweiphasen-Sperrprotokoll. In der Wachstumsphase wird nacheinander das Konto a wie das Gegenkonto b gesperrt, bevor beide Konten sukzessive wieder freigegeben werden. Bei diesem Beispiel wäre auch möglich, beide Sperren gleich zu Beginn der Transaktion zu verfügen, anstatt sie im zeitlichen Ablauf nacheinander errichten zu lassen. Analog könnten die beiden Sperren auch am

Abb. 4-8
Beispiel für ein
Zweiphasen-
Sperrprotokoll
der Transaktion
TRX_1



Ende der Transaktion TRX_1 nicht gestaffelt, sondern insgesamt aufgehoben werden.

Da die Wachstumsphase schrittweise für die Objekte a und b Sperren erwirkt, die Schrumpfungsphase diese schrittweise wieder freigibt, wird der Grad der Parallelisierung der Transaktion TRX_1 erhöht. Würden nämlich die beiden Sperren zu Beginn gesetzt und erst am Ende der Transaktion wieder zurückgegeben, müssten konkurrierende Transaktionen während der gesamten Verarbeitungszeit von TRX_1 auf die Freigabe der Objekte a und b warten.

Allgemein gilt, dass das Zweiphasen-Sperrprotokoll die Serialisierbarkeit parallel ablaufender Transaktionen garantiert.

Pessimistische Synchronisation (engl. *pessimistic concurrency control*)

Jede Menge konkurrierender Transaktionen ist dank Anwendung des Zweiphasen-Sperrprotokolls serialisierbar.

Aufgrund der strikten Trennung der Wachstums- von der Schrumpfungsphase lässt sich zeigen, dass das Zweiphasen-Sperrprotokoll zyklische Abhängigkeiten in sämtlichen Präzedenzgraphen von vornherein verhindert; die konkurrierenden Transaktionen bleiben konfliktfrei. Für die beiden Buchungstransaktionen TRX_1 und TRX_2 bedeutet dies, dass sie bei geschickter Organisation von Sperren und Entsperren parallelisiert werden können, ohne dass die Integritätsbedingung verletzt wird.

Abbildung 4-9 untermauert unsere Behauptung, dass die beiden Transaktionen TRX_1 und TRX_2 konfliktfrei ablaufen können. Dazu werden LOCKs und UNLOCKs nach den Regeln des Zweiphasen-Sperrprotokolls gesetzt. Damit kann beispielsweise das von TRX_2 gesperrte Konto b erst in der Schrumpfungsphase wieder freigegeben werden, und TRX_1 muss beim Anfordern der Sperre für b warten. Sobald TRX_2 das Konto b durch UNLOCK(b) entsperrt, fordert TRX_1 das Konto b an. Diesmal liest die Transaktion TRX_1 den «richtigen» Wert von b, nämlich $b+200$. Die beiden Transaktionen TRX_1 und TRX_2 können somit parallel ausgeführt werden.

Das Zweiphasen-Sperrprotokoll bewirkt im zeitlichen Ablauf von TRX_1 zwar eine Verzögerung, aber nach Ablauf der beiden Transaktionen bleibt die Integrität erhalten. Das Konto a hat sich um 100 Einheiten erhöht ($a+100$), das Konto b ebenfalls ($b+100$), und das Konto c wurde um 200 Einheiten reduziert ($c-200$). Die Summe der Bestände der einzelnen Konten hat sich somit nicht verändert.

Der Vergleich des in Abb. 4-9 gegebenen Logbuches LOG(b) des Kontos b mit dem früher diskutierten Logbuch aus Abb. 4-7 zeigt einen wesentlichen Unterschied: Je ein Lesen (TRX_2: READ) und ein Schreiben (TRX_2: WRITE) wird jetzt strikt zuerst durch TRX_2

Erhöhung des Parallelisierungsgrades

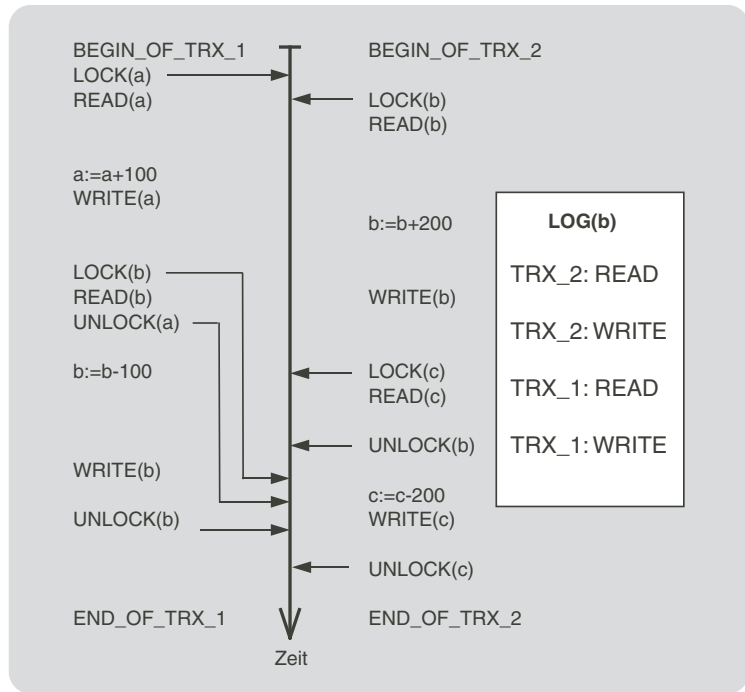
Serialisierbarkeit dank Sperrprotokoll

Zyklen im Präzedenzgraph werden vermieden

Konfliktfreie Buchung dank Sperren

Leichte Zeitverzögerung wird akzeptiert

Abb. 4-9
Konfliktfreie
Buchungs-
transaktionen



durchgeführt, bevor TRX_1 die Kontrolle über das Konto b erhält und ebenfalls Lesen (TRX_1: READ) und Schreiben (TRX_1:WRITE) darf. Der zugehörige Präzedenzgraph enthält weder READ_WRITE- noch WRITE_WRITE-Kanten zwischen den Knoten TRX_2 und TRX_1, er bleibt also zyklensfrei. Die beiden Buchungstransaktionen erfüllen damit die Integritätsbedingung.

Zur Granularität der Sperrobjekte

Bei vielen Datenbankanwendungen verbietet die Forderung nach hoher Parallelität, gleich ganze Datenbanken oder Tabellen als Sperrereinheiten zu verwenden. Man definiert deshalb kleinere Sperrgrößen, wie z.B. einen Datenbankausschnitt, einen Tabellenteil, ein Tupel oder sogar einen Datenwert. *Sperrgrößen* werden vorteilhaft so festgelegt, dass sie bei der Sperrverwaltung *hierarchische Abhängigkeiten* zulassen. Sperren wir beispielsweise eine Menge von Tupeln für eine bestimmte Transaktion, so dürfen während der Sperrzeit die übergeordneten Sperrereinheiten wie Tabelle oder zugehörige Datenbank von keiner anderen Transaktion in vollem Umfang blockiert werden. Falls ein Objekt mit einer exklusiven Sperre versehen wird, können mit Hilfe einer Sperrhierarchie die übergeordneten Objekte automatisch evaluiert und entsprechend gekennzeichnet werden.

Neben Sperrhierarchien sind verschiedene Sperrmodi von Bedeutung. Die einfachste Klassifizierung von Sperren beruht auf der Unterscheidung von Lese- und Schreibsperrern. Eine Lesesperre (engl. *shared lock*) erlaubt einer Transaktion nur den lesenden Zugriff auf das Objekt. Fordert eine Transaktion hingegen eine Schreibsperrre (engl. *exclusive lock*) an, dann darf sie lesend und schreibend auf das Objekt zugreifen.

Zur Bedeutung
von Sperrmodi

Ein weiteres pessimistisches Verfahren, das Serialisierbarkeit gewährleistet, ist die Vergabe von Zeitstempeln, um aufgrund des Alters von Transaktionen streng geordnet die Objektzugriffe durchführen zu können. Solche Zeiterfassungsverfahren erlauben, die zeitliche Reihenfolge der einzelnen Operationen der Transaktionen einzuhalten und damit Konflikte zu vermeiden.

Verwendung von
Zeitstempeln

4.3.4 Optimistische Verfahren

Bei *optimistischen Verfahren* geht man davon aus, dass Konflikte konkurrierender Transaktionen selten vorkommen. Man verzichtet von vornherein auf das Setzen von Sperren, um den Grad der Parallelität zu erhöhen und die Wartezeiten zu verkürzen. Bevor Transaktionen erfolgreich abschließen, werden rückwirkend Validierungen durchgeführt.

Validierung
rückwirkend
durchführen

Transaktionen mit *optimistischer Synchronisation* durchlaufen drei Phasen, nämlich eine *Lese-*, eine *Validierungs-* und eine *Schreibphase*. Ohne irgendwelche präventive Sperren zu setzen, werden in der Lesephase alle benötigten Objekte gelesen und in einem transaktionseigenen Arbeitsbereich gespeichert und verarbeitet. Nach Abschluss der Verarbeitung werden in der Validierungsphase die Objekte geprüft, ob die Veränderungen nicht in Konflikt mit anderen Transaktionen stehen. Ziel dabei ist, die momentan aktiven Transaktionen auf Konfliktfreiheit zu überprüfen. Behindern sich zwei Transaktionen gegenseitig, so wird die in der Validierungsphase stehende Transaktion zurückgestellt. Im Falle einer erfolgreichen Validierung werden durch die Schreibphase die Änderungen aus dem Arbeitsbereich der Transaktion in die Datenbank eingebracht.

Die drei Phasen
optimistischer
Synchronisation

Mit Hilfe transaktionseigener Arbeitsbereiche wird bei den optimistischen Verfahren die Parallelität erhöht. Lesende Transaktionen behindern sich gegenseitig nicht. Erst wenn sie Werte zurückschreiben wollen, ist Vorsicht geboten. Die Lesephasen verschiedener Transaktionen können deshalb parallel ablaufen, ohne dass Objekte durch irgendwelche Sperren blockiert sind. Dafür muss in der Validierungsphase geprüft werden, ob die im Arbeitsbereich eingelese-

Erhöhung der
Parallelität

nen Objekte überhaupt gültig sind, also mit der Wirklichkeit in der Datenbank noch übereinstimmen.

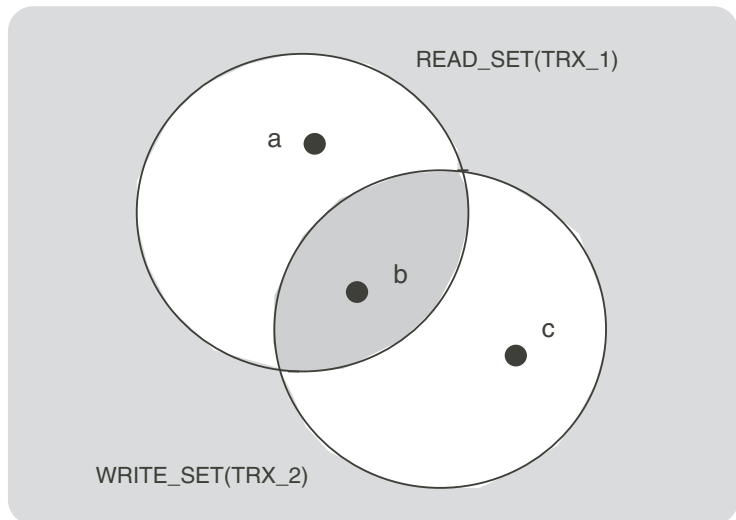
*Kritischer
Zeitpunkt der
Validierungs-
phase*

Der Einfachheit halber wird vorausgesetzt, dass sich die Validierungsphasen verschiedener Transaktionen keinesfalls überlappen. Hierfür wird der Zeitpunkt hervorgehoben, zu welchem die Transaktion in die Validierungsphase tritt. Dadurch lassen sich sowohl die Startzeiten der Validierungsphasen als auch die Transaktionen selbst zeitlich ordnen. Sobald eine Transaktion in die Validierungsphase tritt, wird die Serialisierbarkeit geprüft.

*Lese- und
Schreibmenge
analysieren*

Bei optimistischer Synchronisation wird betreffend Serialisierbarkeit wie folgt vorgegangen: Mit TRX_t sei die zu überprüfende Transaktion, mit TRX_1 bis TRX_k seien alle parallel zu TRX_t laufenden Transaktionen bezeichnet, die während der Lese- und Schreibphase von TRX_t bereits validiert haben. Alle übrigen fallen außer Betracht, da sämtliche Transaktionen streng nach der Eintrittszeit in die Validierungsphase geordnet sind. Hingegen sind die von TRX_t gelesenen Objekte zu überprüfen, sie könnten ja in der Zwischenzeit von den kritischen Transaktionen TRX_1 bis TRX_k bereits verändert worden sein. Wir bezeichnen die von TRX_t gelesene Objektmenge mit dem Ausdruck $READ_SET(TRX_t)$ und die von den übrigen Transaktionen geschriebene Objektmenge mit $WRITE_SET(TRX_1, \dots, TRX_k)$ und erhalten das folgende Serialisierbarkeitskriterium:

*Abb. 4-10
Serialisierbar-
keitsbedingung
für Transaktion
 TRX_1 nicht
erfüllt*



Optimistische Synchronisation (engl. *optimistic concurrency control*)
Bei optimistischer Synchronisation müssen die Mengen `READ_SET(TRX_t)` und `WRITE_SET(TRX_1,...,TRX_k)` *disjunkt* sein, damit die Transaktion `TRX_t` serialisierbar sein kann.

Forderung nach disjunkten Lese- und Schreibmengen

Als Beispiel können wir wiederum die beiden ursprünglichen Buchungstransaktionen `TRX_1` und `TRX_2` von Abb. 4-6 heranziehen und dabei voraussetzen, dass `TRX_2` vor `TRX_1` validiert hat. Ist in diesem Fall nun `TRX_1` serialisierbar oder nicht? Um diese Frage zu beantworten, bemerken wir (Abb. 4-10), dass das von `TRX_1` gelesene Objekt `b` von der Transaktion `TRX_2` bereits zurückgeschrieben worden ist und in der Schreibmenge `WRITE_SET(TRX_2)` liegt. Die Lesemenge `READ_SET(TRX_1)` und die Schreibmenge `WRITE_SET(TRX_2)` überlappen sich aber, was das Prüfkriterium zur Serialisierbarkeit verletzt. Die Buchungstransaktion `TRX_1` muss nochmals gestartet werden.

Buchungstransaktionen mit optimistischer Synchronisation

Eine Verbesserung des optimistischen Verfahrens bringt die präventive Garantie von Disjunktheit der Mengen `READ_SET` und `WRITE_SET`. Dabei wird in der Validierungsphase der Transaktion `TRX_t` geprüft, ob diese eventuell Objekte verändert, die bereits von anderen Transaktionen gelesen worden sind. Bei dieser Prüfvariante bleibt der Validierungsaufwand auf Änderungstransaktionen beschränkt.

Erweiterung der Verfahren

4.4

Speicher- und Zugriffsstrukturen

4.4.1

Baumstrukturen

Speicherstrukturen für relationale Datenbanksysteme müssen darauf ausgelegt sein, Daten in Sekundärspeichern effizient zu verwalten. Sind die Datenbestände umfangreich, so lassen sich Speicherstrukturen für residente Daten nicht ohne weiteres auf Hintergrundspeicher für ausgelagerte Daten übertragen. Vielmehr müssen die Speicherstrukturen modifiziert oder gar neu entworfen werden, um ein Schreiben und Lesen von Tabelleninhalten *auf externen Speichermedien mit möglichst wenig Zugriffen* zu bewerkstelligen.

Effiziente Sekundärspeicherverwaltung ist gefordert

Zur Speicherung von Zugriffsschlüsseln oder Datensätzen können Baumstrukturen verwendet werden. Liegen umfangreiche Datenbestände vor, so ordnet man den Knoten (Wurzel- und Stammknoten) und den Blättern des Baumes nicht einzelne Schlüssel oder Datensätze, sondern ganze *Datenseiten* (engl. *data pages*) zu. Zum Aufsu-

Zur Verwendung einer Baumstruktur

chen eines bestimmten Datensatzes muss dann der Baum durchsucht werden.

*Zur Verwendung
von Binärbäumen*

Bei der Hauptspeicherverwaltung verwendet man normalerweise *Binärbäume*, bei denen der *Wurzelknoten* sowie *jeder Stammknoten* zwei *Teilbäume* aufweist. Solche Bäume können nicht unbesehen für die Speicherung von Zugriffsschlüsseln oder von Datensätzen bei umfangreichen Datenbanken verwendet werden. Sie wachsen nämlich stark in die Tiefe, wenn größere Tabellen abgespeichert werden müssen. Umfangreiche Bäume sind aber für das Suchen und Lesen von Tabellen auf externen Speichermedien unerwünscht, da zu viele Seitenzugriffe notwendig sind.

*Wie können
Zugriffe reduziert
werden?*

Die *Höhe eines Baumes* – der Abstand zwischen Wurzelknoten und Blättern – ist ein *Gradmesser für die Anzahl der Zugriffe* auf externe Speichermedien. Um die Zahl der externen Zugriffe möglichst gering zu halten, versucht man bei Datenbanksystemen, die baumartigen Speicherstrukturen nicht so sehr in die Tiefe, sondern eher in die Breite wachsen zu lassen. Ein wichtiger Vertreter solcher Baumstrukturen ist der *Mehrwegbaum* (vgl. Abb. 4-11).

*Der Mehrweg-
baum mit mehr
als zwei
Teilbäumen*

Ein Mehrwegbaum ist ein Baum, dessen *Wurzel- und Stammknoten im Allgemeinen mehr als zwei Teilbäume* aufweisen. Dabei sollten die durch die einzelnen Stammknoten oder Blätter repräsentierten Datenseiten nicht leer bleiben, sondern möglichst mit Schlüsselwerten oder ganzen Datensätzen gefüllt sein. Meistens wird deshalb verlangt, dass die Seiten mindestens zur Hälfte mit Datensätzen oder Schlüsseln besetzt sind (mit Ausnahme der zum Wurzelknoten gehörenden Seite).

*Definition
des B-Baumes*

Mehrwegbaum (engl. *B-tree*)

Ein Baum ist ein *Mehrwegbaum* oder *B-Baum* der *Ordnung n*, falls

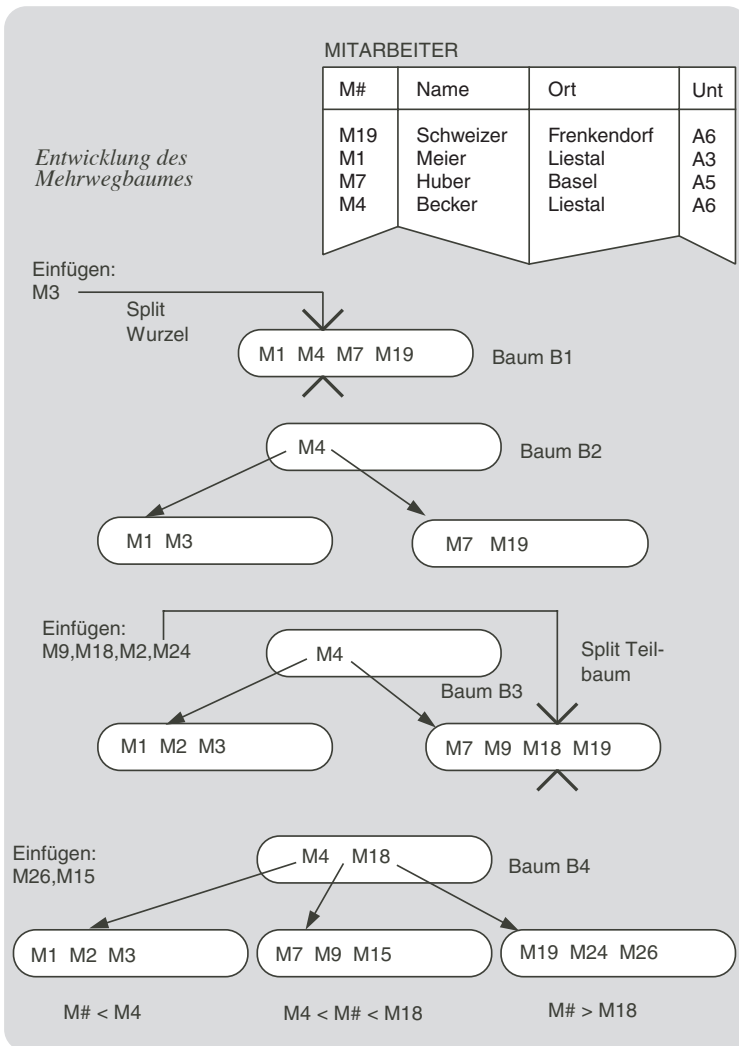
- er vollständig ausbalanciert ist (jeder Weg von der Wurzel zu einem beliebigen Blatt hat eine feste gleiche Länge), und
- jeder Knoten (außer dem Wurzelknoten) und jedes Blatt des Baumes mindestens n und höchstens 2^n Einträge in der entsprechenden Datenseite hat.

Die zweite Bedingung eines Mehrwegbaumes lässt sich auch anders interpretieren: Da jeder Knoten außer der Wurzel mindestens n Einträge aufweist, besitzt jeder Knoten mindestens n Teilbäume. Umgekehrt enthält jeder Knoten höchstens 2^n Einträge, d.h., jeder Knoten eines Mehrwegbaumes kann höchstens auf 2^n Teilbäume verweisen.

*Beispiel eines
B-Baumes der
Ordnung 2*

Betrachten wir dazu unsere Tabelle MITARBEITER mit dem zugehörigen Schlüssel Mitarbeiternummer. Wollen wir den Schlüssel M# in einem Mehrwegbaum der Ordnung $n=2$ als Zugriffsstruktur able-

Abb. 4-11
Mehrwegbaum
in dynamischer
Veränderung



gen, so erhalten wir das in Abb. 4-11 gezeigte Bild. Knoten und Blätter des Baumes können somit nicht mehr als vier Einträge enthalten. Neben den eigentlichen Schlüsseln nehmen wir stillschweigend an, dass die zu den Knoten und Blättern zählenden Seiten nicht nur Schlüsselwerte, sondern auch Zeiger auf Datenseiten aufweisen, die die eigentlichen Datensätze enthalten. Der Baum in Abb. 4-11 repräsentiert somit einen Zugriffsbaum und nicht die Datenverwaltung für die in der Tabelle MITARBEITER gezeigten Tupel bzw. Datensätze.

*Erhalt eines
ausbalancierten
Baumes*

Der Wurzelknoten des Mehrwegbaumes enthält in unserem Beispiel die vier Schlüssel M1, M4, M7 und M19 in Sortierreihenfolge. Beim Einfügen eines zusätzlichen Schlüssels M3 muss der Wurzelknoten geteilt werden, da er keinen Eintrag mehr erlaubt. Die Teilung geschieht so, dass ein ausbalancierter Baum entsteht. Der Schlüssel M4 wird zum Wurzelknoten erklärt, da er die restliche Schlüsselmenge in zwei gleichgroße Hälften zerlegt. Der linke Teilbaum entspricht Schlüsselwerten mit der Bedingung «M# kleiner als M4» (d.h. in unserem Fall M1 und M3), der rechte entspricht «M# größer als M4» (d.h. M7 und M19). Auf analoge Art werden weitere Schlüssel eingefügt, unter Beibehaltung einer festen Baumhöhe.

*Zum
Suchvorgang*

Beim Suchen eines bestimmten Schlüssels geht das Datenbanksystem wie folgt vor: Wird der Schlüsselkandidat M15 im Mehrwegbaum B4 der Abb. 4-11 nachgefragt, so vergleicht es ihn mit den Einträgen des Wurzelknotens. M15 liegt zwischen den Schlüsseln M4 und M18, also wählt es den entsprechenden Teilbaum (hier ein Blatt) aus und setzt seine Suche fort. Schließlich findet es den Eintrag im Blatt. Der Aufwand für die Suche des Schlüssels M15 beträgt in diesem vereinfachten Beispiel lediglich zwei Seitenzugriffe, nämlich einen für den Wurzelknoten und einen für das Blatt.

*Höhe bestimmt
Zugriffszeit*

Die Höhe des Mehrwegbaumes bestimmt die Zugriffszeit der Schlüssel und entsprechend auch der zu einem (Such-)Schlüssel gehörenden Daten. Eine Verbesserung der Zugriffszeiten wird erreicht, indem man beim Mehrwegbaum den Verzweigungsgrad weiter erhöht.

Der B-Baum ist
blattorientiert*

Eine andere Möglichkeit besteht beim so genannten *blattorientierten Mehrwegbaum* (bekannt unter dem Namen B*-Baum). Bei diesem werden die eigentlichen Datensätze nie in inneren Knoten, sondern immer in den Blättern des Baumes gespeichert. Die Knoten weisen allesamt nur Schlüsseleinträge auf, um den Baum möglichst niedrig zu halten.

4.4.2 Hash-Verfahren

*Gestreute
Speicherung
durch Hashing*

Schlüsseltransformations- oder Adressberechnungsverfahren (engl. *key hashing* oder einfach *hashing*) bilden die Grundlage von gestreuten Speicher- und Zugriffsstrukturen. Eine *Schlüsseltransformation* (engl. *hash function*) ist eine Abbildung einer Menge von Schlüsseln in eine Menge von Adressen, die einen zusammenhängenden Adressraum bilden.

*Einfache
Schlüssel-
transformation*

Eine einfache Schlüsseltransformation ordnet jedem Schlüssel einer Tabelle eine natürliche Zahl von 1 bis n als Adresse zu. Diese Adresse wird als relative Seitennummer interpretiert, wobei die Seite

eine fixe Anzahl von Schlüsselwerten aufnimmt, inklusive oder exklusive dazugehörige Datensätze.

An Schlüsseltransformationen werden die folgenden Anforderungen gestellt:

- Die Transformationsvorschrift muss mit einfacher Berechnung und ohne große Kosten eingehalten werden können.
- Die belegten Adressen müssen gleichmäßig über den Adressraum verteilt sein.
- Die Wahrscheinlichkeit für Mehrfachbelegungen, d.h. die Verwendung gleicher Adressen für mehrere Schlüssel, sollte für alle Schlüsselwerte gleich groß sein.

*Forderungen zur
Adress-
berechnung*

Es besteht eine beachtliche Anzahl von Hash-Funktionen, die alle ihre Vor- und Nachteile haben. Als bekanntestes und einfachstes Verfahren gilt die Restklassenbildung, das Hashing mit Divisionsrest.

Hashing mit Divisionsrest

Jeder Schlüssel wird als natürliche Zahl interpretiert, indem die Bitdarstellung verwendet wird. Die Schlüsseltransformation oder *Hash-Funktion H* für einen Schlüssel *k* und eine Primzahl *p* ist durch die Formel

$$H(k) := k \bmod p$$

gegeben. Der ganzzahlige Rest « $k \bmod p$ » – der Division des Schlüsselwertes *k* durch die Primzahl *p* – bildet eine relative Adresse oder Seitennummer. Bei diesem Divisionsrestverfahren bestimmt die Wahl der Primzahl *p* die Speicherausnutzung und den Grad der Gleichverteilung.

*Hash-Funktion
mit Divisionsrest*

In Abb. 4-12 zeigen wir die Tabelle MITARBEITER, die wir mit der obigen Restklassenbildung auf verschiedene Seiten abbilden. Dabei nehmen wir für unser einfaches Beispiel an, dass jede Seite vier Schlüsselwerte aufnehmen kann. Als Primzahl wählen wir die Ziffer 5. Jeder Schlüsselwert wird nun durch 5 dividiert, wobei der ganzzahlige Rest die Seitennummer bestimmt.

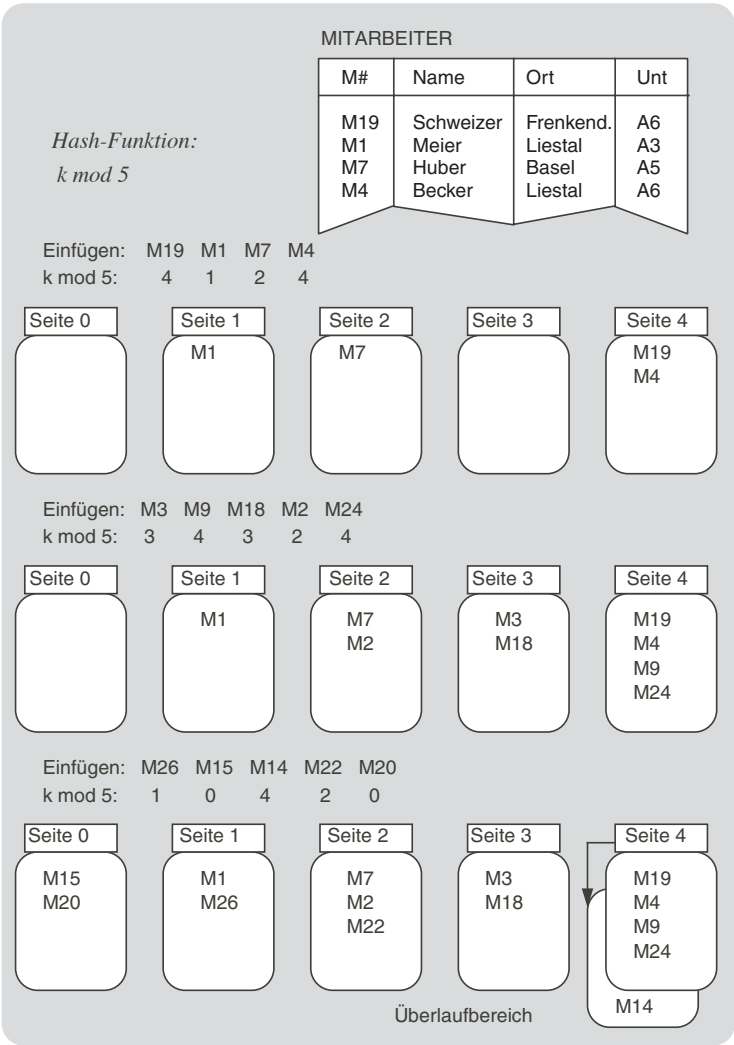
*Beispiel einer
Restklassen-
bildung*

Beim Einfügen des Schlüssels M14 kommt es zu einer Kollision, da die entsprechende Seite bereits gefüllt ist. Der Schlüssel M14 wird in einen *Überlaufbereich* gestellt. Ein Verweis von der Seite 4 auf den Überlaufbereich garantiert die Zugehörigkeit des Schlüssels M14 zur Restklasse 4.

Es existieren unterschiedliche Verfahren zur Behandlung von Überläufern. Anstelle eines Überlaufbereichs können für die Überläufer selbst wieder Schlüsseltransformationen angewendet werden. Bei stark anwachsenden Schlüsselbereichen oder bei größeren Löschoptionen treten bei der Überlaufbehandlung oft Schwierig-

*Zur Behandlung
von Überläufern*

Abb. 4-12
Schlüssel-
transformation
mit Restklassen-
bildung



keiten auf. Um diese Probleme zu entschärfen, sind dynamische Verfahren zur Schlüsseltransformation entwickelt worden.

Dynamische
Adressverfahren

Bei *dynamischen Hash-Verfahren* wird versucht, die Belegung des Speicherplatzes unabhängig vom Wachstum der Schlüssel zu halten. Überlaufbereiche oder umfassende Neuverteilungen von Adressen werden weitgehend vermieden. Bei dynamischen Hash-Verfahren kann ein bestehender Adressraum entweder durch eine geschickte Wahl der Schlüsseltransformation oder durch Verwendung einer hauptspeicherresidenten Seitenzuordnungstabelle erweitert werden, ohne dass alle bereits gespeicherten Schlüssel oder Datensätze neu geladen werden müssen.

4.4.3

Mehrdimensionale Datenstrukturen

Mehrdimensionale Datenstrukturen unterstützen den Zugriff auf Datensätze mit mehreren Zugriffsschlüsselwerten. Die Gesamtheit dieser Zugriffsschlüssel wird *mehrdimensionaler Schlüssel* genannt. Ein solcher ist immer eindeutig, braucht aber nicht in jedem Fall minimal zu sein. Unter einer *mehrdimensionalen Datenstruktur* (engl. *multi-dimensional data structure*) versteht man nun eine Datenstruktur, die einen mehrdimensionalen Schlüssel unterstützt. Beispielsweise lässt sich eine Tabelle MITARBEITER mit den beiden Schlüsselteilen Mitarbeiternummer und Jahrgang als zweidimensionale Datenstruktur auffassen. Die Mitarbeiternummer bildet einen Teil des zweidimensionalen Schlüssels und bleibt nach wie vor eindeutig. Das Merkmal Jahr bildet den zweiten Teil und dient als zusätzlicher Zugriffsschlüssel, wobei diese Angabe nicht eindeutig zu sein braucht.

Bei den mehrdimensionalen Datenstrukturen strebt man – im Gegensatz zu den Baumstrukturen – an, dass kein Schlüsselteil bei der Speicherung der physischen Datensätze die Reihenfolge bestimmt. Eine mehrdimensionale Datenstruktur ist *symmetrisch*, wenn sie den Zugriff über mehrere Zugriffsschlüssel ermöglicht, ohne einen bestimmten Schlüssel oder eine Schlüsselkombination zu bevorzugen. Für unsere Beispieltabelle MITARBEITER ist es wünschenswert, dass beide Schlüsselteile, Mitarbeiternummer wie Jahrgang, gleichberechtigt sind und bei einer konkreten Abfrage den Zugriff effizient unterstützen.

Als bedeutende mehrdimensionale Datenstruktur ist die so genannte Gitterdatei, bekannt unter dem englischen Namen *Grid File*, zu erwähnen:

Gitterdatei

Eine Gitterdatei ist eine mehrdimensionale Datenstruktur mit folgenden Eigenschaften:

- Sie unterstützt den Zugriff bezüglich eines *mehrdimensionalen Zugriffsschlüssels* auf symmetrische Art, d.h., keine Dimension des Schlüssels ist dominant.
- Sie erlaubt das Lesen eines beliebigen Datensatzes mit *zwei Seitenzugriffen*, dem ersten auf das Gitterverzeichnis und dem zweiten auf die Datenseite selbst.

Eine Gitterdatei besteht aus einem Gitterverzeichnis und einer Datei mit den Datenseiten. Das Gitterverzeichnis stellt einen mehrdimensionalen Raum dar, wobei jede Dimension einem Teil des mehrdimen-

Datenstrukturen mit mehrdimensionalen Schlüsseln

Forderung nach Symmetrie

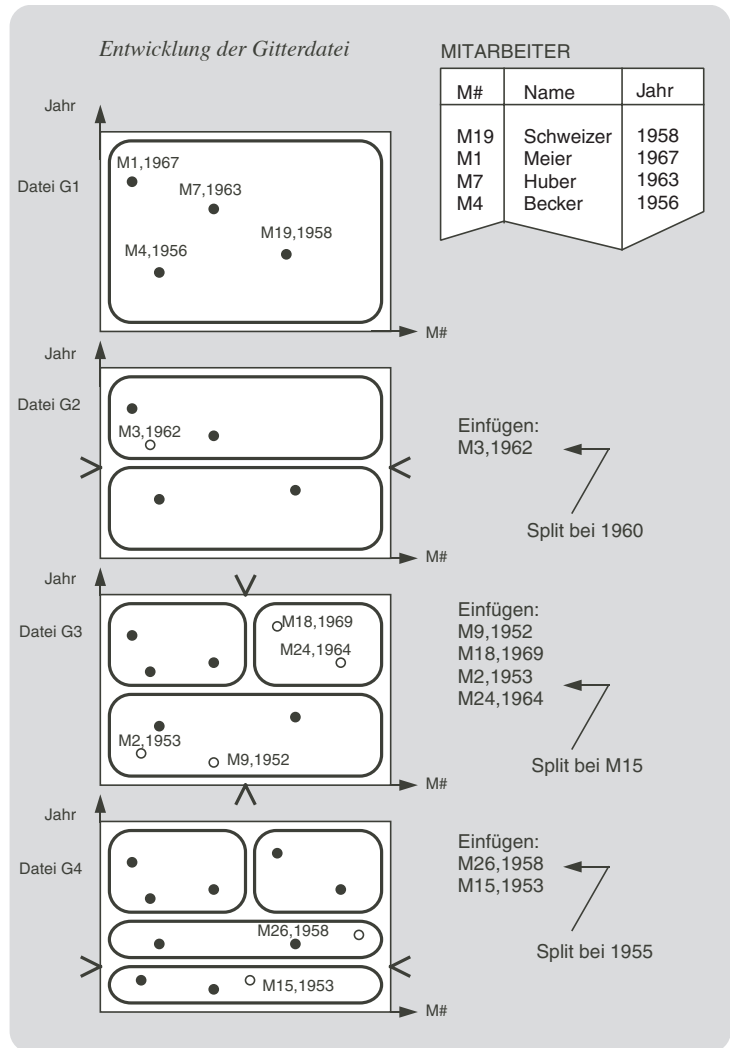
Definition Grid File

Zum Gitterverzeichnis

sionalen Zugriffsschlüssels entspricht. Beim Einfügen von Datensätzen wird das Verzeichnis alternierend in den Dimensionen in Zellen unterteilt. Im Beispiel in Abb. 4-13 wechseln wir für den zweidimensionalen Zugriffsschlüssel alternierend nach Mitarbeiternummer und Jahrgang. Die entsprechenden Unterteilungsgrenzen werden als Skalen des Gitterverzeichnisses bezeichnet.

Eine Zelle des Gitterverzeichnisses entspricht einer Datenseite und enthält mindestens n Einträge und maximal $2 \cdot n$ Einträge. Leere Zellen im Gitterverzeichnis müssen zu größeren Zellen zusammengefasst werden, damit die zugehörigen Datenseiten die Minimalzahl der

Abb. 4-13
Dynamisches
Unterteilen des
Gitter-
verzeichnisses



Einträge aufnehmen können. In unserem Beispiel nehmen wir wiederum an, dass höchstens vier Einträge in den Datenseiten vorkommen können ($n=2$).

Da das Gitterverzeichnis im Allgemeinen sehr groß ist, muss es wie die Datensätze im Sekundärspeicher gehalten werden. Die Menge der Skalen hingegen ist klein und kann resident im Hauptspeicher liegen. Somit geschieht ein Zugriff auf einen spezifischen Datensatz wie folgt: Mit den k Schlüsselwerten einer k -dimensionalen Gitterdatei durchsucht das System die Skalen und stellt fest, in welchem Intervall der jeweilige Teil des Suchschlüssels liegt. Die so bestimmten Intervalle erlauben einen direkten Zugriff auf den entsprechenden Abschnitt des Gitterverzeichnisses. Jede Zelle des Verzeichnisses enthält die Nummer der Datenseite, in der die zugehörigen Datensätze abgespeichert sind. Mit einem weiteren Zugriff auf die Datenseite der Zelle kann schließlich festgestellt werden, ob sie den gesuchten Datensatz enthält oder nicht.

Beim Suchen eines beliebigen Datensatzes in einer Gitterdatei ist das *Zwei-Seiten-Zugriffsprinzip* immer gewährleistet, das heißt, dass höchstens zwei Seitenzugriffe auf den Sekundärspeicher notwendig sind; der erste führt in den richtigen Abschnitt des Gitterverzeichnisses, der zweite zur richtigen Datenseite.

Beispielsweise wird der Mitarbeiter mit der Nummer M18 und dem Jahrgang 1969 in der Gitterdatei G4 der Abb. 4-13 wie folgt gesucht: Die Mitarbeiternummer M18 liegt im Skalenbereich M15 bis M30, d.h. in der rechten Hälfte der Gitterdatei. Der Jahrgang 1969 liegt zwischen den Skalen 1960 und 1970 und somit in der oberen Hälfte. Das Datenbanksystem findet also anhand der Skalen mit einem ersten Zugriff die Adresse der Datenseite im Gitterverzeichnis. Ein zweiter Zugriff auf die entsprechende Datenseite führt zu den gesuchten Datensätzen mit den Zugriffsschlüsseln (M18,1969) und (M24,1964).

Eine k -dimensionale Gitterdatei unterstützt die Anfrage nach einem einzelnen Datensatz oder nach einem Bereich von Datensätzen: Durch eine *Punktfrage* (engl. *point query*) kann anhand von k Zugriffsschlüsseln der entsprechende Datensatz gesucht werden. Es ist auch möglich, mit einer *Teilpunktfrage* nur einen Teil des Schlüssels zu spezifizieren. Mittels einer *Bereichsfrage* (engl. *range query*) kann für jeden der k Schlüsselteile ein Bereich untersucht werden. Sämtliche Datensätze werden bereitgestellt, für die ihre Schlüsselteile in den jeweiligen Bereichen liegen. Es kann auch hier nur für einen Teil der Schlüssel ein Bereich angegeben und ausgewertet werden (so genannte Teil-Bereichsfrage).

Eine Punktfrage entspricht z.B. dem bereits dargelegten Aufsuchen des Datensatzes (M18,1969). Wissen wir lediglich den Jahrgang des Mitarbeiters, so spezifizieren wir zur Suche das Jahr 1969 als

*Verwendung von
Gitterskalen*

*Punktsuche mit
zwei Zugriffen
garantiert*

*Beispiel einer
Suche*

*Effiziente
Unterstützung
von
Bereichsfragen*

*Suchraum ist
reduziert*

Teilpunktfrage. Wir können auch (Teil-)Bereichsfragen stellen, indem wir z.B. sämtliche Mitarbeiter mit Jahrgang 1960 bis 1969 abfragen. Auf unser Beispiel der Abb. 4-13 bezogen, liegen wir damit in der oberen Hälfte des Gitterverzeichnis G4 und müssen deshalb nur die beiden entsprechenden Datenseiten durchsuchen. Bei einer mehrdimensionalen Gitterdatei können auf diese Weise Bereichs- und Teilbereichsfragen beantwortet werden, ohne dass die gesamte Datei durchkämmt werden muss.

*Unterstützung
von
geografischen
Informationssystems*

In den letzten Jahren sind verschiedene mehrdimensionale Datenstrukturen untersucht und beschrieben worden, die auf vorteilhafte Weise mehrere Zugriffsmerkmale symmetrisch unterstützen. Obwohl die Nutzung mehrdimensionaler Datenstrukturen bei relationalen Datenbanksystemen heute noch bescheiden ist, verlangen webbasierte Suchvorgänge vermehrt nach solchen Speicherstrukturen. Insbesondere müssen geografische Informationssysteme sowohl topologische wie geometrische Anfragen effizient unterstützen.

4.5 Fehlerbehandlung

*Vielfalt von
Fehlerquellen*

Beim Betrieb einer Datenbank können verschiedene Fehler auftreten, die normalerweise durch das Datenbanksystem selbst entschärft oder behoben werden können. Einige Fehlersituationen wie Integritätsverletzungen oder Zusammenbrüche im Transaktionenverkehr sind bei der Behandlung parallel ablaufender Transaktionen bereits zur Sprache gekommen. Andere Fehler können durch das Betriebssystem oder durch die Hardware verursacht werden. Beispielsweise kann es vorkommen, dass die Daten nach einem Speicherfehler auf dem externen Medium unlesbar bleiben.

*Zum Recovery
einer Datenbank*

Das *Wiederherstellen eines korrekten Datenbankzustandes nach einem Fehlerfall* steht unter dem Begriff Recovery. Beim Recovery ist es wesentlich zu wissen, wo ein Fehler aufgetreten ist: im Anwendungsprogramm, in der Datenbanksoftware oder bei der Hardware. Bei Integritätsverletzungen oder nach einem «Absturz» eines Anwendungsprogrammes genügt es, eine Transaktion oder auch mehrere rückgängig zu machen und anschließend zu wiederholen. Bei schwerwiegenden Fehlern müssen im Extremfall frühere Datenbestände aus Archiven geholt und durch eine teilweise wiederholte Transaktionsverarbeitung rekonstruiert werden.

*Funktionsweise
der Logdatei*

Um Transaktionen rückgängig zu machen, benötigt das Datenbanksystem gewisse Angaben. Normalerweise wird vor der Veränderung eines Objektes eine Kopie (engl. *before image*) desselben in

eine so genannte *Logdatei*³ (engl. *log file*) geschrieben. Außer den alten Werten des Objektes werden auch Marken in die Logdatei gesetzt, die den Beginn und das Ende einer Transaktion signalisieren.

Damit die Logdatei im Fehlerfall effizient benutzt werden kann, werden entweder auf Grund von Anweisungen im Anwendungsprogramm oder bei bestimmten Systemereignissen *Sicherungspunkte* (engl. *checkpoints*) gesetzt. Ein systemweiter Sicherungspunkt enthält eine Liste der bis zu diesem Zeitpunkt aktiven Transaktionen. Bei einem *Neustart* (engl. *restart*) muss das Datenbanksystem nur den letzten Sicherungspunkt suchen und die noch nicht abgeschlossenen Transaktionen rückgängig machen (z.B. mit dem SQL-Befehl ROLLBACK).

Ein solcher Vorgang ist in Abb. 4-14 dargestellt: Nach dem Systemzusammenbruch muss die Logdatei rückwärts bis zum jüngsten Sicherungspunkt gelesen werden. Von Interesse sind dabei diejenigen Transaktionen, die noch nicht mit einer so genannten EOT-Marke (EOT = End Of Transaction) ihre korrekte Beendigung signalisieren konnten, wie die beiden Transaktionen TRX_2 und TRX_5. Für diese muss nun mit Hilfe der Logdatei der alte Datenbankzustand hergestellt werden (engl. *undo*). Dabei muss im Falle der Transaktion TRX_5 vom Sicherungspunkt weg rückwärts bis zur BOT-Marke (BOT = Begin Of Transaction) gelesen werden, um das Before-Image der Transaktion TRX_5 zu erhalten. Unabhängig von der Art des Sicherungspunktes muss auch der neueste Zustand (engl. *after image*) für mindestens TRX_4 hergestellt werden (engl. *redo*).

Setzen von
Checkpoints

Das Rollback-
Verfahren

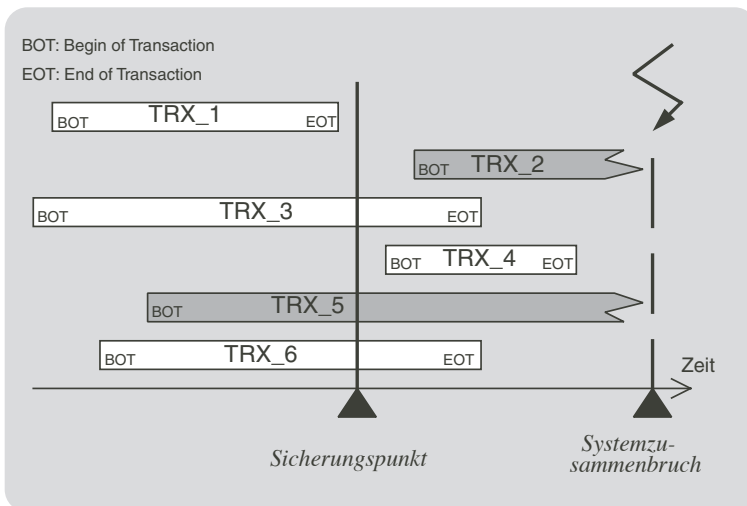


Abb. 4-14
Neustart
eines Datenbanksystems
nach einem Fehlerfall

³ Die Logdatei ist nicht zu verwechseln mit dem Logbuch aus Abschnitt 4.3.

*Zur Organisation
eines Archivs*

Zur Rekonstruktion einer Datenbank nach einem Defekt auf einem externen Speicher benötigt man eine Archivkopie der Datenbank und eine Sammlung sämtlicher in der Zwischenzeit erfolgter Änderungen. Archivkopien werden normalerweise vor und nach der Tagesendverarbeitung gezogen, da dies sehr zeitintensiv ist. Tagüber behilft man sich mit der Protokollierung der Änderungen auf der Logdatei, wobei pro Objekt jeweils die neuesten Zustände festgehalten werden.

*Notwendigkeit
der
Katastrophen-
vorsorge*

Das Sicherstellen von Datenbanken erfordert von den Datenbankspezialisten ein klares Vorgehenskonzept zur *Katastrophenvorsorge*. Normalerweise werden die Sicherheitskopien in Generationen teilweise redundant und physisch getrennt archiviert. Das Bereitstellen solcher Archivkopien und das Löschen alter Bestände muss laufend protokolliert werden. Im Fehlerfall oder bei Katastrophenübungen gilt es, aus alten Archivbeständen und sichergestellten Datenbankänderungen aktuelle Datenbestände innerhalb einer nützlichen Frist zu reproduzieren.

4.6 Die Systemarchitektur im Detail

*Forderung nach
unabhängigen
Systemebenen*

Bei der Systemarchitektur von Datenbanksystemen gilt als unabdingbares Prinzip, künftige Veränderungen oder Erweiterungen lokal begrenzen zu können. Wie beim Implementieren von Betriebssystemen oder von anderen Softwarekomponenten führt man auch bei relationalen Datenbanksystemen *voneinander unabhängige Systemebenen* ein, die über vordefinierte Schnittstellen miteinander kommunizieren.

Die fünf Ebenen der Systemarchitektur erläutern wir im Überblick in Abb. 4-15 und ordnen ihnen im Folgenden die wichtigsten Funktionen aus den vorangehenden Abschnitten zu:

*Schicht der
Anfrage-
übersetzung
und -optimierung*

Schicht 1: Mengenorientierte Schnittstelle

In der ersten Schicht werden Datenstrukturen beschrieben, Operationen auf Mengen bereitgestellt, Zugriffsbedingungen definiert und Konsistenzanforderungen geprüft (vgl. Abschnitt 4.2). Entweder bereits bei vorgezogener Übersetzung und Generierung von Zugriffsmodulen oder erst zur Laufzeit müssen die Syntax geprüft, Namen aufgelöst und Zugriffspfade ausgewählt werden. Bei der Auswahl von Zugriffspfaden können wesentliche Optimierungen vorgenommen werden.

Schicht 2: Satzorientierte Schnittstelle

Die zweite Schicht überführt logische Sätze und Zugriffspfade in physische Strukturen. Ein Cursorkonzept erlaubt das Navigieren oder Durchlaufen von Datensätzen nach der physischen Speicherungsreihenfolge, das Positionieren bestimmter Datensätze innerhalb einer Tabelle oder das Bereitstellen von Datensätzen in wertabhängiger Sortierreihenfolge. Dabei muss mit Hilfe einer Transaktionenverwaltung gemäß Abschnitt 4.3 gewährleistet werden, dass die Konsistenz der Datenbank erhalten bleibt und mehrere Benutzeranforderungen sich nicht gleichzeitig behindern.

Schicht zur Konsistenzgewährung

Schicht 3: Speicher- und Zugriffsstrukturen

Die dritte Schicht implementiert physische Datensätze und Zugriffspfade auf Seiten. Die Zahl der Seitenformate ist zwar begrenzt, jedoch sollten neben Baumstrukturen und Hash-Verfahren künftig auch mehrdimensionale Datenstrukturen unterstützt werden. Diese typischen Speicherstrukturen sind so ausgelegt, dass sie den Zugriff auf externe Speichermedien effizient bewerkstelligen. Daneben kön-

Schicht für effiziente Zugriffe

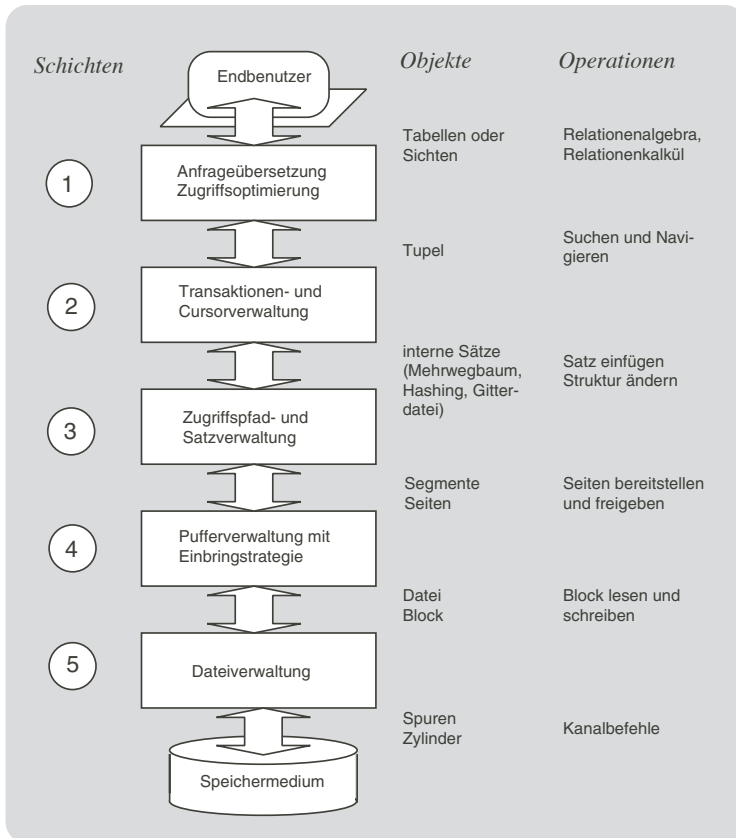


Abb. 4-15
Fünf-Schichten-Modell für relationale Datenbanksysteme

nen physische Clusterbildung und mehrdimensionale Zugriffspfade bei der Satz- und Zugriffspfadverwaltung weitere Optimierungen erzielen.

*Schicht zur
Pufferverwaltung*

Schicht 4: Seitenzuordnung

Aus Effizienzgründen und für die Implementierung von Recovery-Verfahren unterteilt die vierte Schicht den linearen Adressraum in so genannte Segmente mit einheitlichen Seitengrenzen. Auf Anforderung werden durch die Dateiverwaltung in einem Puffer Seiten bereitgestellt. Umgekehrt können durch eine Einbringungs- oder Ersetzungsstrategie Seiten in den Datenbankpuffer eingebracht oder in diesem ersetzt werden. Es gibt nicht nur die direkte Zuordnung von Seiten zu Blöcken, sondern auch indirekte Zuordnungen wie so genannte Schattenspeicher- oder Cacheverfahren, durch die sich mehrere Seiten atomar in den Datenbankpuffer einbringen lassen.

*Schicht des
Dateisystems*

Schicht 5: Speicherzuordnung

Die fünfte Schicht realisiert Speicherzuordnungsstrukturen und bietet der übergeordneten Schicht eine blockorientierte Dateiverwaltung. Dabei bleiben die Hardware-Eigenschaften den datei- und blockbezogenen Operationen verborgen. Normalerweise unterstützt die Dateiverwaltung dynamisch wachsende Dateien mit definierbaren Blockgrößen. Außerdem sollte die Clusterbildung von Blöcken möglich sein sowie die Ein- und Ausgabe mehrerer Blöcke mit Hilfe einer einzigen Operation.

4.7

Bibliografische Angaben

*Literatur zur
System-
architektur*

Einige Arbeiten widmen sich ausschließlich der Architektur von Datenbanksystemen, teilweise beschränkt auf einzelne Ebenen der Schichtarchitektur. Härder (1978) sowie Härder und Rahm (2001) stellen grundlegende Konzepte für die Implementierung relationaler Datenbanksysteme vor (Schichtenmodell). Auch das Datenbankhandbuch von Lockemann und Schmidt (1993) behandelt schwerpunktmäßig Aspekte der Datenarchitektur. Optimierungsfragen werden in Ullman (1982) und Maier (1983) eher von der theoretischen Seite her angegangen.

*Zur
Transaktionen-
verwaltung*

Das ACID-Prinzip geht auf Härder und Reuter (1983) zurück. Gray und Reuter (1993), Weikum (1988) sowie Weikum und Vossen (2002) erläutern Transaktionskonzepte im Detail. Bernstein et al. (1987) beschreiben Mehrbenutzeraspekte und Recovery-Maßnahmen. Reuter (1981) zeigt Verfahren zur Fehlerbehandlung bei Datenbanksystemen. Castano et al. (1995) erklären unterschiedliche Verfahren zur Datenbanksicherung.

Die Dissertation von Schaarschmidt (2001) stellt Konzepte und Sprachen für die Archivierung von Datenbanken zusammen. Eine weitere Dissertation von Störl (2001) widmet sich dem Backup und dem Recovery in Datenbanksystemen.

Wiederhold (1983) befasst sich mit Speicherstrukturen für Datenbanksysteme. Der Mehrwegbaum stammt von Bayer (1972), Hash-Funktionen bieten Maurer und Lewis (1975) im Überblick, und die Gitterdatei geht auf Nievergelt et al. (1984) zurück.

*Speicher- und
Zugriffsstrukturen*

5 Integration und Migration von Datenbanken

5.1

Zur Nutzung heterogener Datenbestände

Viele Datenbestände sind im Laufe der Zeit strukturell verändert und stückweise erweitert worden. Als Resultat existieren heute in vielen Unternehmen inkonsistente und redundante Datensammlungen, die auf eine neue Basis gestellt werden müssen. Man spricht in diesem Zusammenhang von Altlasten (engl. *legacy systems*), die systematisch erneuert oder abgelöst werden müssen.

Eine Lösungsoption besteht darin, Teile der *Informationssysteme ins Web einzubetten*. Damit können die Daten und Informationen sowohl den internen Benutzern wie auch wichtigen Kundengruppen auf einfache Art und Weise zur Verfügung gestellt werden. Die Integration von Daten aus heterogenen Quellen im Web stellt eine besondere Herausforderung dar: Einerseits sind die bestehenden Datensammlungen in der Zwischenzeit vorwiegend in relationalen Datenbanken abgelegt, andererseits bieten die relationalen Datenbanksysteme zur Verwaltung von Hyperdokumenten nicht immer optimale Lösungen.

In Abschnitt 5.2 werden Lösungsvorschläge behandelt, wie man sinnvollerweise Datenbanken im Web einbetten kann. Neben Architekturaspekten wird auch kurz auf die Entwicklung von XML (eXtensible Markup Language) und XML-Datenbanken eingegangen, damit diese Technologie mit der relationalen Datenbanktechnologie verglichen und beurteilt werden kann.

Eine weitere Aufgabe ergibt sich, falls die Daten teilweise noch in Dateisystemen oder in herkömmlichen Datenbanken, z.B. hierarchisch oder netzwerkartig, verwaltet werden. Der Einsatz unterschiedlicher Datenbanksysteme verlangt vom Unternehmen, für die jeweiligen Technologien geeignete Datenbankspezialisten und unterschiedlich geschulte Anwendungsentwickler zu beschäftigen. Schon aus diesen Gründen ist man bestrebt, *mit der Zeit die Altlasten zu*

Zum Umgang mit Altlasten

Datenbanken im Web

XML im Aufwind

*Entwicklungs-
umgebung ist
gefordert*

beheben und sämtliche Datensammlungen in relationale oder postrelationale Datenbanken zu überführen.

*Komplexität bei
heterogenen
Systemen
reduzieren*

Das Nebeneinander von heterogenen Datenbeständen bildet aus wirtschaftlichen Gründen eine Hypothek, da neben personellen Aufwänden auch *Sicherheit und Verfügbarkeit komplexer Systemlandschaften* ihren Preis haben. Zudem müssen sich die Unternehmen fragen, ob sie mit überalterten und teilweise heterogen zusammengesetzten Informationssystemen den Konkurrenzkampf bestehen und die Existenz sichern können.

*Abbildungs-
regeln zwischen
Quell- und
Zielsystem sind
festzulegen*

Bevor in diesem Kapitel verschiedene Migrationsvarianten vorgestellt werden, wird auf die *Integration von Datenbankschemas* eingegangen. Dazu werden in Kapitel 5.3 Abbildungsregeln diskutiert, die sowohl der Integration wie Migration von Datenbeständen dienen. Insbesondere muss festgelegt werden, wie Datenbestände eines Quellsystems (*legacy system*) in Datenbestände des Zielsystems überführt werden können. Falls eine Migration verschiedener Informationssysteme vorgesehen wird, kann ein solches Vorhaben mehrere Monate, wenn nicht Jahre, dauern. Aus diesem Grunde müssen die angesprochenen Abbildungsregeln für die Datenbankschemas eindeutig definiert sein, damit man die veralteten Bestände bei Bedarf noch eine Zeitlang nachführen kann.

*Vorteil der Ko-
existenzvariante*

In Kapitel 5.4 werden nicht nur unterschiedliche Migrationsvarianten diskutiert, sondern eine Koexistenzvariante im Detail besprochen. Eine *Koexistenz paralleler Datenbestände auf Zeit* hat den Vorteil, dass man die Anwendungssysteme nicht unter großem Zeitdruck und entsprechenden Risiken umschreiben muss. Vielmehr kann man sich bei dieser Umstellung nach den Bedürfnissen des Marktes ausrichten und die Ablösung von Altlasten kontinuierlich vorantreiben.

*Planung und
schrittweise
Umsetzung*

Sowohl die Integration von Datenbeständen im Web wie die Migration von Anwendungssystemen muss geplant und schrittweise umgesetzt werden. Zur Illustration einer *Integrations- und Migrationsplanung* wird deshalb in Abschnitt 5.5 auf wichtige Grundsätze hingewiesen.

5.2 Datenbanken im Web

*Informationen im
Internet anbieten*

Das World-Wide Web (WWW oder Web) hat als wichtiger Teil der Internettechnologie in den letzten Jahren eine rasante Entwicklung durchgemacht. Mehr und mehr werden die Informations- und Datenbanksysteme ins Web eingebettet, damit das breite Informationsangebot sowohl von offenen wie geschlossenen Benutzergruppen genutzt werden kann. Insbesondere wird versucht, mit Datenbank- und

Applikationsservern die bestehenden Informationssysteme zu integrieren.

5.2.1

Aufbau eines webbasierten Informationssystems

In einem *webbasierten Informationssystem* (engl. *web-based information system*) werden wichtige Dokumente und Informationen *online* verfügbar gemacht. Solche Systeme dienen nicht nur dem Informationsaustausch, sondern werden für die Beziehungspflege mit den Kunden (engl. *customer relationship management*) und für die Abwicklung elektronischer Geschäfte genutzt. Zudem werden Offertstellungen, Vertragsvereinbarungen, Distribution und Zahlungsverkehr vermehrt online organisiert, vor allem in der Wertschöpfungskette mit den Lieferanten (engl. *supply chain management*).

Webbasierte Systeme unterstützen die Geschäftsprozesse

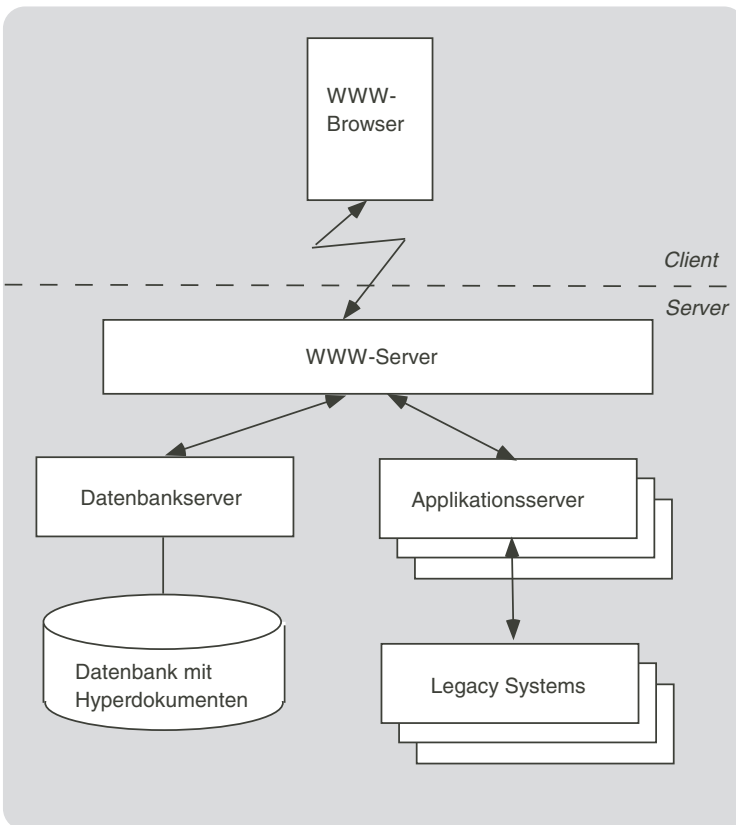


Abb. 5-1
Komponenten
eines
webbasierten
Informationssystems

Zur
Grobarchitektur

In Abb. 5-1 ist die Grobarchitektur eines webbasierten Informationssystems schematisch dargestellt. Kernelement bildet der WWW-Server, der über ein Kommunikationsprotokoll (HTTP = HyperText Transfer Protocol) Informationen in Hypertextdokumenten zur Verfügung stellt. Solche Dokumente werden vorwiegend in der Sprache HTML (HyperText Markup Language) oder der weiterführenden XML (eXtensible Markup Language) abgefasst. Auf die Verwaltung semistrukturierter Daten wird in Abschnitt 5.2.2 näher eingegangen.

Vermehrter
Einsatz mobiler
Endgeräte

Auf die von einem WWW-Server angebotenen Informationen kann im Normalfall rund um die Uhr und von jedem beliebigen Standort aus zugegriffen werden. Voraussetzung dazu ist ein Gerät (engl. *client*) mit einem WWW-Browser. Solche Geräte müssen nicht stationär sein, sondern können mobil eingesetzt werden; Beispiele sind Laptops, Mobiltelefone, Palms, E-Books oder digitale Assistenten.

Arbeitsteilung
zwischen
Datenbank- und
Applikations-
server

Die Hypertextdokumente liegen entweder statisch im Dateisystem des WWW-Servers oder werden dynamisch vom Server beim Zugriff eines Benutzers erzeugt. Für die dynamische Dokumentengenerierung stehen zahlreiche Methoden und Techniken zur Verfügung, die hier nicht näher erläutert werden. In der Regel liegen die für die Dokumentenerzeugung benötigten Informationen auf dem Datenbankserver. Darüber hinaus können Daten aus den bestehenden Informationssystemen (engl. *legacy systems*) mit speziellen Schnittstellen erschlossen werden. Die so genannten Applikationsserver dienen u.a. der Verarbeitung von eingehenden Aufträgen und greifen ebenfalls auf den Datenbankserver oder die bestehenden Informationssysteme zu.

Einsatz von
objekt-
relationalen und
XML-
Datenbanken

Die Betreiber eines webbasierten Informationssystems resp. einer Website sehen sich mit einer *umfangreichen Menge von Hypertextdokumenten* konfrontiert. Aus diesem Grunde benutzen sie Datenbankserver, um die Hypertextdokumente längerfristig ablegen zu können. Der Inhalt von dynamisch generierten HTML-Seiten kann aus relationalen Datenbanken stammen. Ganze Dokumente könnten auch in objektrelationalen Datenbanksystemen abgelegt sein (vgl. Abschnitt 6.4). Eine alternative Speicherungsoption bildet die Verwaltung der semistrukturierten Daten in XML-Datenbanken. Dieser Ansatz soll im nächsten Abschnitt näher betrachtet werden.

5.2.2 XML-Dokumente und XML-Schemas

Die Markup
Language XML

Die *Auszeichnungssprache* XML (eXtensible Markup Language) wurde vom World-Wide Web Consortium (W3C) entwickelt. Die Inhalte von Hypertextdokumenten werden wie bei HTML durch Tags markiert. Ein XML-Dokument ist selbstbeschreibend, da es neben

den eigentlichen Daten auch Informationen über die Datenstruktur mitführt:

```
<Adresse>
  <Straße> Rue Faucigny </Straße>
  <Nummer> 2 </Nummer>
  <Postleitzahl> 1700 </Postleitzahl>
  <Ort> Fribourg </Ort>
</Adresse>
```

*Ausschnitt eines
XML-
Dokumentes*

Die Grundbausteine von XML-Dokumenten sind die so genannten Elemente. Diese bestehen aus einem Start-Tag (in spitzen Klammern <Name>) und einem End-Tag (in spitzen Klammern mit Schrägstrich </Name>), dazwischen steht der Inhalt des Elementes. Die Bezeichner des Start- und End-Tags müssen übereinstimmen.

*Elemente sind
durch Tags
markiert*

Die Tags liefern Informationen über die Bedeutung der konkreten Werte, somit sagen sie etwas über die Datensemantik aus. Elemente können in XML-Dokumenten beliebig geschachtelt werden. Zur Darstellung solcher *hierarchisch strukturierter Dokumente* wird sinnvollerweise ein Graph verwendet; ein Beispiel ist in Abb. 5-2 gegeben.

*XML-
Dokumente
können beliebig
strukturiert sein*

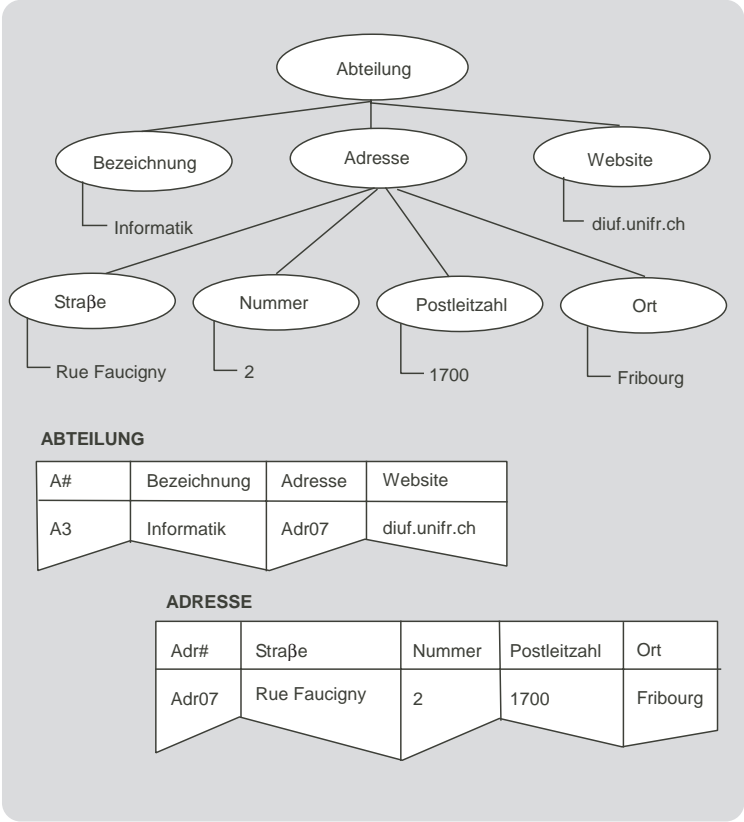
Wie erwähnt, enthalten die XML-Dokumente implizit auch *Informationen über die Struktur* des Dokumentes. Da es für viele Anwendungen wichtig ist, die Struktur der XML-Dokumente zu kennen, sind explizite Darstellungen (DTD = Document Type Definition oder XML-Schema) von W3C vorgeschlagen worden. Mit einem expliziten Schema wird aufgezeigt, welche Tags im XML-Dokument auftreten und wie sie angeordnet sind. Damit lassen sich unter anderem Fehler in XML-Dokumenten lokalisieren und beheben. Im Folgenden soll das XML-Schema illustriert werden, da es für den Einsatz von Datenbanksystemen vorteilhaft ist.

*Struktur-
beschreibung mit
XML*

In Abb. 5-2 zeigen wir die Zuordnung eines XML-Dokumentes zu einem relationalen Datenbankschema. Normalerweise können relationale Datenbankschemas durch eine dreistufige Verschachtelung von Elementen charakterisiert werden, nämlich der Bezeichnung der Datenbank, den Relationennamen sowie den Attributnamen. Der Ausschnitt eines XML-Dokumentes in Abb. 5-2 zeigt die beiden Relationennamen ABTEILUNG und ADRESSE, jeweils mit den zugehörigen Attributnamen resp. Datenwerten. Die Verwendung von Schlüsseln und Fremdschlüsseln ist mit Hilfe eines XML-Schemas möglich, worauf kurz eingegangen wird.

*Zur Zuordnung
von XML-
Dokumenten und
relationalen
Datenbanken*

Abb. 5-2
Darstellung eines
XML-
Dokumentes in
einem
relationalen
Datenbank-
schema



XML- Schemas
definieren
Datentypen

Spezifische
Eigenschaften
sind deklariert

Beispiel eines
einfachen
Datentyps im
XML- Schema

Das grundlegende Konzept von XML-Schemas ist, Datentypen zu definieren und über Deklarationen Namen zu den Datentypen zuordnen zu können. Dadurch lassen sich beliebige XML-Dokumente erstellen. Zudem besteht die Möglichkeit, Integritätsregeln für die Korrektheit von XML-Dokumenten zu beschreiben.

Es gibt eine Vielzahl von Standarddatentypen wie String, Boolean, Integer, Date, Time u.a. Daneben können aber benutzerdefinierte Datentypen eingeführt werden. Spezifische Eigenschaften der Datentypen lassen sich durch so genannte Facets deklarieren. So kann die Ordnungseigenschaft eines Datentyps angegeben werden, die Beschränktheit der Werte durch Ober- und Untergrenzen, Längenbeschränkungen oder Aufzählung erlaubter Werte:

```
<xs:simpleType name=«Ort»>
  <xs:restriction base=«xs:string»>
    <xs:length value=«20»/>
  </xs:restriction>
```

</xs:simpleType>

Zur Darstellung der Ortschaften wird ein einfacher Datentyp vorgeschlagen, der auf dem vordefinierten Datentyp String basiert. Zudem wird verlangt, dass die Ortsnamen nicht mehr als 20 Zeichen umfassen sollen.

Bei der Verwendung von XML-Schemas ist es möglich, die Schlüssel aus den Datenbanktabellen mittels «key» zu definieren. Mit der Deklaration von «keyref» lassen sich Verweise auf bereits definierte Schlüssel im XML-Schema bewerkstelligen. Mit den beiden Konstrukten können Primärschlüssel wie Fremdschlüssel definiert werden.

Es sind verschiedene XML-Editoren entwickelt worden, die eine grafische Darstellung eines XML-Dokumentes resp. eines XML-Schemas erlauben. Diese Editoren können sowohl für die Deklaration der Struktureigenschaften wie für die Erfassung von Dateninhalten verwendet werden. Durch das Ein- und Ausblenden von Teilstrukturen lassen sich umfangreiche XML-Dokumente resp. XML-Schemas übersichtlich anordnen.

Schlüssel- und Referenzbeziehungen

Darstellung von XML-Dokumenten resp. XML-Schemas

5.2.3

Die Abfragesprache XQuery

Es ist wünschenswert, dass XML-Dokumente oder XML-Datenbanken ausgewertet werden können. Im Gegensatz zu relationalen Abfragesprachen werden nicht nur Selektionsbedingungen an Werte geknüpft (Wertselektion), sondern auch an Elementstrukturen (Strukturselektion). Weitere Grundoperationen einer XML-Abfrage betreffen die Extraktion von Subelementen eines XML-Dokumentes resp. das Verändern ausgewählter Subelemente. Das Zusammensetzen von einzelnen Elementen aus unterschiedlichen Quellstrukturen ist ebenfalls gefordert. Damit lassen sich neue Elementstrukturen erzeugen. Last but not least muss eine geeignete Abfragesprache auch mit Hyperlinks resp. Referenzen umgehen können; so genannte Pfadausdrücke sind deshalb unentbehrlich.

XQuery ist von W3C vorgeschlagen worden, beeinflusst durch die Sprachen SQL, unterschiedliche XML-Sprachen (z.B. XPath als Navigationssprache von XML-Dokumenten) sowie objektorientierte Abfragesprachen. Grundelement von XQuery bilden *FOR-LET-WHERE-RETURN-Ausdrücke*: FOR und LET binden eine oder mehrere Variablen an die Ergebnisse der Auswertung von Ausdrücken. Mit der WHERE-Klausel können analog zu SQL weitere Einschränkungen an die Ergebnismenge vorgenommen werden. Das Ergebnis der Abfrage wird durch RETURN angezeigt.

Wert- und Strukturselektion ist mit XQuery möglich

FLWR als Grundstruktur von XQuery

Ein einfaches Beispiel soll das Grundkonzept von XQuery skizzieren. Es soll eine Anfrage an das XML-Dokument «Abteilung» (siehe Abb. 5-2) gestellt werden. Dabei interessieren die Straßennamen sämtlicher Abteilungen:

*Einfaches
Beispiel eines
FLWR-Ausdrucks*

```
<StraßenNamen>
  {FOR $Abteilung IN //Abteilung
   RETURN $Abteilung/Adresse/Straße }1
</StraßenNamen>
```

*Zur Bindung von
Variablen*

Die obige Abfrage bindet die Variable \$Abteilung im Laufe der Bearbeitung jeweils an die Knoten vom Typ <Abteilung>. Für jede dieser Bindungen wird durch den RETURN-Ausdruck die jeweilige Adresse evaluiert und die Straße ausgegeben. Die Anfrage in XQuery produziert das folgende Ergebnis:

*Das Resultat
einer XQuery ist
eine Sequenz*

```
<StraßenNamen>
  <Straße> Rue Faucigny </Straße>
  <Straße> ..... </Straße>
  <Straße> ..... </Straße>
</StraßenNamen>
```

*Variablen-
bindung durch
FOR und LET*

In XQuery werden Variablen mit einem durch das \$-Zeichen ergänzten Namen eindeutig gekennzeichnet, um eine Unterscheidung zu den Namen von Elementen zu machen. Im Gegensatz zu einigen Programmiersprachen können Variablen in XQuery keine Werte zugewiesen werden. Vielmehr müssen Ausdrücke ausgewertet und das Ergebnis an die Variablen gebunden werden. Diese Variablenbindung erfolgt bei XQuery mit den FOR- und LET-Ausdrücken.

*Ergebnis einer
Anfrage wird mit
RETURN
angezeigt*

Im obigen Anfragebeispiel wird auf die Spezifikation des LET-Ausdrucks verzichtet. Mit der WHERE-Klausel ließe sich zudem die Ergebnismenge weiter einschränken. Die Auswertung der RETURN-Klausel erfolgt für jeden Schleifendurchlauf mit FOR, muss aber nicht zwingend ein Resultat liefern. Die einzelnen Resultate hingegen werden aneinandergereiht und bilden das Ergebnis des FOR-LET-WHERE-RETURN-Ausdrucks.

*XQuery als
Anfragesprache
für XML-
Dokumente und
XML-
Datenbanken*

XQuery ist eine *mächtige Abfragesprache für Hyperdokumente* und wird sowohl für XML-Datenbanken wie auch für einige postrelationale Datenbanksysteme angeboten. Damit relationale Datenbanksysteme XML-Dokumente speichern können, müssen Erweiterungen in der Speicherkomponente vorgenommen werden. Beispielsweise erlauben objektrelationale Datenbanksysteme dank ihren hierarchischen Strukturierungskonzepten (siehe Abschnitt 6.4), mit XML zu arbeiten.

¹ Geschweifte Klammern stammen von der Navigationssprache XPath. Entsprechende Ausdrücke werden vom Parser direkt ausgewertet.

5.3

Abbildungsregeln für Integration und Migration

Bei der Integration heterogener Datenbestände im Web oder bei der Migration von Datenbanken müssen eindeutige Abbildungsregeln (engl. *mapping rules*) zwischen dem Datenbankschema des Quellsystems und demjenigen des Zielsystems definiert werden; unter Umständen werden mehrere Quellsysteme in ein Zielsystem überführt. Um diese Abbildungsregeln für unterschiedliche Datenmodelle nicht gesondert führen zu müssen, soll das Quellsystem hier mit einem Entitäten-Beziehungsmodell abstrahiert und das Zielsystem als relationales Datenmodell angenommen werden; analoge Abbildungsregeln ließen sich auch für eine XML-Datenbank formulieren.

Wichtig für die Datenintegration wie -migration ist der Grundsatz, dass die Entitätsmengen des Quellsystems auf eindeutige Weise in Tabellen des Zielsystems abgebildet werden und umgekehrt. Diese Eindeutigkeit ist vor allem dann zwingend, wenn bei der Integration oder Migration noch Anwendungen auf den Quellsystemen nachgeführt werden müssen. Aber auch bei der rechnergestützten Konversion bestehender Anwendungsprogramme in solche für das Zielsystem ist die Eindeutigkeit eine wichtige Voraussetzung für eine fehlerfreie und korrekte Datenmigration.

*Die
Notwendigkeit
von Abbildungs-
regeln*

*Eindeutige
Beziehung
zwischen Quell-
und Zielsystem*

5.3.1

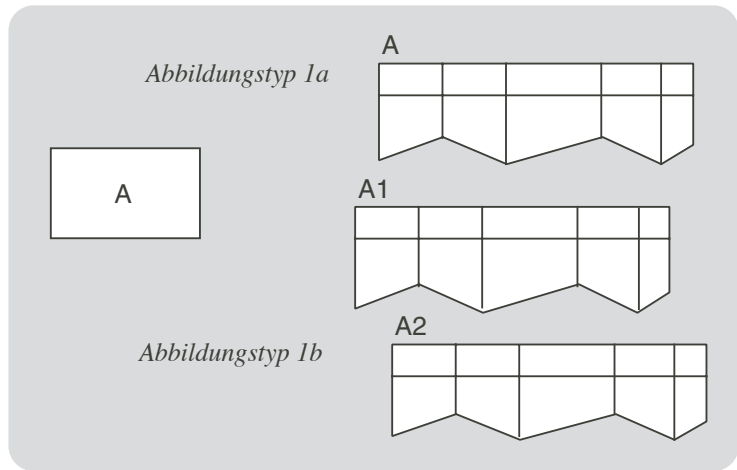
Abbildungen für einfache Entitätsmengen und Wiederholungsgruppen

Um ein Entitäten-Beziehungsmodell in ein relationales Datenbankschema überführen und eine entsprechende Datenintegration oder -migration vorbereiten zu können, lassen sich verschiedene Abbildungstypen für Entitäts- und Beziehungsmengen angeben.

Der erste Abbildungstyp aus Abb. 5-3 bezieht sich ausschließlich auf einzelne Entitätsmengen. Die *Abbildung vom Typ 1a* definiert aus einer Entitätsmenge A eine eigenständige Tabelle A, wobei als (eventuell künstlicher) Schlüssel der Tabelle eine eindeutige und minimale Merkmalskombination dient. Bei allen Abbildungstypen können *jeweils alle oder nur ein Teil der Merkmale* aus den Entitätsmengen des Quellsystems in die Tabellen des Zielsystems übernommen werden. Es ist also nicht zwingend, beim Abbildungstyp 1a alle Merkmale aus der Entitätsmenge A in entsprechende Attribute der Tabelle A zu übertragen.

*Abbildungstyp 1a
überführt
Entitätsmengen
in Tabellen*

Abb. 5-3
Abbildungs-
regeln für
einfache
Entitätsmengen



Abbildungstyp 1b
klassifiziert
Entitätsmengen

Als zulässige Erweiterung des Abbildungstyps 1a kann eine Entitätsmenge durch ein Selektionsprädikat auf zwei oder auf mehrere Tabellen aufgeteilt werden. So erlaubt die *Abbildung vom Typ 1b* das Überführen der Entitätsmenge A in die beiden Tabellen A1 und A2. Diese Abbildungsregel ist für die Praxis interessant, wenn anhand einer Selektionsbedingung auf der Entitätsmenge A eine Klassenbildung gewünscht wird, um *neue Entitätsmengen A1 und A2 bei einer Integration oder Migration einführen* zu können (Bereinigung von Altlasten).

Beispiel einer
Klassifizierung

In Abb. 5-3 werden also die Entitäten der Entitätsmenge A nach einer festen Aufteilungsregel in die beiden Tabellen A1 und A2 übertragen. Würde beispielsweise A der Entitätsmenge MITARBEITER entsprechen, so könnte Tabelle A1 die in Basel wohnhaften und Tabelle A2 die außerhalb der Stadt wohnenden Mitarbeiter als Teilklassen enthalten. Die beiden Tabellen A1 und A2 würden in diesem Fall die Mitarbeitereinträge in genau zwei Teilklassen unterteilen. Eine Rückspiegelung von den Tabellen A1 und A2 in den herkömmlichen Mitarbeiterbestand A wäre gewährleistet.

Natürlich ist bei der Abbildung vom Typ 1b nicht zwingend, dass sämtliche Entitäten einen Eintrag in den zugeordneten Teiltabellen aufweisen. Wäre die Tabelle A2 lediglich für Mitarbeiter aus Liestal vorgesehen, so würden die beiden Tabellen A1 und A2 nur die Mitarbeitenden aus den Städten Basel und Liestal umfassen, die außerhalb wohnenden und nicht von der Abbildung erfassten Mitarbeiter wären weiterhin in der herkömmlichen Datenbank enthalten. Dieser unvollständigen Klassenbildung zum Trotz wäre eine Rückspiegelung auf dieser Teilmenge erlaubt und eindeutig.

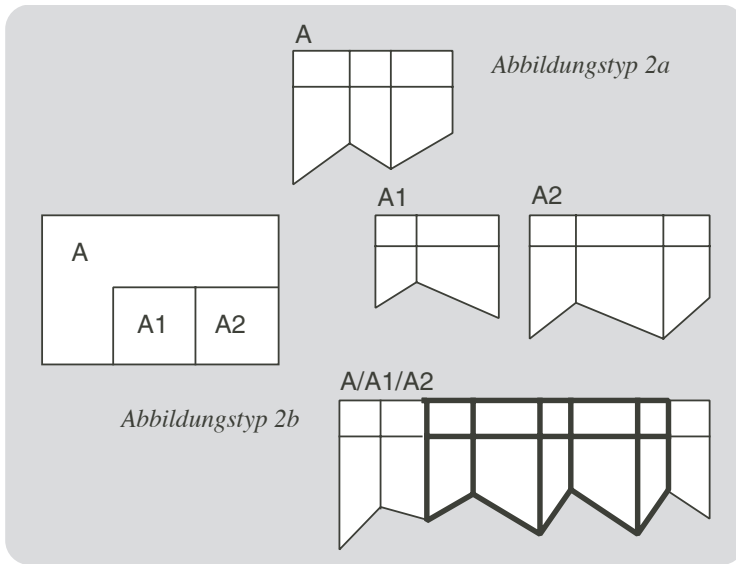


Abb. 5-4
Abbildungs-
regeln für
Wiederholungs-
gruppen

Wiederholungsgruppen, d.h. Entitätsmengen mit Subentitätsmengen, müssen im Normalfall in einzelne Tabellen aufgelöst werden und umgekehrt. So bringt der *Abbildungstyp 2a* aus Abb. 5-4 die Entitätsmenge A mit den beiden Wiederholungsgruppen A1 und A2 in die drei Tabellen A, A1 und A2. Die Auflösung der Wiederholungsgruppen A1 und A2 in die eigenständigen Tabellen A1 und A2 hängt damit zusammen, weil die erste Normalform nur atomare Merkmalswerte zulässt (vgl. Abschnitt 2.4.2).

Neuerdings werden in einem erweiterten Relationenmodell auch Wiederholungsgruppen als Merkmalswerte zugelassen, was zu geschachtelten Tabellen führt. Diese unnormalisierten, d.h. nicht in erster Normalform befindlichen Tabellen werden im Abschnitt 6.4 näher behandelt, hier aber schon zu Abbildungszwecken bei der Integration und Migration verwendet. Der *Abbildungstyp 2b* aus Abb. 5-4 lässt die Entitätsmenge A mit den Wiederholungsgruppen A1 und A2 als eigenständige Tabelle A mit den Untertabellen A1 und A2 bestehen. Ein Beispiel dazu wäre die Tabelle BUCHOBJEKT mit den Wiederholungsgruppen für Autoren und Schlagworte aus Abb. 6-6. Solche geschachtelte Tabellen werden zwar nicht von jedem Datenbanksystem unterstützt, setzen sich aber immer mehr durch.

Abbildungstyp 2a
löst
Wiederholungs-
gruppen auf

Abbildungstyp 2b
überführt
Wiederholungs-
gruppen in
geschachtelte
Tabellen

5.3.2

Abbildungen für abhängige Entitätsmengen

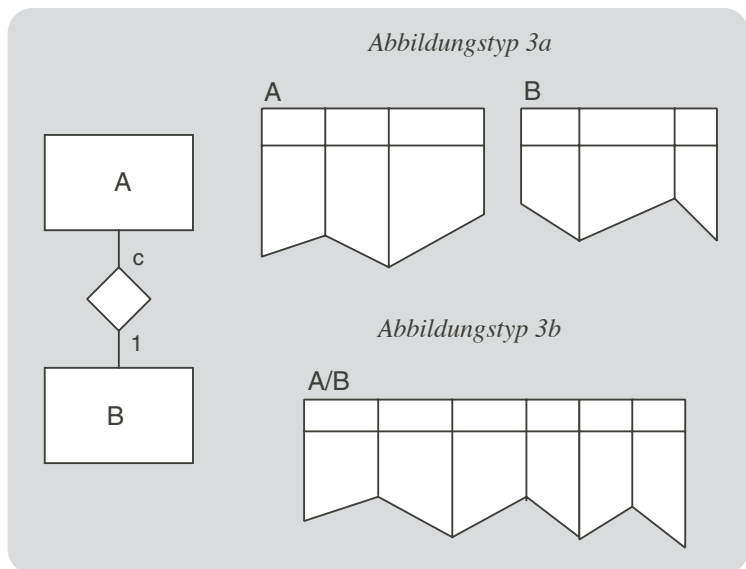
In diesem Abschnitt werden Abbildungstypen behandelt, die abhängige Entitätsmengen in abhängige Tabellen überführen und umgekehrt. Eine erste Abbildungsregel ist möglich, wenn zwischen den Entitätsmengen A und B eine (c,1)-Beziehung existiert, d.h., zu jeder Entität in der Entitätsmenge A darf «höchstens eine» Entität in der Entitätsmenge B vorliegen ($c=0$ oder $c=1$), umgekehrt gehört zu jeder Entität aus B «genau eine» (1) aus A.

*Abbildungstyp 3
überführt eine
(c,1)-Beziehung
in eine oder zwei
Tabellen*

Hier lassen sich zwei Abbildungstypen unterscheiden: Die *Abbildung vom Typ 3a* aus Abb. 5-5 ist der Normalfall, indem eine (c,1)-Beziehung zwischen den beiden Entitätsmengen A und B durch zwei Tabellen A und B dargestellt wird. Dabei wird in der Tabelle B das Merkmal A# als Fremdschlüssel geführt, um die Referenz auf die abhängige Tabelle A auszudrücken.

Als wichtige Ergänzung gilt in Abb. 5-5 die *Abbildung vom Typ 3b*, die die beiden Entitätsmengen in eine einzige Tabelle A/B zusammenfasst. Wie erwähnt, bedeutet der Assoziationstyp c, dass zu jeder Entität in A «höchstens» eine Entität in B vorliegt. Da also nicht jede Entität in A auf eine abhängige Entität in B verweist, können in der Tabelle A/B auch Nullwerte auftreten. Die Abbildung vom Typ 3b ist somit nur sinnvoll, wenn zu den meisten Entitäten in A genau eine Entität in B existiert.

*Abb. 5-5
Abbildungs-
regeln für
einfach-
abhängige
Datensatztypen*



Typ 3b hilft auch, bestehende Datenbankschemas zu bereinigen. Aus Gründen der Wirtschaftlichkeit hat man nämlich bei Datenbankerweiterungen in herkömmlichen Datenbanken zu einer bestimmten Entitätsmenge trotz Datenmodellierungsgrundsätzen oft ergänzende Entitätsmengen eingeführt, um die bestehenden Anwendungen bei diesen Schema-Erweiterungen nicht anpassen zu müssen. Mit Hilfe der Abbildung vom Typ 3b lassen sich jedoch zwei (oder auch mehrere) Entitätsmengen bei einer Integration oder Migration entsprechend der Normalformenlehre wieder zusammenfassen (Bereinigung von Altlasten).

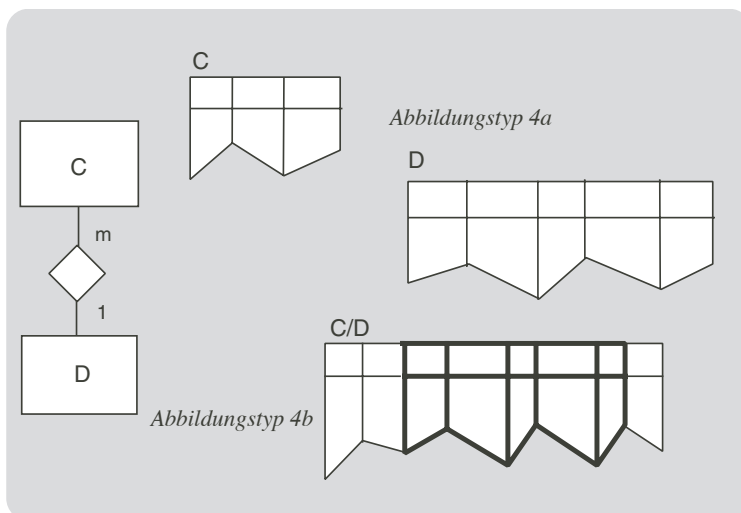
Bei der Abbildung vom Typ 4 aus Abb. 5-6 werden zwei Entitätsmengen, die in einer echten hierarchischen (m,1)-Beziehung zueinander stehen, entweder in zwei Tabellen oder in eine Tabelle übertragen. Hier bestehen zu jeder Entität aus der Entitätsmenge C «mehrere» Entitäten in der Entitätsmenge D, umgekehrt existiert zu jeder Entität in D «genau eine» Entität in C.

Die *Abbildung vom Typ 4a* überführt die Entitätsmenge C in die Tabelle C und die Entitätsmenge D in die Tabelle D. Die hierarchische Beziehung dazwischen wird durch den Fremdschlüssel C# in der Tabelle D ausgedrückt. Falls das darunterliegende relationale Datenbanksystem auch geschachtelte Tabellen unterstützt, kann die *Abbildung vom Typ 4b* verwendet werden. Solche geschachtelte oder rekursiv definierte Tabellen können als Merkmale wiederum Tabellen aufweisen (vgl. Abschnitt 6.4). Die entsprechende Tabelle C/D ist zwar nicht mehr in erster Normalform, kann jedoch jederzeit durch Projektionsoperatoren in die beim Typ 4a vorliegenden Tabellen C und D zerlegt werden (und umgekehrt).

Behebung von Altlasten

Wie werden hierarchische Beziehungen aufgelöst?

Abbildungstyp 4 überführt eine Hierarchie in eine oder zwei Tabellen



*Abb.5-6
Abbildungs-
regeln für
einfach-
komplexe Daten-
satztypen*

Abbildungstyp 5
überträgt eine
komplex-
komplexe
Beziehung in drei
Tabellen

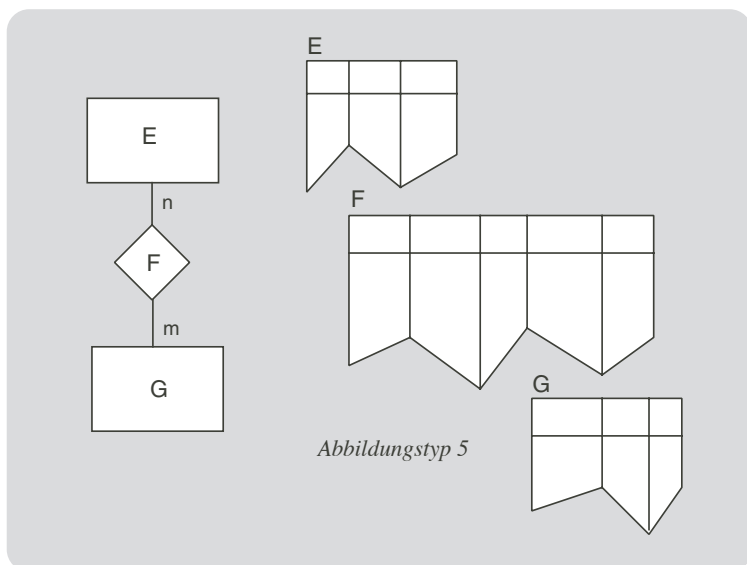
Aufgrund der bekannten Normalformen müssen mehrfach-mehrfa-
che Beziehungsmengen in eigenständigen Tabellen dargestellt wer-
den, um redundante Tabellen zu vermeiden. Deshalb überführt die
Abbildung vom Typ 5 gemäss Abb. 5-7 die Entitätsmengen E und G
in die Tabellen E und G sowie die Beziehungsmenge F in die Tabelle
F, möglicherweise ergänzt mit Regeln der referentiellen Integrität.

Neben den Abbildungen vom Typ 1 bis 5 kann man sich weitere
vorstellen, die z.B. Generalisationshierarchien oder Aggregations-
strukturen konform in Tabellen überführen. Da die relationalen
Datenbanksysteme in der Praxis solche Abstraktionskonzepte noch
selten unterstützen, gehen wir auf diese Thematik nicht weiter ein.

Kombinieren von
Abbildungstypen
ist zugelassen

Die Abbildungen vom Typ 1 bis 5 helfen, auf der logischen Ebene
netzwerkartige oder hierarchische Datenstrukturen auf Tabellen
abzubilden. Selbstverständlich lassen sich diese Abbildungstypen
auch beliebig miteinander kombinieren. Teilweise werden sie durch
kommerzielle Reengineering-Tools mit dem Ziel unterstützt, aus her-
kömmlichen Datenstrukturen relationale herzuleiten. Damit ist im
Falle eines Übergangs von nichtrelationalen zu relationalen Daten-
banken die Migration noch nicht vollständig durchgeführt, müssen
doch auch die dazugehörigen Anwendungsprogramme eventuell neu
geschrieben oder zumindest angepasst werden.

Abb. 5-7
Abbildungsregel
für komplex-
komplexe
Datensatztypen



5.3.3

Indirekte Abbildungen für die Datenintegration und -migration

Auf den ersten Blick erscheinen die Abbildungen vom Typ 1 bis 5 zu restriktiv. Weshalb kann bei einer Integration oder Migration nicht einfach eine bestimmte Entitätsmenge auf beliebig viele Tabellen aufgeteilt oder mehrere Entitätsmengen einer herkömmlichen Datenbank in genau eine Tabelle übertragen werden und umgekehrt?

Der Grund liegt in den Datenmodellierungsgrundsätzen, die in Abschnitt 2.3 ausführlich dargelegt sind. Die dortigen Regeln 1 bis 7 zur Datenmodellierung schreiben vor, wie ein Entitäten-Beziehungsmodell sauber in ein relationales Datenbankschema abgebildet wird. Ähnliche Regeln gelten bei der Integration oder Migration von Datenbeständen. Deshalb ist es beim Abbilden von Datenbanken nicht immer sinnvoll, einzelne Entitätsmengen beliebig auf mehrere Tabellen zu verteilen oder mehrere beliebige Entitätsmengen in genau eine Tabelle abzubilden. Erst beim Auswerten von relationalen Datenbanken steht es frei, auf beliebige Art Informationen zu selektieren oder zu kombinieren.

Falls die historisch gewachsenen Anwendungssysteme mit ihren heterogenen Datenbanken sauber definiert und implementiert worden sind, lassen sie sich mit den Abbildungen vom Typ 1 bis 5 ohne Schwierigkeiten in relationale Datenbanken überführen. Nun sind aber in der Praxis einzelne Datenbanken oft weniger sorgfältig und isoliert aus Sicht eines einzelnen Anwendungsgebietes heraus aufgebaut worden. So fehlt heute noch vielerorts eine unternehmensweite Datenarchitektur, oder eine solche ist erst im Entstehen begriffen. Wie kann man mit Unzulänglichkeiten umgehen, falls die Abbildungen vom Typ 1 bis 5 nicht genügen?

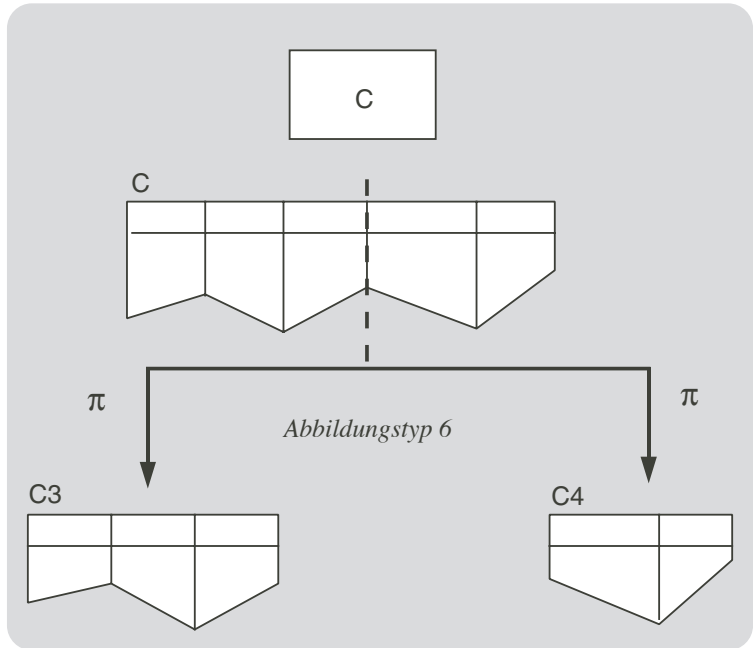
Die *Abbildung vom Typ 6* ermöglicht es auf indirekte Art, die Einträge einer bestimmten Entitätsmenge auf verschiedene Tabellen aufzuteilen. Bei diesem in Abb. 5-8 dargestellten Abbildungstyp werden einzelne Merkmale der Entitätsmenge C in die Tabelle C3 und die restlichen in die Tabelle C4 indirekt abgebildet. Bei einer notwendigen Rückspiegelung könnte ein Problem auftauchen: Wenn zu jedem Eintrag in C3 nicht mehr genau ein Eintrag in C4 vorliegt, da beispielsweise in der Zwischenzeit eine Löschoperation gewisse Tupel in C4 entfernt hat, ist eine eindeutige Rückführung der beiden Tabellen C3 und C4 in die Entitätsmenge C nicht mehr gewährleistet.

Integration muss Grundsätze der Datenmodellierung respektieren

Wege aus dem Datenchaos

Abbildungstyp 6 teilt Entitätsmenge auf

Abb. 5-8
Indirekte
Abbildungsregel
mit Projektion



Veränderungs-
operation auf
Sichten
unterliegt
Restriktionen

Die Abbildungsregel vom Typ 6 ist *indirekt*, da sie auf indirektem Weg, über eine Zwischenstufe, konfliktfrei realisiert werden kann. Zuerst werden die Entitäten aus C mit Hilfe der Spiegelungsabbildung vom Typ 1a (gegebenenfalls auch vom Typ 1b) in die zugehörige Tabelle C übertragen. Nun steht es dem Administrator einer relationalen Datenbank jederzeit frei, die Tabelle C zum Beispiel mit Projektionsoperatoren auf die Teiltabellen C3 und C4 aufzugliedern, damit dem Benutzer zusätzlich die beiden Sichten C3 und C4 angeboten werden können. Dadurch bleibt die relationale Datenbank konsistent, da Veränderungsoperationen durch die Sichten C3 und C4 gewissen Restriktionen unterworfen sind. Beispielsweise lässt ein relationales Datenbanksystem eine Löschoption auf der Sicht C4 nicht zu, falls es das Entfernen der zugehörigen Tupel in der Basistabelle C nicht eindeutig nachvollziehen kann.

Abbildungstyp 7
führt
unabhängige
Entitätsmengen
zusammen

Die *Abbildung vom Typ 7* (vgl. Abb. 5-9) ist ebenfalls *indirekt*, da die beiden Entitätsmengen B und T nicht direkt in das Kombinat B/T übertragen werden. Vielmehr werden die beiden Entitätsmengen B und T zunächst mit der Abbildung vom Typ 1a gesondert in die eigenständigen Tabellen B und T übertragen, bevor sie mit einem gemeinsamen Verbundoperator \bowtie kombiniert werden können. Auch hier gilt, dass ein Verbund zweier Tabellen im Normalfall nur sinnvoll ist, wenn gemeinsame Merkmale sie verbinden. Die gewünschte

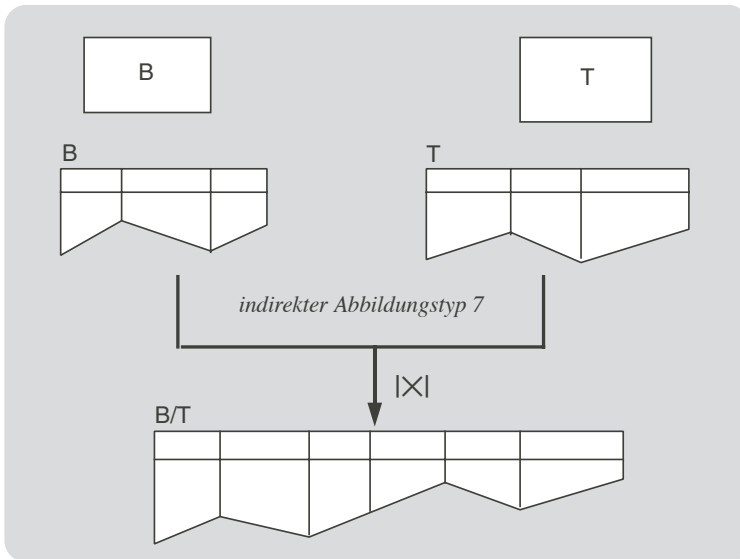


Abb. 5-9
Indirekte
Abbildungsregel
mit Verbund

Tabelle B/T kann also als Sicht, als Verbund der beiden Tabellen B und T, aufgefasst werden. Die Rückführung der Abbildung vom Typ 7 ermöglicht jederzeit eine Rückführung der Tabellen B und T in die Entitätsmengen B und T.

Bei der Diskussion der Normalformen wurde in Abschnitt 2.4 darauf hingewiesen, dass eine Tabelle mit Projektionsoperatoren nicht in jedem Fall in Teiltabellen zerlegt und durch Verbundoperatoren wieder zurückgewonnen werden kann (vgl. Verbundabhängigkeit Abschnitt 2.4.5). Die fünfte Normalform ist somit ein Grund, für das beliebige Aufteilen und Vereinen von Entitätsmengen indirekte Abbildungsregeln vorauszusetzen, die die direkten ergänzen.

*Vorsicht bei
Verbund-
abhängigkeit*

5.4 Migrationsvarianten für heterogene Datenbanken

Obwohl heute viele Unternehmen herkömmliche und relationale Datenbanken gleichzeitig betreiben müssen, bildet ein solches Nebeneinander eine kaum zu bewältigende Schwierigkeit, da *in heterogenen Datenbanken weder die Aktualität noch die Konsistenz der Daten gewährleistet* ist. Erschwerend kommt hinzu, dass unzählige Anwendungsprogramme mit herkömmlicher Datenbanksoftware und unter großen Investitionen entwickelt worden sind. Wie können nun

*Wie lassen sich
Investitionen
schützen?*

Unternehmen mit dieser oft zitierten Erblast längerfristig zurecht-
kommen?

5.4.1

Charakterisierung unterschiedlicher Migrationsvarianten

Im Hochschulbereich und in vielen Forschungszentren ist der kompatible Übergang von einer Datenbanksystemgeneration zur nächsten selten ein Thema, doch haben sich verschiedene Softwarehäuser und einige Firmen mit umfangreichen Informationssystemen als Pioniere hervorgetan, um ein «Generationenlifting» oder eine Koexistenz unterschiedlicher Datenbanksysteme softwaremäßig zu unterstützen.

Grob lassen sich diese Maßnahmen gemäß Abb. 5-10 wie folgt charakterisieren:

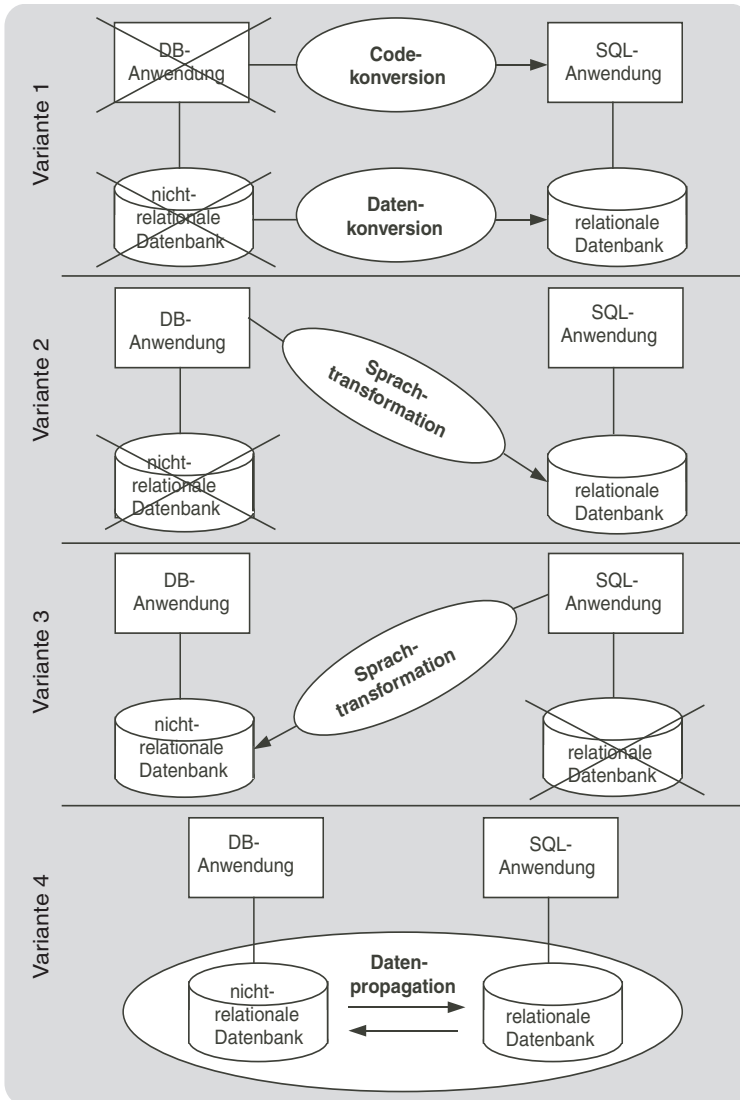
*Migrations-
variante: Daten
und Programme
konvertieren*

- *Daten- und Codekonversion von Anwendungsprogrammen (vgl. Abb. 5-10, Variante 1):* Herkömmliche Datenbanken werden mit allgemeinen Abbildungsregeln (vgl. Abschnitte 5.3.1 und 5.3.2) auf relationale abgebildet und mit Hilfe von Extrakt- und Lade-Programmen in die entsprechenden Datenbanken übertragen. In einem separaten Arbeitsschritt übersetzt man die prozeduralen Datenbankaufrufe eines Anwendungsprogramms mit Hilfe von Konversionsroutinen weitgehend automatisch in relationale Datenbankaufrufe für das Zielsystem. Bei Spezialfällen oder bei Performance-Problemen müssen diese Konvertierungen von Hand nachgebildet oder abgeändert werden. In gewissen Fällen können Datenmigration und Programmkonvertierungen zum *Wechsel der gesamten Anwendungsseite* führen, so dass im besten Fall auf den Weiterbetrieb des herkömmlichen Datenbanksystems verzichtet werden kann.

*Migrations-
variante: Daten-
bankaufrufe
bestehender
Programme
transformieren*

- *Übertragen der prozeduralen Zielsprache auf die gewünschte deskriptive Schnittstelle (Variante 2):* Zu diesem Zweck wird das relationale Datenbanksystem um eine allgemeine Software-schicht (engl. *wrapper*) ergänzt, die jeden prozeduralen Aufruf in die relationale Abfrage- und Manipulationssprache übersetzt. Die existierenden Anwendungsprogramme werden nicht tangiert, da nur die Datenbankaufrufe softwaremäßig auf das relationale Zielsystem umgeformt werden. Konkrete Untersuchungen an herkömmlichen und relationalen Datenbanksystemen haben allerdings – von Performance-Problemen ganz zu schweigen – grundsätzliche Schwierigkeiten aufgedeckt, für jeden prozeduralen Aufruf mit all seinen möglichen Verarbeitungsregeln einen äquivalenten mengenorientierten Aufruf zu finden.

Abb. 5-10
Übersicht über
grundlegende
Migrations-
varianten



- **Rückführen der relationalen Zielsprache auf die gewünschte prozedurale Schnittstelle (Variante 3):** Ein herkömmliches Datenbanksystem wird durch eine generelle Softwareschicht ergänzt, die erlaubt, deskriptive Aufrufe zu formulieren und an das herkömmliche Datenbanksystem zu richten. Dabei werden die mengenorientierten Ausdrücke auf die vorhandene prozedurale Schnittstelle des herkömmlichen Datenbanksystems zurechtgestutzt, was meist mit hohen Performance-Einbußen verbunden

*Migrations-
variante: SQL-
Anwendungen
auf
herkömmliche
Datenbanken
ausrichten*

ist. Nachteilig wirkt sich zusätzlich aus, dass die Datenhaltung nach wie vor mit dem herkömmlichen Datenbanksystem betrieben werden muss. Bestehende Anwendungen müssen dafür nicht umgeschrieben werden. Neue Anwendungen lassen sich mit einer relationalen Zielsprache und mit entsprechenden Entwicklungswerkzeugen realisieren.

Migrations-
variante: Zeitlich
befristete
Koexistenz

- *Konsistentes Nachführen paralleler Datenbanken durch Spiegelungsabbildungen (Variante 4):* Lediglich die Änderungen in herkömmlichen Datenbanken werden im relationalen Datenbanksystem gespiegelt. Umgekehrt können Änderungen in relationalen Datenbanken im Bedarfsfall in den Datenbanken des herkömmlichen Systems nachgeführt werden. In beiden Fällen müssen so genannte *Spiegelungsabbildungen* definiert werden, die die Datenbankstrukturen aufeinander beziehen. Eine eventuell befristete *Koexistenz von herkömmlichen und relationalen Datenbanken* (vgl. Abschnitt 5.4.2) macht es möglich, bei Neuentwicklungen aktuelle oder periodisch nachgeführte Datenbestände mit einzubeziehen oder bestehende Anwendungen ohne Termindruck umzuschreiben.

Im Folgenden wird die Koexistenzvariante etwas ausführlicher behandelt, da sie in der Praxis oft verwendet wird und z.B. beim Aufbau eines Data Warehouse (vgl. Abschnitt 6.5) an Bedeutung gewonnen hat.

5.4.2 Systemkonforme Spiegelung von Datenbanken

Vorteile
physischer
Datenredundanz

Bei der Datenmodellierung gilt zwar die Maxime, dass in Anbetracht latent vorhandener Anomalien oder Konsistenzprobleme redundante Informationen vermieden werden sollten. Ein Verzicht auf Datenredundanz lässt sich aber in der Praxis nicht immer konsequent aufrechterhalten. Beim physischen Entwurf relationaler Datenbanken beispielsweise können sogar unnormalisierte Tabellen von Vorteil sein. Dies zeigen Performance-Überlegungen und die Erkenntnis, dass Tabellen redundanten Inhalts einfachere und effizientere Selektionsoperationen anstelle teurer Verbundoperationen zulassen. Deshalb müssen aus Optimierungsgründen beim physischen Datenbankentwurf oft Kompromisse eingegangen werden.

Koexistenz
garantiert
system-
kontrollierte
Datenredundanz

Eine andere Form von Datenredundanz ergibt sich aus dem Wunsch nach Koexistenz unterschiedlicher Datenbanksysteme oder nach längerfristiger Migration in ein relationales Zielsystem. Eine solche *Datenredundanz sollte aus Gründen der Datenkonsistenz systemkontrolliert bleiben*, wie es die Spiegelung zwischen herkömmlichen und relationalen Datenbanken nahelegt.

Die Spiegelung einer herkömmlichen Datenbank auf eine relationale Datenbank geschieht gemäß Abb. 5-11 in drei Phasen, einer Definitions-, einer Initialisierungs- und einer Überführungsphase:

- **Definitionsphase:** Zuerst werden die *Spiegelungsabbildungen* definiert und im Datenkatalog abgelegt. Damit wird festgelegt, welche Datensatztypen in welche Tabellen abgebildet werden sollen. Für direkt abhängige Datensätze einer herkömmlichen Datenbank werden die entsprechenden Tabellen mit einer referenziellen Integritätsregel ausgerüstet.

Die drei Phasen der Koexistenzvariante

Spiegelungsabbildungen im Datenkatalog festhalten

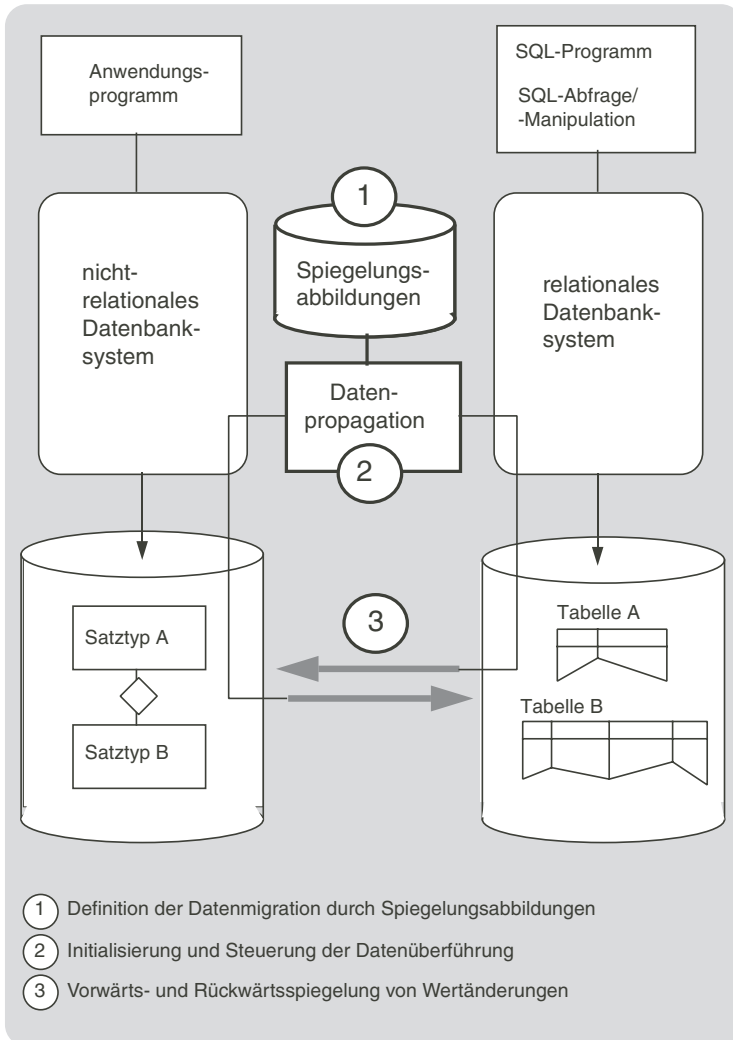


Abb. 5-11
System-
kontrollierte
Datenredundanz
bei der Daten-
propagation

Identische Datenbestände als Ausgangsbasis bereitstellen

- *Initialisierungsphase:* Ein Anwendungsprogramm, das eine herkömmliche Datenbank aufruft, muss nicht angepasst oder erweitert werden. Lediglich vor einem erstmaligen Gebrauch der Datenspiegelung müssen bereits existierende Dateninhalte vorab aus der herkömmlichen Datenbank in die entsprechenden Tabellen geladen werden. Damit soll ein *identischer Datenbestand* zwischen den zu spiegelnden herkömmlichen und relationalen Datenbanken als Ausgangsbasis geschaffen werden.

Wertveränderungen synchron oder asynchron nachführen

- *Überführungsphase:* Aufgrund der Spiegelungsabbildungen werden zur Laufzeit sämtliche Änderungsoperationen auf der herkömmlichen Datenbank automatisch durch relationale Aufrufe ergänzt (synchrone Spiegelung), so dass *lediglich Wertveränderungen auf der relationalen Seite nachgeführt* werden müssen. Es ist auch möglich, die Änderungen der herkömmlichen Datenbank zu sammeln und zu bestimmten Zeitpunkten auf der relationalen Seite nachzutragen (asynchrone Spiegelung). Die Spiegelungsabbildungen halten zudem fest, welche Datensatztypen auf die relationale Datenbank gespiegelt und welche Felder innerhalb eines Datensatzes in die entsprechenden Tabellenmerkmale transferiert werden.

Rückspiegelung mindert Risiken und schützt Investitionen

Die Spiegelung herkömmlicher Datenbanken auf relationale ermöglicht die Verwirklichung neuer Anwendungen mit Hilfe relationaler Datenbanksprachen oder darauf aufbauender Entwicklungswerkzeuge. Falls bestimmte Teile eines Anwendungssystems mit relationaler Technologie neu geschrieben und die Daten nach wie vor auf den herkömmlichen Datenbanken parallel für bestimmte Funktionen bereitgestellt werden müssen, kommt die Rückspiegelungsvariante zum Zuge. Die Rückspiegelung von Änderungen einer relationalen Datenbank auf die entsprechende herkömmliche Datenbank des Quellsystems ist natürlich nur dann notwendig, wenn bereits bestehende Anwendungsprogramme von diesen Datenbeständen abhängig sind. Die *Rückspiegelung bildet somit einen Investitionsschutz* für die zahlreichen Programme, die nicht schlagartig umgeschrieben werden können. Auch aus Risikoüberlegungen heraus ist es vorteilhaft, dass die Koexistenz von herkömmlichen und relationalen Datenbanken ein zeitlich abgestuftes Hinüberwechseln in die relationale Datenbanktechnologie erlaubt.

5.5

Grundsätze der Integrations- und Migrationsplanung

Die Nutzung von Internettechnologien, der mögliche Wechsel eines Datenbanksystems sowie die Auswahl einer geeigneten Integrations- und Migrationsvariante bedarf einer exakten Planung, da ein solcher Schritt mit erheblichen Investitionen verbunden ist. Gleichzeitig sollte beim Beheben von Altlasten die Chance genutzt werden, mit den historisch gewachsenen *Unzulänglichkeiten in den Datenbeständen aufzuräumen* und die *Datenarchitektur auf die strategischen Ziele des Unternehmens auszurichten*. Als Grundlage für einen abgestimmten Integrations- und Migrationsplan dient deshalb eine unternehmensweite Datenarchitektur. Diese abstrahiert die strategischen Erfolgspositionen der Geschäftsbereiche und konzentriert sich auf die längerfristig gültigen Informationsbedürfnisse (vgl. Abschnitt 2.6).

Eine Integrations- und Migrationsplanung kann nicht alleinige Aufgabe eines Spezialistenteams sein, vielmehr muss sie im Unternehmen unter den einzelnen Geschäftsbereichen abgesprochen werden, da die Restaurierung bestehender Informationssysteme und ein Wechsel der darunterliegenden Datenbanktechnologie mit großen Investitionen verbunden ist. So ist es von Vorteil, die wichtigsten Grundsätze bei einem Integrationsprojekt resp. bei einer Datenmigration in Verhaltensregeln festzuhalten, denen in der alltäglichen Entwicklungsarbeit nachgelebt werden kann.

Die folgenden Grundsätze sind von den Bedürfnissen eines international tätigen Unternehmens aus dem Dienstleistungssektor geprägt. Sie können nicht ohne weiteres auf andere Firmen übertragen werden. Sie sollen aber beispielhaft illustrieren, wie eine Integration und Migration von heterogenen Datenbeständen im Rahmen strategischer Erfolgspositionen gesehen und umgesetzt werden kann:

Aufbau einer unternehmensweiten Datenarchitektur

Die Datenintegration und -migration hat im Rahmen der unternehmensweiten Datenarchitektur zu erfolgen.

Dieser Grundsatz scheint offensichtlich, er ist aber in der Praxis nach wie vor umstritten. Erstaunlicherweise scheut man sich oft vor dem Aufwand, der der Strukturierung der Daten aus globaler Sicht des Unternehmens dient. Mit einer solchen Haltung wird die Datenintegration und eventuell notwendige Datenmigration auf die technischen Komponenten reduziert. Um nun doch einen Technologiewechsel im Datenbankbereich mit der Erarbeitung einer zukunftsgerichteten Datenarchitektur kombinieren zu können, wird dieser Grundsatz an den Anfang gestellt.

Integration und Migration müssen langfristig geplant werden

Behebung von Altlasten zählt zur Unternehmensverantwortung

Planungsgrundsätze auf Erfolgspositionen ausrichten

Grundsatz 1

Technologiewechsel alleine genügt nicht!

- Grundsatz 2 Nutzung relationaler oder postrelationaler Datenbanktechnologie
Neue Anwendungen und neue Geschäftsfunktionen sind mit relationaler und objektrelationaler Datenbanktechnologie zu realisieren.*
- System-
komplexität
reduzieren und
Schnittstellen
abbauen* Die Komplexität bei bestehenden Informationssystemen und bei der Nutzung des Webs verlangen, heterogene Architekturansätze zu vereinheitlichen und die Schnittstellen zu reduzieren. Für die Anwendungsentwicklung macht es deshalb Sinn, verbindliche Grundsätze bezüglich der Datenbanktechnologie und der Benutzung des Webs aufzustellen und durchzusetzen. Die unter stetigem Zeitdruck leidenden Entwicklungsabteilungen stehen sonst zu oft vor Entscheidungsproblemen und folgen der Macht der Gewohnheit mit herkömmlichen Methoden und Techniken.
- Grundsatz 3 Verbot für unkontrollierte Extrakte
Datenbestände dürfen nur in bewilligten Ausnahmen auf Extraktbasis den Fachabteilungen zur Verfügung gestellt werden.*
- Extrakte sind nur
im Rahmen einer
Data Warehouse
Strategie
zugelassen* Das regelmäßige Extrahieren von Datenbeständen aus bestehenden Datenbanken ist für ein Dienstleistungsunternehmen äußerst problematisch und sollte nur im Rahmen einer abgestimmten Data Warehouse Strategie (siehe auch Abschnitt 6.5) erfolgen. Ein unkontrolliertes Extrahieren produktiver Datenbestände widerspricht der Zielsetzung, einen 24-Stunden-Betrieb aufrechterhalten zu können. Hinzu kommt, dass periodisch erstellte Datenbestände immense Abstimmungsprobleme in den Fachbereichen aufwerfen. *Die Datenintegrität ist nur bei aktuellen oder zeitpunktbezogenen Informationen gewährleistet.* Unüberwindbare Schwierigkeiten ergeben sich deshalb, wenn bei Auswertungen durch die Fachabteilungen aktuelle Datenbestände und periodisch extrahierte Datensammlungen kombiniert werden.
- Grundsatz 4 Pflicht zu ISO-Normen
Internationale Normen der International Organization for Standardization sind zu berücksichtigen.*
- Chance nutzen
und ISO-Normen
einführen* Bei mehreren tausend Anwendungsprogrammen hat man normalerweise keine Chance, international ausgerichtete Codewerte für Länder, Branchen, Währungen oder Adressen einzuführen, da der Umstellungsaufwand zu groß ist. Ein Wechsel in der Datenbanktechnik resp. die Nutzung von Webtechnologien bietet deshalb die einmalige Gelegenheit, für die neuen Datenstrukturen die ISO-Normen von Anfang an zu verwenden.
- Grundsatz 5 Ausrichtung auf Branchendatenmodelle und Online-Datenbanken
Branchendatenmodelle und Online-Datenbanken, die sich auf dem Markt durchsetzen, sind bei der Datenintegration und -migration mit einzubeziehen.*

In den letzten Jahren sind für unterschiedliche Anwendungsgebiete kommerziell erhältliche Datenmodelle und Frameworks aufgetaucht. Auf dem Markt zukaufbare Informationen aus Online-Datenbanken wie Wirtschaftsdaten, Marktanalysen und Geschäftsbilanzen ergänzen zudem das Angebot. Wo immer möglich und sinnvoll, sollten diese externen Datenlieferanten bei einer Umstellung der Informationssysteme berücksichtigt werden.

Externe Datenlieferanten und Informationsbroker berücksichtigen

Ein Integrations- und Migrationsplan wird nun anhand der obigen Grundsätze entworfen. Er hat die bestehenden und historisch gewachsenen Datenbestände zu berücksichtigen und umfasst folgende Schritte:

- *Verfeinern der unternehmensweiten Datenarchitektur:* Diejenigen Entitäts- und Beziehungsmengen, die bei der Integration oder Migration berücksichtigt werden, sollten anhand der unternehmensweiten Datenarchitektur verfeinert und auf ein relationales Datenbankschema abgebildet werden.
- *Festlegen von Abbildungen zwischen Quell- und Zielsystem:* Das aus der unternehmensweiten Datenarchitektur hergeleitete relationale Datenbankschema für das Zielsystem wird mit den bereits existierenden Datenbanken der Quellsysteme verglichen. Bei einer Abweichung muss geprüft werden, ob und gegebenenfalls welche der zur Verfügung stehenden Abbildungstypen ins gewünschte relationale Datenbankschema des Zielsystems führen.
- *Wahl der Integrations- und Migrationsvarianten:* Jetzt kann entschieden werden, welcher Ansatz der Integration oder Migration im jeweiligen Fall geeignet ist. Eventuell ist dies eine Kombination verschiedener Integrations- und Migrationswege. Bei der Koexistenz muss zusätzlich berücksichtigt werden, ob die Spiegelung synchron oder asynchron erfolgen soll. Bei einer synchronen Datenspiegelung liegen sowohl auf der nichtrelationalen wie auf der relationalen Seite aktuelle und konsistent nachgeführte Datenbestände vor. Die asynchrone Spiegelung kommt dann zum Zuge, wenn periodisch übermittelte Änderungen für die relationalen Datenbanken des Zielsystems genügen.
- *Bereinigen von Abbildungskonflikten:* Liegen für einzelne Datensatztypen keine geeigneten Abbildungen vor, so muss entweder eine Bereinigung auf der entsprechenden herkömmlichen Datenbank vorgenommen werden (was eventuell zu Anpassungen bei einigen Anwendungen führen kann) oder es müssen individuelle Abbildungsregeln für diese Datensatztypen vorgesehen werden. Für die auf den Datenbanken des Quellsystems gültigen Verar-

Zielsystem mit Datenarchitektur abstimmen

Abbildungstypen auswählen

Eventuell Integrations- und Migrationsvarianten kombinieren

Bei Bedarf Quellsysteme korrigieren

beitungsregeln müssen die entsprechenden referenziellen Integritätsbedingungen auch fürs Zielsystem spezifiziert werden.

Benutzerschnittstelle auf künftige Datenarchitektur ausrichten

- *Definieren externer Schemas:* Können aufgrund der Abbildungstypen die Tabellen der relationalen Datenbank des Zielsystems nicht so ausgelegt werden, wie es die unternehmensweite Datenarchitektur empfiehlt, so sollten mit Hilfe von Sichten (engl. *views*) entsprechende externe Schemas definiert werden. Dieses starke Konzept relationaler Datenbanken ermöglicht, an der Benutzerschnittstelle dem Anwender schon frühzeitig die auf die unternehmensweite Datenarchitektur ausgerichtete Datensicht aufzuzeigen.

Durchhaltewillen für Umstellungsarbeiten aufbringen

Diese groben Planungsschritte illustrieren, wie vielfältig ein Integrationsprojekt resp. ein Wechsel in die relationale oder postrelationale Datenbanktechnik ausgestaltet werden kann. So können mehrere Integrations- und Migrationsvarianten miteinander kombiniert werden, und eventuell muss für einzelne leistungskritische Anwendungen auf einen Technologiewechsel ganz verzichtet werden. Auch der Zeitaspekt spielt bei einer Umstellung eine wesentliche Rolle, besitzen doch die Anwendungssysteme normalerweise eine mehrjährige Lebensdauer, bevor sie in größerem Umfang überarbeitet werden.

5.6 Bibliografische Angaben

Seit kurzem gibt es eine ansehnliche Fachliteratur zu XML und Web-Datenbanken

Seit kurzem sind einige Fachbücher über Web-Datenbanken resp. über XML und Datenbanken entstanden. Loeser (2001) zeigt in seiner Arbeit den Einsatz objektrelationaler Datenbanken für webbasierte Informationssysteme auf. Rahm und Vossen (2003) haben verschiedene Spezialisten auf dem Gebiet Webservices und Datenbanken ermuntert, ihre Kenntnisse in einem Sammelband zu veröffentlichen. Meier und Wüst (2003) haben ein Werk über objektorientierte und objektrelationale Datenbanken verfasst, wobei unterschiedliche postrelationale Datenbanksysteme vorgestellt werden. Kazakos et al. (2002), Klettke und Meyer (2003) sowie Schöning (2003) widmen ihre Werke dem Thema XML und Datenbanken. Darin werden unter anderem die Auszeichnungssprache XML, das XML-Schema sowie die Anfragesprache XQuery behandelt und an Beispielen illustriert.

Forschungsliteratur zum Datenbankwechsel

Zur Abbildung von Datenbankschemas gibt es einige Forschungsarbeiten, die beim Aufkommen relationaler Datenbanksysteme Mitte der 70er Jahre publiziert wurden. Chen (1976) zeigt auf der logischen Ebene, wie das Entitäten-Beziehungsmodell in ein relationales, hierarchisches oder netzwerkartiges Datenbankschema überführt werden

kann. Date (1986) geht in seinem Buch über ausgewählte Themen relationaler Datenbanken auf die Abbildungsproblematik hierarchischer und relationaler Datenbanken ein. Gillenson (1990) stellt Konversionsregeln für physische Datenbanken unterschiedlicher Systeme auf.

Brodie und Stonebraker (1995) beschreiben allgemeine Migrationsstrategien für überalterte Informationssysteme. Meier und Dippold (1992) behandeln die Migration und die Koexistenz heterogener Datenbanken. Den Schutz der Investitionen bei einem Datenbankwechsel illustrieren Meier et al. (1993) und Meier (1997). Das Fachbuch von Dippold et al. (2005) geht neben Aspekten des Datenmanagements grundlegend auf die Datenbankmigration ein. Hüseemann (2002) illustriert in seiner Arbeit die Migration von relationalen zu objektorientierten Datenbanken, eine Methodik für die Datenbankmigration sowie softwaretechnische Unterstützungsmaßnahmen.

*Spezifische
Arbeiten zur
Migration von
Datenbanken*

6 Postrelationale Datenbanksysteme

6.1

Weiterentwicklung – weshalb und wohin?

Die relationale Datenbanktechnologie hat sich in den letzten Jahren breit im Markt durchgesetzt. Ein Ende dieser erfolgreichen Entwicklung ist noch nicht abzusehen. Trotzdem stellt sich die Frage, wohin die Reise führt. Da ist die Rede von verteilten Datenbanksystemen, von temporalen, von deduktiven, von semantischen, von objektorientierten Systemen, von unscharfen, von versionenbehafteten etc. Was verbirgt sich hinter all diesen schillernden Adjektiven? Das vorliegende Kapitel erläutert einige dieser Begriffe und zeigt künftige Methoden und Entwicklungstendenzen auf, wobei die Auswahl subjektiv bleiben muss.

Beim klassischen Relationenmodell und bei den entsprechenden relationalen Datenbanksystemen treten zugegebenermaßen einige Mängel in Erscheinung, die vor allem von *erweiterten Anforderungen in neuen Anwendungsgebieten* herrühren. Die Hardwarehersteller und Softwarelieferanten haben eine derart rasante Entwicklung der relationalen Technologie offenbar nicht voraussehen können. Nun suchen sie z.B. für Webapplikationen nach Mitteln und Wegen, neben formatierten Datenbeständen auch Textdokumente, Grafiken und Bilder in ein und derselben Datenbank abspeichern und verwalten zu können. Daneben haben auch technische Anwendungen Bedarf an erweiterter Datenbanktechnologie, so der rechnergestützte Entwurf integrierter Schaltungen, die Prozess- und Fertigungssteuerung von Maschinen- und Bauteilen, die Speicherung und Verarbeitung von Satellitenbildern oder die Herstellung geobezogener Pläne und Karten.

Heute gibt es eine Auswahl kommerzieller Produkte mit erweiterter Datenbankfunktionalität. Der Praktiker tut sich oft schwer, sich im Dschungel der Ankündigungen zurechtzufinden. Auch ist im Allgemeinen der Umstellungsaufwand einerseits und der wirtschaftliche

*Relationale
Datenbank-
technologie ist
weiterhin
marktdominant*

*Schwachstellen
relationaler
Datenbank-
systeme*

*Einheitliche
Anwendungs-
architektur*

Nutzen andererseits zu wenig ersichtlich. Einige Unternehmen brauchen deshalb noch manche Kopfarbeit, um ihre Anwendungsarchitektur zukunftsgerichtet zu planen und eine sinnvolle Produktwahl zu treffen. Kurz gesagt, es fehlt an klaren Architekturkonzepten und Migrationsstrategien für den optimalen Einsatz postrelationaler Datenbanktechnologie.

Ansätze
postrelationaler
Datenbank-
systeme

Im Folgenden geben wir eine Auswahl von Problemstellungen und Lösungsansätzen, um die möglichen Entwicklungen künftiger Datenbanksysteme etwas auszuloten. Einige der heute noch nicht abgedeckten Anforderungen können mit punktuellen Erweiterungen relationaler Datenbanksysteme befriedigt werden, andere müssen mit grundlegend neuen Konzepten und Methoden angegangen werden. Beide Entwicklungstendenzen sind unter dem Begriff der *postrelationalen Datenbanksysteme* zusammengefasst.

6.2

Verteilte Datenbanken

Verteilte
Datenbanken
behalten eine
logisch
einheitliche Sicht

Dezentrale oder *verteilte Datenbanken* (engl. *distributed databases*) finden in öffentlich- und privatrechtlichen Unternehmen Anwendung, in denen Daten an verschiedenen Orten gesammelt, gepflegt und verarbeitet werden sollen. Eine *Datenbank ist dezentral oder verteilt*, wenn sie zwar durch ein *einziges logisches Datenbankschema* beschrieben, aber durch *mehrere physische Tabellenfragmente auf örtlich verteilten Rechnern* gehalten wird. Der Anwender einer verteilten Datenbank hat sich lediglich mit der logischen Sicht auf die Daten zu befassen, um die physischen Fragmente braucht er sich nicht zu kümmern. Das Datenbanksystem selbst übernimmt es, Datenbankoperationen lokal oder bei Bedarf verteilt auf verschiedenen Rechnern durchzuführen.

Der Datenbank-
administrator
bestimmt
die physischen
Fragmente

Ein einfaches Beispiel einer verteilten Datenbank zeigt Abb. 6-1. Das Zerlegen der Tabellen MITARBEITER und ABTEILUNG in verschiedene physische Fragmente ist eine wesentliche Aufgabe des Datenbankadministrators, nicht des Anwenders. Gemäß unserem Beispiel seien die Abteilungen Informatik und Personal geografisch in Basel, die Abteilung Finanz in Zürich lokalisiert. Fragment F1 als Teiltabelle der Tabelle MITARBEITER enthält nur Mitarbeiter der Informatik- und der Personalabteilung. Fragment F2 aus der ursprünglichen Tabelle ABTEILUNG beschreibt auf analoge Art diejenigen Abteilungen, die in Basel lokalisiert sind. Die beiden Fragmente F3 und F4 beziehen sich auf den Standort Zürich, und zwar hinsichtlich der Mitarbeiter und der Abteilungen.

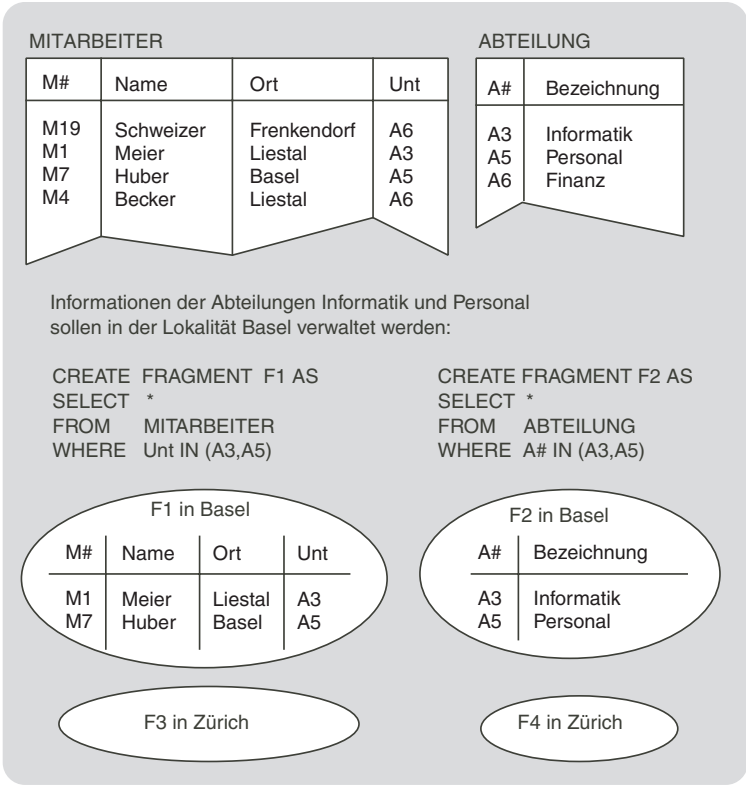


Abb. 6-1
Horizontale
Fragmentierung
der Tabelle
MITARBEITER und
ABTEILUNG

Man spricht von *horizontalen Fragmenten*, wenn eine bestimmte Tabelle horizontal zerlegt worden ist, die ursprüngliche Form der Tabellenzeilen also erhalten bleibt. Dabei sollten sich die verschiedenen Fragmente im Normalfall nicht gegenseitig überlappen, zusammengekommen jedoch die ursprüngliche Tabelle bilden.

Zur horizontalen
Fragmentierung

Anstelle einer horizontalen Zerlegung kann eine Tabelle in *vertikale Fragmente* unterteilt werden, indem mehrere Spalten mit dem Identifikationsschlüssel versehen zusammengefasst, die Tupel also zergliedert werden. Ein Beispiel dafür wäre eine Tabelle MITARBEITER, bei der Teile wie Lohn, Qualifikationsstufe, Entwicklungspotenzial etc. aus Gründen der Diskretion in einem vertikalen Fragment in der Personalabteilung gehalten würden. Die restlichen Merkmale hingegen könnten als weiteres Fragment in den verschiedenen Fachabteilungen vorliegen. Mischformen zwischen horizontalen und vertikalen Fragmenten sind ebenfalls möglich.

Vertikale und
gemischte
Fragmentierung

Eine wichtige Aufgabe eines verteilten Datenbanksystems ist die *Gewährleistung der lokalen Autonomie*. Der Anwender kann auf seinen lokalen Datenbeständen autonom arbeiten, und zwar auch dann,

Lokale Autonomie muss garantiert bleiben

wenn verschiedene Rechnerknoten im Netz nicht zur Verfügung stehen¹.

Verteiltes
Transaktions-
konzept
ist gefordert

Neben lokaler Autonomie ist das *Prinzip der dezentralen Verarbeitung* äußerst wichtig. Es bedeutet, dass das Datenbanksystem lokal in den verschiedenen Netzknoten Abfragen verarbeiten kann. Für solche dezentralen Anwendungen, die Daten aus verschiedenen Fragmenten beanspruchen, muss das Datenbanksystem einen entfernten Zugriff zum Lesen und Verändern von Tabellen ermöglichen. Dazu muss es ein verteiltes Transaktions- und Recoverykonzept zur Verfügung stellen. Solche Konzepte erfordern bei verteilten Datenbanken besondere Kontrollmechanismen, auf die näher einzugehen ist.

Wie funktioniert
das Two-Phase
Commit?

Beim Verarbeiten dezentraler Datenbanken garantiert das *Zweiphasen-Freigabeprotokoll* (engl. *two-phase commit protocol*) jederzeit Konsistenz. Dazu wird ein globales Koordinationsprogramm benötigt, das die lokalen Datenbanksysteme wie folgt überwacht: In einer ersten Phase signalisieren alle Teilhaberjobs dem Koordinationsprogramm, dass ihre lokalen Transaktionen beendet sind. Das Koordinationsprogramm sendet daraufhin ein PREPARE FOR COMMIT als Vorbereitung zum regulären Abschluss der Transaktion. Durch diese Aufforderung müssen alle Teilhaberprogramme ihre Datenbankänderungen definitiv in die lokalen Datenbanken einbringen. Je nach Erfolg dieser Aktion übermitteln sie dem zentralen Koordinationsprogramm ein OK oder NOT OK. In der zweiten Phase sendet das Koordinationsprogramm ein COMMIT, falls es von allen Teilhaberjobs ein OK erhalten hat. Falls mindestens eines der Teilhaberprogramme mit NOT OK in Phase 1 Misserfolg signalisiert hat, gibt das Koordinationsprogramm in Phase 2 ein NOT OK aus. In einem solchen Fehlerfall müssen alle Transaktionen, die bereits erfolgreich abgeschlossen sind, ihre Änderungen wieder rückgängig machen. Der Vorteil eines Zweiphasen-Freigabeprotokolls ist, dass entweder alle Teilhaberjobs mit Erfolg enden und die Datenbank korrekt nachführen oder dass überhaupt keine Wirkung in der Datenbank erzielt wird.

Optimale
Verarbeitung bei
verteilten
Datenbank-
anfragen

Von Bedeutung ist die *interne Verarbeitungsstrategie für verteilte Datenbankabfragen*, wie das Beispiel eines Interessenten für eine Liste der Mitarbeiternamen und der Abteilungsbezeichnungen in Abb. 6-2 darlegt. Die Abfrage kann ohne Einbeziehung von Fragmentangaben mit dem üblichen SQL formuliert werden. Die Aufgabe des Datenbanksystems besteht darin, für diese dezentrale Abfrage eine optimale Berechnungsstrategie festzusetzen. Sowohl

¹ Periodisch extrahierte Tabellenteile (sog. Snapshots) erhöhen diese lokale Autonomie.

die Tabelle MITARBEITER als auch die Tabelle ABTEILUNG liegen fragmentiert in Basel und Zürich vor. Deshalb werden *gewisse Berechnungen lokal und parallel* durchgeführt. Jeder Knoten organisiert unabhängig vom anderen den Verbund zwischen dem Mitarbeiter- und dem Abteilungsfragment. Nach diesen Teilberechnungen wird das Ergebnis durch Vereinigung der Teilresultate gebildet.

Zur zusätzlichen Optimierung werden in den einzelnen Knoten zunächst Projektionen auf die im Resultat gewünschten Merkmale Name und Bezeichnung realisiert. Sodann werden auf den reduzierten Tabellenfragmenten die Verbundoperatoren in Basel und in Zürich getrennt berechnet. Schließlich werden die beiden Zwischenergebnisse vor der Vereinigung ein weiteres Mal reduziert, indem auf die gewünschten Namen und Bezeichnungen projiziert wird.

Beispiel einer verteilten Anfrage

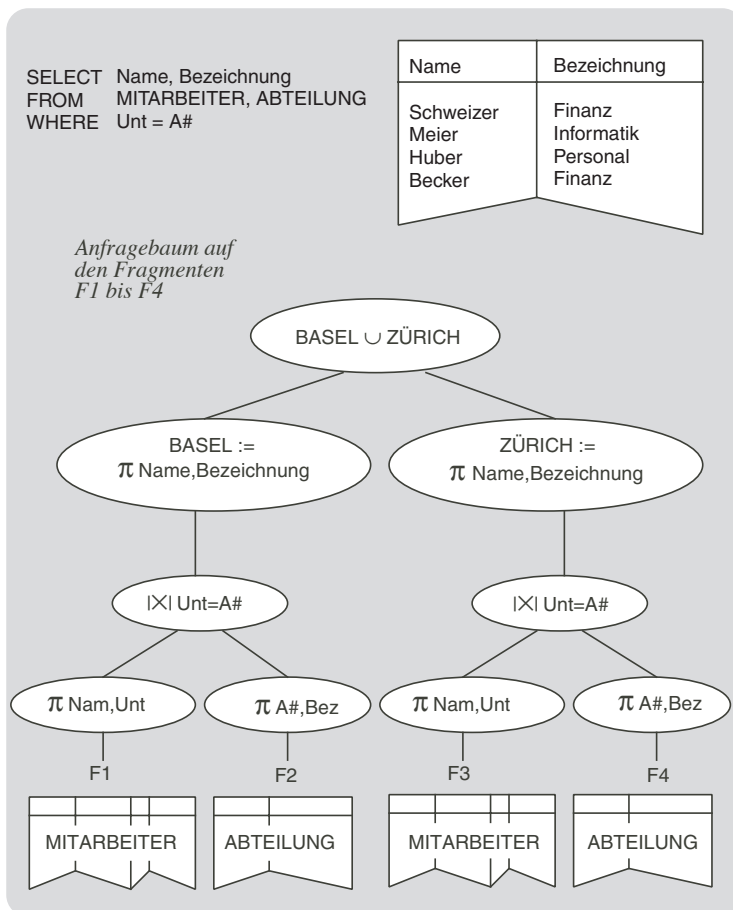


Abb. 6-2
*Optimierter
Anfragebaum für
eine verteilte
Verbundstrategie*

*Parallelität bei
verteilten
Anfragen*

Allgemein ist bei der Berechnung dezentraler Abfragen typisch, dass Vereinigungs- und Verbundoperatoren² spät evaluiert werden. Dies unterstützt *hohe Parallelität* in der Verarbeitung und fördert die Performance bei verteilten Abfragen. Die Optimierungsmaxime lautet also, Vereinigungsoperatoren im Anfragebaum möglichst in die Nähe des Wurzelknotens zu bringen, hingegen Selektionen und Projektionen im Blattbereich des Anfragebaumes anzusetzen.

*Replikate
erhöhen
Verfügbarkeit*

Die in Abb. 6-2 dargestellte Verbundstrategie ist optimal, da an verschiedenen Orten die Fragmente parallel benutzt werden können. Beim Entwurf einer verteilten Datenbank gelingt es normalerweise nicht, die Fragmente so zu legen, dass jeweils alle späteren Abfragen lokal als Zwischenresultate vorbereitet werden können. Ein verteiltes Datenbanksystem wird deshalb auf Grund der Anfrageprofile versuchen, im Laufe der Zeit die Fragmente von sich aus den jeweiligen Gegebenheiten anzupassen. Manchmal ist es zweckmäßig, Tabellenteile redundant unter Kontrolle des Datenbanksystems zu halten, da solche Replikate hohe Verfügbarkeit und große Auskunftsbereitschaft gewährleisten.

Verteiltes Datenbanksystem

*Haupt-
forderungen an
ein verteiltes
Datenbank-
system*

Ein verteiltes Datenbankmanagementsystem (abgekürzt VDBMS) muss folgende Bedingungen erfüllen:

- Es unterstützt *ein einziges logisches Datenbankschema und mehrere physische Fragmente auf örtlich verteilten Rechnern*.
- Es garantiert *Transparenz bezüglich der Verteilung von Datenbanken*, so dass Ad-hoc-Abfragen oder Anwendungsprogramme auf die physische Verteilung der Daten, d.h. auf die Fragmentbildung, keine Rücksicht nehmen müssen.
- Es gewährleistet lokale Autonomie, d.h., es erlaubt das lokale Arbeiten auf seinen dezentralen Datenbeständen, selbst wenn einzelne Rechnerknoten nicht verfügbar sind.
- Es garantiert mit dem *Zweiphasen-Freigabeprotokoll* die Konsistenz der verteilten Datenbanken und optimiert intern die verteilten Abfragen und Manipulationen mit einem Koordinationsprogramm.

*Entwurf verteilter
Datenbanken
bleibt
anspruchsvoll*

Erste Prototypen verteilter Datenbanksysteme sind Anfang der achtziger Jahre entstanden. Heute sind kommerzielle Produkte verfügbar, die die oben formulierten Anforderungen an ein VDBMS weitgehend erfüllen: So wird zwar die Verteilung ganzer Tabellen auf

² Mit dem so genannten Semi-Join wird ebenfalls versucht, bei einer Verbundoperation die Übermittlungskosten durch geschickte Wahl von Projektionen zu reduzieren.

verschiedene Rechner unterstützt, die Fragmentierung der Tabellen jedoch nur teilweise. Hinsichtlich eines optimalen Entwurfs einer verteilten Datenbank und hinsichtlich effizienter verteilter Manipulationsoperationen sind ebenfalls Verbesserungen zu erwarten.

6.3 Temporale Datenbanken

Heutige relationale Datenbanksysteme sind darauf ausgelegt, gegenwartsbezogene (aktuelle) Informationen in Tabellen verwalten zu können. Möchten die Anwender ihre relationalen Datenbanken hingegen allgemeiner zeitbezogen abfragen und auswerten, so müssen sie ihre vergangenheits- und zukunftsgerichteten Sachbestände individuell verwalten und nachführen. Das Datenbanksystem unterstützt sie nämlich beim Abspeichern, Suchen oder Auswerten zeitbezogener Informationen in keiner Weise.

Unter dem Begriff Zeit wird eine *eindimensionale physikalische Größe* verstanden, deren Werte total geordnet sind, so dass je zwei Werte auf der Zeitachse durch die Ordnungsrelationen «kleiner als» oder «größer als» miteinander verglichen werden können. Als Zeitangaben interessieren nicht nur Tag und Zeitpunkt wie z.B. «1. April 1989, 14:00 Uhr», sondern auch die Zeitdauer in Form von Zeitintervallen. Ein Beispiel dazu wäre das Alter eines Mitarbeiters, festgelegt durch eine Anzahl Jahre. Es gilt zu beachten, dass eine Zeitangabe je nach Auffassung des Anwenders als Zeitpunkt oder als Zeitdauer interpretiert werden kann.

Bei *temporalen Datenbanken* (engl. *temporal databases*) geht es darum, Datenwerte, einzelne Tupel oder ganze Tabellen mit der *Zeitachse in Beziehung* zu setzen. Die Zeitangabe selbst hat für ein bestimmtes Objekt in der Datenbank unterschiedliche Bedeutung, denn unter einer *Gültigkeitszeit* wird entweder die Angabe eines Zeitpunktes, zu dem ein bestimmtes Ereignis stattfindet, oder auch die Angabe eines Zeitintervalls verstanden, falls die entsprechenden Datenwerte während einer Zeitdauer gültig sind. So ist die Adresse eines Mitarbeiters bis zur nächsten Adressänderung gültig.

Um eine andere Art von Zeitangabe handelt es sich bei der *Aufzeichnungszeit*, die festhält, zu welchem Zeitpunkt ein bestimmtes Objekt in die Datenbank eingefügt, dort verändert oder gelöscht worden ist. Normalerweise verwaltet das Datenbanksystem die verschiedenen Aufzeichnungszeiten mit Hilfe eines Journals in eigener Regie, weshalb in den nun folgenden Ausführungen unter Zeit immer die Gültigkeitszeit verstanden wird.

Heutige Datenbanksysteme ignorieren temporale Konzepte

Zeitpunkte und Zeitintervalle

Die Gültigkeitszeit steht im Zentrum

Aufzeichnungszeit wird im Journal protokolliert

Abb. 6-3
Tabelle
MITARBEITER mit
Datentyp DATE

| MITARBEITER | | | | | |
|-------------|-----------|------------|-------------|------------|-------------|
| M# | Name | Geb.datum | Ort | Ein.datum | Funktion |
| M19 | Schweizer | 1948-02-19 | Frenkendorf | 1979-10-01 | Sachbearb. |
| M1 | Meier | 1958-07-09 | Liestal | 1984-07-01 | Analystiker |
| M7 | Huber | 1969-03-28 | Basel | 1988-01-01 | Pers.chef |
| M4 | Becker | 1952-12-06 | Liestal | 1978-04-15 | Revisor |

Gesucht sind alle Mitarbeiter, die vor dem zwanzigsten Altersjahr in die Firma eingetreten sind:

```
SELECT  M#, Name
FROM    MITARBEITER
WHERE   Eintrittsdatum - Geburtsdatum <= 20
```

| M# | Name |
|----|-------|
| M7 | Huber |

Temporale
Datentypen
DATE und TIME

Um Zeitpunkte der Gültigkeit erfassen zu können, werden von den meisten relationalen Datenbanksystemen bereits heute zwei Datentypen unterstützt: *DATE* dient der Angabe eines Datums in der Form Jahr, Monat und Tag, *TIME* der Angabe einer Uhrzeit durch Stunden, Minuten und Sekunden. Für die Angabe einer Zeitdauer muss kein spezieller Datentyp gewählt werden, ganze Zahlen und Dezimalzahlen genügen. Damit lässt sich auf natürliche Art mit Zeitangaben rechnen.

Beispiel
zeitbezogener
Merkmale

Ein Beispiel ist die in Abb. 6-3 dargestellte Mitarbeitertabelle, deren Merkmalskategorien durch Geburtsdatum und Eintrittsdatum ergänzt sind. Somit sind diese Merkmale zeitbezogen, und vom System kann nun beispielsweise eine Liste aller Mitarbeiter verlangt werden, die vor dem zwanzigsten Altersjahr in die Firma eingetreten sind.

Tabelle mit
Datumsfeldern ist
noch nicht
temporal

Die genannte Tabelle MITARBEITER bildet nach wie vor eine Momentaufnahme des aktuellen Datenbankbestandes. Wir können also weder Abfragen in die Vergangenheit noch Abfragen in die Zukunft stellen, da wir keinen Aufschluss über die *Gültigkeitszeit* der einzelnen Datenwerte erhalten. Wird in der Tabelle beispielsweise die Funktion des Mitarbeiters Huber verändert, so überschreiben wir den jetzigen Datenwert und betrachten die neue Funktion als die aktuelle. Hingegen wissen wir nicht, ab wann und bis wann Mitarbeiter Huber in einer bestimmten Funktion tätig gewesen ist.

Zeitstempel
drücken die
Gültigkeit aus

Um die *Gültigkeit einer Entität* auszudrücken, werden oft zwei Merkmale verwendet. Der Zeitpunkt «gültig von» (engl. *valid from*) gibt an, ab wann ein Tupel oder ein Datenwert gültig ist. Das Merk-

| TEMP_MITARBEITER (Auszug) | | | | | |
|---------------------------|------------|-------|---------|------------|----------------|
| M# | VALID_FROM | Name | Ort | Ein.datum | Funktion |
| M1 | 01.07.1984 | Meier | Basel | 1984-07-01 | Programmierer |
| M1 | 13.09.1986 | Meier | Liestal | 1984-07-01 | Programmierer |
| M1 | 04.05.1987 | Meier | Liestal | 1984-07-01 | Progr.-Analyt. |
| M1 | 01.04.1989 | Meier | Basel | 1984-07-01 | Analytiker |

Gesucht ist die Funktion des Mitarbeiters Meier am 01.01.1988:

ursprüngliches SQL:

```

SELECT Funktion
FROM TEMP_MITARBEITER A
WHERE A.M# = M1 AND
      A.VALID_FROM =
      (SELECT MAX(VALID_FROM)
       FROM TEMP_MITARBEITER B
       WHERE B.M# = M1 AND
             B.VALID_FROM <='01.01.1988')

```

temporales SQL:

```

SELECT Funktion
FROM TEMP_MITARBEITER
WHERE M# = M1 AND
      VALID_AT = (01.01.1988)

```

| Funktion |
|----------------|
| Progr.-Analyt. |

Abb. 6-4
Auszug aus einer
temporalen
Tabelle
TEMP_MITARBEITER

mal «gültig bis» (engl. *valid to*) drückt durch den entsprechenden Zeitpunkt das Ende des Gültigkeitsintervalls aus. Anstelle der beiden Zeitpunkte VALID_FROM und VALID_TO kann auf der Zeitachse der Zeitpunkt VALID_FROM bereits genügen. Die Zeitpunkte VALID_TO sind nämlich implizit durch die jeweils nächstfolgenden Zeitpunkte VALID_FROM gegeben, da sich die Gültigkeitsintervalle einer bestimmten Entität nicht überlappen können.

Die in Abb. 6-4 dargestellte temporale Tabelle TEMP_MITARBEITER listet für die Mitarbeitertupel M1 (Meier) im Merkmal VALID_FROM sämtliche Gültigkeitsangaben auf. Dieses Merkmal *muss* zum Schlüssel gezählt werden, damit nicht nur die aktuellen Zustände, sondern auch Vergangenheitswerte und Zukunftsangaben eindeutig identifiziert werden können. Die vier Tupeleinträge lassen sich wie folgt interpretieren: Der Mitarbeiter Meier wohnte vom 01.07.84 bis zum 12.09.86 in Basel, anschließend bis zum 31.03.89 in Liestal und ab 01.04.89 wieder in Basel. Seit Eintritt in die Firma bis zum 03.05.87 war er als Programmierer tätig, vom 04.05.87 bis zum 31.03.89 als Programmierer-Analytiker, seit 01.04.89 ist er Analytiker. Die Tabelle TEMP_MITARBEITER ist in der Tat temporal, da sie neben aktuellen Zuständen auch Angaben über vergangenheitsbezogene Datenwerte aufzeigt. Insbesondere kann sie Abfragen beant-

Temporale
Tabelle drückt
historische
Entwicklung aus

worten, die nicht nur aktuelle Zeitpunkte oder Zeitintervalle betreffen.

*Beispiel einer
Abfrage in der
Vergangenheit*

Als Beispiel interessieren wir uns in Abb. 6-4 für die Funktion des Mitarbeiters Meier am 1. Januar 1988. Anstelle des ursprünglichen SQL-Ausdrucks einer geschachtelten Abfrage mit der Funktion MAX (vgl. Abschnitt 3.4.1 resp. Tutorium in SQL) könnte man sich eine Sprache vorstellen, die temporale Abfragen direkt unterstützt. Mit dem Schlüsselwort *VALID_AT* wird ein Zeitpunkt festgelegt, zu dem alle gültigen Einträge gesucht werden sollen.

*Grund-
forderungen an
temporale Daten-
banksysteme*

Temporales Datenbanksystem

Ein temporales Datenbankmanagementsystem (TDBMS)

- unterstützt als *Gültigkeitszeit* die *Zeitachse*, indem es Merkmalswerte oder Tupel nach der Zeit ordnet und
- umfasst *temporale Sprachelemente* für Abfragen in die Zukunft, Gegenwart und Vergangenheit.

*Vorschläge für
temporales SQL*

Auf dem Gebiet der *temporalen Datenbanken* liegen verschiedene Sprachmodelle vor, die das Arbeiten mit zeitbezogenen Informationen vereinfachen. Speziell müssen die Operatoren der Relationenalgebra bzw. des Relationenkalküls erweitert werden, um z.B. einen Verbund von temporalen Tabellen zu ermöglichen. Auch die Regeln der referenziellen Integrität müssen angepasst und zeitbezogen ausgelegt werden.

*Zur Speicherung
temporaler
Datenbanken*

Vergangenheits- oder zukunftsbezogene Dateninhalte vergrößern das Volumen relationaler Datenbanken beträchtlich, so dass ausgeklügelte Implementierungsverfahren entwickelt worden sind. Beim *Differenzverfahren* werden lediglich die einzelnen mutierten Datenwerte als Differenzen registriert, nicht mehr ganze Tupel oder Tabellen. Dabei müssen Algorithmen implementiert werden, um aus veralteten Zuständen und Differenzen den aktuellen Zustand rekonstruieren und vergangenheitsbezogene Abfragen verarbeiten zu können. Obwohl sich solche Speicherungsverfahren und entsprechende Spracherweiterungen als Prototypen in Forschungs- und Entwicklungslabors bereits bewährt haben, unterstützen heute die wenigsten kommerziellen Datenbanksysteme temporale Konzepte.

6.4

Objektrelationale Datenbanken

Im Relationenmodell werden Informationen in einfachen Tabellen abgelegt. Sind die zur Speicherung in einer relationalen Datenbank vorgesehenen Daten schon von sich aus strukturiert, so werden sie auf Grund der Normalformen zerlegt und auf einzelne Tabellen aufgeteilt. Auch Wiederholungsgruppen sind innerhalb einer Tabelle nicht zugelassen. Darüber hinaus müssen alle Beziehungen zwischen den Daten durch gemeinsame Merkmalswerte vom Benutzer definiert und unterhalten werden. Das Datenbanksystem kann somit die Struktureigenschaften für Speicherungs- und Abfragezwecke nicht effizient nutzen.

Wollen wir in eine relationale Datenbank Informationen über Bücher ablegen, so müssen wir mehrere Tabellen definieren, drei davon sind in Abb. 6-5 dargestellt: In der Tabelle BUCH versehen wir ein Buch mit dem Titel und dem Verlag. Da an einem bestimmten Buch mehrere Autoren beteiligt sein können und umgekehrt ein Autor mehrere Bücher schreiben kann, reihen wir jede einzelne Urheberschaft an einem Buch in einer zusätzlichen Tabelle AUTOR auf. Das Merkmal «Name» ist nicht voll funktional vom zusammengesetzten Schlüssel der Autoren- und Buchnummer abhängig, weshalb sich die Tabelle weder in zweiter noch in höherer Normalform befindet. Dasselbe gilt für die Tabelle SCHLAGWORT, da zwischen den Büchern und ihren Schlagwörtern ebenfalls eine komplex-komplexe Beziehung besteht. Die «Gewichtung» ist zwar ein typisches Beziehungsmerkmal, hingegen ist die «Bezeichnung» nicht voll funktional vom Schlüssel der Schlagwort- und Buchnummer abhängig. Für die Verwaltung von Büchern würden somit aus Gründen der Normalisierung fünf Tabellen resultieren, da zusätzlich zu den Beziehungstabellen AUTOR und SCHLAGWORT eigenständige Tabellen für die Autoren- und Schlagwortmerkmale hinzukommen müssten.

Es ist von Nachteil und aus Anwendersicht kaum verständlich, Buchinformationen auf mehrere Tabellen zu zerstreuen, möchte doch der Benutzer die Eigenschaften eines bestimmten Buches strukturiert in einer einzigen Tabelle aufgelistet haben. Die relationale Abfrage- und Manipulationssprache wäre eigentlich dazu da, durch einfache Operatoren unterschiedliche Buchinformationen zu verwalten. Auch hinsichtlich der Leistung (Performance) ergeben sich Nachteile, wenn das Datenbanksystem mehrere Tabellen durchsuchen und zeitaufwändige Verbundoperatoren berechnen muss, um ein bestimmtes Buch aufzufinden. Wegen solcher Nachteile sind für das Relationenmodell Erweiterungen vorgeschlagen worden.

Strukturierte und semi-strukturierte Daten eignen sich schlecht für relationale Datenbanken

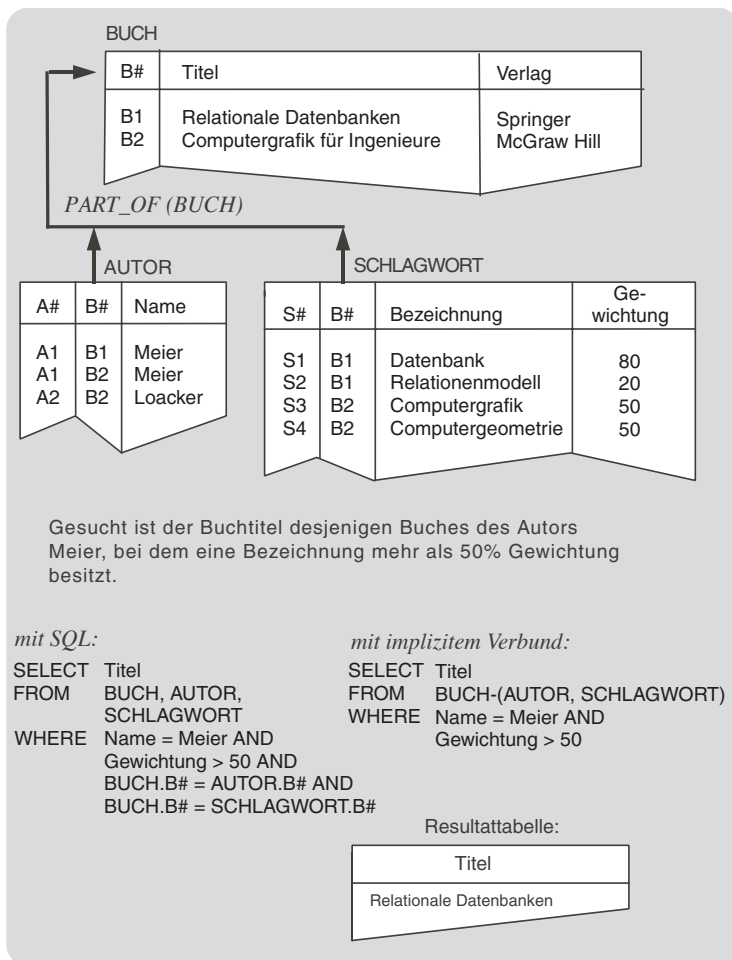
Ein Buch besteht aus mehreren Tabellen und bildet ein strukturiertes Objekt

Relationale Datenbanken unterstützen strukturierte Objekte mangelhaft

Surrogate
dienen der
Strukturierung
und werden vom
Datenbank-
system verwaltet

Eine erste Erweiterung relationaler Datenbanktechnologie besteht darin, dem *Datenbanksystem Struktureigenschaften explizit bekannt-zugeben*. Dies kann durch die Vergabe so genannter *Surrogate* geschehen. Ein Surrogat ist ein *vom System definierter, fixer und unveränderbarer (invarianter) Schlüsselwert*, der jedes Tupel in der Datenbank eindeutig identifiziert. Surrogate können als invariante Werte zur Definition systemkontrollierter Beziehungen benutzt werden, und zwar an verschiedenen Stellen innerhalb einer relationalen Datenbank. Solche Surrogate unterstützen die referenzielle Integrität, aber auch Generalisations- und Aggregationsstrukturen.

Abb. 6-5
Abfrage eines
strukturierten
Objektes mit und
ohne impliziten
Verbundoperator



Kehren wir nun zurück zur Tabelle BUCH und zur Abb. 6-5. Dort sei die Buchnummer B# als Surrogat definiert. Zusätzlich wird diese Nummer in den beiden abhängigen Tabellen AUTOR und SCHLAGWORT unter dem Hinweis PART_OF(BUCH) nochmals verwendet (vgl. Regel 7 zur Aggregation in Abschnitt 2.3.3). Diese Referenz sorgt dafür, dass das Datenbanksystem die Struktureigenschaft der Bücher-, Autoren- und Schlagwortangaben explizit kennt und diese bei Datenbankabfragen ausnutzen kann, sofern die relationale Abfrage- und Manipulationssprache entsprechend erweitert wird. Als Beispiel diene der implizite hierarchische Verbundoperator, der in der FROM-Klausel steht und die zur Tabelle BUCH gehörenden Teiltabellen AUTOR und SCHLAGWORT miteinander verknüpft. Die Verbundprädikate in der WHERE-Klausel anzugeben erübrigt sich, da diese mit der expliziten Definition der PART_OF-Struktur dem Datenbanksystem bereits bekannt sind.

Wird eine PART_OF- oder auf analoge Art eine IS_A-Struktur (vgl. Abschnitt 2.2.3) dem Datenbanksystem mitgeteilt, so lassen sich auch die Speicherstrukturen effizienter implementieren. Beispielsweise wird zwar die logische Sicht der drei Tabellen BUCH, AUTOR und SCHLAGWORT beibehalten, physisch hingegen werden die Buchinformationen als strukturierte Objekte³ abgespeichert, so dass ein einziger Datenbankzugriff das Auffinden eines Buches bewerkstelligt. Die klassische Sicht der Tabellen bleibt erhalten, da einzelne Tabellen der Aggregation wie bisher abgefragt werden können.

Eine andere Möglichkeit zur Verwaltung strukturierter Informationen liegt darin, die *erste Normalform aufzugeben*⁴ und als Merkmale selbst Tabellen zuzulassen, wie das in Abb. 6-6 abgewandelte Buchbeispiel illustriert, wo Informationen über Bücher, Autoren und Schlagwörter in einer Tabelle untergebracht sind. Auch dieses zeigt einen objektrelationalen Ansatz, da ein Buch als ein Objekt in einer einzigen Tabelle BUCHOBJEKT verwaltet werden kann. Ein *objektrelationales Datenbanksystem* (engl. *object-relational database system*) kann Struktureigenschaften explizit aufnehmen und Operatoren für Objekte und Teilobjekte anbieten.

Unterstützt ein Datenbanksystem strukturierte Objekttypen wie in den Beispieldarstellungen Abb. 6-5 und 6-6, so ist es strukturell objektrelational. Zusätzlich zur Objektidentifikation, zur Strukturbeschreibung und zum Angebot generischer Operatoren (Methoden wie impliziter Verbund etc.) sollte ein vollständig objektrelationales Datenbanksystem dem Benutzer auch das Definieren neuer Objekttypen

Surrogate erlauben PART_OF-Strukturen

PART_OF- und IS_A-Strukturen sind dem Datenbanksystem explizit bekannt

Geschachtelte Tabellen sind nützlich

Objektrelationale Datenbanksysteme sollten das objektorientierte Paradigma unterstützen

³ Die Forschungsliteratur nennt diese auch «komplexe Objekte».

⁴ Das so genannte NF²-Modell (NF² = Non First Normal Form) erlaubt geschachtelte Tabellen.

pen (Klassen) und Methoden anbieten. Der Anwender sollte dabei die für einen individuellen Objekttyp notwendigen Methoden selbst festlegen können. Auf tatkräftige Unterstützung von «Vererbungseigenschaften» sollte er zählen können, damit er nicht alle Objekttypen und Methoden von Grund auf neu zu definieren braucht, sondern auf bereits bestehende Konzepte zurückgreifen kann.

Generische Operatoren für Aggregation und Generalisation

Objektrelationale Datenbanksysteme machen es möglich, strukturierte Objekte als Einheiten zu behandeln und entsprechende generische Operatoren auf diese Objekte anzuwenden. Klassenbildung durch PART_OF- und IS_A-Strukturen ist erlaubt und wird durch Methoden zum Speichern, Abfragen und Manipulieren unterstützt.

Die Hauptforderungen an objektrelationale Datenbanksysteme

Objektrelationales Datenbanksystem

Ein objektrelationales Datenbankmanagementsystem (ORDBMS) lässt sich wie folgt umreißen:

- Es erlaubt die *Definition von Objekttypen* (in Anlehnung an die objektorientierte Programmierung oft Klassen genannt), die ihrerseits aus weiteren Objekttypen zusammengesetzt sein können.
- Jedes Datenbankobjekt kann aufgrund von *Surrogaten strukturiert und identifiziert* werden.
- Es unterstützt *generische Operatoren* (Methoden), die auf Objekte oder Teilobjekte wirken. Dabei bleibt die interne Darstellung der Objekte nach außen unsichtbar (Datenkapselung).
- Eigenschaften von Objekten lassen sich vererben. Diese *Vererbungseigenschaft* bezieht sich sowohl auf die Struktur als auch auf die zugehörigen Operatoren.

Der neueste SQL-Standard enthält objektorientierte Erweiterungen

Ein objektrelationales Datenbanksystem muss den *erweiterten SQL-Standard* (unter dem Namen SQL: 2003) unterstützen, der die folgenden objektorientierten Erweiterungen verlangt: Objektidentifikationen (Surrogate), vordefinierte Datentypen für Menge, Liste und Feld,

Abb. 6-6
Tabelle
BUCHOBJEKT mit
Merkmale vom
Typ Relation

| BUCHOBJEKT | | | | | | | |
|------------|---------|-------------|-------|---------|------------|-------------------------|------|
| B# | Titel | Verlag | Autor | | Schlagwort | | |
| | | | A# | Name | S# | Bezeichnung | Gew. |
| B1 | Rel... | Springer | A1 | Meier | S1 | Datenbank Rel.modell | 80 |
| | | | | | S2 | | 20 |
| B2 | Comp... | McGraw Hill | A1 | Meier | S3 | Comp.grafik C.geometrie | 50 |
| | | | A2 | Loacker | S4 | | 50 |

allgemeine abstrakte Datentypen mit Kapselungsmöglichkeiten, parametrisierbare Typen, Typ- und Tabellenhierarchien mit Mehrfachvererbung sowie benutzerdefinierte Funktionen (Methoden).

6.5

Multidimensionale Datenbanken

Bei operativen Datenbanken und Anwendungen konzentriert man sich auf einen klar definierten, funktionsorientierten Leistungsbe- reich. Transaktionen bezwecken, Daten für die Geschäftsabwicklung schnell und präzise bereitzustellen. Diese Art der Geschäftstätigkeit wird oft als *Online Transaction Processing* oder OLTP bezeichnet.

Da die operativen Datenbestände täglich neu überschrieben wer- den, gehen für den Anwender wichtige Entscheidungsgrundlagen verloren. Zudem sind diese Datenbanken primär für das laufende Geschäft und nicht für Analyse und Auswertung konzipiert worden. Aus diesen Gründen werden seit einigen Jahren neben transaktions- orientierten Datenbeständen auch eigenständige Datenbanken und Anwendungen entwickelt, die der Datenanalyse und der Entschei- dungsunterstützung dienen. Man spricht in diesem Zusammenhang von *Online Analytical Processing* oder OLAP.

Kernstück von OLAP ist eine *multidimensionale Datenbank* (engl. *multi-dimensional database*), in der alle entscheidungsrelevanten Sachverhalte nach beliebigen Auswertungsdimensionen abgelegt werden (mehrdimensionaler Datenwürfel oder Data Cube). Eine sol- che Datenbank kann recht umfangreich werden, da sie Entschei- dungsgrößen zu unterschiedlichen Zeitpunkten enthält. Beispiels- weise können in einer multidimensionalen Datenbank Absatzzahlen quartalsweise, nach Verkaufsregionen und Produkten abgelegt und ausgewertet werden.

Betrachten wir dazu ein Beispiel in Abb. 6-7, das zugleich den Entwurf einer multidimensionalen Datenbank illustrieren soll. In die- sem Beispiel interessieren uns drei Auswertungsdimensionen, näm- lich Produkt, Region und Zeit. Der Begriff *Dimension* (engl. *dimen- sion*) beschreibt die Achsen des mehrdimensionalen Würfels. Der Entwurf dieser Dimensionen ist bedeutend, werden doch entlang die- ser Achsen Analysen und Auswertungen vorgenommen. Die Reihen- folge der Dimensionen spielt keine Rolle, jeder Anwender kann und soll aus unterschiedlichen Blickwinkeln seine gewünschten Auswer- tungen vornehmen können. Beispielsweise priorisiert ein Produkte- manager die Produktdimension oder ein Verkäufer möchte Ver- kaufszahlen nach seiner Region auflisten.

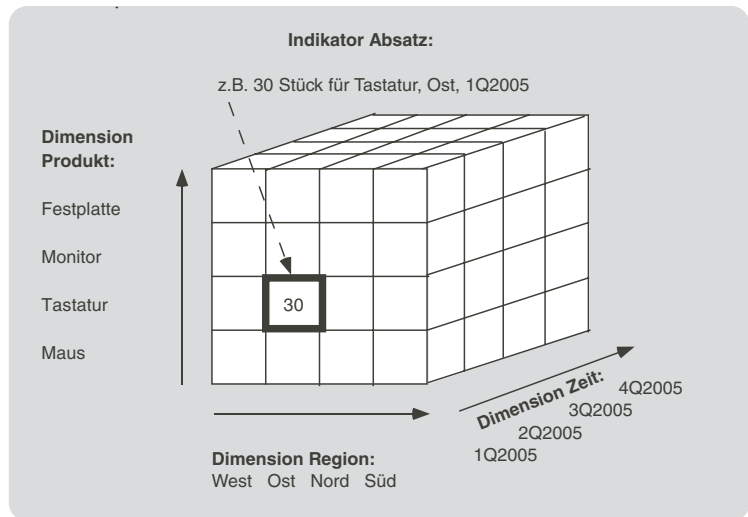
Was versteht man unter OLTP?

Wozu benötigt man OLAP?

Kernstück einer mehr- dimensional- en Datenbank

Dimensionen beschreiben die Achsen des Data Cube

Abb. 6-7
 Mehr-
 dimensionaler
 Würfel mit
 unterschiedlichen
 Auswertungs-
 dimensionen



Hierarchische Gliederung der Dimensionen

Die Dimensionen selber können weiter strukturiert sein: Die Dimension Produkt kann Produktgruppen enthalten; die Dimension Zeit könnte neben Quartalsangaben auch Tage, Wochen und Monate pro Jahr abdecken. Eine Dimension beschreibt somit die gewünschten *Aggregationsstufen*, die für die Auswertung des mehrdimensionalen Würfels gelten.

Indikatoren sind die Kenngrößen für die Entscheidungs- unterstützung

Aus logischer Sicht müssen bei einer mehrdimensionalen Datenbank resp. beim Datenwürfel neben den Dimensionen auch die Indikatoren⁵ spezifiziert werden. Unter einem *Indikator* (engl. *indicator*) versteht man eine Kennzahl resp. Kenngröße, die für die Entscheidungsunterstützung gebraucht wird. Indikatoren können quantitative wie qualitative Eigenschaften der Geschäftstätigkeit betreffen. Neben finanziellen Kenngrößen sind Indikatoren über Markt und Absatz, Kundenstamm und Kundenbewegung, Geschäftsprozesse, Innovationspotenzial oder Know-how der Belegschaft von Bedeutung. Die Indikatoren bilden neben den Dimensionen die Grundlage für die Entscheidungsunterstützung des Managements, für interne und externe Berichterstattung sowie für ein rechnergestütztes Performance Measurement System.

Indikatoren- und Dimensions- tabellen

Hauptmerkmal eines *Sternschemas* (engl. *star schema*) ist die Klassifikation der Daten in zwei Gruppen, nämlich Indikatorendaten und Dimensionsdaten. Beide Gruppen werden in Abb. 6-8 tabellarisch illustriert. Die Indikatorentabelle steht im Zentrum, zudem sind

⁵ In der Literatur werden die Indikatoren oft als Fakten bezeichnet, vgl. dazu auch Abschnitt 6.7 über Fakten und Regeln wissensbasierter Datenbanken.

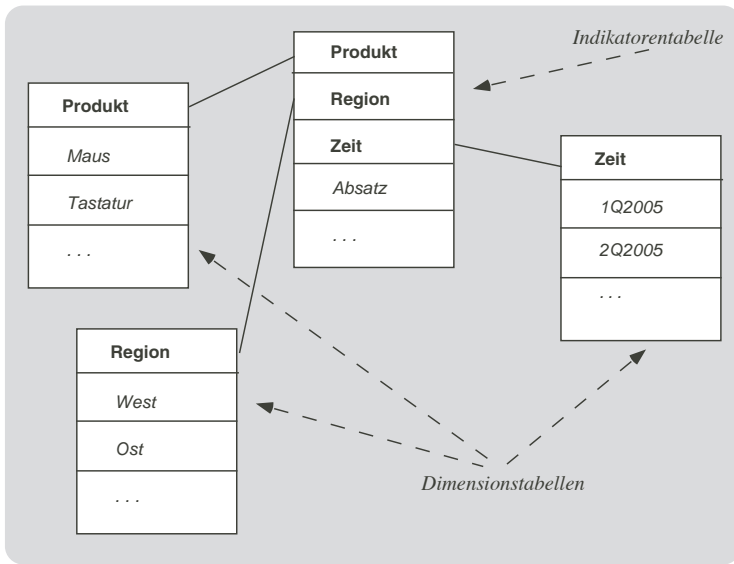


Abb. 6-8
Sternschema für
eine mehr-
dimensionale
Datenbank

die deskriptiven Dimensionstabellen angesiedelt, pro Dimension je eine Tabelle. Die Dimensionstabellen hängen also sternartig an der Indikatorentabelle.

Falls einzelne Dimensionen strukturiert sind, können an der entsprechenden Dimensionstabelle weitere Dimensionstabellen angehängt werden. Dadurch entsteht ein *Schneeflocken-Schema* (engl. *snowflake schema*), das Aggregationsstufen einzelner Dimensionen zeigt. Beispielsweise könnte in der Abb. 6-8 an der Dimensionstabelle Zeit für das erste Quartal 2005 eine weitere Dimensionstabelle angeschlossen werden, die die Tage der Monate Januar bis März 2005 aufzählt. Falls die Dimension Monat ebenfalls für Analysezwecke gebraucht wird, würde man eine weitere Dimensionstabelle Monat definieren und mit der Dimensionstabelle Tag verbinden etc.

Für die Implementierung einer mehrdimensionalen Datenbank kann das klassische Relationenmodell verwendet werden, allerdings resultieren daraus gewichtige Nachteile. In Abb. 6-9 zeigen wir, wie die Indikatoren- und Dimensionstabellen des Sternschemas umgesetzt werden.

Die Indikatorentabelle wird mit der Relation F_ABSATZ dargestellt, wobei ein mehrdimensionaler Schlüssel resultiert. Dieser zusammengesetzte Schlüssel muss als Komponenten die jeweiligen Schlüssel der Dimensionstabellen von D_PRODUKT, D_REGION und D_ZEIT enthalten. Möchte man nun z.B. den Umsatz des Verkaufsleiters Müller für das erste Quartal 2005 und für die Apple-Geräte ermitteln, so muss ein aufwändiger Verbund aller beteiligter Indika-

*Erweiterungen
des
Sternschemas*

*Relationale
Datenbanken
sind nur begrenzt
geeignet*

*Auswertung des
mehr-
dimensionalen
Würfels*

Nachteile
relationaler
Datenbanken zur
Verwaltung mehr-
dimensionaler
Würfel

toren- und Dimensionstabellen formuliert werden (siehe SQL-State-
ment in Abb. 6-9).

Die Leistungsfähigkeit eines relationalen Datenbanksystems stößt
bei umfangreichen multidimensionalen Datenbanken an Grenzen.
Auch das Formulieren von Abfragen ist mit dem Sternschema auf-
wändig und fehleranfällig. Darüber hinaus gibt es eine weitere
Anzahl von Nachteilen, falls mit dem klassischen Relationenmodell
und dem herkömmlichen SQL gearbeitet wird: Für die Bildung von
Aggregationsstufen muss das Sternschema zu einem Schneeflocken-
Schema ausgebaut werden, die entsprechenden physischen Tabellen
verschlechtern das Antwortzeitverhalten zusätzlich. Möchte ein
Anwender einer multidimensionalen Datenbank zu Analysezwecken
Details nachfragen (sog. Drill-down) oder weitere Aggregationsstu-
fen auswerten (sog. Roll-up), so hilft ihm das klassische SQL nicht
weiter. Auch das Herausschneiden oder Rotieren einzelner Teile aus
dem mehrdimensionalen Würfel, was zu Analysezwecken sehr
gebräuchlich ist, muss mit spezifischer Software oder sogar Hard-
ware erkaufte werden.

Von der
Datenbank zum
Data Warehouse

Wegen dieser Mängel haben sich einige Hersteller von Datenbank-
produkten entschlossen, ihre Softwarepalette mit geeigneten Werk-
zeugen für ein Data Warehouse resp. Data Mining anzureichern. Das
Data Warehouse ist nichts anderes als eine Softwareumgebung zum
Betreiben einer multidimensionalen Datenbank, wobei auch Werkzeug-

Abb. 6-9
Implementierung
eines
Sternschemas
mit dem
Relationenmodell

Gesucht ist der Apple-Umsatz fürs 1. Quartal 2005 von Verkaufsleiter Müller:

```
SELECT SUM(Umsatz)
FROM D_PRODUKT D1, D_REGION D2, D_ZEIT D3, F_ABSATZ F
WHERE D1.P# = F.P# AND
      D2.R# = F.R# AND
      D3.Z# = F.Z# AND
      D1.Lieferant = 'Apple' AND
      D2.Verkaufsleiter = 'Müller' AND
      D3.Jahr = 2005 AND
      D3.Quartal = 1
```

D_ZEIT

| Z# | Jahr | Quartal |
|----|------|---------|
| Z1 | 2005 | 1 |

F_ABSATZ

| P# | R# | Z# | Menge | Umsatz |
|----|----|----|-------|--------|
| P2 | R2 | Z1 | 30 | 160000 |

D_PRODUKT

| P# | Bezeichnung | Lieferant |
|----|-------------|-----------|
| P2 | Tastatur | Apple |

D_REGION

| R# | Name | Verkaufsleiter |
|----|------|----------------|
| R2 | Ost | Müller |

ge für die Datenbeschreibung (Wartung der Dimensionstabellen) sowie für das Integrieren unterschiedlicher Datenbestände miteingeschlossen werden. Auf der Sprachebene ist das SQL erweitert worden, um die verlangten Würfeloperationen inkl. Aggregationen einfach formulieren zu können:

Multidimensionales Datenbanksystem

Ein multidimensionales Datenbankmanagementsystem (MDBMS), oft als Data Warehouse bezeichnet, unterstützt einen mehrdimensionalen Würfel mit folgender Funktionalität:

- Für den Entwurf können unterschiedliche Dimensionstabellen mit beliebigen *Aggregationsstufen* definiert werden,
- die Dimension Zeit gehört zwingend zum Data Warehouse,
- die Auswertungs- und Analysesprache stellt Funktionen für *Drill-down* und *Roll-up* zur Verfügung,
- das Auswählen einer Scheibe aus dem mehrdimensionalen Würfel (*Slicing*) und ein *Wechsel der Dimensionsreihenfolge* (*Dicing*, *Rotation*) ist unproblematisch,
- bei der Implementierung der Indikatorentabelle wird keine Dimension ausgezeichnet, d.h. die Daten werden in *mehrdimensionalen Datenstrukturen* gehalten.

Die Forderungen an ein multidimensionales Datenbanksystem (Data Warehouse)

Im Data Warehouse lassen sich unterschiedliche interne und externe Datenquellen integrieren, um einen konsistenten Datenbestand für unterschiedliche betriebswirtschaftliche Sichten zu erhalten und auswerten zu können. In den meisten Fällen wird die mehrdimensionale Datenbank nur gelesen und analysiert, mit Ausnahme periodischer Nachführungen. Die Dimension Zeit ist fester Bestandteil einer solchen Datensammlung, wodurch statistische Auswertungen an Aussagekraft gewinnen. Falls Werkzeuge des sog. Data Mining wie Klassifikation, Prognose und Wissensakquisition verwendet werden, verwandelt sich die mehrdimensionale Datenbank in eine Wissensbank.

Vom Data Warehouse zum Data Mining

6.6

Fuzzy Datenbanken

Bei herkömmlichen Datenbanksystemen werden die Merkmalswerte als präzise, sicher und scharf vorausgesetzt und Abfragen ergeben klare Ergebnisse:

*Merkmalswerte
klassischer
Datenbestände
sind präzise,
sicher und scharf*

- Die Merkmalswerte in den Datenbanken sind *präzise*, d.h. sie sind eindeutig. Die erste Normalform verlangt, dass die Merkmalswerte atomar sind und aus einem wohldefinierten Wertebereich stammen. Vage Merkmalswerte wie «2 oder 3 oder 4 Tage» oder «ungefähr 3 Tage» beim Terminverzug eines Lieferanten sind nicht zugelassen.
- Die in einer relationalen Datenbank abgelegten Merkmalswerte sind *sicher*, d.h., die einzelnen Werte sind entweder bekannt oder sie sind unbekannt. Eine Ausnahme bilden Nullwerte, d.h. Merkmalswerte, die nicht oder noch nicht bekannt sind. Darüber hinaus bieten die Datenbanksysteme keine Modellierungskomponente für vorhandene Unsicherheit an. So sind Wahrscheinlichkeitsverteilungen für Merkmalswerte ausgeschlossen; es bleibt schwierig auszudrücken, ob ein bestimmter Merkmalswert dem wahren Wert entspricht oder nicht.
- Abfragen an die Datenbank sind *scharf*. Sie haben immer einen dichotomen Charakter, d.h., ein in der Abfrage vorgegebener Abfragewert muss mit den Merkmalswerten in der Datenbank entweder übereinstimmen oder nicht übereinstimmen. Eine Auswertung der Datenbank, bei der ein Abfragewert mit den gespeicherten Merkmalswerten «mehr oder weniger» übereinstimmt, ist nicht zulässig.

*Fuzzy Logic und
Datenbanken*

Seit einigen Jahren werden Erkenntnisse aus dem *Gebiet der unscharfen Logik* (engl. *fuzzy logic*) auf Datenmodellierung und Datenbanken angewendet. Falls man unvollständige oder vage Sachverhalte zulässt, lässt sich ein größeres Anwendungsspektrum erschließen. Die meisten dieser Arbeiten sind theoretischer Natur; einige Forschungsgruppen versuchen allerdings, mit Implementierungen die Nützlichkeit unscharfer Datenbankmodelle und Datenbanksysteme aufzuzeigen.

*Kontextmodell mit
Membership
Function*

Der hier aufgezeigte Forschungsansatz basiert auf einem *Kontextmodell*, um Klassen von Datensätzen im relationalen Datenbankschema festlegen zu können. Bei der Klassifikation kann man zwischen scharfen und unscharfen Verfahren unterscheiden. Bei einer scharfen Klassifikation wird eine dichotome Zuweisung der Datenbankobjekte zur Klasse vorgenommen, d.h., die Mengenzuge-

hörigkeitsfunktion des Objekts zur Klasse beträgt 0 für nicht enthalten oder 1 für enthalten. Ein klassisches Verfahren würde daher einen Kunden entweder der Klasse «Kunden mit Umsatzproblemen» oder der Klasse «Kunden, zu denen die Geschäftsbeziehung ausgebaut werden soll» zuordnen. Ein unscharfes Verfahren dagegen lässt für die *Mengenzugehörigkeitsfunktion* (engl. *membership function*) Werte zwischen 0 und 1 zu: Ein Kunde kann z.B. mit einem Wert von 0.3 zur Klasse «Kunden mit Umsatzproblemen» gehören und gleichzeitig mit einer Zugehörigkeit von 0.7 zur Klasse der «Kunden, zu denen die Geschäftsbeziehung ausgebaut werden soll». Eine unscharfe Klassifikation ermöglicht daher eine *differenziertere Interpretation der Klassenzugehörigkeit*: Man kann bei Datenbankobjekten einer unscharfen Klasse zwischen Rand- oder Kernobjekten unterscheiden, zudem können Datenbankobjekte zu zwei oder mehreren unterschiedlichen Klassen gleichzeitig gehören.

Im fuzzy-relationalen Datenmodell mit Kontexten, kurz im Kontextmodell, ist jedem Attribut A_j definiert auf einem Wertebereich $D(A_j)$ ein Kontext zugeordnet. *Ein Kontext $K(A_j)$ ist eine Partition von $D(A_j)$ in Äquivalenzklassen.* Ein relationales Datenbankschema mit Kontexten besteht daher aus einer Menge von Attributen $A=(A_1,...,A_n)$ und einer Menge assoziierter Kontexte $K=(K_1(A_1),...,K_n(A_n))$.

Für eine Bewertung von Kunden dienen Umsatz und Treue als Beispiel. Zusätzlich werden die beiden qualifizierenden Attribute je in zwei Äquivalenzklassen zerlegt. Die entsprechenden Attribute und Kontexte für das Kundenbeziehungsmanagement lauten:

*Partition in
Äquivalenz-
klassen*

*Beispiel
Customer
Relationship
Management*

- Umsatz in Euro pro Monat: Der Wertebereich für den Umsatz in Euro soll durch $[0..1000]$ definiert sein. Zudem werden die beiden Äquivalenzklassen $[0..499]$ für kleinen Umsatz und $[500..1000]$ für großen Umsatz gebildet.
- Treue der Kunden: Der Wertebereich $\{\text{top, gut, medioker, schlecht}\}$ gilt für das Attribut der Kundentreue. Der Wertebereich wird in die beiden Äquivalenzklassen $\{\text{top, gut}\}$ für starke Treue und $\{\text{medioker, schlecht}\}$ für schwache Treue zerlegt.

Die beiden vorgeschlagenen Merkmale mit ihren Äquivalenzklassen zeigen je ein Beispiel für ein numerisches Attribut und für ein qualitatives. Die entsprechenden Kontexte sind:

- $K(\text{Umsatz}) = \{ [0..499], [500..1000] \}$
- $K(\text{Treue}) = \{ \{\text{medioker, schlecht}\}, \{\text{top, gut}\} \}$

Die Partitionierung der Wertebereiche Umsatz und Treue ergibt in Abb. 6-10 die vier Äquivalenzklassen C1, C2, C3 und C4. Die inhalt-

Abb. 6-10
Klassifikations-
raum
aufgespannt
durch die
Attribute Umsatz
und Treue

| | top | gut | medioker | schlecht | D(Treue) |
|------|----------------------------------|-----|---------------------------------|----------|----------|
| 1000 | C1 Commit Customer | | C2 Improve Loyalty | | |
| 500 | | | | | |
| 499 | C3 Augment Turnover | | C4 Don't Invest | | |
| 0 | | | | | |

D(Umsatz)

liche Bedeutung der Klassen wird durch semantische Klassennamen ausgedrückt; so wird z.B. für Kunden mit kleinem Umsatz und schwacher Treue die Bezeichnung «Don't Invest» für die Klasse C4 gewählt. Es gehört zu den Aufgaben des Datenbankadministrators, in Zusammenarbeit mit den Marketingspezialisten sowohl die Attribute wie die Äquivalenzklassen festzulegen und diese als Erweiterung des Datenbankschemas zu spezifizieren.

Verwendung
linguistischer
Variablen

Gemäß Abb. 6-11 kann für einen bestimmten Kunden die Treue als *linguistische Variable* (engl. *linguistic variable*) gleichzeitig «stark» und «schwach» sein; zum Beispiel ist die Zugehörigkeit von Becker zur scharfen Menge stark 0.66 und diejenige zur Menge schwach ist 0.33. Der Treuegrad von Becker ist also nicht ausschließlich stark oder schwach wie bei scharfen Klassen.

Klassen mit
kontinuierlichen
Übergängen

Die linguistische Variable Treue mit den vagen Termen «stark» und «schwach» und den Zugehörigkeitsfunktionen stark und schwach bewirkt, dass der Wertebereich D(Treue) unscharf partitioniert wird. Analog wird der Wertebereich D(Umsatz) durch die Terme «groß» und «klein» unterteilt. Dadurch entstehen im Kontextmodell Klassen mit kontinuierlichen Übergängen, d.h. unscharfe Klassen.

Aggregation von
Zugehörigkeits-
werten

Die Zugehörigkeit eines Objekts zu einer Klasse ergibt sich aus der Aggregation über alle Terme, die die Klasse definieren. Die Klasse C1 z.B. wird durch die Terme «groß» (für die linguistische Variable Umsatz) und «stark» (für die linguistische Variable Treue) beschrieben. Die Aggregation muss daher einer Konjunktion der einzelnen Zugehörigkeitswerte entsprechen. Dazu sind in der Theorie der unscharfen Mengen verschiedene Operatoren entwickelt worden.

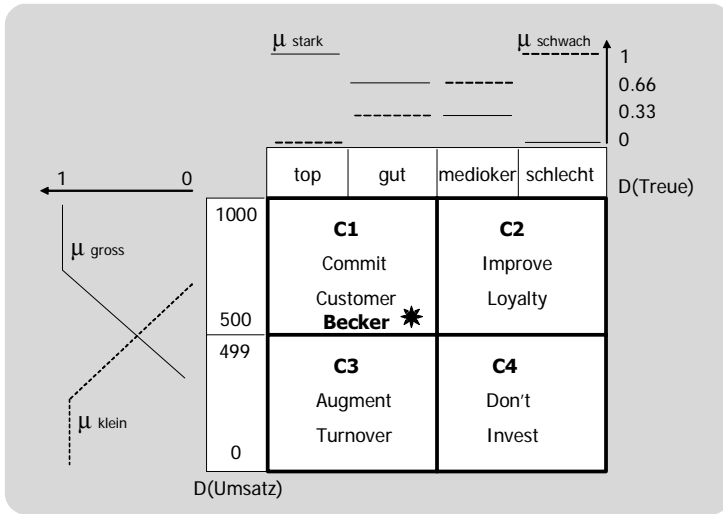


Abb. 6-11
Unscharfe
Partitionierung
der Werte-
bereiche mit
Zugehörigkeits-
funktionen

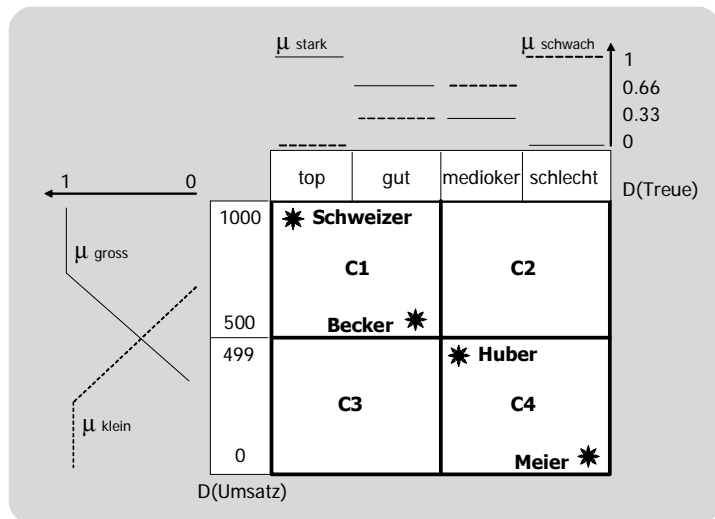
Das Kundenbeziehungsmanagement bezweckt, anstelle produktbezogener Argumentationslisten und Anstrengungen, die kundenindividuellen Wünsche und das Kundenverhalten miteinzubeziehen. Sollen Kunden als Vermögenswert (customer value) aufgefasst werden, so müssen sie entsprechend ihrem Markt- und Ressourcenpotenzial behandelt werden. Mit scharfen Klassen, d.h. traditionellen Kundensegmenten, ist dies kaum möglich, da alle Kundinnen und Kunden in einer Klasse gleich behandelt werden. In Abb. 6-12 beispielsweise besitzen Becker und Huber einen ähnlichen Umsatz und zeigen ein ähnliches Treueverhalten. Trotzdem werden sie bei einer scharfen Segmentierung unterschiedlich klassifiziert: Becker gehört zur Premiumklasse C1 (Commit Customer) und Huber zur Verliererklasse C4 (Don't Invest). Zusätzlich wird der topgesetzte Kunde Schweizer gleich behandelt wie Becker, da beide zum Segment C1 gehören.

Kunden mit ähnlichem Verhalten liegen in unterschiedlichen Segmenten

Gemäß Abb. 6-12 können bei einer scharfen Kundensegmentierung folgende Konfliktsituationen auftreten:

- Kunde Becker hat wenige Anreize, seinen Umsatz zu steigern oder die Kundenbindung und -treue zu verbessern. Er liegt in der Premiumklasse C1 und genießt die entsprechenden Vorteile.
- Kunde Becker kann überrascht werden, falls sein Umsatz ein wenig zurückgeht oder sein Treuebonus abnimmt. Plötzlich sieht er sich einem andern Kundensegment zugeordnet; im Extremfall fällt er von der Premiumklasse C1 in die Verliererklasse C4.
- Kunde Huber verfügt über einen ordentlichen Umsatz und eine mittlere Kundentreue, wird aber als Verlierer behandelt. Es wird

Abb. 6-12
Konflikt-
situationen bei
scharfen Kunden-
segmenten



kaum überraschen, wenn sich Huber im Markt umsieht und abspringt.

- Eine scharfe Kundensegmentierung lässt auch für Kunde Schweizer eine kritische Situation entstehen. Er ist im Moment der profitabelste Kunde mit ausgezeichnetem Ruf, wird aber vom Unternehmen nicht entsprechend seinem Kundenwert wahrgenommen und behandelt.

Die hier exemplarisch aufgezeigten Konfliktsituationen können entschärft oder eliminiert werden, falls unscharfe Kundenklassen gebildet werden. Die Positionierung eines Kunden im zwei- oder mehrdimensionalen Datenraum entspricht dem Kundenwert, der jetzt aus unterschiedlichen Klassenzugehörigkeitsanteilen besteht.

Unscharfe Klassifikations- sprache

Klassifikationsabfragen mit der *Sprache fCQL* (fuzzy Classification Query Language) operieren auf der linguistischen Ebene mit vagen Kontexten. Das hat den Vorteil, dass der Anwender keinen scharfen Zielwert und keinen Kontext kennen muss, sondern lediglich den Spaltennamen des objektidentifizierenden Merkmals und die Tabelle oder Sicht, in der die Merkmalswerte enthalten sind. Für eine gezielte Betrachtung einzelner Klassen kann der Anwender eine Klasse spezifizieren oder Merkmale mit einer verbalen Beschreibung ihrer Ausprägungen angeben. Klassifikationsabfragen arbeiten also mit verbalen Beschreibungen auf Merkmals- und Klassenebene:

```

CLASSIFY   Objekt
FROM       Tabelle
WITH       Klassifikationsbedingung

```

Die Sprache fCQL ist an SQL angelehnt, wobei die Projektionsliste bei der SELECT-Klausel durch den Spaltennamen des zu klassifizierenden Objekts ersetzt wird. Während die WHERE-Klausel bei SQL eine Selektionsbedingung enthält, wird mit der WITH-Klausel die Klassifikationsbedingung festgelegt. Als Beispiel einer fCQL-Abfrage erzeugt «CLASSIFY Kunde FROM Kundentabelle» eine Klassifikation sämtlicher in der Tabelle vorhandener Kunden. Mit «CLASSIFY Kunde FROM Kundentabelle WITH CLASS IS Augment Turnover» wird gezielt die Klasse C3 abgefragt. Verzichtet man auf die Definition der Klasse, so kann man mit den linguistischen Beschreibungen der Äquivalenzklassen eine bestimmte Objektmenge selektieren. Als Beispiel gilt die Abfrage «CLASSIFY Kunde FROM Kundentabelle WITH Umsatz IS klein AND Treue IS stark». Diese Abfrage besteht aus dem Bezeichner des zu klassifizierenden Objekts (Kunde), dem Namen der Grundtabelle (Kundentabelle), den kritischen Merkmalsnamen (Umsatz und Treue) sowie dem Term «klein» der linguistischen Variablen Umsatz sowie dem Term «stark» der linguistischen Variablen Treue.

Anlehnung an
SQL

Aufgrund des obigen Beispiels und der gemachten Erläuterungen lassen sich unscharfe Datenbanken wie folgt charakterisieren:

Fuzzy Datenbanksystem

Ein fuzzy Datenbankmanagementsystem (FDBMS) ist ein Datenbanksystem, das die folgenden Eigenschaften aufweist:

Grundkonzepte
unscharfer
Datenbank-
systeme

- Das Datenmodell ist unscharf relational, d.h., es lässt *impräzise, vage und unsichere Merkmalswerte* zu,
- Abhängigkeiten zwischen den Attributen oder Merkmalen werden mit *unscharfen Normalformen* ausgedrückt,
- sowohl der Relationenkalkül wie die Relationenalgebra können mit Hilfe der Fuzzy Logic zum *unscharfen Relationenkalkül* resp. zur *unscharfen Relationenalgebra* ausgebaut werden, und
- mit einer durch linguistische Variablen erweiterten Klassifikationssprache lassen sich *unscharfe Abfragen formulieren*.

Seit Jahren forschen vereinzelt Informatiker auf dem Gebiet der unscharfen Logik und relationaler Datenbanksysteme. Diese Arbeiten werden vorwiegend auf dem Gebiet der Fuzzy Logic und weniger im Bereich der Datenbanken publiziert und entsprechend honoriert. Es ist zu hoffen, dass die beiden Forschungsgebiete sich in Zukunft nähern und die Exponenten der Datenbanktechnologie das Potenzial unscharfer Datenbanken und unscharfer Abfragesprachen erkennen.

Zusammenarbeit
zwischen
Fachbereichen ist
gefordert

6.7 Wissensbasierte Datenbanken

Datenbanken mit Fakten und Regeln

Darstellung von Fakten in Tabellen

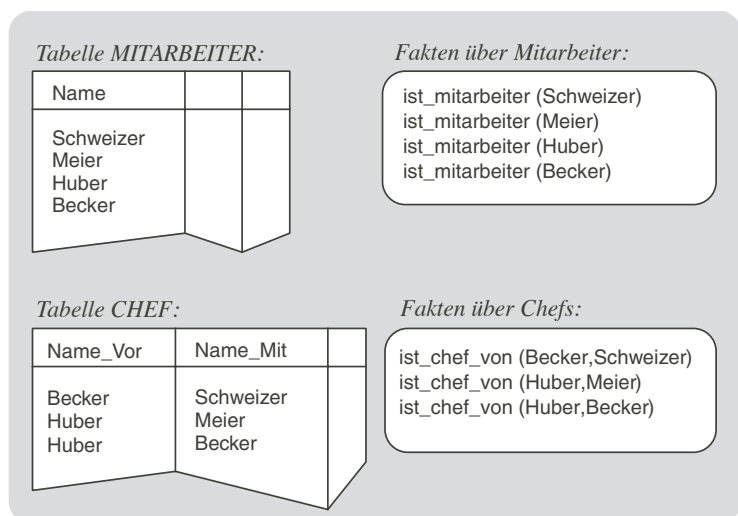
Wissensbasierte Datenbanken (engl. *knowledge* oder *deductive databases*) können nicht nur eigentliche Datenvorkommen – *Fakten* genannt – sondern auch *Regeln* verwalten, mit deren Hilfe neue Tabelleninhalte oder Fakten hergeleitet werden.

Die Tabelle MITARBEITER ist in Abb. 6-13 der Einfachheit halber auf die Namen der Mitarbeitenden reduziert. Zu dieser Tabelle lassen sich nun Fakten oder Aussagen definieren, in diesem Fall über Mitarbeiter. Allgemein bezeichnet man diejenigen Aussagen als Fakten, die *bedingungslos den Wahrheitsgehalt TRUE annehmen*. So ist es Tatsache, dass Herr Huber ein Mitarbeiter ist. Dieser Sachverhalt wird durch den Fakt «ist_mitarbeiter (Huber)» ausgedrückt. Für die direkten Vorgesetzten der Mitarbeiter kann eine neue Tabelle CHEF errichtet werden, die den Namen des direkten Vorgesetzten und des ihm unterstellten Mitarbeiters paarweise pro Tupel aufzeigt. Entsprechend werden Fakten «ist_chef_von (A,B)» formuliert, um die Tatsache «A ist direkter Chef von B» auszudrücken.

Fakten auswerten

Die Vorgesetztenhierarchie spiegelt sich anschaulich im Baum von Abb. 6-14. Auf die Frage, wer der direkte Chef des Mitarbeiters Meier sei, wertet die SQL-Abfrage die Tabelle CHEF aus und stößt auf den Vorgesetzten Huber. Dasselbe Resultat zeigt die Auswertung mit Hilfe einer logischen Programmiersprache (angelehnt an Prolog).

Abb. 6-13
Gegenüberstellung von Tabellen und Fakten



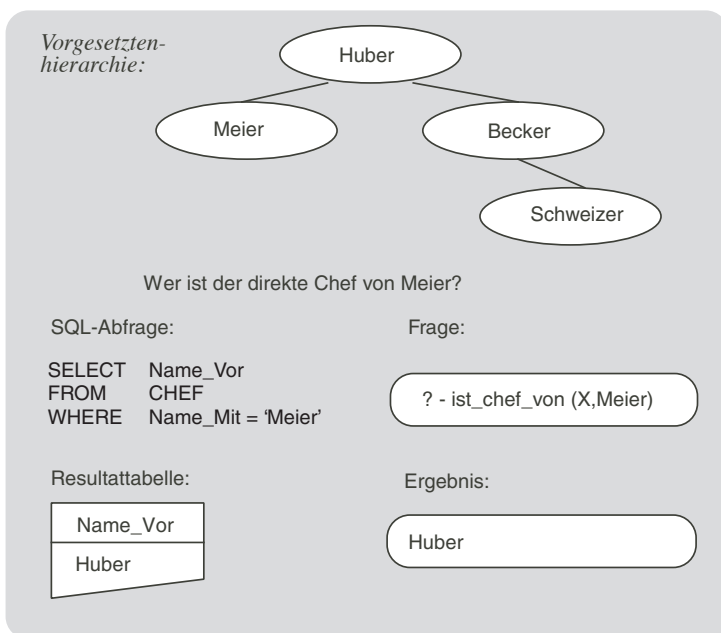


Abb. 6-14
Auswerten von
Tabellen und
Fakten

Neben den eigentlichen Fakten können *Regeln zur Herleitung unbekannter Tabelleninhalte* definiert werden. Beim Relationenmodell spricht man in diesem Fall von abgeleiteten Tabellen (engl. *derived* oder *deduced relation*). Ein einfaches Beispiel einer abgeleiteten Tabelle und einer entsprechenden Ableitungsregel zeigt Abb. 6-15. Daraus geht hervor, wie für jeden Mitarbeiter der übernächste Vorgesetzte herausgefunden werden kann. Dies ist z.B. für Großfirmen oder Firmen mit entfernten Niederlassungen interessant, falls der direkte Vorgesetzte des Mitarbeiters abwesend ist und mit elektronischer Post der nächsthöhere Vorgesetzte angesprochen werden soll.

Die Definition einer abgeleiteten Tabelle entspricht der Festlegung einer Sicht. Im Beispiel dient eine solche Sicht unter dem Namen VORGESETZTE zur Bestimmung des nächsthöheren Chefs für jeden Mitarbeiter, sie entspricht einem Verbund der Tabelle CHEF auf sich selbst. Zu dieser Sicht lässt sich eine Ableitungsregel spezifizieren. Die Regel «ist_vorgesetzter_von (X,Y)» ergibt sich aus der Tatsache, dass es ein Z gibt, so dass X direkter Chef ist von Z und Z wiederum direkter Chef von Y. Damit drücken wir aus, dass X übernächster Vorgesetzter von Y ist, da Z dazwischenliegt.

Eine mit Fakten und Regeln bestückte Datenbank wird unvermittelt zu einer *Methoden- oder Wissensbank*, da sie nicht nur offenkundige Tatsachen wie «Huber ist Mitarbeiter» oder «Huber ist direkter

Wie werden die
Regeln mit dem
Relationen-
modell
nachgebildet?

Abgeleitete
Tabellen durch
Sichten festlegen

Der Schritt zur
Wissensbank

Chef von Meier und Becker» enthält, sondern auch abgeleitete Feststellungen wie z.B. «Huber ist übergeordneter Chef von Schweizer».

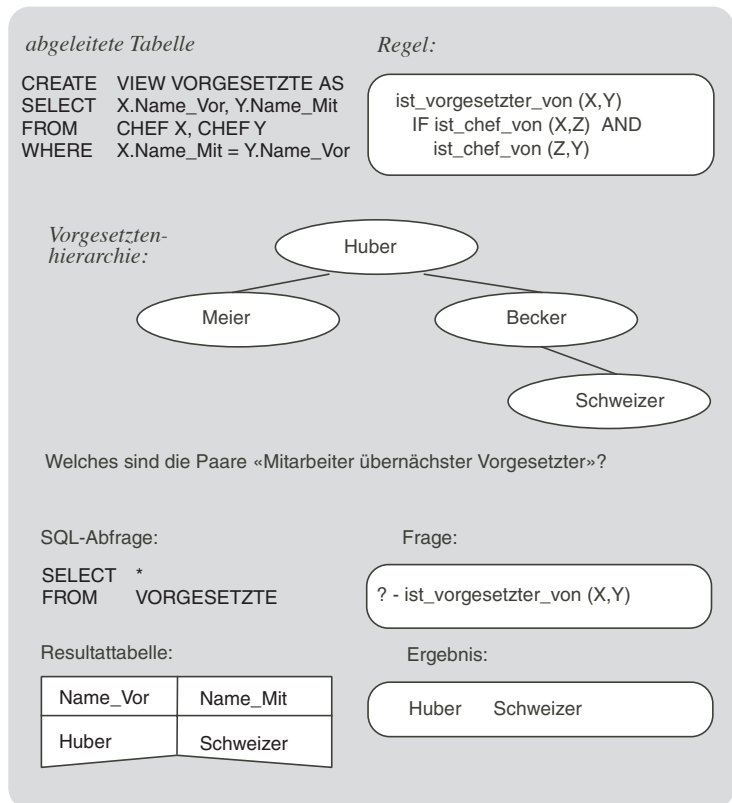
Logische
Auswertung mit
SQL

Die übergeordneten Vorgesetzten aufzufinden, dazu dient die in Abb. 6-15 definierte Sicht VORGESETZTE. Die SQL-Abfrage über diese Sicht liefert mit der Resultattabelle die Tatsache, dass im Beispiel nur eine einzige übergeordnete Vorgesetztenbeziehung existiert, nämlich Mitarbeiter Schweizer und sein übergeordneter Vorgesetzter Huber. Zu dieser Erkenntnis führt auch die Anwendung der entsprechenden Ableitungsregel «ist_vorgesetzter_von».

Deduktive
Datenbanken
ermöglichen
rekursive
Auswertungen

Eine deduktive Datenbank als Gefäß für Fakten und Regeln unterstützt zusätzlich das *Prinzip der Rekursion*, das die Möglichkeit eröffnet, aufgrund der in der deduktiven Datenbank enthaltenen Regeln beliebig viele korrekte Schlussfolgerungen zu ziehen. Aus einer wahrheitsgetreuen Aussage ergeben sich jeweils neue Aussagen.

Abb. 6-15
Herleitung neuer
Erkenntnisse



Das Prinzip der Rekursion kann sich entweder *auf die Objekte der Datenbank oder auf die Ableitungsregeln* beziehen. Unter rekursiv definierten Objekten werden Strukturen verstanden, die selbst wiederum aus Strukturen zusammengesetzt sind und wie die beiden Abstraktionskonzepte Generalisation und Aggregation als hierarchische oder netzwerkartige Objektstrukturen aufgefasst werden können. Überdies lassen sich auch Aussagen rekursiv berechnen, etwa im Beispiel der Vorgesetztenhierarchie aus den Fakten «ist_mitarbeiter_von» und «ist_chef_von» alle direkten und indirekten Vorgesetztenbeziehungen.

Zum Prinzip der
Rekursion

Der Berechnungsvorgang, der aus einer bestimmten Tabelle alle transitiv abhängigen Tupel herleitet, bildet die *transitive Hülle* der Tabelle. Dieser Operator zählt nicht zu den ursprünglichen Operatoren der Relationenalgebra. Die transitive Hülle stellt vielmehr eine natürliche Erweiterung der relationalen Operatoren dar. Dabei kann sie nicht durch eine feste Anzahl von Berechnungsschritten, sondern nur durch eine nach dem jeweiligen Inhalt der Tabelle variierenden Anzahl von relationalen Verbund-, Projektions- und Vereinigungsoperatoren gebildet werden.

Forderung nach
transitiver Hülle

Diese Erläuterungen ergeben zusammengefasst die folgende Definition:

Wissensbasiertes Datenbanksystem

Ein wissensbasiertes Datenbankmanagementsystem (WDBMS) unterstützt deduktive Datenbanken oder Wissensbanken,

- wenn es neben den eigentlichen Daten, also den *Fakten*, auch *Regeln* enthält,
- wenn die *Ableitungskomponente* es erlaubt, aus Fakten und Regeln weitere Fakten herzuleiten und
- wenn es die *Rekursion* unterstützt, mit der unter anderem die transitive Hülle einer Tabelle berechnet werden kann.

Haupt-
komponenten
eines wissens-
basierten
Datenbank-
systems

Unter dem Begriff *Expertensystem* versteht man ein Informationssystem, das für einen abgegrenzten Anwendungsbereich fachspezifische Kenntnisse und Schlussfolgerungen verfügbar macht. Wichtige Komponenten bilden eine Wissensbank mit Fakten und Regeln und eine Ableitungskomponente zur Herleitung neuer Erkenntnisse. Die Fachgebiete Datenbanken, Programmiersprachen und künstliche Intelligenz werden sich mehr und mehr beeinflussen und künftig effiziente Problemlösungsverfahren für die Praxis bereitstellen (vgl. Forschung zu Knowledge Discovery in Databases resp. Data Mining).

Von der
Wissensbank
zum
Expertensystem

6.8

Literatur zur Forschung und Entwicklung

Forschungs- literatur zu verteilten Datenbanken

Ein Standardbuch über verteilte Datenbanksysteme stammt von Ceri und Pelagatti (1985), eine weitere Übersicht über das Gebiet bieten Özsu und Valduriez (1991). Dadam (1996) illustriert verteilte Datenbanken und Client-Server-Systeme. Das deutschsprachige Werk von Rahm (1994) behandelt neben Aspekten verteilter Datenbanksysteme auch Architekturfragen paralleler Mehrrechner. Grundlegende Arbeiten über verteilte Datenbanksysteme basieren auf den Erweiterungen von «System R» nach Williams et al. (1982) und von «Ingres» nach Stonebraker (1986). Die Arbeiten von Rothnie et al. (1980) sind ebenfalls von Interesse, da das entsprechende Datenbanksystem «SDD-1» den ersten Prototypen eines verteilten Systems darstellte.

Literatur zur Zeit- behandlung in Datenbanken

Zur Integration der Zeit in relationale Datenbanken liegen verschiedene Vorschläge vor; erwähnt seien die Arbeiten von Clifford und Warren (1983), Gadia (1988) und Snodgrass (1987). Snodgrass (1994) hat zusammen mit weiteren Forschungskollegen die SQL-Sprache um temporale Konstrukte erweitert; die Sprache ist unter dem Namen TSQL2 bekannt geworden. Aktuelle Forschungsarbeiten über temporale Datenbanken finden sich in Etzion et al. (1998). Myrach (2005) behandelt temporale Aspekte in betrieblichen Informationssystemen.

Arbeiten zu objekt- orientierten und objekt- relationalen Datenbanken

Objektorientierte Ansätze zur Erweiterung relationaler Datenbanken werden von Dittrich (1988), Lorie et al. (1985), Meier (1987) sowie von Schek und Scholl (1986) aufgezeigt. Bücher über objektorientierte Datenbanken stammen von Bertino und Martino (1993), Cattell (1994), Geppert (2002), Heuer (1997), Hughes (1991) und Kim (1990). Lausen und Vossen (1996) beschreiben grundlegende Aspekte objektrelationaler und objektorientierter Datenbanksprachen. Die deutschsprachigen Werke von Kemper und Eickler (2006), Lang und Lockemann (1995) sowie Saake et al. (1997) erklären Entwicklungen relationaler und objektorientierter Datenbanksysteme. Meier und Wüst (2003) haben eine Einführung in das Gebiet objektorientierter und objektrelationaler Datenbanken für den Praktiker verfasst. Stonebraker (1996) erläutert objektrelationale Datenbanksysteme. Die neueste Entwicklung zur Erweiterung des SQL-Standards ist in Türker (2003) zusammengefasst. Die Veröffentlichungen von Coad und Yourdon (1991) sowie von Martin und Odell (1992) enthalten Bausteine für den objektorientierten Datenbankentwurf. Stein (1994) gibt in seinem Fachbuch einen Vergleich verschiedener objektorientierter Analysemethoden.

In den letzten Jahren sind einige Werke über die Thematik Data Warehouse und Data Mining erschienen. Als Standardwerk für Data Warehouse gilt das Buch von Inmon (2002). Kimball et al. (1998) widmen sich ebenfalls dieser Thematik. Im deutschsprachigen Raum sind die Werke von Gluchowski et al. (1997) sowie Mucksch und Behme (1996) bekannt. Jarke et al. (2000) vermitteln Grundlagen über das Data Warehouse und zeigen Forschungsprojekte auf. Über Data Mining haben Berson und Smith (1997) sowie Witten und Frank (1999) Bücher verfasst. Die Integration von Datenbanken im World Wide Web illustrieren verschiedene Autoren in einem Schwerpunkttheft von Meier (2000).

*Standardwerke
zu
Data Warehouse
und Data Mining*

Seit einigen Jahren werden Erkenntnisse aus dem Gebiet der unscharfen Logik (Fuzzy Logic) auf Datenmodellierung und Datenbanken angewendet, siehe z.B. Bordogna und Pasi (2000), Bosc und Kacprzyk (1995), Chen (1998), Petry (1996) oder Pons et al. (2000). Mit Hilfe der Fuzzy Logic wurden verschiedene Modellerweiterungen vorgeschlagen, sowohl für das Entitäten-Beziehungsmodell wie für das Relationenmodell. Beispielsweise hat Chen (1992) in seiner Dissertation die klassischen Normalformen der Datenbanktheorie zu unscharfen erweitert, indem er bei den funktionalen Abhängigkeiten Unschärfe zuließ; siehe dazu auch Shenoï et al. (1992). Weitere Vorschläge zu unscharfen Datenmodellen für Datenbanken finden sich in Kerre und Chen (1995). Takahashi (1995) schlägt eine Fuzzy Query Language (FQL) auf der Basis des Relationenkalküls vor. Die Sprache FQUERY von Kacprzyk und Zadrozny (1995) verwendet unscharfe Terme und ist im Microsoftprodukt Access prototypartig implementiert worden. Ein etwas anderer Ansatz wird mit unscharfer Klassifikation gewählt, ursprünglich von Schindler (1998) vorgeschlagen. Dazu wurde die unscharfe Klassifikationssprache fCQL (fuzzy Classification Query Language) entworfen und in einem Prototypen (vgl. Meier et al. 2001, Meier et al. 2005, Meier und Werro 2006) realisiert.

*Forschungs-
arbeiten zu
unscharfen
Datenbanken*

Die regelbasierte Sprache bei deduktiven Datenbanksystemen wird oft als Datalog bezeichnet, in Anlehnung an den Begriff Data und die bekannte Logikprogrammiersprache Prolog. Ein Klassiker unter den Prolog-Lehrbüchern ist das Werk von Clocksin und Mellish (1994), eine formale Abhandlung der logischen Datenbankprogrammierung geben Maier und Warren (1988). Das deutschsprachige Werk von Cremers et al. (1994) behandelt auf ausführliche Weise das Gebiet der deduktiven Datenbanken. Die Werke von Gallaire et al. (1984) sowie von Gardarin und Valdriez (1989) befassen sich zu einem großen Teil mit deduktiven Datenbanken.

*Fachbücher zu
deduktiven
Datenbanken*

Repetitorium

1. a) *Was versteht man unter dem Begriff Datenmanagement?*

Das Datenmanagement umfasst alle betrieblichen, organisatorischen und technischen Aufgaben und Funktionen der Datenarchitektur, der Datenadministration und der Datenbanktechnik, die der unternehmensweiten Datenhaltung und Datennutzung dienen.

Abschnitt 1.5

b) *Welche spezifischen Berufsbilder existieren fürs Datenmanagement? (drei Nennungen)*

Datenarchitekten sind für den Unterhalt der unternehmensweiten Datenarchitektur sowie für den logischen Entwurf von Datenbanken zuständig. Datenadministratoren verwalten mit Hilfe eines Data-Dictionary-Systems die für das Unternehmen gültigen Beschreibungen von Tabellen und Merkmalen. Die Datenbankspezialisten legen den physischen Datenbankentwurf fest, installieren die Datenbanken und definieren die Abläufe für die Archivierung und Wiederherstellung von Datenbanken nach einem Fehlerfall.

Abschnitte 1.5, 2.6, 3.1, 3.7, 4.5

2. a) *Was versteht man unter einem relationalen Datenbankmanagementsystem (RDBMS)?*

Ein RDBMS besteht aus einer «relationalen» Speicherungs- und Verwaltungskomponente: Die Speicherkomponente dient dazu, sowohl Daten als auch Beziehungen zwischen den Daten in Tabellen abzulegen. Neben Tabellen mit Benutzerdaten existieren vordefinierte Systemtabellen, die Beschreibungsinformationen enthalten. Die Verwaltungskomponente enthält als wichtigsten Bestandteil eine relationale Datendefinitions- und -manipulationssprache (z.B. die standardisierte Sprache SQL). Daneben umfasst diese Sprache auch Dienstfunktionen zur Gewährung der Datenintegri-

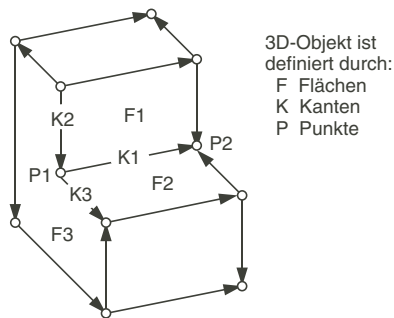
tät, des Datenschutzes und der Datensicherung.
Abschnitte 1.3, 1.4

- b) *Was versteht man unter Datenunabhängigkeit, und welche Komponente des RDBMS unterstützt diese Eigenschaft?*

Bei Datenbanksystemen spricht man von Datenunabhängigkeit, falls die Daten von den Anwendungsprogrammen mittels Systemfunktionen voneinander getrennt bleiben. Dadurch kann man gewisse Änderungen in den Datenbanken vornehmen, ohne Programme anpassen zu müssen. Die Verwaltungskomponente des RDBMS ermöglicht Datenunabhängigkeit.

Abschnitt 1.4

3. a) *Erstellen Sie eine Liste der relevanten Informationssachverhalte für dreidimensionale, flächenbegrenzte Objekte (Polyeder)!**



Datenanalyse für flächenbegrenzte 3D-Objekte:

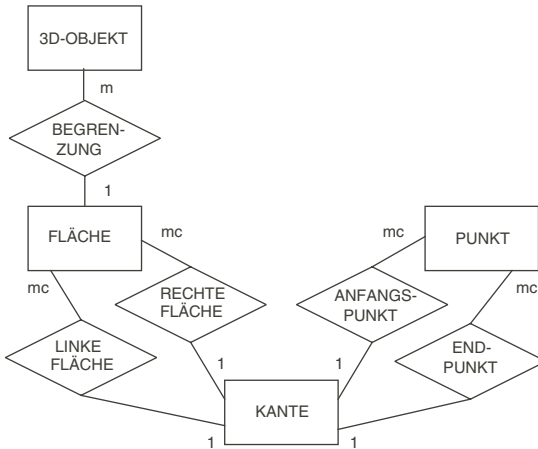
- Ein 3D-Objekt besteht aus mehreren Flächen, jede Fläche gehört zu genau einem 3D-Objekt.
- Eine Fläche ist durch mehrere gerichtete Kanten definiert. Jede Kante zeigt auf *genau zwei* angrenzende Flächen (Beispiel: Die Kante K1 hat als linke angrenzende Fläche F1 und als rechte F2).
- Jede Kante hat genau zwei Grenzpunkte, nämlich einen als Startpunkt und einen als Endpunkt (Beispiel: Der Startpunkt von K1 ist P1, der Endpunkt ist P2). In jedem Grenzpunkt können mehrere eingehende oder ausgehende Kanten münden.

Abschnitt 2.1

- b) Wie sieht das Entitäten-Beziehungsmodell für flächenbegrenzte 3D-Objekte aus? (Annahme: 3D-Objekte ohne Löcher)

Die Beziehungen zwischen Flächen, Kanten und Punkten werden durch zwei duale Beziehungsmengen ausgedrückt. Die Entitätsmenge PUNKT enthält die kartesischen Koordinaten.

Abschnitt 2.2

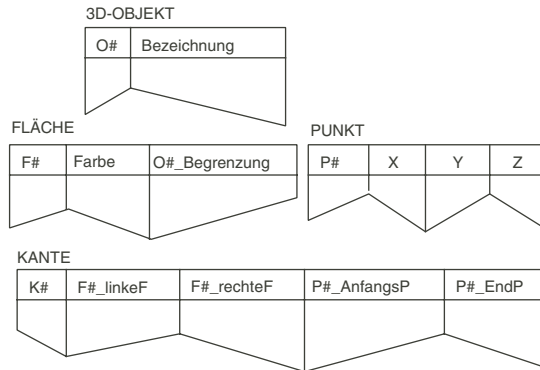


4. a) Überführen Sie das Entitäten-Beziehungsmodell für flächenbegrenzte 3D-Objekte (vgl. Aufgabe 3b) in ein relationales Datenbankschema. Welche Überführungsregeln kommen sinnvollerweise zum Zuge?

Werden die vier Entitätsmengen und die fünf Beziehungsmengen eins zu eins (mit Regel 1 und 2) in Tabellen überführt, so resultieren insgesamt 9 Tabellen. Ein solches Datenbankschema kann optimiert werden, indem die Regel 4 für einfachkomplexe Beziehungen angewendet wird. Es entsteht ein Datenbankschema mit 4 Tabellen: Die Beziehungsmenge BEGRENZUNG wird durch die Tabelle FLÄCHE ausgedrückt, indem ein Fremdschlüsselverweis O#_Begrenzung in die Tabelle aufgenommen wird. Analog werden die zwei dualen Beziehungsmengen der angrenzenden Flächen und Punkte in die Tabelle KANTE integriert, nämlich durch die Fremdschlüsselverweise F#_linkeFläche und F#_rechteFläche resp. P#_AnfangsPunkt und P#_EndPunkt.

Abschnitt 2.2

- b) *Wie sieht das optimierte relationale Datenbankschema für flächenbegrenzte 3D-Objekte aus?*



Abschnitt 2.2

5. a) *Was versteht man unter einer relational vollständigen Sprache?*

Eine relationale Abfragesprache heißt vollständig im Sinne der Relationenalgebra, wenn sie mindestens die mengenorientierten Operatoren Vereinigung, Subtraktion und kartesisches Produkt sowie die relationenorientierten Operatoren Projektion und Selektion unterstützt.

Abschnitte 3.2, 3.3

Bemerkung: Die Operatoren Durchschnitt, Verbund (Join) und Division sind fakultativ, da sie auf die obigen Operatoren zurückgeführt werden können.

- b) *Um welche Sprachelemente muss eine relational vollständige Sprache ergänzt werden, damit sie für die Praxis tauglich ist? (mindestens drei Nennungen)*

Es sollten Sprachkonstrukte für die Datendefinition (Datendefinitionssprache), für die Datenmanipulation (Datenmanipulationssprache), für die Vergabe von Benutzungsrechten, für Kontrollfunktionen (z.B. Recovery/Restart) und für die referenzielle Integrität vorhanden sein.

Abschnitt 3.3

6. a) *In der Datenbank für ebenbegrenzte Polyeder (vgl. relationales Datenbankschema in Aufgabe 4b) sollen alle Bezeichnungen der Objekte zusammengestellt werden, die rote Flächen aufweisen.*

```

SELECT Bezeichnung
FROM 3D-OBJEKT, FLÄCHE
WHERE O# = O#_Begrenzung AND Farbe = 'rot'
Abschnitte 3.2, 3.4

```

- b) Welche Flächen des ebenbegrenzten Objektes aus Aufgabe 3a) stoßen im Punkt P1 zusammen?

```

SELECT F#_linkeF
FROM KANTE
WHERE P#_AnfangsP = 'P1'
UNION
SELECT F#_rechteF
FROM KANTE
WHERE P#_EndP = 'P1'

```

Lösungsidee: Es werden alle linken Flächen ausgehender Kanten von P1 berechnet (nämlich F1 und F2) und mit den rechten Flächen eingehender Kanten von P1 kombiniert (F3).

Abschnitte 3.2, 3.4

7. a) Was versteht man unter referenzieller Integrität?

Eine relationale Datenbank erfüllt die Regel der referenziellen Integrität, wenn jeder Fremdschlüsselwert als Wert beim zugehörigen Primärschlüssel (der referenzierten Tabelle) vorkommt.

Abschnitt 2.5

- b) Geben Sie eine Einfügeoperation *INSERT* (in *SQL*-Notation) für die Tabelle *STADT* an, die die referenzielle Integrität verletzt.

LAND

| Name | Hauptstadt | Code |
|-------------|------------|------|
| Schweiz | Bern | CH |
| Deutschland | Berlin | D |
| Italien | Rom | I |

STADT

| Bezeichnung | Einwohner | Staat |
|-------------|-----------|-------|
| Bern | 35000 | CH |
| Berlin | 1860000 | D |
| Florenz | 40000 | I |
| Basel | 60000 | CH |
| München | 1290000 | D |
| Zürich | 900000 | CH |
| Rom | 1140000 | I |

Antwort:

```
INSERT INTO STADT
VALUES ('Paris', 1400000, 'F')
```

Bemerkung: Der Wert «F» (für Frankreich) existiert (noch) nicht in der zugehörigen Tabelle LAND.

Abschnitte 2.5, 3.4

8. a) *Gesucht ist eine Liste aller Städte des Landes Italien (vgl. Aufgabe 7). Wie lautet die Abfrage in SQL?*

```
SELECT Bezeichnung
FROM LAND, STADT
WHERE Code=Staat AND Name='Italien'
```

Bemerkung: Die Abfrage ergibt eine Tabelle mit den Städten Florenz und Rom.

Abschnitt 3.4

- b) *Wie lautet die Abfrage mit Hilfe von Operatoren der Relationalalgebra (algebraischer Ausdruck)?*

Die Resultattabelle kann durch den algebraischen Ausdruck

$\pi_{\text{Bezeichnung}}(\sigma_{\text{Name}=\text{Italien}}(\text{LAND} \bowtie_{\text{Code}=\text{Staat}} \text{STADT}))$ berechnet werden.

Abschnitt 3.2

9. a) *Gesucht ist die Abfrage in SQL-Notation, die aus den Beispieltabellen LAND und STADT die folgende Resultattabelle erzeugt:*

| Bezeichnung | Einwohner | Staat |
|-------------|-----------|-------|
| Berlin | 1860000 | D |
| München | 1290000 | D |
| Rom | 1140000 | I |

Antwort:

```
SELECT *
FROM STADT
WHERE Einwohner > 1000000
```

oder

```
SELECT Bezeichnung, Einwohner, Staat
FROM STADT
WHERE Einwohner > 1000000
```

Abschnitt 3.4

- b) *Gesucht ist dieselbe Resultattabelle wie in a), ausgedrückt mit der Relationenalgebra.*

Die Resultattabelle kann durch den algebraischen Ausdruck $\sigma_{\text{Einwohner} > 1000000}(\text{STADT})$ berechnet werden.
Abschnitt 3.2

10. a) *Es sollen die Namen aller Länder und Städte ermittelt und in einer einzigen Tabelle dargestellt werden. Wann ist diese Abfrage zulässig?*

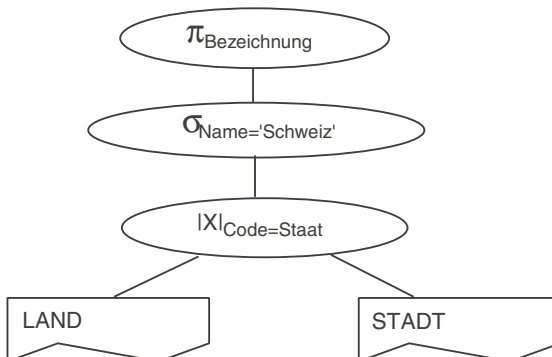
Die Tabellen $\pi_{\text{Name}}(\text{LAND})$ und $\pi_{\text{Bezeichnung}}(\text{STADT})$ müssen vereinigungsverträglich sein, d.h. dieselbe Dimension aufweisen und über dieselben Wertebereiche definiert sein.
Abschnitt 3.2

- b) *Wie sieht die Abfrage in SQL aus?*

```
SELECT Name
FROM LAND
UNION
SELECT Bezeichnung
FROM STADT
```

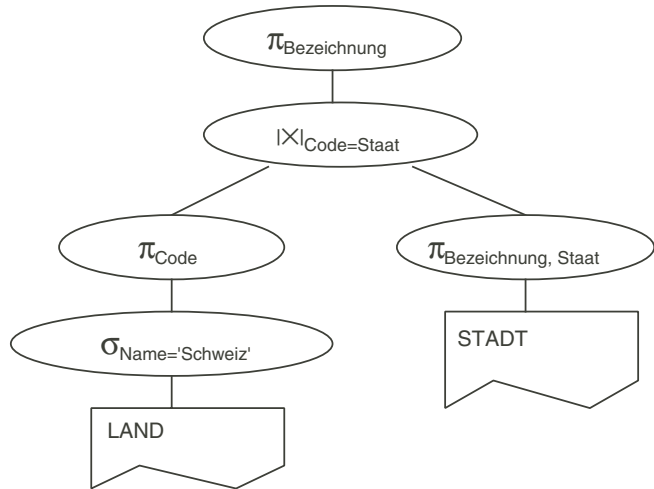
Bemerkung: Wir nehmen hier an, dass die Namen der Länder und die Bezeichnungen der Städte über denselben Wertebereich definiert sind. In diesem Fall ist die Vereinigung (engl. UNION) zulässig.
Abschnitt 3.4

11. a) *Gegeben ist der folgende algebraische Ausdruck:
 $\pi_{\text{Bezeichnung}}(\sigma_{\text{Name}='Schweiz'}(\text{LAND} \bowtie_{\text{Code}=\text{Staat}} \text{STADT}))$
Wie sieht der zugehörige Anfragebaum aus?*



Abschnitt 4.2

b) *Wie sieht der optimierte Anfragebaum aus?*



Abschnitt 4.2

12.a) *Weshalb ist es sinnvoll, bei der Architektur von relationalen Datenbanksystemen ein Schichtenmodell zu verwenden?*

Bei der Systemarchitektur von Datenbanksystemen gilt als unabdingbares Prinzip, künftige Veränderungen oder Erweiterungen lokal begrenzen zu können. Wie beim Implementieren von Betriebssystemen oder anderen Softwarekomponenten führt man auch bei relationalen Datenbanksystemen voneinander unabhängige Systemebenen ein, die miteinander über vordefinierte Schnittstellen kommunizieren.

Abschnitt 4.6

b) *Welche wichtigen Aufgaben gehören zur obersten Schicht, d.h. zur mengenorientierten Schnittstelle? (mindestens drei Nennungen)*

Die oberste Schicht dient der Anfrageübersetzung und Anfrageoptimierung. Es werden folgende Aufgaben durchgeführt: Anfrageübersetzung und Namensauflösung, Zugriffskontrolle und Autorisierung, algebraische Optimierungen, Prüfen von Zugriffspfaden, Integritätskontrolle sowie eventuell Codegenerierung.

Abschnitt 4.6

13.a) *Was versteht man unter XML-Datenbanken und wozu sind solche nützlich?*

Eine XML-Datenbank enthält Hyperdokumente (XML-Dokumente) und ist über ein XML-Schema definiert. Damit lassen sich semistrukturierte Daten wie Text, Bilder, Grafiken etc. beschreiben und verwalten. XML-Datenbanken werden u.a. für webbasierte Anwendungen bei Unternehmen (E-Commerce) und in der Verwaltung (E-Government) benötigt.

Abschnitte 5.1, 5.2

- b) *Wie lassen sich bereits getätigte Investitionen beim Wechsel eines Datenbanksystems schützen?*

Es bestehen unterschiedliche Migrationsvarianten, wie rechnergestützte Konversion von Datenbanken und Anwendungssystemen, Transformation der Sprachschnittstelle oder zeitlich befristete Koexistenz mit parallelem Nachführen (synchron oder asynchron) der Datenbestände.

Abschnitte 5.3, 5.4, 5.5

- 14.a) *Wo liegen die heutigen Schwachstellen beim Verwenden relationaler Datenbanken? Nennen Sie mindestens drei Kritikpunkte.*

In der Büroautomation, beim rechnergestützten Entwurf und bei der Konstruktion von Schaltelementen oder Bauteilen, bei webbasierten Anwendungen mit Hyperdokumenten sowie bei geografischen Informationssystemen treten Mängel auf, die auf folgende Tatsachen zurückzuführen sind:

Objektbeschreibung: Die Daten sind unterschiedlich stark strukturiert, neben formatierten und strukturierten Daten existieren auch semistrukturierte Daten wie Texte, Bilder und Grafiken. Neben den vom Datenbanksystem vorgegebenen Datentypen sollten benutzerdefinierte Datentypen zugelassen werden.

Abschnitt 6.4

Transaktionenverwaltung: Eine Transaktion gilt als Einheit für Konsistenz und Recovery. Da im Ingenieurbereich normalerweise während Tagen oder Wochen an einem Entwurf gearbeitet wird, können solche Transaktionen im Fehlerfall nicht vollständig zurückgesetzt werden. Mit anderen Worten, langandauernde Transaktionen sollten künftig erlaubt sein.

Abschnitte 4.3, 4.5

Archivierung: Das Datenbanksystem sollte eine Versionskontrolle anbieten. Zudem sollten Zeitaspekte durch das Datenbanksystem selbst verwaltet werden können (z.B. Zeit- und Messreihen), um Abfragen in die Vergangenheit und in die Zukunft richten zu können.

Abschnitt 6.3

- b) *Welche Erweiterungen oder Verbesserungen sind zu erwarten?*

Künftig werden Erweiterungen relationaler Datenbanken in der Praxis Fuß fassen oder es werden neuartige Datenbankgenerationen auf dem Markt erscheinen. Postrelationale Datenbanksysteme unterstützen verteilte, temporale, objektrelationale, multidimensionale, unscharfe und/oder deduktive Datenbanken softwaremäßig.

Abschnitte 6.2 bis 6.7

Tutorium in SQL

Am IBM Research Lab in San José in Kalifornien wurde Anfang der siebziger Jahre die Sprache SEQUEL definiert, um über eine «Structured English QUERy Language» für relationale Datenbanken zu verfügen. Die Sprache sollte einfach und verständlich sein und gleichzeitig die Forderung nach relationaler Vollständigkeit erfüllen. Anstelle von Operatoren aus der Relationenalgebra (Projektion, Selektion, Verbund u.a.) wurde die einfache Grundform SELECT FROM WHERE vorgeschlagen. Aus Tabellen (FROM) sollen demnach Merkmalswerte ausgewählt werden (SELECT), die eine vom Anwender definierte Selektionsbedingung (WHERE) erfüllen.

SEQUEL wurde aus juristischen Gründen auf Structured Query Language oder SQL umgetauft und international standardisiert (vgl. Abschnitt 1.3). Abfragen jeglicher Art basieren auf der folgenden formalen Beschreibung (Syntax):

```
SELECT      merkmalsname [,merkmalsname ... ]  
FROM        tabellenname [,tabellenname ... ]  
[WHERE      selektionsbedingung] ;
```

Von SEQUEL zu
SQL

Grundstruktur
von SQL

Die Schlüsselwörter SELECT, FROM und WHERE der Sprache SQL sind hier in Großbuchstaben gegeben. Bei der Verwendung von SQL können die einzelnen Buchstaben dieser Schlüsselwörter beliebig in Groß- oder Kleinschrift erscheinen; zwingend ist hingegen die buchstabengenaue Formulierung der Schlüsselwörter. Im folgenden Tutorium werden die Schlüsselwörter ausschließlich groß geschrieben, damit die Grammatik von SQL besser zur Geltung kommt.

Alle Angaben, die bei der Syntaxdefinition in eckigen Klammern [...] stehen, können weggelassen werden. Bei einer Grammatik einer formalen Sprache hat man sich geeinigt, fakultative Angaben durch eckige Klammern anzugeben.

Die kursiv geschriebenen Ausdrücke wie *merkmalsname*, *tabellenname* oder *selektionsbedingung* müssen vom Anwender spezifiziert werden. Er gibt an, welche Merkmale ihn bei einer Abfrage interessieren, aus welchen Tabellen diese Merkmalswerte stammen und welche Einschränkungen (*selektionsbedingung*) gelten.

Einfache Abfragen

Die Selektion Möchte man alle Mitarbeitenden aus der Tabelle MITARBEITER aus Abb. 3-7 auflisten, so geschieht das mit SQL wie folgt:

```
SELECT      M#, Name, Straße, Ort, Unt      (1)
FROM        MITARBEITER;
```

Zum Sternsymbol Diese SQL-Abfrage zeigt alle Tupel aus der Mitarbeitertabelle mit ihren entsprechenden Merkmalswerten. Da solche Abfragen oft vorkommen, kann man in der SELECT-Klausel anstelle der Angabe sämtlicher Attribute einer Tabelle das Zeichen * verwenden:

```
SELECT      *      (2)
FROM        MITARBEITER;
```

Diese verkürzte Form einer Abfrage ergibt dieselbe Resultatstabelle wie die Anweisung (1). Die Ausdrücke (1) und (2) sind also äquivalent.

Die Verwendung des Sternsymbols in der SELECT-Klausel verlangt, die ursprüngliche Definition der SQL-Syntax wie folgt zu erweitern:

```
SELECT      { * | merkmalsname [,merkmalsname ... ] }
FROM        tabellenname [,tabellenname ... ]
[WHERE      selektionsbedingung] ;
```

Nun steht in der SELECT-Klausel ein Ausdruck in geschweiften Klammern. Die Teilausdrücke in der geschweiften Klammer sind durch das Trennsymbol « | » separiert, was eine Auswahl vom SQL-Anwender verlangt: Entweder spezifiziert er in der SELECT-Klausel die gewünschten Merkmale durch das Sternsymbol oder er nennt die einzelnen Merkmale beim Namen.

Projektionen

Projektionen mit SQL Interessiert man sich bei den Mitarbeitenden lediglich um deren Mitarbeiternummer, dem Namen und Ort, so ergibt sich in SQL die folgende Abfrage:

```
SELECT      M#, Name, Ort      (3)
FROM        MITARBEITER;
```

Soll eine Liste der Wohnorte der Mitarbeitenden zusammengestellt werden, genügt folgende Anweisung:

```
SELECT      Ort      (4)
FROM        MITARBEITER;
```

Als Resultatstabelle erhält man eine einspaltige Tabelle mit den Ortschaften Frenkendorf, Liestal, Basel und Liestal. Da SQL im Gegen-

satz zur Relationenalgebra Duplikate nicht eliminiert, muss bei der SELECT-Klausel das Wort DISTINCT eingefügt werden:

```
SELECT      DISTINCT Ort                               (5)
FROM        MITARBEITER;
```

Die Abfrage (5) ergibt die gewünschte Tabelle mit den Ortschaften Frenkendorf, Liestal und Basel. Der Ort Liestal erscheint in der Resultatstabelle nur einmal, da Mehrfachnennungen durch DISTINCT eliminiert wurden. Korrekterweise muss hier angefügt werden, dass die Resultatstabelle der Abfrage (4) gar keine Tabelle im Sinne des Relationenmodells darstellt, da jede Relation per Definition eine Menge ist und die in Abschnitt 1.2 genannten Anforderungen erfüllen muss.

*DISTINCT
eliminiert
Duplikate*

Mit der Einführung des neuen Schlüsselwortes DISTINCT muss die Syntax von SQL erweitert werden:

```
SELECT      [DISTINCT] { * | merkmalsname [,merk-
                                     malsname ... ] }
FROM        tabellenname [,tabellenname ... ]
[WHERE      selektionsbedingung] ;
```

Projektionen können mit Selektionen kombiniert werden, worauf im nächsten Absatz näher eingegangen wird.

Qualifizierte Abfragen

Mit der Hilfe von Selektionsbedingungen lassen sich qualifizierte Abfragen durchführen. Interessiert man sich für die Mitarbeitenden, die in Liestal wohnen und in der Abteilung A6 arbeiten, so lautet die entsprechende Abfrage in SQL:

```
SELECT      *                                           (6)
FROM        MITARBEITER
WHERE       Ort = 'Liestal' AND Unt = 'A6';
```

Wie wir wissen, erhalten wir aus unserer Mitarbeitertabelle den Mitarbeiter Becker mit seinen Merkmalen (vgl. Abb. 3-8).

Selektionsbedingungen können die logischen Operationen AND (logisches Und), OR (logisches Oder) und NOT enthalten. Falls notwendig, müssen die Teilaussagen durch Klammern kenntlich gemacht werden, da ein logischer Ausdruck immer von links nach rechts ausgewertet wird. Zudem gilt die Regel, dass NOT vor AND und OR, und dass AND vor OR ausgewertet werden.

*Abfragen mit
Selektions-
prädikaten*

Möchte man in der Abfrage (6) lediglich die Mitarbeiternummer und den Namen aufgelistet haben, so kann die Selektion (6) mit einer Projektion zur folgenden Abfrage umformuliert werden:

| | | |
|--------|---------------------------------|-----|
| SELECT | M#, Name | (7) |
| FROM | MITARBEITER | |
| WHERE | Ort = 'Liestal' AND Unt = 'A6'; | |

*Kombination von
Projektion und
Selektion*

Der Abfrage (7) sieht man nicht direkt an, dass es sich hier um die Kombination einer Selektion mit einer Projektion handelt. Erst beim genaueren Hinsehen fällt auf, dass in der SELECT-Klausel nur eine Teilmenge der Merkmale aus MITARBEITER steht (Projektion) und in der WHERE-Klausel zusätzlich eine Einschränkung spezifiziert ist (Selektion).

Interessiert man sich für Mitarbeitende aus der Tabelle PERSONAL (vgl. Abb. 3-12), die mehr als 50000 Schweizer Franken verdienen im Jahr, genügt folgende Selektion:

| | | |
|--------|---------------|-----|
| SELECT | * | (8) |
| FROM | PERSONAL | |
| WHERE | Lohn > 50000; | |

*Zulässige
Vergleichs-
operationen*

Als Resultat erhält man die Tupel von Schweizer, Huber und Becker.

Verschiedene Vergleichsoperationen sind bei den Selektionsbedingungen zugelassen. Die bekannten Symbole sind «=» für Gleichheit, «<» für kleiner als, «>» für größer als, «<=» für kleiner oder gleich als, «>=» für größer oder gleich als und «<>» für ungleich.

*Einfache
Berechnungen*

Möchte man anstelle des Jahreslohns das monatliche Gehalt berechnen, kann man direkt in der SELECT-Klausel die entsprechende Berechnung vornehmen:

| | | |
|--------|-------------------------------------|-----|
| SELECT | M#, Name, Lohn / 12 AS Monatsgehalt | (9) |
| FROM | PERSONAL; | |

Die Resultattabelle hat nun die Spaltenüberschriften M#, Name und Monatsgehalt, da das Merkmal Lohn mit dem Schlüsselwort AS auf Monatsgehalt umbenannt wurde. Gleichzeitig wird die Berechnung des Monatsgehalts als Division des Jahreslohns durch 12 vorgenommen.

Die Syntax von SQL müsste nun in der SELECT-Klausel wiederum angepasst werden, indem man neben *merkmalsname* auch *merkmalsberechnung* (z.B. für Addition, Subtraktion, Multiplikation und Division) zulässt resp. Attribute umnennen kann (durch [AS *neuermerkmalsname*]).

Abfragen mit Zeichenerkennung

Der SQL-Standard verwendet zwei Symbole für das Erkennen von Zeichen (pattern matching):

- Das %-Zeichen steht für eine beliebige Zeichenkette von einem oder mehreren Buchstaben. Mit dem Schlüsselwort LIKE kann

in der WHERE-Klausel nach unterschiedlichen Wörtern oder Teilwörtern gesucht werden.

- Das Underscore-Zeichen «_» stellt einen einzelnen Buchstabenkandidaten dar. Es kann ebenfalls mit dem LIKE-Schlüsselwort für die Suche von Textstrings verwendet werden.

Als Beispiel für die beiden Suchoptionen «%» und «_» soll das Merkmal Ort aus der Tabelle MITARBEITER dienen, um bestimmte Ortschaften gezielt suchen zu können. Interessiert man sich für die Mitarbeitenden, die in einer Ortschaft mit dem Anfangsbuchstaben B wohnen, so hilft das LIKE-Schlüsselwort mit dem %-Zeichen:

Suchen nach Zeichenketten

```
SELECT      *                                (10)
FROM        MITARBEITER
WHERE       Ort LIKE 'B%';
```

Die Abfrage ergibt als Resultat ein einziges Tupel, nämlich den Mitarbeitenden Huber, der in Basel wohnt. Würde man nach Ortschaften mit LIKE '%a%' suchen, würde man alle Wohnorte finden, die den Buchstaben «a» enthalten. In der Abfrage (10) würde man damit die Mitarbeiter Meier, Huber und Becker erhalten, da diese in Basel resp. Liestal wohnen.

Möchte man alle Ortschaften in der Tabelle MITARBEITER suchen, die fünf Buchstaben haben und mit B beginnen, so dient das Underscore-Symbol:

```
SELECT      *                                (11)
FROM        MITARBEITER
WHERE       Ort LIKE 'B_ _ _ _';
```

Mit der Kombination NOT LIKE können in der WHERE-Klausel auch Wortkombinationen ausgeschlossen werden, um Resultatstupel zu erhalten.

Abfragen mit Aggregationsfunktionen

Wie bei der Einführung von SQL in Abschnitt 3.4.1 erwähnt, erlaubt die Sprache die folgenden Aggregationsfunktionen:

- COUNT(*merkmalsname*) zählt die Anzahl der Werte in der Spalte
- SUM(*merkmalsname*) gibt die Summe der Spaltenwerte zurück
- AVG(*merkmalsname*) berechnet den Durchschnitt der Spaltenwerte
- MIN(*merkmalsname*) evaluiert den kleinsten Wert innerhalb der Spalte

Berechnungsvarianten für Spalten

- **MAX(*merkmalsname*)** nennt den größten Wert in der Spalte

Soll die Anzahl der Mitarbeitenden berechnet werden, geschieht dies durch:

```
SELECT      COUNT(M#)
FROM        MITARBEITER;           (12)
```

Die Resultatstabelle der Abfrage (9) ist degeneriert zu einer einspaltigen Tabelle mit einem einzigen Wert; im aktuellen Beispiel der Tabelle MITARBEITER ergibt die Zählfunktion den Wert 4 für die Mitarbeiter Schweizer, Meier, Huber und Becker.

Muss für die Budgetierung die gesamte Lohnsumme des Personalbestands bekannt sein, kann die Aggregationsfunktion SUM verwendet werden:

```
SELECT      SUM(Lohn) AS Lohnsumme
FROM        MITARBEITER;           (13)
```

Auch hier ergibt sich eine degenerierte Tabelle mit dem Merkmal Lohnsumme und dem einzigen Wert von 270000 Schweizer Franken, als Summe der Jahresgehälter der Mitarbeitenden.

Zusammensetzen von Tabelleninhalten (Join)

*Mehrfacher
Verbund*

Der Verbundsoperator wurde in Abschnitt 3.2.3 behandelt, die Realisierung in SQL in Abschnitt 3.4.1 gegeben. Ein Verbund oder Join kann mehr als zwei Tabellen betreffen. Allerdings müssen dann die entsprechenden Verbundsprädikate in der WHERE-Klausel korrekt spezifiziert sein. Als Beispiel eines mehrfachen Verbundes soll die Tabelle EINKAUF aus den drei Tabellen KÄUFER, WEIN und PRÄFERENZ zurückgewonnen werden (vgl. Abschnitt 2.4.5 resp. Abb. 2-22):

```
SELECT      *
FROM        KÄUFER, WEIN, PRÄFERENZ
WHERE       KÄUFER.Weinsorte = WEIN.Weinsorte AND
           WEIN.Jahrgang = PRÄFERENZ.Jahrgang;           (14)
```

Das Resultat dieses Verbunds gibt exakt die Tabelle EINKAUF mit den vier Tupeln der Käufer Schweizer, Meier, Huber und Becker. Die FROM-Klausel in der Abfrage (14) listet drei Tabellen, die in der WHERE-Klausel durch zwei Verbundsprädikate kombiniert werden. Das erste Verbundsprädikat verbindet die Tabelle KÄUFER mit der Tabelle WEIN über das gemeinsame Merkmal Weinsorte, das zweite die Tabelle WEIN mit PRÄFERENZ über den Jahrgang. Würden die Verbundsprädikate weggelassen, so erhielte man eine fehlerhafte Einkaufsliste (kartesisches Produkt), bei der jeder Mitarbeitende alle Weine und Jahrgänge beanspruchen würde.

Für die Formulierung von komplizierten SQL-Abfragen drängt sich die Verwendung von Ersatznamen (Alias) auf. So kann die Abfrage (14) vereinfacht wie folgt formuliert werden:

*Verwendung von
Ersatznamen
(Alias)*

```
SELECT      *                               (15)
FROM        KÄUFER k, WEIN w, PRÄFERENZ p
WHERE       k.Weinsorte = w.Weinsorte AND
            w.Jahrgang = p.Jahrgang;
```

Hier wird das Alias als Ersatz für den Tabellennamen verwendet. Dies ist vor allem dann wichtig, wenn ein und dieselbe Tabelle in der WHERE-Klausel mehrfach angesprochen werden soll. Ein Verbund einer Tabelle mit sich selbst ist zwar ein seltener Fall, doch muss dann auf die Verwendung von Ersatznamen zurückgegriffen werden.

Sortieren und Gruppieren von Tupelinträgen

Gemäß der Definition einer Tabelle ist die Reihenfolge der Tupel innerhalb einer Tabelle irrelevant (vgl. Abschnitt 1.2). Bei der Auswertung von Tabellen hingegen ist man oft interessiert, die Tupelinträge oder Zeilen in einer bestimmten Reihenfolge aufgelistet zu erhalten. Dies ermöglicht das Schlüsselwort ORDER BY mit den möglichen Zusätzen ASC (ascending oder aufsteigend) resp. DESC (descending oder absteigend):

*Sortieren mit
ORDER BY*

```
SELECT      *                               (16)
FROM        MITARBEITER
ORDER BY    Name ASC;
```

Die Anweisung (16) erstellt eine Mitarbeitertabelle, wobei die Tupelinträge nach aufsteigenden Namen sortiert sind. Somit erscheinen zuerst die Angaben von Becker, dann von Huber, Meier und Schweizer.

Manchmal ist es wichtig, innerhalb einer Resultatstabelle einzelne Teilgruppen zu bilden. Beispielsweise könnte man sich bei der Tabelle PERSONAL aus Abb. 3-12 für die Lohnsummen einzelner Abteilungen interessieren. Eine solche Gruppenbildung wird durch die Schlüsselworte GROUP BY ermöglicht:

*Gruppenbildung
mit GROUP BY*

```
SELECT      Unt, COUNT(M#) AS Anzahl,      (17)
            SUM(Lohn) AS Summe
FROM        PERSONAL
GROUP BY    Unt
ORDER BY    Unt;
```

Die Abfrage (17) ergibt eine Liste der Abteilungen mit der Nennung der jeweiligen Anzahl Mitarbeitenden sowie der Lohnsumme pro

Abteilung; sie ist sortiert nach Abteilungen (Unt). Die Resultatstabelle sieht demnach wie folgt aus:

| Unt | Anzahl | Summe |
|-----|--------|--------|
| A3 | 1 | 50000 |
| A5 | 1 | 80000 |
| A6 | 2 | 140000 |

Möchte man anstelle der Abteilungsnummern die Namen der Abteilungen in der ersten Spalte genannt haben, so muss die Abfrage (17) um einen Verbund erweitert werden:

```

SELECT      Bezeichnung, COUNT(M#) AS      (18)
            Anzahl, SUM(Lohn) AS Summe
FROM        PERSONAL, ABTEILUNG
WHERE       PERSONAL.Unt = ABTEILUNG.A#
GROUP BY    Bezeichnung
ORDER BY    Bezeichnung;
```

Entsprechend lautet nun die Resultatstabelle der Abfrage (18) wie folgt:

| Bezeichnung | Anzahl | Summe |
|-------------|--------|--------|
| Finanz | 2 | 140000 |
| Informatik | 1 | 50000 |
| Personal | 1 | 80000 |

*Zum HAVING
Schlüsselwort*

Die GROUP BY Anweisung kann durch das Schlüsselwort HAVING erweitert werden, falls bei Gruppierungen weitere Einschränkungen gelten sollen. Der Zusatz HAVING ist nur bei der Verwendung von Aggregationsfunktionen zulässig, da er sich nicht auf einzelne Tupel sondern auf Tupelgruppierungen bezieht. Möchte man eine Liste der Abteilungen mit der Anzahl Mitarbeitenden und der Lohnsumme, allerdings nur für größere Abteilungen mit mehr als einem Mitarbeitenden, so kann die Abfrage (18) wie folgt ergänzt werden:

```

SELECT      Bezeichnung, COUNT(M#) AS      (19)
            Anzahl, SUM(Lohn) AS Summe
FROM        PERSONAL, ABTEILUNG
WHERE       PERSONAL.Unt = ABTEILUNG.A#
```

| | |
|----------|----------------------------------|
| GROUP BY | Bezeichnung HAVING Count(M#) > 1 |
| ORDER BY | Bezeichnung; |

Das Resultat der Abfrage (19) ist eine einzige Zeile, nämlich die Finanzabteilung mit 2 Mitarbeitenden und der Lohnsumme 140000 Schweizer Franken.

Vereinigung, Durchschnitt, Differenz und kartesisches Produkt

Die mengenorientierten Operatoren der Relationenalgebra finden ihre Entsprechung im SQL-Standard. Möchte man z.B. die vereinigungsverträglichen Tabellen SPORTCLUB mit dem FOTOCLUB vereinen, so geschieht das in SQL mit dem Schlüsselwort UNION:

*Der Operator
UNION*

```
SELECT      *                               (20)
FROM        SPORTCLUB
UNION
SELECT      *
FROM        FOTOCLUB;
```

In der Anweisung (20) werden die Mitglieder des Sportclubs und des Fotoclubs selektiert. Da die beiden Tabellen vereinigungsverträglich sind, enthält die Resultattabelle alle Sport- und Fotoclubmitglieder, wobei Duplikate eliminiert werden.

Möchte man alle Sportclubmitglieder herausfinden, die nicht gleichzeitig im Fotoclub mitwirken, so erfolgt die Abfrage mit dem Differenzoperator (Schlüsselwort EXCEPT):

*Subtraktion von
Tabellen
(EXCEPT)*

```
SELECT      *                               (21)
FROM        SPORTCLUB
EXCEPT
SELECT      *
FROM        FOTOCLUB;
```

Gemäß Abschnitt 3.2.2 ist Huber sowohl im Sport- wie im Fotoclub engagiert und er wird deshalb von den Sportclubmitgliedern subtrahiert. Als Resultat dieser Subtraktion verbleiben die beiden Sportclubmitglieder Meier und Schweizer.

Bei vereinigungsverträglichen Tabellen können Durchschnitte gebildet werden. Interessiert man sich für die Mitglieder, die sowohl im Sportclub wie im Fotoclub mitwirken, so kommt das Schlüsselwort INTERSECT zum Zuge:

*Durchschnitts-
bildung mit
INTERSECT*

```
SELECT      *                               (22)
FROM        SPORTCLUB
INTERSECT
SELECT      *
FROM        FOTOCLUB;
```


Da sich nur das Mitglied Huber in beiden Clubs eingeschrieben hat, besteht die Resultatstabelle lediglich aus einem Tupeleintrag.

*Kartesisches
Produkt*

Zu den mengenorientierten Operatoren zählt das kartesische Produkt, bei dem beliebige Tabellen miteinander multipliziert werden. Wie in Abschnitt 3.2.3 erwähnt, macht die Multiplikation beliebiger Tabellen oft wenig Sinn. Werden die Tupeleinträge der Tabelle MITARBEITER mit den Informationen aus der Tabelle ABTEILUNG miteinander kombiniert, so erhält man gemäß der Abb. 3-9 auch Kombinationen von Tupeln, die in der Realität nicht vorkommen. Das entsprechende SQL-Statement würde wie folgt lauten:

```
SELECT      *
FROM        MITARBEITER, ABTEILUNG;           (23)
```

*Verbund mit
Verbundsprädikat*

Anstelle des kartesischen Produkts wäre hier ein Verbund angesagt, der die Mitarbeitenden mit ihren zugehörigen Abteilungsinformationen zusammenstellen würde. Allerdings müsste für einen korrekten Verbund dann auch das Verbundsprädikat spezifiziert werden:

```
SELECT      *
FROM        MITARBEITER, ABTEILUNG
WHERE       Unt = A#;                         (24)
```

Mit den Abfragen (23) und (24) wird nochmals aufgezeigt, dass kartesisches Produkt und Verbund miteinander verwandt sind; der Verbund stellt ein eingeschränktes kartesisches Produkt dar.

Geschachtelte Abfragen

*Schachtelung
von
SQL-Statements*

Es ist erlaubt und manchmal notwendig, innerhalb eines SQL-Statements einen weiteren SQL-Aufruf zu formulieren. Man spricht in diesem Zusammenhang von geschachtelten Abfragen. Solche Abfragen sind z.B. bei der Suche des Mitarbeitenden mit dem höchsten Lohn sinnvoll:

```
SELECT      M#, Name
FROM        PERSONAL
WHERE       Lohn >= ALL (SELECT  Lohn
                        FROM      PERSONAL);   (25)
```

*Zur Bedeutung
des ALL
Schlüsselwortes*

Die Anweisung (25) enthält innerhalb der WHERE-Klausel ein weiteres SQL-Statement, um die Gehälter aller Mitarbeitenden zu selektieren (innerer SQL-Ausdruck resp. Subquery). Im äußeren SQL-Statement wird nochmals die Tabelle PERSONAL konsultiert, um denjenigen Mitarbeiter mit M# und Namen zu erhalten, der den höchsten Lohn erzielt. Das Schlüsselwort ALL¹ bedeutet, dass die Bedingung «Lohn größer gleich ...» für alle selektierten Lohnanteile (Resultat der Subquery) gelten muss.

Das Schlüsselwort **SOME** verlangt mindestens ein Element, das die entsprechende Bedingung erfüllt. Aus diesem Grunde erlaubt der SQL-Standard, anstelle des Schlüsselwortes **SOME** das reservierte Wort **ANY** zu verwenden, da es für den gelegentlichen Benutzer eventuell verständlicher ist.

*Verwendung von
SOME oder ANY*

Der Existenzquantor der Aussagenlogik («... es existiert ein Element, für welches gilt ...») wird im SQL-Standard durch das Schlüsselwort **EXISTS** ausgedrückt. Dieses Schlüsselwort wird bei einer SQL-Auswertung auf «wahr» gesetzt, falls die nachfolgende Subquery mindestens ein Element resp. eine Zeile selektiert.

Als Beispiel einer Abfrage mit einem **EXISTS**-Schlüsselwort können wir die Projektzugehörigkeit **ZUGEHÖRIGKEIT** aus Abb. 2-18 heranziehen, die aufzeigt, welche Mitarbeitenden an welchen Projekten arbeiten. Interessieren wir uns für die Mitarbeiter, die keine Projektarbeit leisten, so lautet das SQL-Statement wie folgt:

*Nutzung des
EXISTS-
Operators*

```
SELECT      M#, Name, Straße, Ort                (26)
FROM        MITARBEITER m
WHERE       NOT EXISTS
            (SELECT      *
             FROM        ZUGEHÖRIGKEIT z
             WHERE       m.M# = z.M#);
```

In der äußeren Anweisung (26) wird die vollständige Tabelle **MITARBEITER** benutzt, wie sie z.B. in der Abb. 2-19 gegeben ist. Aus dieser Tabelle werden die Namen und Adressen derjenigen Mitarbeitenden selektiert, die keine Projektzugehörigkeit haben. Dazu wird eine Subquery formuliert, um alle Mitarbeiter-Projekt-Zugehörigkeiten (Beziehungen) zu erhalten. Im Ausschlussverfahren (**NOT EXISTS**) erhalten wir die gewünschten Mitarbeitenden, die keine Projektarbeit leisten.

In der Abfrage (26) wird nochmals ersichtlich, wie nützlich Ersatznamen (Alias) bei der Formulierung von SQL-Anweisungen sind. Um das Verbundsprädikat im inneren SQL-Statement auszudrücken, wird die Tabelle **MITARBEITER** mit dem neuen Namen **m** und die Tabelle **ZUGEHÖRIGKEIT** mit dem Namen **z** umbenannt. Das Verbundsprädikat **m.M# = z.M#** ersetzt demnach das umständlichere Prädikat **MITARBEITER.M# = ZUGEHÖRIGKEIT.M#**.

*Zur Nützlichkeit
von Ersatznamen*

¹ Das Schlüsselwort **ALL** darf nicht mit dem Allquantor der Aussagenlogik («... für alle Elemente gilt ...») verwechselt werden. Ein solcher Quantor ist im SQL-Standard nicht vorgesehen. Das Schlüsselwort **ALL** führt lediglich einen Vergleich eines selektierten Wertes mit einer Menge von Werten (Resultat einer Subquery) durch.

Datenwerte einfügen, verändern oder löschen

Data Manipulation Language oder DML

Wie wir wissen, ist SQL nicht nur eine Abfragesprache (query language), sondern auch eine Datenmanipulationssprache (Data Manipulation Language oder DML). Die entsprechenden Sprachelemente sind Einfüge- (INSERT), Änderungs- (UPDATE) und Löschoperationen (DELETE).

*Der Operator
INSERT*

Soll der neue Mitarbeiter Müller in der Finanzabteilung A6 seine Tätigkeit aufnehmen, so muss die Tabelle MITARBEITER um ein Tupel erweitert werden:

```
INSERT INTO MITARBEITER (27)
VALUES ('M20', 'Müller', 'Riesweg', 'Olten', 'A6');
```

*Änderungen
durch UPDATE*

Möchte man einzelne Werte in der Tabelle PERSONAL aus Abb. 3-12 ändern, so geschieht dies mit dem UPDATE-Statement. Eine generelle Lohnanpassung von 5% für alle Mitarbeitenden erfolgt durch die Anweisung:

```
UPDATE PERSONAL (28)
SET Lohn = Lohn * 1.05;
```

Die Anweisung (28) zeigt nochmals auf, dass Manipulationsoperationen mengenorientiert erfolgen können. Möchte man individuelle Lohnanpassungen vornehmen, müsste man die Anweisung (28) um eine WHERE-Klausel erweitern.

*Vorsicht beim
DELETE
Operator*

Die Löschanweisung ist insofern heikel, weil hier die Regeln der referenziellen Integrität berücksichtigt werden. Ist beispielsweise eine fortgesetzte Löschregel (ON DELETE CASCADE, siehe Abschnitt 3.8) für die Tabelle ABTEILUNG vorgesehen, so erwirkt der folgende SQL-Befehl Löschvorgänge in der abhängigen Tabelle MITARBEITER:

```
DELETE (29)
FROM ABTEILUNG
WHERE Bezeichnung = 'Informatik';
```

Die Anweisung (29) löscht demnach nicht nur die Informatikabteilung in der Tabelle ABTEILUNG, sondern auch alle Mitarbeitenden der Tabelle MITARBEITER, die in dieser Abteilung arbeiten. Im Beispiel betrifft dieser Löschvorgang den Mitarbeiter Meier.

Merkmale und Tabellen definieren

Data Definition Language oder DDL

Die Sprache SQL erlaubt, Merkmale und Tabellen zu definieren (Data Definition Language oder DDL). Als Datentypen gibt der SQL-Standard unterschiedliche Formate vor:

- CHARACTER(n) oder CHAR(n) bedeutet eine Sequenz von Buchstaben fester Länge n.
- CHARACTER VARYING oder VARCHAR erlaubt die Spezifikation von Buchstabenfolgen beliebiger Länge.
- Numerische Daten werden mit den Datentypen NUMERIC oder DECIMAL festgelegt, wobei Angaben zur Größe und Genauigkeit spezifiziert werden müssen.
- Ganze Zahlen lassen sich durch INTEGER oder SMALLINT angeben.
- Der Datentyp DATE gibt Datumsangaben durch YEAR, MONTH und DAY an. Dabei gelangen unterschiedliche Formate zur Anwendung, so z.B. (yyyy,mm,dd) für Jahres-, Monats- und Tagesangaben (vgl. Abschnitt 6.3 über temporale Datenbanken).
- Der Datentyp TIME liefert Zeitangaben in HOUR, MINUTE und SECOND.
- Der Datentyp TIMESTAMP ist eine Kombination des Typs DATE und TIME. Zusätzlich können die Präzision der Zeitangabe sowie die Zeitzone festgelegt werden.
- Daneben gibt es noch weitere Datentypen für Bit-Strings (BIT oder BIT VARYING) sowie für umfangreiche Objekte (CHARACTER LARGE OBJECT oder BINARY LARGE OBJECT). Zudem wird die Einbindung von XML (Extensible Markup Language, siehe Abschnitt 5.2) unterstützt.

*Die wichtigsten
Datentypen von
SQL*

Der Benutzer kann weitere Datentypen definieren, die ihm das Arbeiten mit relationalen Datenbanken erleichtern. Spielt beispielsweise das Geschlecht für eine Anwendung eine Rolle, lässt sich dies wie folgt festlegen:

*Benutzer-
definierte
Datentypen*

```
CREATE DOMAIN Geschlecht AS VARCHAR      (30)
      DEFAULT 'männlich'
      CHECK (VALUE IN ('männlich','weiblich'));
```

Der Wertevorrat des neuen Datentyps Geschlecht besteht aus den beiden Werten männlich und weiblich. Das entsprechende Schlüsselwort CHECK erlaubt, Integritätsbedingungen für den Wertevorrat zu formulieren (vgl. Abschnitt 3.8).

*Der CHECK
Operator*

Falls keine Angaben bei einer Einfügeoperation betreffend des Geschlechts gemacht werden, soll das Geschlecht den Wert männlich enthalten (Schlüsselwort DEFAULT).

Vorgegebene oder selbst definierte Datentypen werden bei der Definition einer Tabelle benötigt. Das Grundgerüst einer Tabellendefinition sieht wie folgt aus:

Der
CREATE TABLE
Befehl

```
CREATE TABLE tabellenname
    (merkmalsname datatype [UNIQUE] [NOT NULL] [, ... ]
    [PRIMARY KEY (merkmalsliste),]
    {[FOREIGN KEY (merkmalsliste)
    REFERENCES referenziertetabelle
    [ON DELETE löschoption]]});
```

UNIQUE und
NOT NULL
Optionen

Bei der Definition einer Tabelle müssen somit die Merkmale (Attribute) mit ihren Wertebereichen (Datentypen) spezifiziert werden. Der Zusatz UNIQUE bedeutet, dass alle Werte des Merkmals eindeutig sind; Wiederholungen von Datenwerten werden vom Datenbanksystem verweigert. Das fakultative Schlüsselwort NOT NULL untersagt Nullwerte.

Neben der Angabe von Primärschlüsseln können auch Fremdschlüssel angegeben werden, falls auf weitere Tabellen referenziert wird. Regeln der referenziellen Integrität werden ebenfalls spezifiziert, so z.B. Löschregeln mit Nullsetzten, restriktives Löschen oder fortgesetzte Löschung (vgl. Abschnitt 3.8).

Die Definition der Tabelle MITARBEITER aus Abb. 3-7 lautet demnach:

```
CREATE TABLE MITARBEITER                                     (31)
    (M#                CHAR (6) UNIQUE,
    Name               CHAR (20) NOT NULL,
    Strasse            VARCHAR,
    Ort                VARCHAR,
    Unt                CHAR (2)
    PRIMARY KEY (M#),
    FOREIGN KEY (Unt)
    REFERENCES ABTEILUNG
    ON DELETE SET NULL);
```

Referenzielle
Integrität

Die Tabelle MITARBEITER enthält die Merkmale M#, Name, Straße und Ort sowie das Fremdschlüsselmerkmal Unt (Unterstellung). Der Primärschlüssel der Tabelle ist die Mitarbeiternummer. Als Fremdschlüssel gilt das Merkmal Unt, welches die Tabelle ABTEILUNG mit dem Primärschlüssel A# referenziert. Bei einem Löschvorgang in der Tabelle ABTEILUNG sollen die betroffenen Fremdschlüsselwerte in der Mitarbeitertabelle auf Null gesetzt werden (ON DELETE SET NULL), damit die Angaben der Mitarbeiter nicht verloren gehen.

ALTER und
DROP TABLE

Mit dem SQL-Befehl ALTER TABLE können Tabellendefinitionen verändert werden. Der Befehl DROP TABLE erlaubt das Löschen von Tabellen, eventuell mit der Eliminierung aller Tupelinträge.

Soll der Zugriff auf einzelne Merkmale einer Tabelle beschleunigt werden, lassen sich mit dem Befehl `CREATE INDEX` Zugriffsstrukturen definieren. `DROP INDEX` entfernt entsprechende Indices.

Datenschutz und Vergabe von Rechten

Die Definition von Sichten ermöglicht gemäß Abschnitt 3.7, personenbezogene Angaben besser zu schützen. Die Definition einer Sicht erfolgt mit dem SQL-Befehl `CREATE VIEW`:

Sichtdefinition

```
CREATE VIEW sichtenname AS
  SELECT  merkmalsname [,merkmalsname ... ]
  FROM    tabellenname [,tabellenname ... ]
  [WHERE  selektionsbedingung] ;
```

Sichten erhalten einen Namen und können durch eine Selektion beliebiger Datenwerte festgelegt werden.

Möchte man die Lohnangaben in der Tabelle `PERSONAL` nicht allen Mitarbeitenden zukommen lassen, drängt sich eine Sicht mit dem Namen `MITARBEITER` auf:

```
CREATE VIEW MITARBEITER AS                                (32)
  SELECT  M#, Name, Ort, Unt
  FROM    PERSONAL;
```

Damit die Angehörigen der Personalabteilung nur einen eingeschränkten Zugriff auf die Lohnangaben erhalten, können verschiedene Lohnklassen und Zugriffsrechte definiert werden (vgl. Abb. 3-12):

```
CREATE VIEW GRUPPE_A AS                                  (33)
  SELECT  M#, Name, Lohn, Unt
  FROM    PERSONAL
  WHERE   Lohn BETWEEN 70000 AND 90000;
```

Für die Lohnbandbreite zwischen 70000 und 90000 Schweizer Franken wird mit der Hilfe des Schlüsselwortes `BETWEEN` eine Sicht mit dem Namen `GRUPPE_A` definiert. Ausgewählte Mitarbeitende der Personalabteilung erhalten einen Zugriff auf diese Sicht, um Lohnangaben nachführen zu können.

Zugriffsrechte auf Tabellen oder Sichten können mit dem SQL-Befehl `GRANT` vergeben werden:

Zur Vergabe von Rechten (GRANT)

```
GRANT  { privilegienliste | ALL PRIVILEGES }
ON     { tabellenname | viewname }
TO     { berechtigtenliste | PUBLIC }
      [WITH GRANT OPTION];
```

Als *privilegienliste* gelten die SQL-Befehle SELECT, DELETE, INSERT und UPDATE. In der *berechtigtenliste* werden die Benutzer aufgeführt, die ein Zugriffsrecht erhalten sollen; das Schlüsselwort PUBLIC vergibt das entsprechende Recht an jedermann. Der Zusatz WITH GRANT OPTION erlaubt, dass der berechtigte Benutzer seine Rechte weitergeben kann.

Möchte man den lesenden Zugriff auf die Tabelle (resp. Sicht) MITARBEITER allen Angestellten ermöglichen, lautet der GRANT-Befehl:

```
GRANT      SELECT
ON         MITARBEITER
TO         PUBLIC;                                (34)
```

Soll der Zugriff und ein Änderungsrecht auf die Sicht GRUPPE_A selektiv an den Personalverantwortlichen mit der Identifikation ID37289 vergeben werden, geschieht das wie folgt:

```
GRANT      UPDATE
ON         GRUPPE_A
TO         ID37289 WITH GRANT OPTION;            (35)
```

Der Personalverantwortliche kann mit der GRANT OPTION seine Rechte z.B. während Abwesenheiten an Stellvertreter weitergeben.

Rücknahme von
Rechten
(REVOKE)

Die Rücknahme von Rechten erfolgt mit dem Befehl REVOKE auf ähnliche Weise:

```
REVOKE     [GRANT OPTION FOR]
ON         { privilegienliste | ALL PRIVILEGES }
FROM       { tabellenname | viewname }
           { berechtigtenliste | PUBLIC }
           [RESTRICT | CASCADE];
```

Die Schlüsselworte RESTRICT und CASCADE erlauben bei der Rücknahme von Privilegien, nur die Rechte gemäß der *berechtigtenliste* zurückzunehmen (RESTRICT) oder die weitergegebenen Rechte mit CASCADE ebenfalls einzufordern.

Verlässt der Mitarbeiter mit der Identifikation ID37289 die Firma, werden ihm sinnvollerweise sämtliche Rechte weggenommen:

```
REVOKE     ALL PRIVILEGES
ON         GRUPPE_A, MITARBEITER
FROM       ID37289 CASCADE;                      (36)
```

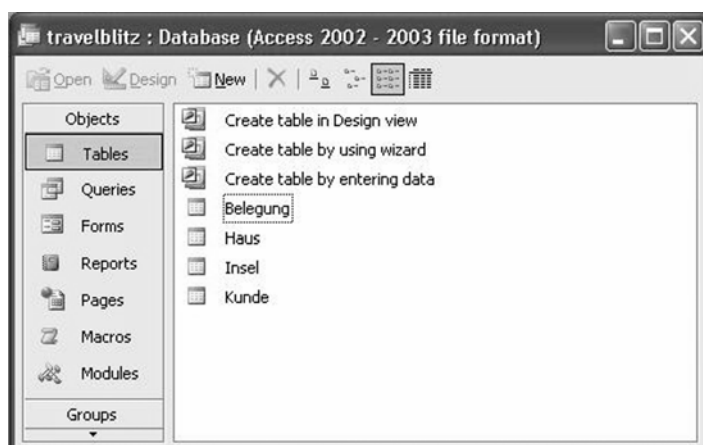
Es wird offensichtlich, dass das Verwalten von Sichten und Zugriffsrechten sauber geplant und mit den entsprechenden SQL-Befehlen laufend nachgeführt werden muss.

Eine Datenbank mit Access erstellen

Mit dieser Anleitung können Sie ein einfaches Datenbankprojekt aus der Reisebranche selbstständig durchspielen. Ausgehend von einem Entitäten-Beziehungsmodell werden Sie die Tabellenstruktur der Datenbank herleiten und in Access definieren. Nachdem Sie die Tabellen mit Daten gefüllt haben, können Sie mit selbst formulierten Abfragen die wichtigsten Informationen aus der Datenbank extrahieren.

Access ist ein Softwareprodukt mit grafischer Benutzeroberfläche zur Verwaltung von Tabellen. Alle Ihre Eingaben werden in SQL-Befehle übersetzt und ausgeführt. Access bietet Ihnen aber auch die Möglichkeit, direkt mit SQL-Befehlen zu arbeiten, was sich beim Erstellen komplexer Datenbankabfragen empfiehlt.

Die Elemente einer Access-Datenbank werden mit Hilfe von Registern verwaltet, wie der Blick auf eine geöffnete Access-Datenbank zeigt:



Das «Datenbankfenster» einer Access-Datenbank (Version 2003)

Im Beispiel werden die in der Datenbank *travelblitz* enthaltenen Register (links) und als Elemente des ersten Registers die Tabellen (Fenster) angezeigt.

Minimal enthält eine Access-Datenbank die folgenden Register:

- Die *Tabellen (Tables)* bilden die Grundlage jeder Access-Datenbank. Sie enthalten die in der Datenbank gespeicherten Informationen in Form von Tupeln (Datensätzen).
- Mit *Abfragen (Queries)* lassen sich Informationen aus der Datenbank selektieren.

Im obigen Beispiel sind weitere Register enthalten, deren Elemente die Bedienbarkeit der Datenbank erleichtern:

- *Formulare (Forms)* gestatten die komfortable Eingabe, Anzeige und Verwaltung der Daten mit Hilfe von Datenmasken.
- *Berichte (Reports)* dienen zum Ausdrucken von Daten in übersichtlicher und ansprechender Form (z.B. für die Rechnungserstellung).
- *Pages, Makros (Macros) und Module (Modules)* ermöglichen die Automatisierung komplexer Datenbankabläufe.

In dieser Anleitung beziehen sich die Abbildungen und Befehle auf die englische Version von Access 2003; es sollte Ihnen aber nicht schwerfallen, den Bezug zu anderen Versionen herzustellen.

Zur Fallstudie *travelblitz*

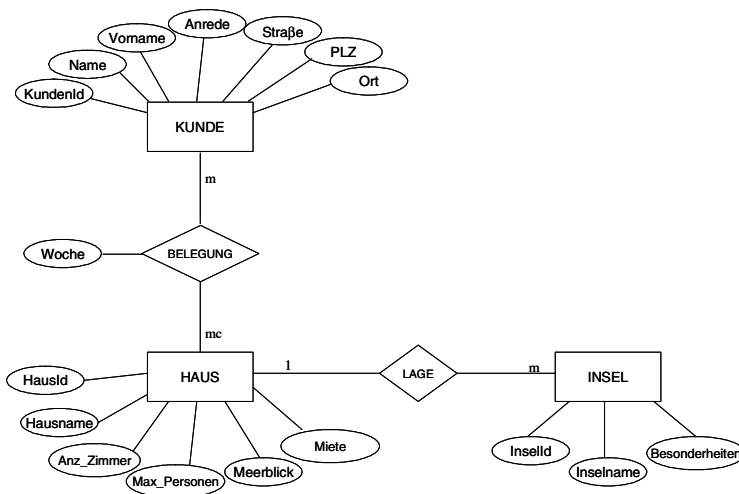
Als Beispiel realisieren Sie eine Datenbankanwendung für das Reisebüro *travelblitz*, das sich auf die Vermietung von Ferienhäusern auf griechischen Inseln spezialisiert hat. Die Ferienhäuser befinden sich auf verschiedenen Inseln. Es ist damit zu rechnen, dass in Zukunft weitere Häuser auf neuen Inseln dazukommen. Momentan werden Kunden- und Häuserdaten traditionell mit der Hilfe eines Karteisystems verwaltet, die Abfragemöglichkeiten sind entsprechend eingeschränkt. Beispielsweise ist es zeitraubend herauszufinden, welche Häuser weniger als 400 Euro pro Woche kosten und in einem bestimmten Zeitraum (z.B. für die ersten drei Juliwochen) frei sind. Sie möchten sich nun besser auf die Kundenanfragen einstellen und beschließen, den Missstand durch den Einsatz einer Datenbank zu beseitigen.

Schritt 1: Entitäten-Beziehungsmodell entwickeln

Für die Erstellung eines Entitäten-Beziehungsmodells treffen Sie die folgenden vereinfachenden Annahmen:

1. Die Saison dauert von Woche 10 bis 40, d.h. von Anfang April bis Ende September. Der Mietpreis ist während der gesamten Saison konstant; die Häuser werden wochenweise vermietet.
2. Eine Buchung enthält die folgenden Informationen: Haus, Kunde und Nummer der Woche. Falls ein Kunde ein Haus mehrere Wochen hintereinander mietet, müssen mehrere Buchungen angelegt werden (eine pro Woche).
3. Der Zeitraum jeder Buchung, d.h. die betreffende Woche, wird in Form einer Zahl zwischen 10 und 40 angegeben. Für jedes Jahr wird eine neue Datenbank angelegt.
4. Die Datenbank enthält vorläufig keinerlei Informationen über Zahlungsfristen, Rechnungen usw.

Diese Annahmen dienen dazu, das Datenmodell zu Übungszwecken so klein wie möglich zu halten. Sie sind nun aufgefordert, das Entitäten-Beziehungsmodell für *travelblitz* zu entwickeln: Welche Entitätsmengen und Beziehungsmengen legen Sie fest? Welche Merkmale ordnen Sie den Entitätsmengen zu, welche den Beziehungsmengen? Welche Identifikationsschlüssel sehen Sie für die Entitätsmengen vor?



Mit den Entitätsmengen KUNDE, HAUS und INSEL legen Sie das Entitäten-Beziehungsmodell für die Ferienhaus-Verwaltung fest. Die

Beziehungsmenge BELEGUNG zeigt Ihnen die Ausleihe der Ferienhäuser an die Kunden; die Beziehung ist komplex-komplex. Das Merkmal Woche haben Sie als typisches Beziehungsmerkmal erkannt, da es die Belegung eines Ferienhauses durch einen Kunden zeitlich festlegt. Schließlich drücken Sie die hierarchische Zuordnung der Ferienhäuser zu den Inseln mit der Beziehungsmenge LAGE aus.

Schritt 2: Datenbankschema entwerfen

Nun stellen Sie sich folgende Fragen: Wie sieht das relationale Datenbankschema für die Ferienhaus-Verwaltung aus? Welche Abbildungsregeln verwenden Sie, um das obige Entitäten-Beziehungsmodell in Tabellen zu überführen?

Aus dem Entitäten-Beziehungsmodell leiten Sie gemäß Abschnitt 2.3.2 die Tabellenstruktur der Datenbank her:

1. Gemäß der Abbildungsregel 1 müssen Sie für jede Entitätsmenge eine eigenständige Tabelle festlegen. Als Namen für die Tabellen wählen Sie der Einfachheit halber die Namen der entsprechenden Entitätsmengen. Damit erhalten Sie folgende Tabellen:
 - KUNDE (*KundenId*, Name, Vorname, Anrede, Straße, PLZ, Ort)
 - HAUS (*HausId*, Hausname, Anz_Zimmer, Max_Personen, Meerblick, Miete)
 - INSEL (*InselId*, Inselname, Besonderheiten)

Wie üblich heben Sie die Identifikationsschlüssel kursiv hervor.

2. Gemäß der Abbildungsregel 3 müssen Sie auf alle Fälle für die komplex-komplexe Beziehungsmenge BELEGUNG eine eigenständige Tabelle definieren, zur Speicherung der Mietbeziehungen. Diese Tabelle enthält neben den Fremdschlüsselattributen für Häuser (*HausId*) und Kunden (*KundenId*) das Beziehungsmerkmal Woche, das die Zeitangabe der Vermietung in Form der Wochennummer enthält:
 - BELEGUNG (*HausId*, *KundenId*, *Woche*)

Als zusammengesetzten Schlüssel zeichnen Sie die *HausId* und die Wochennummer aus. Damit verhindern Sie auf effiziente Art Doppelbelegungen resp. Überbuchungen.

3. Die einfach-komplexe Beziehungsmenge LAGE zwischen Ferienhäusern und Inseln können Sie auf zwei Arten im Datenbankschema darstellen: entweder als eigenständige Tabelle (Abbil-

dungsregel 2) oder durch einen Fremdschlüsselverweis in der Ferienhaustabelle (Abbildungsregel 4). Sie wählen die zweite Variante und erweitern die Tabelle HAUS wie folgt:

- HAUS (*HausId*, Hausname, Anz_Zimmer, Max_Personen, Meerblick, Miete, InseId_Lage)
4. Als Resultat erhalten Sie die folgenden vier Tabellen, die Sie noch hinsichtlich der dritten Normalform überprüfen müssen:
- KUNDE (*KundenId*, Name, Vorname, Anrede, Straße, PLZ, Ort)
 - HAUS (*HausId*, Hausname, Anz_Zimmer, Max_Personen, Meerblick, Miete, InseId_Lage)
 - INSEL (*InseId*, Inselname, Besonderheiten)
 - BELEGUNG (*HausId*, KundenId, Woche)

Bis auf die Tabelle KUNDE erfüllen alle Tabellen die dritte Normalform. Da das Merkmal Ort transitiv via PLZ vom Merkmal KundenId abhängig ist, müssten Sie eine zusätzliche Tabelle ORTSCHAFT definieren, mit den beiden Merkmalen PLZ und Ort. Aus praktischen Gründen verzichten Sie jedoch auf diese Aufteilung.

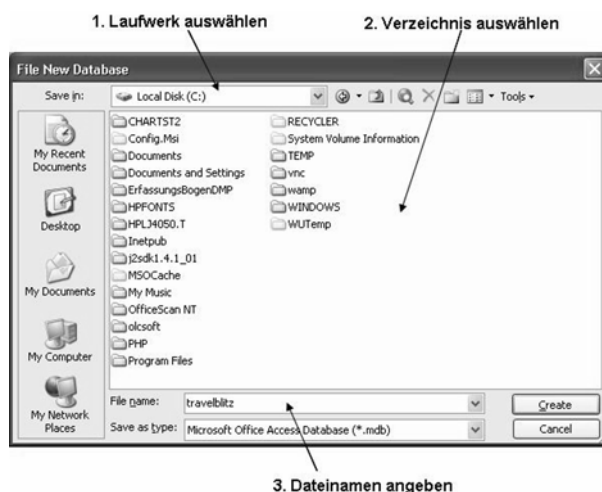
Schritt 3: Datenbankschema in Access definieren

Starten Sie Access und geben Sie an, dass sie eine neue leere Datenbank erstellen wollen ((1) oder (2)):



Das Anlegen einer neuen Datenbank

Damit Access weiß, wo die Datei für die Datenbank angelegt werden soll, müssen Sie im Fenster *File New Database* Laufwerk, Verzeichnis und Dateinamen angeben:

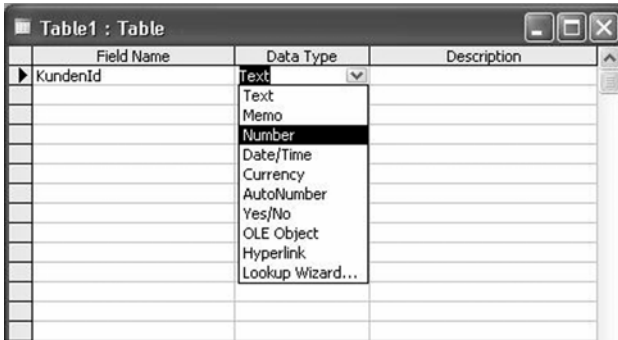


Anschließend erscheint das bereits bekannte Datenbankfenster, wobei die Register zu Beginn natürlich alle noch leer sind.

Nun können Sie mit dem Definieren des Datenbankschemas beginnen. Definieren Sie als erstes die Kundentabelle wie folgt:

1. Wählen Sie im Register *Tables* die Schaltfläche *Create Table* (in der Design View). Access präsentiert anschließend eine Tabelle, in der man die Attribute der zu definierenden Tabelle zeilenweise einträgt. Für jedes Attribut müssen Sie die folgenden Angaben festlegen:
 - Name,
 - Datentyp,
 - eine kurze Beschreibung des Attributs.

Den Attributnamen und die -beschreibung müssen Sie eingeben. Den Datentyp können Sie aus einer Liste auswählen: Klicken Sie in der Spalte *Data Type* in die entsprechende Auswahlliste (Pfeil zum Öffnen der Liste verwenden). Wählen Sie aus der Liste einen Datentyp für das gewünschte Merkmal:



Tabellen-
definition in
Access
(Datentyp
auswählen)

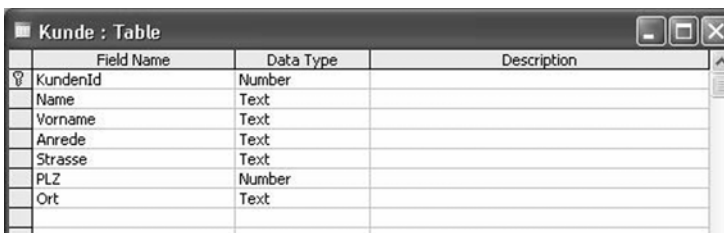
Die wichtigsten Datentypen in Access sind: Text, Zahl, Währung, Wahrheitswert (ja/nein) und Datum.

2. Zum Auszeichnen des Primärschlüssels markieren Sie das betreffende Attribut (resp. die betreffenden Attribute) und verwenden dann den Befehl *Edit/Primary Key*. Falls der Primärschlüssel aus mehreren Attributen zusammengesetzt ist, markieren Sie zunächst nur ein Schlüsselmerkmal, drücken dann die CTRL-Taste und markieren die anderen zum Schlüssel gehörenden Attribute.
3. Wenn Sie den Tabellenentwurf schließen, können Sie den Tabellennamen im aufgehenden Fenster eingeben:



Tabellennamen
eingeben

Falls Sie die Tabelle KUNDE vollständig definiert haben, sollte die Entwurfssicht wie folgt aussehen:



Definition der
Kundentabelle

Definieren Sie nun analog die Tabellen HAUS, INSEL und BELEGUNG.

Als Nächstes vereinbaren Sie strukturelle Integritätsbedingungen, wie sie in den Abschnitten 2.5 und 3.8 behandelt wurden. In Access lassen sich drei Typen von Integritätsregeln definieren:

- Wertebereichsbedingungen (siehe Schritt 4),
- Prüfregeln für Tupel (siehe Schritt 5) und
- referenzielle Integrität (siehe Schritt 6).

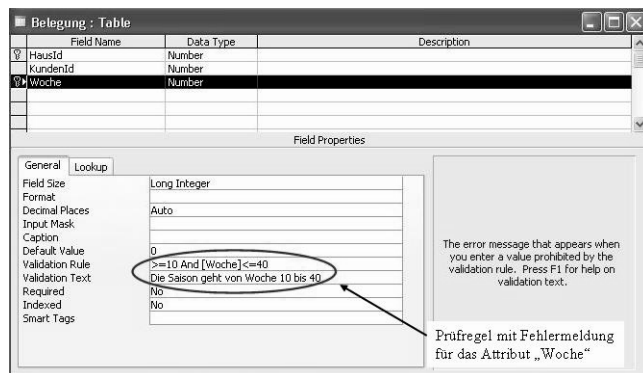
Diese Integritätsbedingungen können Sie unabhängig voneinander definieren. Im Folgenden vereinbaren Sie jeweils Bedingungen für Wertebereiche und Tupel sowie die referenzielle Integrität für Fremdschlüsselmerkmale.

Schritt 4: Wertebereichsbedingungen spezifizieren

Wie können Sie für ein bestimmtes Attribut die zulässigen Datenwerte einschränken? Beispielsweise möchten Sie für das Merkmal Woche der Tabelle BELEGUNG nur Zahlen zwischen 10 und 40 eintragen, da die Ferienhäuser in den übrigen Wochen nicht vermietet werden. Wie realisieren Sie diese Integritätsbedingung in Access?

Um eine Wertebereichsbedingung für das Merkmal Woche zu formulieren, öffnen Sie die betreffende Tabelle im Entwurfsmodus und markieren darin das Attribut. Im unteren Bereich des Fensters erscheinen jeweils detailliertere Informationen zum momentan markierten Attribut. Im Feld *Validation Rule* können Sie eine Prüfregel für das markierte Attribut eintragen. Falls ein Anwender später einen Wert in dieses Feld eingeben möchte, der die eingetragene Integritätsregel verletzt, so weigert sich die Datenbanksoftware und reagiert mit einer Fehlermeldung. Im Feld *Validation Text* können Sie einen möglichst aussagekräftigen Text für die Fehlermeldung eintragen.

Wertebereichs-
bedingungen für
Merkmale



Wird dieses Feld leer gelassen, so enthält die Fehlermeldung einen nichtssagenden Standardtext wie z.B. «ungültige Eingabe».

Definieren Sie die Integritätsbedingung für das Merkmal Woche und erfassen Sie eine aussagekräftige Fehlermeldung.

Als Anrede der Kunden möchten Sie «Herr» und «Frau» zulassen. Wie formulieren Sie eine entsprechende Integritätsbedingung in Access?

An dieser Stelle können Sie zwei weitere Einstellungen vornehmen, die ebenfalls zu den Wertebereichsbedingungen zählen. Es handelt sich um die Merkmalseigenschaften *Field-Size* und *Required*:

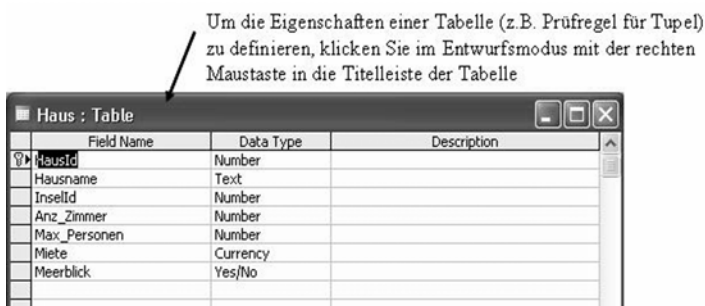
- Mit dem Schlüsselwort *Field-Size* können Sie den Datentyp für das markierte Attribut näher spezifizieren. Bei Textmerkmalen ist hier die maximale Länge des Textes, bei Zahlmerkmalen der Typ der Zahl (ganze Zahl als Integer, Gleitkommazahl als Single oder Double) gemeint.
- Mit dem Schlüsselwort *Required* kann das Ausfüllen eines Attributes erzwungen werden. Ein Merkmal mit *Required*=Yes darf beim Ausfüllen nicht leer gelassen werden.

Bemerkung: Sie können Schritt 4 simultan mit Schritt 3 durchführen, indem Sie während der Definition jedes Attributes die jeweiligen Eigenschaften im unteren Bereich des Fensters mit vereinbaren.

Schritt 5: Prüfregeln für Tupel festlegen

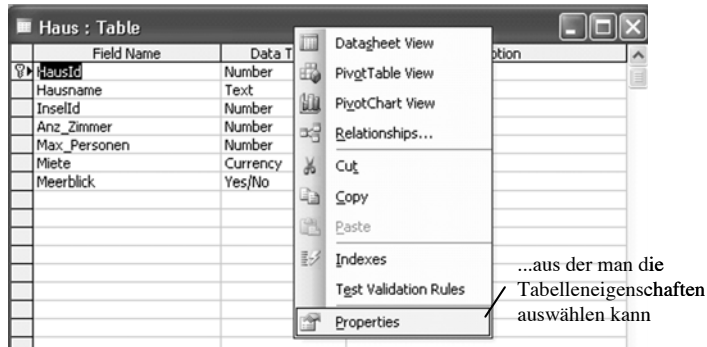
Diese Integritätsbedingung bezieht sich auf mehrere Merkmale desselben Tupels und setzt deren Inhalte zueinander in Beziehung. Ein Beispiel für eine solche Regel wäre, dass die Miete für jedes Haus mindestens 100 Euro je Zimmer betragen soll. Diese Bedingung lässt sich durch die beiden Attribute *Anz_Zimmer* und *Miete* durch « $Miete \geq Anz_Zimmer * 100$ » ausdrücken.

Um eine Prüfregel für Tupel zu vereinbaren, öffnen Sie die Tabelle im Entwurfsmodus und machen einen Rechtsklick in die Titelleiste:



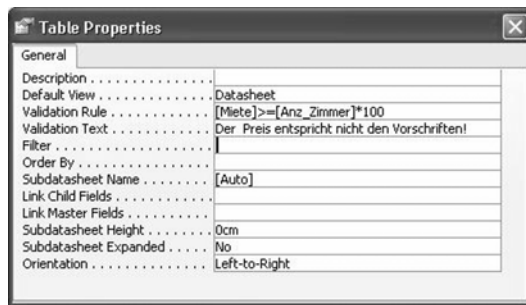
Das Öffnen eines kontext-sensitiven Menüs ...

Es erscheint eine Auswahlliste. Wählen Sie die *Properties* aus:



Nun erhalten Sie ein Fenster mit den Eigenschaften der Tabelle. Unter *Validation Rule* können Sie eine Regel eintragen, die jedes Tupel der Tabelle erfüllen muss. Unter *Validation Text* erfassen Sie die Fehlermeldung, die angezeigt wird, falls ein Benutzer beim Eintrag eines Tupels in die Datenbank diese Regel verletzt:

*Prüfregel für die
Tupel der Tabelle
HAUS (inkl.
Fehlermeldung)*



Sie können nun die angesprochene Integritätsregel für die Tabelle HAUS erfassen.

Schritt 6: Referenzielle Integrität formulieren

Erinnern Sie sich an die Regel der referenziellen Integrität aus den Abschnitten 2.5 und 3.8? Sie können damit verhindern, dass in einem Fremdschlüsselmerkmal ein Wert eingetragen wird, zu dem in der referenzierten Tabelle kein Tupel existiert. In der Datenbank von *travelblitz* möchten Sie sich für jedes Fremdschlüsselattribut durch referenzielle Integrität gegen derartige ungültige Dateneingaben absichern.

Die Fremdschlüsselbeziehungen werden in der Entwurfssicht von Access als Linien eingezeichnet, die die Fremdschlüsselmerkmale mit den entsprechenden Primärschlüsseln der referenzierten Tabellen

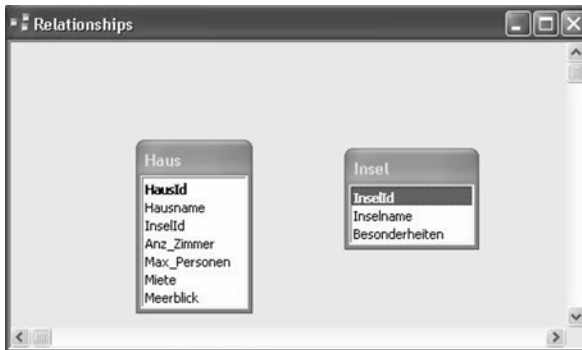
verbinden. Vereinbaren Sie die referenzielle Integrität zunächst für das Fremdschlüsselmerkmal *InselId* in der Tabelle der Ferienhäuser. Dazu müssen Sie die Teilschritte 1 bis 4 durchlaufen:

1. Wählen Sie den Befehl *Tools/Relationships*. Anschließend erscheint das Fenster *ShowTable* zum Auswählen der Tabellen für die referenzielle Integrität:



Das Fenster Show Table zum Auswählen der Tabellen für die referenzielle Integrität

2. Wählen Sie mit *Add* die Tabellen HAUS und INSEL aus, und beenden Sie die Tabellenauswahl mit *Close*. Nun werden beide Tabellen schematisch in einem Fenster namens *Relationships* angezeigt (Primärschlüssel sind durch Fettdruck hervorgehoben):



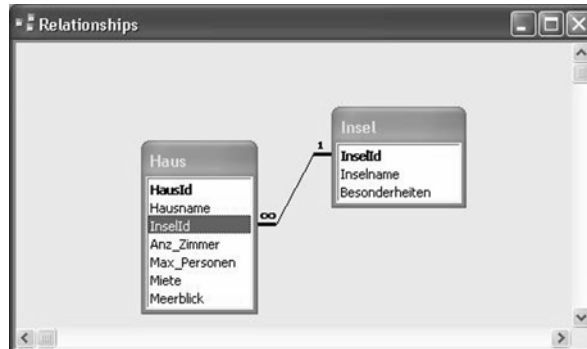
Tabellen vor dem Vereinbaren der referenziellen Integrität ...

Vergrößern Sie zunächst den Rahmen um das Schema der Häusertabelle, damit alle Attribute angezeigt werden (wie abgebildet), und verschieben Sie die Inseltabelle etwas nach rechts, damit mehr Platz zwischen den beiden Schemas ist (zum Verschieben eines Fensters klicken Sie in dessen Titelleiste und ziehen das Fenster bei gedrückter Maustaste an die gewünschte Stelle).


3. Positionieren Sie den Mauszeiger nun über dem Attribut *Haus.InselId*, drücken Sie die Maustaste nach unten, halten Sie

sie weiter gedrückt und ziehen Sie den Mauszeiger zum Attribut *Insel.InselId*. Es erscheint ein Eingabefenster. Machen Sie darin einen Haken bei *Enforce Referential Integrity* und wählen Sie *Create*. Nun sind beide Attribute durch eine Linie miteinander verbunden:

... und danach

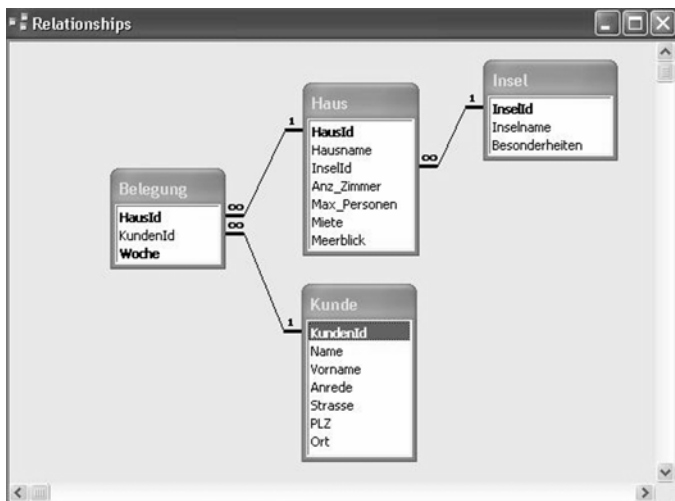


Falls die Linie nicht so aussieht wie abgebildet, haben Sie etwas falsch gemacht. Prüfen Sie in diesem Fall, ob die referenzielle Integrität vereinbart wurde. Machen Sie dazu einen Rechtsklick auf die Verbindungslinie und wählen Sie *Edit Relationship*. Falls dies nichts nützt, überprüfen Sie, ob die Datentypen beider Attribute übereinstimmen, und korrigieren Sie diese gegebenenfalls. Falls auch dies nichts nützt, löschen Sie die Verbindungslinie und beginnen von vorne.

4. Beenden Sie die Definition der Fremdschlüsselbeziehungen, indem Sie den Befehl *File/Close* wählen oder auf das kleine Kreuz  in der rechten oberen Ecke des *Relationships*-Fensters klicken. Speichern Sie die durchgeführten Änderungen ab.

Vereinbaren Sie die referenzielle Integrität nun auch für die übrigen Fremdschlüsselmerkmale. Verwenden Sie wiederum den Befehl *Tools/Relationships*. Um weitere Tabellen anzeigen zu lassen, machen Sie einen Rechtsklick in den Hintergrund des *Relationships*-Fensters, wählen *ShowTable* und fahren anschließend fort wie gehabt.

Wenn alles richtig funktioniert, sieht das *Relationships*-Fenster anschließend wie folgt aus:



Nun ist das Datenbankschema fertig definiert. Sie können mit der Dateneingabe beginnen!

Schritt 7: Daten eingeben

Öffnen Sie zunächst die Tabelle KUNDE. Sie müssen sich dazu im Datenbankfenster im Register *Tables* befinden. Zum Öffnen einer Tabelle haben Sie zwei Möglichkeiten: Entweder klicken Sie doppelt darauf oder Sie markieren die gewünschte Tabelle und wählen *Open*.

Es erscheint die (zur Zeit noch leere) Tabelle, die momentan nur aus einer Zeile für die Dateneingabe besteht. Attribute für Zahlen sind in Access standardmäßig durch Nullen gekennzeichnet, damit der Benutzer weiß, wo Zahlen eingegeben werden müssen.


Geben Sie nun einige Datenwerte in die Tabelle ein:

| Kunde : Table | | | | | | |
|---------------|-------|---------|--------|---------|-----|-----|
| KundenId | Name | Vorname | Anrede | Strasse | PLZ | Ort |
| 1 | Meier | Ursula | | | 0 | |
| 2 | | | | | 0 | |

Dateneingabe in Access

Beachten Sie folgende Hinweise bei der Datenerfassung:

1. Mit der Tabulator- oder Eingabetaste wechseln Sie in das jeweils nächste Feld. Nehmen Sie die Umschalt-Taste (\uparrow , auch *Shift*-Taste genannt) hinzu, so kehren Sie in das vorherige Feld zurück.
2. Mit den Pfeiltasten erreichen Sie beliebige Tabellenabschnitte.

3. Zum Widerrufen einer Eingabe dient Ihnen die *Escape* Taste: Durch einmaliges Drücken der Taste setzen Sie den Inhalt der aktuellen Zelle zurück, durch zweimaliges das gesamte Tupel.
4. Zum Löschen eines Datensatzes markieren Sie die entsprechende Tupelzeile und drücken Delete.
5. Zum Eingeben logischer Datenwerte (z.B. für das Attribut Meerblick) benutzen Sie die Maus oder Leertaste.
6. Zum Schließen einer Tabelle wählen Sie *File/Close* oder klicken auf  .

Erfassen Sie folgende Beispieltabellen:

KUNDE

| KundenId | Name | Vorname | Anrede | Strasse | PLZ | Ort |
|----------|---------|---------|--------|---------------------|------|----------|
| 1 | Meier | Ursula | Frau | Lindenstr. 13 | 4051 | Basel |
| 2 | Aeby | Paul | Herr | Rosenweg 26 | 3001 | Bern |
| 3 | Gessner | Heidi | Frau | Reichengasse 11 | 1700 | Fribourg |
| 4 | Zumsteg | Irene | Frau | Brückenstr. 23 | 3005 | Bern |
| 5 | Wyss | Beat | Herr | Wallisellenstr. 243 | 8050 | Zürich |

INSEL

| InselId | Inselname | Besonderheiten |
|---------|-----------|---|
| 1 | Rhodos | Windsurfen |
| 2 | Samos | Golfplatz, Wandern |
| 3 | Kreta | historische Sehenswürdigkeiten, Felsklippen |

HAUS

| HausId | Hausname | InselId | Anz_Zimmer | Max_Personen | Miete | Meerblick |
|--------|------------|---------|------------|--------------|----------|-------------------------------------|
| 1 | Paphos | 1 | 3 | 5 | € 500.00 | <input checked="" type="checkbox"/> |
| 2 | Arethoussa | 2 | 2 | 4 | € 400.00 | <input type="checkbox"/> |
| 3 | Malia | 1 | 4 | 6 | € 750.00 | <input checked="" type="checkbox"/> |
| 4 | Atrium | 3 | 3 | 4 | € 450.00 | <input checked="" type="checkbox"/> |

BELEGUNG

| HausId | KundenId | Woche |
|--------|----------|-------|
| 1 | 2 | 28 |
| 1 | 2 | 29 |
| 1 | 4 | 31 |
| 1 | 4 | 32 |
| 1 | 4 | 33 |
| 2 | 5 | 17 |
| 2 | 1 | 31 |
| 2 | 1 | 32 |
| 4 | 3 | 25 |
| 4 | 3 | 26 |
| 4 | 3 | 27 |

Falls Sie die Tabellen in der vorgegebenen Reihenfolge ausgefüllt haben, gibt es keine Verletzungen der referenziellen Integrität. Dank

des zusammengesetzten Primärschlüssels in der Tabelle BELEGUNG vermeiden Sie zudem eventuelle Doppelbuchungen.

Testen Sie nun, ob die vereinbarten Integritätsregeln von der Datenbank auch unterstützt werden. Versuchen Sie,

- in der Tabelle BELEGUNG eine ungültige Wochennummer einzutragen,
- bei einer Kundin die Anrede von «Frau» auf «Fräulein» abzuändern,
- die Miete eines Hauses auf einen zu tiefen Wert herabzusetzen (vgl. die Prüfregel zur Miete),
- bei einem Haus die Inselnummer auf einen nicht vorhandenen Wert abzuändern oder
- das Haus Arethoussa aus der Datenbank zu löschen (Warum geht dies nicht?).

Nun können Sie mit der Datenbank arbeiten und Geschäfte abwickeln. Beispielsweise möchten Sie Herrn Ernst Bircher, wohnhaft in der Seestraße 10 in 6004 Luzern erfassen, der Ende August das Ferienhaus Malia für drei Wochen (Wochen 33 bis 35) buchen möchte. Tragen Sie diesen Geschäftsfall in Ihre Datenbank ein.

Das Reisebüro *travelblitz* hat ein neues Ferienhaus mit dem Namen Pegasus auf der Insel Kreta akquiriert. Dieses Haus hat 5 Zimmer und kann maximal 8 Personen aufnehmen. Die Miete beträgt 650 Euro. Leider gibt es keine Sicht auf das Meer. Geben Sie nun all diese Angaben in die Datenbank ein.

Schritt 8: Datenbank mit QBE auswerten

Sie interessieren sich für Fragen wie: Welche Häuser befinden sich auf der Insel Kreta? Welches Haus hat die Kundin Ursula Meier gebucht? Welche Häuser sind in den Wochen 31 bis 33 frei? Natürlich können Sie die Antworten anhand der obigen Tabelle ablesen. Sobald die Datenbank jedoch realistischere Ausmaße annimmt, lohnt sich der Einsatz von Datenbankauswertungssprachen.

In Access können Sie eine Abfrage entweder mit einer grafischen Oberfläche im QBE-Modus (vgl. Abschnitt 3.4.3) durchführen oder als SQL-Befehl eingeben (vgl. Abschnitt 3.4.1 und folgenden Schritt 9).

Sie wünschen eine Liste aller Häuser mit Preisangaben, wobei die Häuser nach der Höhe des Mietpreises sortiert sind:

Eine einfache
Selektions-
abfrage ...

| Hausname | Miete |
|------------|----------|
| Arethoussa | € 400.00 |
| Atrium | € 450.00 |
| Paphos | € 500.00 |
| Pegasus | € 650.00 |
| Malia | € 750.00 |

Um eine Abfrage mit QBE durchzuführen, müssen Sie vier Schritte ausführen:

1. Wechseln Sie ins Register Abfragen (*Queries*) und klicken Sie auf *New*. Wählen Sie den Entwurfsmodus (*Design View*).
2. Selektieren Sie im Fenster *Show Table* die Tabellen, die Sie für die Abfrage benötigen. Es erscheint eine Entwurfstabelle, in die Sie im QBE-Modus die Abfrage eintragen können.
3. Pro Attribut der Entwurfstabelle können Sie eine der folgenden Optionen aktivieren:
 - Anzeigen des Attributs (*Show*),
 - Sortieren der Datensätze nach diesem Attribut (*Sort*), nach auf- oder absteigender Folge oder
 - Heraussuchen von Datensätzen (Filtern) anhand einer Suchbedingung (*Criteria*).

... in QBE
formuliert
(Entwurfsmodus)

Query1 : Select Query

| Field: | Hausname | Miete | | |
|-----------|-------------------------------------|-------------------------------------|--------------------------|--|
| Table: | Haus | Haus | | |
| Sort: | | Ascending | | |
| Show: | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | |
| Criteria: | | | | |
| or: | | | | |

- Um die Abfrage ausführen zu können, wechseln Sie mit dem Befehl *View/Datasheet View* in den Datenmodus. Access präsentiert dann das Resultat der Abfrage:

| Hausname | Miete |
|------------|----------|
| Arethoussa | € 400.00 |
| Atrium | € 450.00 |
| Paphos | € 500.00 |
| Pegasus | € 650.00 |
| Malia | € 750.00 |
| | € 0.00 |

Das Resultat der Abfrage im Datenmodus

Bemerkung: Der leere Datensatz am Schluss bedeutet, dass über die Abfrage prinzipiell ein neuer Datensatz in die zugrunde liegende Tabelle eingetragen werden könnte. Wir raten Ihnen von dieser Eingabemöglichkeit jedoch ab.

Falls Sie eine Abfrage abändern wollen, wechseln Sie mit *View/Design View* zurück in den Entwurfsmodus. Dort führen Sie die gewünschten Änderungen durch und prüfen das Resultat im Datenmodus.

- Falls Sie die Abfrage in Zukunft benötigen, speichern Sie sie ab (vgl. nächster Abschnitt).

Sie speichern eine Abfrage entweder mit dem Befehl *File/Save*, oder Sie schließen die Abfrage, wobei sich automatisch die Gelegenheit zum Speichern ergibt. Nach der Speicherung erscheint die Abfrage im Register *Queries*:



Das Register Queries enthält die Datenbank-abfragen als virtuelle Tabellen

Mit der Schaltfläche *Open* können Sie die Abfrage jederzeit starten. Mit der Schaltfläche *Design* öffnen Sie die Abfrage im Entwurfsmodus.

Beachten Sie: Das Resultat einer Abfrage sieht zwar aus wie eine Tabelle, die angezeigten Daten sind aber nicht in Tabellenform abgespeichert, sondern werden bei jedem Aufruf neu generiert. Falls Daten in einer Tabelle abgeändert werden, ändern sich also die Resultate aller darauf basierenden Abfragen automatisch.

Schritt 9: Abfragen mit SQL durchführen

Access wandelt jede Abfrage intern in einen SQL-Befehl um und führt diesen anschließend aus. Sie können diesen SQL-Befehl auch anschauen. Dazu öffnen Sie die Abfrage im Daten- oder Entwurfsmodus und wählen den Befehl *View/SQL View*.

Führen Sie nun die soeben erzeugte Abfrage «Hauspreise» durch und wählen Sie anschließend die SQL-Ansicht aus:

*Anzeigen des
entsprechenden
SQL-Befehls*



Access stellt den Attributnamen immer die jeweiligen Tabellennamen voran. In den meisten Fällen können Sie in der SQL-Ansicht auf diese Präzisierung verzichten und den einfachen SQL-Befehl

```
SELECT      Hausname, Miete
FROM        Haus
ORDER BY    Miete;
```

eingeben.

Sie wünschen eine Liste mit Namen, Vornamen, Postleitzahl und Ort aller Kunden aus der Ostschweiz (mit Postleitzahl ≥ 3000). Zum Erstellen dieser Abfrage wählen Sie das Register Abfragen und den Befehl *View/SQL View* für den SQL-Modus. Nun geben Sie den benötigten Befehl in das Eingabefenster ein:

*Beispiel einer
SQL-Abfrage ...*



Führen Sie diese SQL-Abfrage aus und benutzen Sie den Befehl *View/Datasheet View*, um das Resultat anzuschauen:

Query2 : Select Query

| | Name | Vorname | PLZ | Ort |
|--|---------|---------|------|--------|
| | Meier | Ursula | 4051 | Basel |
| | Aeby | Paul | 3001 | Bern |
| | Zumsteg | Irene | 3005 | Bern |
| | Wyss | Beat | 8050 | Zürich |
| | Bircher | Ernst | 6004 | Luzern |
| | | | 0 | |

Record: 1 2 3 4 5 6 of 6

... mit dem gewünschten Resultat

Falls es Sie interessiert, wie Sie diese Abfrage in QBE formulieren könnten, wechseln Sie mit dem Befehl *View/Design View* in den QBE-Modus:

Query2 : Select Query

Kunde

- * KundenId
- Name
- Vorname
- Anrede
- Strasse
- PLZ
- Ort

| | | | | |
|-----------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| Field: | Name | Vorname | PLZ | Ort |
| Table: | Kunde | Kunde | Kunde | Kunde |
| Sort: | | | | |
| Show: | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Criteria: | | | >=3000 | |
| or: | | | | |

... dieselbe Abfrage in QBE formuliert

Vergleichen Sie den SQL-Befehl mit der Formulierung im QBE-Modus, so bemerken Sie, dass für beide Abfragetypen dieselben Angaben notwendig sind. Sie können deshalb einfache Abfragen wahlweise in SQL oder QBE erzeugen, je nachdem, wo Sie sich sicherer fühlen. Komplexere Abfragen (z.B. geschachtelte SQL-Befehle, siehe Schritt 10) müssen Sie in SQL formulieren.

Schritt 10: Komplexere Abfragen (Join oder geschachtelte SQL) erfassen

Häufig möchten Sie Daten aus mehreren Tabellen auswerten. Sie wünschen beispielsweise eine Übersicht über alle Häuser mit den jeweiligen Inselangaben:

Abfrage mit
Daten aus zwei
Tabellen

| Hausname | Inselname | Anz_Zimmer | Max_Personen | Meerblick | Miete |
|------------|-----------|------------|--------------|-------------------------------------|----------|
| Arethoussa | Samos | 2 | 4 | <input type="checkbox"/> | € 400.00 |
| Atrium | Kreta | 3 | 4 | <input checked="" type="checkbox"/> | € 450.00 |
| Malia | Rhodos | 4 | 6 | <input checked="" type="checkbox"/> | € 750.00 |
| Paphos | Rhodos | 3 | 5 | <input checked="" type="checkbox"/> | € 500.00 |
| Pegasus | Kreta | 5 | 8 | <input type="checkbox"/> | € 650.00 |

Für dieses Resultat benötigen Sie die Tabellen HAUS und INSEL, indem Sie die beiden Tabellen über das Attribut *InselId* miteinander verknüpfen:

```
SELECT      Hausname, Inselname, Anz_Zimmer,
            Max_Personen, Meerblick, Miete
FROM        Haus, Insel
WHERE       Haus.InselId=Insel.InselId
ORDER BY    Hausname;
```

Erzeugen Sie diese Abfrage und speichern Sie das Resultat ab.

Wie verknüpfen Sie mehr als zwei Tabellen miteinander? Beispielsweise wollen Sie eine Liste aller Buchungen mit den Namen der Kunden inklusive der gebuchten Häuser:

Abfrage mit
einem
zweifachen Join

| Hausname | Woche | Name |
|------------|-------|---------|
| Arethoussa | 17 | Wyss |
| Arethoussa | 31 | Meier |
| Arethoussa | 32 | Meier |
| Atrium | 25 | Gessner |
| Atrium | 26 | Gessner |
| Atrium | 27 | Gessner |
| Malia | 33 | Bircher |
| Malia | 34 | Bircher |
| Malia | 35 | Bircher |
| Paphos | 28 | Aeby |
| Paphos | 29 | Aeby |
| Paphos | 31 | Zumsteg |
| Paphos | 32 | Zumsteg |
| Paphos | 33 | Zumsteg |

Für diese Abfrage benötigen Sie die Tabellen HAUS, BELEGUNG und KUNDE:

```
SELECT      Hausname, Woche, Name
FROM        Haus, Belegung, Kunde
```

```

WHERE      Haus.HausId=Belegung.HausId AND
           Kunde.KundenId=Belegung.KundenId
ORDER BY   Hausname, Woche;

```

Erzeugen Sie diese Abfrage und speichern Sie das Resultat unter dem Namen Buchungsdaten ab.

Bei einer geschachtelten Abfrage enthält ein SQL-Befehl weitere SQL-Befehle. Interessieren Sie sich beispielsweise für die Häuser auf der Insel Kreta, so können Sie mit einem inneren SQL-Befehl die Inselnummer von Kreta ermitteln und diese an den äußeren SQL-Befehl weiterreichen:

```

SELECT      Hausname
FROM        Haus
WHERE       InselId = (SELECT      InselId
                       FROM        Insel
                       WHERE        Inselname = 'Kreta');

```

Erzeugen und testen Sie diese Abfrage, wobei Sie jeweils verschiedene Inselnamen in die Abfrage einsetzen.

Noch mächtiger wird eine geschachtelte SQL-Abfrage, wenn der innere Befehl nicht mit einem einzigen Vergleichswert arbeitet, sondern nacheinander mit mehreren Werten, die aus den Tupeln der äußeren Abfrage stammen. Interessieren Sie sich für Häuser, die in den Wochen 31 bis 33 frei sind, so können Sie ein geschachteltes SQL formulieren:

```

SELECT      HausId, Hausname
FROM        Haus
WHERE       NOT EXISTS (SELECT *
                       FROM      Belegung
                       WHERE      Belegung.HausId =
                                Haus.HausId
                                AND Woche >= 31
                                AND Woche <= 33);

```

Access liefert Ihnen das korrekte Resultat aus der obigen Tabelle der Buchungsdaten:

| HausId | Hausname |
|--------|----------|
| 4 | Atrium |
| 5 | Pegasus |

Wir zeigen Ihnen noch zwei Techniken, durch die Sie die Möglichkeiten der Sprache SQL weiter ausschöpfen. Sie können nämlich Abfragen selbst wie Tabellen behandeln:

```

SELECT      *
FROM        Buchungsdaten
WHERE       Name='Meier';

```

Dadurch erhalten Sie die Buchungssätze der Kunden mit Namen Meier:

| Hausname | Woche | Name |
|------------|-------|-------|
| Arethoussa | 31 | Meier |
| Arethoussa | 32 | Meier |

Access gestattet Ihnen zudem Abfragen mit variablen Vergleichswerten. Diese müssen Sie erst beim Starten der Abfrage mit Inhalt füllen. Dadurch können Sie die obige Abfrage so verallgemeinern, dass sie für jeden Kunden funktioniert.

Ersetzen Sie in Ihrer Abfrage den Vergleichswert Meier durch eine Eingabeaufforderung in eckigen Klammern:

```

SELECT      *
FROM        Buchungsdaten
WHERE       Name=[Name des Kunden eingeben:];

```

Der Text in eckigen Klammern erscheint beim Starten der Abfrage in einem Fenster, in das der noch fehlende Vergleichswert eingetippt werden kann:



Access setzt den eingetippten Wert in die Abfrage ein, bevor es den SQL-Befehl ausführt.

Mit Hilfe dieser Technik können Sie die obigen beiden geschachtelten Befehle so abändern, dass sie für beliebige Inseln oder Zeiträume gelten.

Glossar

Abfragesprache

Eine relationale Abfragesprache erlaubt, Tabellen durch die Angabe von Selektionsbedingungen mengenorientiert zu bearbeiten, d.h., das Resultat einer Abfrage ist immer eine Menge von Tabellentupeln.

Aggregation

Aggregation ist das Zusammenfügen von Entitätsmengen zu einem Ganzen. Die Aggregationsstruktur kann netzwerkartig oder hierarchisch (Stückliste) sein.

Anomalie

Anomalien sind von der Realität abweichende Sachverhalte, die bei Einfüge-, Änderungs- und Löschooperationen in einer Datenbank entstehen können.

Assoziation

Unter einer Assoziation von einer Entitätsmenge zu einer zweiten versteht man die Bedeutung der Beziehung in dieser Richtung. Jede Assoziation kann durch einen Assoziationstyp gewichtet werden, der die Mächtigkeit der Beziehungsrichtung angibt.

Baum

Ein Baum ist eine Datenstruktur, bei der jeder Knoten außer dem Wurzelknoten genau einen Vorgängerknoten besitzt und bei dem zu jedem Blatt ein eindeutiger Weg zur Wurzel existiert.

Cursorverwaltung

Die Cursorverwaltung ermöglicht, mit Hilfe eines Zeigers eine Menge von Tupeln satzweise zu verarbeiten.

Data-Dictionary-System

Ein Data-Dictionary-System dient der Beschreibung und Dokumentation von Datenelementen, Datenbankstrukturen, Transaktionen etc. sowie deren Verknüpfungen untereinander.

Data Warehouse

Ein Data Warehouse ist ein multidimensionales Datenbanksystem, das unterschiedliche Analyseoperationen auf dem mehrdimensionalen Würfel zulässt.

Datenadministrator

Der Datenadministrator verwaltet mit Hilfe des Data-Dictionary-Systems die für das Unternehmen gültigen Beschreibungen von Daten und Funktionen.

Datenbankmanagementsystem

Ein Datenbankmanagementsystem besteht aus einer Speicherungs- und einer Verwaltungskomponente. Die Speicherungskomponente erlaubt, Daten und Beziehungen abzulegen; die Verwaltungskomponente stellt Funktionen und Sprachmittel zur Pflege und Verwaltung der Daten zur Verfügung.

Datenbankschema

Unter einem relationalen Datenbankschema versteht man die formale Spezifikation der Datenbanken und Tabellen unter Angabe von Schlüssel- und Nichtschlüsselmerkmalen sowie von Integritätsbedingungen.

Datenmodell

Ein Datenmodell beschreibt auf strukturierte und formale Art die für ein Informationssystem notwendigen Daten und Datenbeziehungen.

Datenschutz

Unter Datenschutz versteht man den Schutz der Daten vor unbefugtem Zugriff und Gebrauch.

Datensicherheit

Bei der Datensicherheit geht es um technische und programmäßige Vorkehrungen gegen Verfälschung, Zerstörung oder Verlust von Datenbeständen.

Datenunabhängigkeit

Bei Datenbanksystemen spricht man von Datenunabhängigkeit, wenn die Daten von den Anwendungsprogrammen mittels Systemfunktionen getrennt bleiben.

Deadlock

Ein Deadlock ist eine Verklemmung, d.h. eine gegenseitige Behinderung oder Blockierung von Transaktionen im Mehrbenutzerbetrieb. Deadlocks müssen vom Datenbankmanagementsystem erkannt und aufgelöst werden.

Endbenutzer

Ein Endbenutzer ist ein Anwender in der Fachabteilung des Unternehmens, der Grundkenntnisse in Informatik besitzt.

Entität

Entitäten entsprechen Objekten der realen Welt oder unserer Vorstellung. Sie werden durch Merkmale charakterisiert und zu Entitätsmengen zusammengefasst.

Entitäten-Beziehungsmodell

Das Entitäten-Beziehungsmodell ist ein Datenmodell, das Datenklassen (Entitätsmengen) und Beziehungen freilegt. Entitätsmengen werden grafisch durch Rechtecke, Beziehungsmengen durch Rhomben dargestellt.

Erblast

Bei Datenbanken spricht man von einer Erblast, wenn historisch gewachsene Unzulänglichkeiten in den Datenstrukturen vorliegen. Dies betrifft auch Datenbestände, die bei einem Wechsel des Datenbanksystems nicht automatisch bereinigt werden können.

Generalisation

Unter Generalisation versteht man das Verallgemeinern von Entitätsmengen zu einer übergeordneten Entitätsmenge; die Subentitätsmengen der Generalisationshierarchie werden Spezialisierungen genannt.

Hashing

Hashing ist eine gestreute Speicherorganisation, bei der aufgrund einer Transformation (Hash-Funktion) aus den Schlüsseln direkt die zugehörigen Adressen der Datensätze berechnet werden.

Index

Ein Index ist eine physische Datenstruktur, die für ausgewählte Merkmale einer Tabelle die internen Adressen der Datensätze liefert.

Integritätsbedingung

Integritätsbedingungen sind formale Spezifikationen über Schlüssel, Merkmale und Wertebereiche. Sie dienen dazu, die Widerspruchsfreiheit der Daten zu gewährleisten.

Manipulationssprache

Eine relationale Manipulationssprache dient zum Einfügen, Ändern und Löschen von Tabelleneinträgen, wobei die Datenbankoperationen auf Mengen wirken können.

Migration

Unter der Migration von Datenbankprogrammen versteht man den rechnergestützten Wechsel von einem Datenbanksystem zu einem anderen, unter Konvertierung der Daten wie der Anwendungen vom herkömmlichen System ins Zielsystem.

Normalform

Normalformen sind Regeln, mit denen innerhalb von Tabellen Abhängigkeiten freigelegt werden können, zur Vermeidung redundanter Informationen und damit zusammenhängender Anomalien.

Nullwert

Ein Nullwert ist ein Datenwert, der dem Datenbanksystem zur Zeit noch nicht bekannt ist.

Objektorientierung

Bei der Objektorientierung werden die Daten durch geeignete Methoden gekapselt. Zudem lassen sich Eigenschaften von Datenklassen vererben.

Optimierung

Unter der Optimierung einer Datenbankabfrage versteht man das Umformen des entsprechenden Ausdrucks (algebraische Optimierung) sowie das Ausnutzen von Speicher- und Zugriffsstrukturen zwecks Reduktion des Berechnungsaufwandes.

Recovery

Recovery bedeutet das Wiederherstellen eines korrekten Datenbankzustandes nach einem Fehlerfall.

Redundanz

Die mehrfache Speicherung desselben Sachverhaltes in einer Datenbank wird als Redundanz bezeichnet.

Relationenalgebra

Die Relationenalgebra bildet den formalen Rahmen für die relationalen Datenbanksprachen. Sie setzt sich aus den Operatoren Vereinigung, Subtraktion, kartesisches Produkt, Projektion und Selektion zusammen.

Relationenkalkül

Der Relationenkalkül basiert auf der Aussagenlogik, wobei neben der logischen Verknüpfung von Prädikaten auch Quantoren («für alle gilt ...» oder «es existiert ...») zugelassen sind.

Relationenmodell

Das Relationenmodell ist ein Datenmodell, das sowohl Daten als auch Datenbeziehungen in Form von Tabellen ausdrückt.

Schlüssel

Ein Schlüssel ist eine minimale Merkmalskombination, die alle Tupel innerhalb einer Tabelle eindeutig identifiziert.

Selektion

Die Selektion ist eine Datenbankoperation, die aufgrund einer benutzerspezifisierten Bedingung die entsprechenden Tupel einer Tabelle bereitstellt.

SQL

SQL (Structured Query Language) ist die wichtigste relationale Abfrage- und Manipulationssprache; sie wurde durch die ISO (International Organization for Standardization) normiert.

Synchronisation

Beim Mehrbenutzerbetrieb versteht man unter der Synchronisation die Koordination gleichzeitiger Zugriffe auf eine Datenbank. Bei der pessimistischen Synchronisation werden Konflikte parallel ablaufender Transaktionen von vornherein verhindert, bei der optimistischen werden konfliktträchtige Transaktionen im Nachhinein zurückgesetzt.

Tabelle

Eine Tabelle (Relation) ist eine Menge von Tupeln (Datensätzen) bestimmter Merkmalskategorien, wobei ein Merkmal oder eine Merkmalskombination die Tupel innerhalb der Tabelle eindeutig identifiziert.

Transaktion

Eine Transaktion ist eine Folge von Operationen, die atomar, konsistent, isoliert und dauerhaft ist. Die Transaktionsverwaltung dient dazu, mehreren Benutzern ein konfliktfreies Arbeiten zu ermöglichen.

Unschärfe Datenbank

Eine unscharfe Datenbank unterstützt unvollständige, vage oder unpräzise Sachverhalte durch Anwendung der Fuzzy Logic.

Verbund

Ein Verbund ist eine Datenbankoperation, die zwei Tabellen über ein gemeinsames Merkmal verbindet und eine Resultattabelle erzeugt.

XML

Die Auszeichnungssprache XML (eXtensible Markup Language) beschreibt semi-strukturierte Daten, Inhalt und Form, auf hierarchische Art und Weise.

Zweiphasen-Freigabeprotokoll

Das Zweiphasen-Freigabeprotokoll garantiert, dass bei einer verteilten Datenbank alle lokalen Transaktionen mit Erfolg enden und die Datenbestände korrekt nachführen oder dass überhaupt keine Wirkung in der Datenbank erzielt wird.

Zweiphasen-Sperrprotokoll

Das Zweiphasen-Sperrprotokoll untersagt es einer Transaktion, nach dem ersten Entsperren eines Datenbankobjektes eine weitere Sperre anzufordern.

Fachbegriffe englisch/deutsch

| | |
|----------------------------------|--|
| access path | Zugriffspfad |
| after image | Kopie nach Änderung |
| aggregation | Aggregation, Zusammenfassung |
| association | Assoziation, Verknüpfung |
| atomicity | Atomarität, unteilbare Einheit |
| attribute | Merkmal, Attribut |
| before image | Kopie vor Änderung |
| B-tree | B-Baum |
| B*-tree | B*-Baum |
| built-in function | eingebaute Funktion |
| candidate key | Schlüsselkandidat |
| cascaded deletion | fortgesetzte Löschung |
| checkpoint | Sicherungspunkt |
| commit | Datenbankänderung freigeben |
| concurrency control | Synchronisation |
| conditional | bedingt |
| consistency | Konsistenz |
| corporate-wide data architecture | unternehmensweite Daten- architektur |
| create | erstellen |
| cursor | Zeiger |
| database management system | Datenbankmanagementsystem |
| database schema | Datenbankschema |
| data definition language | Datendefinitionssprache |
| data dictionary system | Data-Dictionary-System |
| data manipulation language | Datenmanipulationssprache |
| data mining | Suche nach wertvollen Informa- tionen |
| data model | Datenmodell |
| data protection | Datenschutz |
| data security | Datensicherheit |

| | |
|---------------------------|---|
| data warehouse | Datenbank für Entscheidungs- unterstützung |
| date | Datum |
| deadlock | Verklemmung, Blockierung |
| declare | vereinbaren |
| deductive database | deduktive Datenbank |
| delete | löschen |
| descriptive language | deskriptive Sprache |
| design | Entwurf, entwerfen |
| difference | Differenz, Subtraktion |
| distributed database | verteilte Datenbank |
| domain | Wertebereich |
| durability | Dauerhaftigkeit |
| enduser | Endbenutzer |
| entity | Entität, Ganzheit |
| entity-relationship model | Entitäten-Beziehungsmodell |
| entity set | Entitätsmenge |
| equi-join | Gleichheitsverbund |
| exclusive lock | exklusive Sperre |
| fetch | einbringen |
| foreign key | Fremdschlüssel |
| functional dependency | funktionale Abhängigkeit |
| fuzzy database | unscharfe Datenbank |
| generalization | Generalisation, Verallgemeinerung |
| grant | bewilligen |
| grid file | Gitterdatei |
| hash-function | Schlüsseltransformation |
| hashing | Adressberechnung |
| identification key | Identifikationsschlüssel |
| index | Index |
| information system | Informationssystem |
| insert | einfügen |
| integrity | Integrität, Widerspruchsfreiheit |
| intersection | Durchschnitt |
| is-a-structure | Generalisationsstruktur |
| isolation | Isolation |
| join | Verbund, zusammenfügen |
| key | Schlüssel |
| key hashing | Schlüsseltransformation |
| knowledge base | Wissensbank |
| lock | Sperre, sperren |

| | |
|----------------------------------|---------------------------------|
| locking protocol | Sperrprotokoll |
| log | Logbuch, aufzeichnen |
| log file | Logdatei |
| loop | Schleife |
| manipulation language | Manipulationssprache |
| multi-dimensional data structure | mehrdimensionale Datenstruktur |
| multi-valued dependency | mehrwertige Abhängigkeit |
| nested join | geschachtelter Verbund |
| normal form | Normalform |
| null value | Nullwert |
| object-relational database | objektrelationale Datenbank |
| optimistic concurrency control | optimistische Synchronisation |
| optimization | Optimierung |
| page | Datenseite |
| part-of-structure | Stückliste |
| performance | Leistung, Performance |
| pessimistic concurrency control | pessimistische Synchronisation |
| point query | Punktfrage |
| precedence graph | Präzedenzgraph |
| primary key | Primärschlüssel |
| procedural language | prozedurale Sprache |
| projection | Projektion |
| query language | Abfragesprache |
| query tree | Anfragebaum |
| range query | Bereichsfrage |
| record | Datensatz |
| recovery | Wiederherstellungsverfahren |
| redundancy | Redundanz, mehrfaches Vorkommen |
| referential integrity | referenzielle Integrität |
| relation | Relation, Tabelle |
| relational algebra | Relationenalgebra |
| relational calculus | Relationenkalkül |
| relationship | Beziehung |
| repeating group | Wiederholungsgruppe |
| restart | Wiederanlaufverfahren |
| restricted deletion | restriktive Löschung |
| retrieve | erhalten |
| revoke | zurücknehmen |
| role | Rolle |
| rollback | zurücksetzen |

| | |
|----------------------------|--------------------------------------|
| save | sicherstellen |
| selection | Selektion, Auswahl |
| serializability | Serialisierbarkeit |
| snapshot | periodisch erstellter Tabellenauszug |
| sort-merge join | Sortier-Verschmelzungsverbund |
| synchronization | Synchronisation |
| temporal database | temporale Datenbank |
| time | Zeit |
| transaction | Transaktion |
| transitive dependency | transitive Abhängigkeit |
| transitive closure | transitive Hülle |
| tree | Baum |
| two-phase commit protocol | Zweiphasen-Freigabeprotokoll |
| two-phase locking protocol | Zweiphasen-Sperrprotokoll |
| union | Vereinigung |
| unlock | entsperren |
| update | verändern, aktualisieren |
| view | Sicht |

Literaturverzeichnis

- Astrahan M. M. et al. (1976) System R: A Relational Approach to Data Base Management. ACM Transactions on Database Systems, Vol.1, No. 2, pp. 97–137
- Balzert H. (Hrsg.) (1993) CASE-Systeme und Werkzeuge. Bibliographisches Institut
- Balzert H. (2004) Lehrbuch der Objektmodellierung – Analyse und Entwurf. Spektrum Akademischer Verlag
- Bayer R. (1972) Symmetric Binary B-Trees: Data Structures and Maintenance Algorithms. Acta Informatica, Vol. 1, No. 4, pp. 290–306
- Beaulieu A. (2006): Einführung in SQL. O'Reilly
- Bernstein P. A. et al. (1987) Concurrency Control and Recovery in Database Systems. Addison-Wesley
- Berson A., Smith S. (1997) Data Warehousing, Data Mining and OLAP. McGraw-Hill
- Bertino E., Martino L. (1993) Object-Oriented Database Systems. Addison-Wesley
- Biethahn et al. (2000) Ganzheitliches Informationsmanagement (Band II: Daten- und Entwicklungsmanagement). Oldenbourg
- Blaha M., Rumbaugh J. (2004) Object Oriented Modelling and Design with UML. Prentice-Hall
- Booch G. (2006) Object-Oriented Analysis and Design with Applications. Benjamin/Cummings
- Bordogna G., Pasi G. (Eds.) (2000) Recent Issues on Fuzzy Databases. Physica-Verlag
- Bosc P., Kacprzyk J. (Eds.) (1995) Fuzzyness in Database Management Systems. Physica-Verlag

- Brodie M. L., Stonebraker M. (1995) Migrating Legacy Systems – Gateways, Interfaces & The Incremental Approach. Morgan-Kaufmann
- Castano S. et al. (1995) Database Security. Addison-Wesley
- Cattell R. G. G. (1994) Object Data Management – Object-Oriented and Extended Relational Database Systems. Addison-Wesley
- Ceri S., Pelagatti G. (1985) Distributed Databases – Principles and Systems. McGraw-Hill
- Chen G. (1992) Design of Fuzzy Relational Databases Based on Fuzzy Functional Dependency. Ph.D. Dissertation Nr. 84, Leuven Belgium
- Chen G. (1998) Fuzzy Logic in Data Modeling – Semantics, Constraints, and Database Design. Kluwer Academic Publishers
- Chen P. P.-S. (1976) The Entity-Relationship Model – Toward an Unified View of Data. ACM Transactions on Database Systems, Vol. 1, No. 1, pp. 9–36
- Clifford J., Warren D. S. (1983) Formal Semantics for Time in Databases. ACM Transactions on Database Systems, Vol. 8, No. 2, pp. 214–254
- Clocksin W. F., Mellish C. S. (1994) Programming in Prolog. Springer
- Coad P., Yourdon E. (1991) Object-Oriented Design. Yourdon Press
- Codd E. F. (1970) A Relational Model for Large Shared Data Banks. Communications of the ACM, Vol. 13, No. 6, pp. 377–387
- Connolly T., Begg C. (2004) Database Systems – A Practical Approach to Design, Implementation and Management. Addison-Wesley
- Cremers A. B. et al. (1994) Deduktive Datenbanken – Eine Einführung aus der Sicht der logischen Programmierung. Vieweg
- Dadam P. (1996) Verteilte Datenbanken und Client/Server-Systeme. Springer
- Darwen H., Date C. J. (1997) A Guide to the SQL Standard. Addison-Wesley
- Date C. J. (1986) Relational Database – Selected Writings. Addison-Wesley
- Date C. J. (2004) An Introduction to Database Systems. Addison-Wesley

- Dippold R., Meier A., Schnider W., Schwinn K. (2005) Unternehmensweites Datenmanagement – Von der Datenbankadministration bis zum Informationsmanagement. Vieweg
- Dittrich K. R. (Ed.) (1988) Advances in Object-Oriented Database Systems. Lecture Notes in Computer Science, Vol. 334, Springer
- Dürr M., Radermacher K. (1990) Einsatz von Datenbanksystemen – Ein Leitfaden für die Praxis. Springer
- Dutka A. F., Hanson H. H. (1989) Fundamentals of Data Normalization. Addison-Wesley
- Elmasri R., Navathe S. B. (2006) Fundamentals of Database Systems. Addison-Wesley
- Etzion O., Jajodia S., Sripada S. (Eds.) (1998) Temporal Databases – Research and Practice. Lecture Notes in Computer Science, Springer
- Fagin R. (1979) Normal forms and relational database operators. Proc. Int. Conf. on Management of Data, SIGMOD, pp. 153–160
- Ferstl O. K., Sinz E. J. (1991) Ein Vorgehensmodell zur Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM). Wirtschaftsinformatik, Jhrg. 33, Nr. 6, S. 477–491
- Findler N. V. (Ed.) (1979) Associative Networks – Representation and Use of Knowledge by Computers. Academic Press
- Gadia S. K. (1988) A Homogeneous Relational Model and Query Languages for Temporal Databases. ACM Transactions on Database Systems, Vol. 13, No. 4, pp. 418–448
- Gallaire H. et al. (1984) Logic and Databases: A Deductive Approach. ACM Computing Surveys, Vol. 16, No. 2, pp. 153–185
- Gardarin G., Valduriez P. (1989) Relational Databases and Knowledge Bases. Addison-Wesley
- Gemünden G., Schmitt M. (1991) Datenmanagement in deutschen Großunternehmen – Theoretischer Ansatz und empirische Untersuchung. Information Management, Jhrg. 6, Nr. 4, S. 22–34
- Geppert A. (2002) Objektrelationale und objektorientierte Datenbankkonzepte und -systeme. dpunkt
- Gillenson M. L. (1990) Physical Design Equivalences in Database Conversion. Communications of the ACM, Vol. 33, Nr. 8, pp. 120–131

- Gluchowski P., Gabriel R., Chamoni P. (1997) Management Support Systeme – Computergestützte Informationssysteme für Führungskräfte und Entscheidungsträger. Springer
- Gray J., Reuter A. (1993) Transaction Processing – Concepts and Techniques. Morgan-Kaufmann
- Härder T. (1978) Implementierung von Datenbanksystemen. Hanser
- Härder T., Rahm E. (2001) Datenbanksysteme – Konzepte und Techniken der Implementierung. Springer
- Härder T., Reuter A. (1983) Principles of Transaction-Oriented Database Recovery. ACM Computing Surveys, Vol. 15, Nr. 4, pp. 287–317
- Heinrich L. J., Lehner F. (2005) Informationsmanagement – Planung, Überwachung und Steuerung der Informationsinfrastruktur. Oldenbourg
- Heuer A. (1997) Objektorientierte Datenbanken. Addison-Wesley
- Heuer A., Saake G. (2000) Datenbanken – Konzepte und Sprachen. MITP
- Hitz M., Kappel G., Kapsammer E., Retschitzegger W. (2005) UML@Work – Objektorientierte Modellierung mit UML2. dpunkt
- Hoffer I.A., Prescott M.B., McFadden F.R. (2005) Modern Database Management. Pearson Prentice Hall
- Hughes J. G. (1991) Object-Oriented Databases. Prentice-Hall
- Hüsemann F. C. (2002) Datenbankmigration – Methodik und Softwareunterstützung. VDI Verlag
- Inmon W. H. (2002) Building the Data Warehouse. 3rd Edition. Wiley
- Jarke M., Lenzerini M., Vassiliou Y., Vassiliadis P. (2000) Fundamentals of Data Warehouses. Springer
- Kacprzyk J., Zadrozny S. (1995) FQUERY for Access – Fuzzy Querying for a Windows-Based DBMS. In: Bosc und Kacprzyk (1995), pp. 415–433
- Kazakos W., Schmidt A., Tomczyk P. (2002) Datenbanken und XML – Konzepte, Anwendungen, Systeme. Springer
- Kemper A., Eickler A. (2006) Datenbanksysteme – Eine Einführung. Oldenbourg

- Kerre E. E., Chen G. (1995) An Overview of Fuzzy Data Models. In: Bosc und Kacprzyk (1995), pp. 23–41
- Kimball R., Reeves L., Ross M. (1998) The Datawarehouse Lifecycle Toolkit. Wiley
- Kim W. (1990) Introduction to Object-Oriented Databases. The MIT Press
- Klettke M., Meyer H. (2003) XML & Datenbanken – Konzepte, Sprachen und Systeme. dpunkt
- Kuhlmann G., Müllmerstadt F. (2004) SQL – Der Schlüssel zu relationalen Datenbanken. Rowohlt
- Kurbel K., Strunz H. (Hrsg.) (1990) Handbuch der Wirtschaftsinformatik. C. E. Poeschel
- Lang S. M., Lockemann P. C. (1995) Datenbankeinsatz. Springer
- Lausen G., Vossen G. (1996) Objekt-orientierte Datenbanken: Modelle und Sprachen. Oldenbourg
- Lockemann P. C., Schmidt J. W. (Hrsg.) (1993) Datenbank-Handbuch. Springer
- Loeser H. (2001) Web-Datenbanken – Einsatz objekt-relationaler Datenbanken für Web-Informationssysteme. Springer
- Lorie R. A. et al. (1985) Supporting Complex Objects in a Relational System for Engineering Databases. In: Kim W. et al. (Ed.) Query Processing in Database Systems. Springer, pp. 145–155
- Maier D. (1983) The Theory of Relational Databases. Computer Science Press
- Maier D., Warren D. S. (1988) Computing with Logic – Logic Programming with Prolog. Benjamin/ Cummings
- Martin J. (1986) Einführung in die Datenbanktechnik. Hanser
- Martin J. (1990) Information Engineering – Planning and Analysis. Prentice-Hall
- Martin J., Odell J. J. (1992) Object-Oriented Analysis and Design. Prentice-Hall
- Martiny L., Klotz M. (1989) Strategisches Informationsmanagement. Oldenbourg
- Maurer W. D., Lewis T. G. (1975) Hash Table Methods. ACM Computing Surveys, Vol. 7, Nr. 1, pp. 5–19

- Meier A. (1987) Erweiterung relationaler Datenbanksysteme für technische Anwendungen. Informatik-Fachberichte, Bd. 135, Springer
- Meier A. (1994) Ziele und Aufgaben im Datenmanagement aus der Sicht des Praktikers. Wirtschaftsinformatik, Jhrg. 36, Nr. 5, S. 455–464
- Meier A. (1997) Datenbankmigration – Wege aus dem Datenchaos. Praxis der Wirtschaftsinformatik, Jhrg. 34, Nr. 194, S. 24–36
- Meier A. (Hrsg.) (2000) WWW & Datenbanken. Praxis der Wirtschaftsinformatik, Jhrg. 37, Heft 214, dpunkt
- Meier A., Dippold R. (1992) Migration und Koexistenz heterogener Datenbanken – Praktische Lösungen zum Wechsel in die relationale Datenbanktechnologie. Informatik-Spektrum, Jhrg. 15, Nr. 3, S. 157–166
- Meier A., Graf H., Schwinn K. (1991) Ein erster Schritt zu einem globalen Datenmodell. Information Management, Jhrg. 6, Nr. 2, S. 42–48
- Meier A., Haltinner R., Widmer-Itin B. (1993) Schutz der Investitionen beim Wechsel eines Datenbanksystems. Wirtschaftsinformatik, Jhrg. 35, Nr. 4, S. 331–338
- Meier A., Johner W. (1991) Ziele und Pflichten der Datenadministration. Theorie und Praxis der Wirtschaftsinformatik, Jhrg. 28, Nr. 161, S. 117–131
- Meier A., Savary C., Schindler G., Veryha Y. (2001) Database Schema with Fuzzy Classification and Classification Query Language. Proc. of the International Congress on Computational Intelligence – Methods and Applications, CIMA 2001, University of Wales, Bangor U.K.
- Meier A., Werro N., Albrecht M., Sarakinos M. (2005) Using a Fuzzy Classification Query Language for Customer Relationship Management. Proc. 31st International Conference on Very Large Databases (VLDB), Trondheim, Norway, pp. 1089–1096
- Meier A., Werro N. (2006) Extending a Webshop with a Fuzzy Classification Model for Online Customers. Proc. of the IADIS International Conference on e-Society 2006, Trinity College Dublin, Ireland, June 13–16, Volume I, pp. 305–312
- Meier A., Wüst T. (2003) Objektorientierte und objektrelationale Datenbanken – Ein Kompass für die Praxis. dpunkt

- Melton J., Simon A.R., (2002) SQL1999 - Understanding Relational Language Components. Morgan Kaufmann
- Mucksch H., Behme W. (Hrsg.) (1996) Das Data-Warehouse-Konzept – Architektur, Datenmodelle, Anwendungen. Gabler
- Myrach T. (2005) Temporale Datenbanken in betrieblichen Informationssystemen - Prinzipien, Konzepte, Umsetzung. Teubner
- Nievergelt J., Hinterberger H., Sevcik K. C. (1984) The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Transactions on Database Systems, Vol. 9, No. 1, pp. 38–71
- Olle T. W. et al. (1988) Information Systems Methodologies – A Framework for Understanding. Addison-Wesley
- Ortner E. et al. (1990) Entwicklung und Verwaltung standardisierter Datenelemente. Informatik-Spektrum, Jhr. 13, Nr. 1, S. 17–30
- Österle H. et al. (1991) Unternehmensführung und Informationssystem – Der Ansatz des St. Galler Informationssystem-Managements. Teubner
- Özsu M. T., Valduriez P. (1991) Principles of Distributed Database Systems. Prentice-Hall
- Panny W., Taudes (2000) Einführung in den Sprachkern SQL-99. Springer
- Paredaens J. et al. (1989) The Structure of the Relational Database Model. Springer
- Petry F. E. (1996) Fuzzy Databases – Principles and Applications. Kluwer Academic Publishers
- Pistor P. (1993) Objektorientierung in SQL3: Stand und Entwicklungstendenzen. Informatik-Spektrum, Jhr. 16, Nr. 2, S. 89–94
- Pons O., Vila M. A., Kacprzyk J. (Eds.) (2000) Knowledge Management in Fuzzy Databases. Physica-Verlag
- Rahm E. (1994) Mehrrechner-Datenbanksysteme. Addison-Wesley
- Rahm E., Vossen G. (Hrsg.) (2003) Web & Datenbanken – Konzepte, Architekturen, Anwendungen. dpunkt
- Reuter A. (1981) Fehlerbehandlung in Datenbanksystemen. Hanser
- Rothnie J. B. et al. (1980) Introduction to a System for Distributed Databases. ACM Transactions on Database Systems, Vol. 5, No. 1, pp. 1–17

- Saake G. et al. (1997) Objektdatenbanken. International Thomson Publishing
- Sauer H. (2002) Relationale Datenbanken – Theorie und Praxis inklusive SQL-2. Addison-Wesley
- Schaarschmidt R. (2001) Archivierung in Datenbanksystemen - Konzepte und Sprache. Teubner
- Scheer A.-W. (1991) Architektur integrierter Informationssysteme – Grundlagen der Unternehmensmodellierung. Springer
- Schek H.-J., Scholl M. H. (1986) The Relational Model with Relation-Valued Attributes. Information Systems, Vol. 11, No. 2, pp. 137–147
- Schindler G. (1998): Fuzzy Datenanalyse durch kontextbasierte Datenbankabfragen. Deutscher Universitäts-Verlag
- Schlageter G., Stucky W. (1983) Datenbanksysteme: Konzepte und Modelle. Teubner
- Schöning H. (2003) XML und Datenbanken – Konzepte und Systeme. Hanser
- Shenoi S., Melton A., Fan L.T. (1992) Functional Dependencies and Normal Forms in the Fuzzy Relational Database Model. Information Sciences, Vol. 60, pp. 1–28
- Silberschatz A., Korth H. F., Sudarshan S. (2005) Database Systems Concepts. McGraw-Hill
- Silverston L. (2001) The Data Model Resource Book. Vol. 2., John Wiley
- Simsion G. C, Witt G. C. (2005) Data Modeling Essentials. Morgan Kaufmann
- Smith J. M., Smith D. C. P. (1977) Database Abstractions: Aggregation and Generalization. ACM Transactions on Database Systems, Vol. 2, No. 2, pp. 105–133
- Snodgrass R. T. (1987) The Temporal Query Language TQuel. ACM Transactions on Database Systems. Vol. 12, No. 2, pp. 247–298
- Snodgrass R. T. et al. (1994) A TSQL2 Tutorial. SIGMOD-Record, Vol. 23, No. 3, pp. 27–33
- Stein W. (1994) Objektorientierte Analysemethoden – Vergleich, Bewertung, Auswahl. Bibliographisches Institut

- Störl U. (2001) Backup und Recovery in Datenbanksystemen - Verfahren, Klassifikation, Implementierung und Bewertung. Teubner
- Stonebraker M. (Ed.) (1986) The Ingres Papers. Addison-Wesley
- Stonebraker M. (1996) Object-Relational DBMS's – The Next Great Wave. Morgan-Kaufmann
- Takahashi Y. (1995) A Fuzzy Query Language for Relational Databases. In: Bosc und Kacprzyk (1995), pp. 365–384
- Tsichritzis D. C., Lochovsky F. H. (1982) Data Models. Prentice-Hall
- Türker C. (2003) SQL:1999 & SQL:2003 – Objektrelationales SQL, SQLJ & SQL/XML. dpunkt
- Ullman J. (1982) Principles of Database Systems. Computer Science Press
- Ullman J. (1988) Principles of Database and Knowledge-Base Systems. Computer Science Press
- Vetter M. (1998) Aufbau betrieblicher Informationssysteme mittels pseudo-objektorientierter, konzeptioneller Datenmodellierung. Teubner
- Vossen G. (2000) Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme. Oldenbourg
- Wedekind H. (1981) Datenbanksysteme – Eine konstruktive Einführung in die Datenverarbeitung in Wirtschaft und Verwaltung. Spektrum Akademischer Verlag
- Weikum G. (1988) Transaktionen in Datenbanksystemen – Fehlertolerante Steuerung paralleler Abläufe. Addison-Wesley
- Weikum G., Vossen G. (2002) Transactional Information Systems – Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann
- Wiederhold G. (1983) Database Design. McGraw-Hill
- Williams R. et al. (1982) R*: An Overview of the Architecture. In: Scheuermann P. (Ed.) Improving Database Usability and Responsiveness. Academic Press, pp. 1–27
- Witten I. H., Frank E. (1999) Data Mining. Morgan Kaufmann
- Zehnder C. A. (2002) Informationssysteme und Datenbanken. vdf Hochschulverlag
- Zloof M. M. (1977) Query-by-Example: A Data Base Language. IBM Systems Journal. Vol. 16, No. 4, pp. 324–343

Stichwortverzeichnis

- Abbildungsregel 28, 30f., 35f., 139f.
- Abfragesprache
 - siehe Sprache
- Abhängigkeit 38f., 41, 43, 47f.
 - funktionale 41f., 43
 - mehrwertige 40, 45f.
 - transitive 40, 43f.
 - Verbund- 47f.
- Adressberechnung
 - siehe Hashing
- Aggregation 24f., 35f., 170f.
- Algebra
 - siehe Relationenalgebra
- Anfragebaum 96f., 163
 - optimierter 99
- Anomalie 39f.
- ANSI 7
- Assoziation
 - einfache 23, 30, 33f.
 - konditionelle 23, 31
 - mehrfache 23, 31f.
 - mehrfach-konditionelle 23, 31
- Assoziationsstyp 22f., 30f.
- Atomarität 104
- Attribut
 - siehe Merkmal
- Aufzeichnungszeit 165
- Autonomie 161, 164
- Baum 95, 116f.
- B-Baum 116f.
 - B*-Baum 118
- Benutzer
 - End- 62
 - gelegentlicher 9
- Bereichsfrage 123
- Beziehung 21, 30f.
 - einfach-einfache 30f., 34
 - einfach-komplexe 31, 33
 - komplex-komplexe 31f.
- Beziehungsmenge 19, 29f., 55
- CASE 27, 58
- CREATE 77, 86, 90
- COMMIT 162
- CURSOR 82f.
- Cursor
 - konzept 62, 82f.
 - verwaltung 94, 127
- Data-Dictionary-System 61
- Data Mining 177, 187
- Data Warehouse 177
- DATE 166
- Dateiverwaltung 127
- Daten
 - administrator 61f.
 - analyse 13f., 17f., 53f.
 - architekt 61f.
 - architektur 13, 53f.
 - definitionssprache 11
 - integrität 11, 50f., 88f.
 - manipulationssprache 11
 - satz 5, 119f., 127
 - unabhängigkeit 11
- Datenbank 10



multidimensionale 173f.
 objektrelationale 169f.
 relationale 10f.
 -spezialist 61f.
 temporale 165f.
 unscharfe 178f.
 verteilte 160f.
 wissensbasierte 184f.
 Datenbankentwurf 57f.
 Datenbankmanagementsystem
 siehe Datenbanksystem
 Datenbankschema 19, 28f.
 relationales 19, 28f., 58
 Datenbanksystem
 fuzzy 178f.
 multidimensionales 173f.
 objektrelationales 169f.
 relationales 10f., 93f.
 temporales 165f.
 verteiltes 160f.
 wissensbasiertes 184f.
 Datenmodell 17f., 54
 relationales 28f., 35f.
 Datenmodellierung 17f.
 Datenschutz 85f.
 Datensicherheit 85
 Datenstruktur
 mehrdimensionale 121f., 127,
 177
 physische 57, 127
 Dauerhaftigkeit 105
 DELETE 77
 Differenz 67
 Division 65, 72
 DROP 77
 Durchschnitt 64, 66

 Einbringstrategie 128
 Eindeutigkeit 5, 51f.
 Entität 20f., 28f.
 Entitäten-Beziehungsmodell
 20f., 28f., 139f.
 Entitätsmenge 20f., 28f., 35f.,
 57
 Sub- 24f.
 Erblast 148f.
 Expertensystem 187

 Fakt 174f., 184f.
 Fehlerbehandlung 95, 124f.
 FETCH 83
 Fragment 161f.
 horizontales 161
 vertikales 161
 Fremdschlüssel 21, 31f, 88f.
 Funktion
 eingebaute 76

 Generalisation 24f., 35f., 57,
 172
 Gitter
 -datei 121f.
 -verzeichnis 122
 GRANT 87
 Grid File
 siehe Gitterdatei
 Gültigkeitszeit 165f.

 Hashing 95, 118f.
 dynamisches 120
 mit Divisionsrest 119f.
 Hash-Verfahren
 siehe Hashing
 Hülle
 transitive 187

 Identifikationsschlüssel
 siehe Schlüssel
 Index 102f.
 Indikator 174
 Information 1f., 17f.
 Informationsmanagement 2
 Informationssystem 2f., 17,
 53f.
 INSERT 77
 Integrität 11, 50f., 88f.
 referenzielle 51, 90
 Integritätsbedingung 50f., 57,

- 88f.
- ISO 7, 75, 154
- Isolation 105
- Konsistenz 57, 105, 147
- Kopie 124f.
- LOCK 110f.
- Log
 - buch 107f.
 - datei 125
- Löschung
 - fortgesetzte 52f., 90
 - restriktive 52, 90
- Mächtigkeit 23f., 73
- Manipulationssprache
 - siehe Sprache
- Mehrbenutzerbetrieb 11, 104f.
- Mehrwegbaum 116f., 127
 - blattorientierter 118
- Merkmal 4f., 20f., 41
- Methoden 172
- Migration 139f., 145f., 147f.
- Minimalität 5
- Normalform 38f., 57, 141, 171
 - Boyce-Codd- 40, 45
 - dritte 40, 44
 - erste 40, 141, 143, 171
 - fünfte 40, 49
 - Projektion-Verbund- 49
 - vierte 40, 47
 - zweite 40, 41
- Nullwert 83f.
- Objekt
 - strukturiertes 169f.
- Operator
 - binärer 97f.
 - Divisions- 72
 - generischer 172
 - mengenorientierter 63, 65f., 74
 - Projektions- 64, 68f., 99, 146
 - relationenorientierter 64, 68f., 74
 - Selektions- 64, 69f., 74, 76, 100
 - unärer 98
 - Verbund- 64, 70f., 101f., 170
- Optimierung 98f.
- Parallelität 106f., 113, 164
- Performanz 12, 58, 169
- Präzedenzgraph 108f.
- Primärschlüssel 6, 29, 51
- Prinzip
 - ACID- 105
 - Alles-oder-Nichts- 94, 104
 - der Rekursion 186
 - der Serialisierbarkeit 106
- Privileg 87
- Produkt
 - kartesisches 63, 70f.
- Projektion 64, 68, 74, 100
- Protokoll
 - Zweiphasenfreigabe- 162
 - Zweiphasensperr- 110f.
- Pufferverwaltung 128
- Punktfrage 123
- QBE 79f.
- QUEL 78f.
- READ 106f., 112, 114f.
- Recovery
 - siehe Wiederherstellungsverfahren
- Redundanz 38f., 150
- Regel 185f.
- Rekursion 186
- Relation
 - siehe Tabelle
- Relationen
 - algebra 63f., 74, 96
 - kalkül 73, 78

- modell 4f., 38f.
- Restart
 - siehe Wiederanlaufverfahren
- REVOKE 87
- ROLLBACK 125
- Rolle 32
- Rücksetzbarkeit 104, 124f.
- Schichtenmodell 126f.
- Schlüssel 5f., 116f.
 - kandidat 29, 46
 - künstlicher 6
 - mehrdimensionaler 121f.
 - zusammengesetzter 42f.
- Schlüsseltransformation
 - siehe Hashing
- Schnittstelle
 - mengenorientierte 126
 - satzorientierte 127
- Seiten
 - zugriff 121f.
 - zuordnung 128
- SELECT 7, 75f.
- Selektion 7f., 64, 69f., 75f.
- Serialisierbarkeit 106f., 109
- Sicherungspunkt 125
- Sicht 86f., 156
- Speicher
 - struktur 94, 127
 - verwaltung 94, 127
 - zuordnung 128
- Sperre
 - exklusive 109
- Sperrgröße 112
- Sperrprotokoll
 - Zweiphasen- 110f.
- Spiegelung 150f.
- Spiegelungsabbildung 151
- Sprache
 - Abfrage- 7f., 61f., 75f.
 - deskriptive 8f.
 - eingebettete 82f.
 - Manipulations- 8f., 61f., 77f.
 - mengenorientierte 7f., 65f., 83
- natürliche 9
- prozedurale 9
- relational vollständige 73f.
- SQL 7f., 75f.
- Struktur 26, 170f.
 - IS-A- 26
 - PART-OF- 26, 171
- Surrogat 170f.
- Synchronisation
 - optimistische 113
 - pessimistische 111
- Systemarchitektur 126f.
- Tabelle 4, 7f.
 - abgeleitete 185
 - System- 10, 61f.
- TIME 166
- Transaktion 94f., 104f.
- Überlaufbereich 119f.
- Übersetzung 93, 96f., 126f.
- UNLOCK 110f.
- UPDATE 77
- Verbund 64, 70f., 76, 78
 - abhängigkeit 40, 47f., 147
 - berechnung 101f.
 - geschachtelter 102f.
 - impliziter 170
 - prädikat 70f., 76, 81
 - Sortier-Verschmelzungs 103f.
 - strategie 95, 101f.
- Vereinigung 63, 66
- Vereinigungsverträglichkeit 65f.
- Verfahren
 - optimistisches 113f.
 - pessimistisches 109f.
- VIEW 86f.
 - Konzept 86
- Wertebereich 4f., 51

Widerspruchsfreiheit 105
Wiederherstellungsverfahren
124f.
Wiederholungsgruppe 41
Wissensbank 3, 187
WRITE 106f., 111f.

Zeit 159f.

Zugriffs

-pfad 57, 95, 115f.

-struktur 95, 127

Zweiphasen

-freigabeprotokoll 162

-sperrprotokoll 110

Zyklus 109