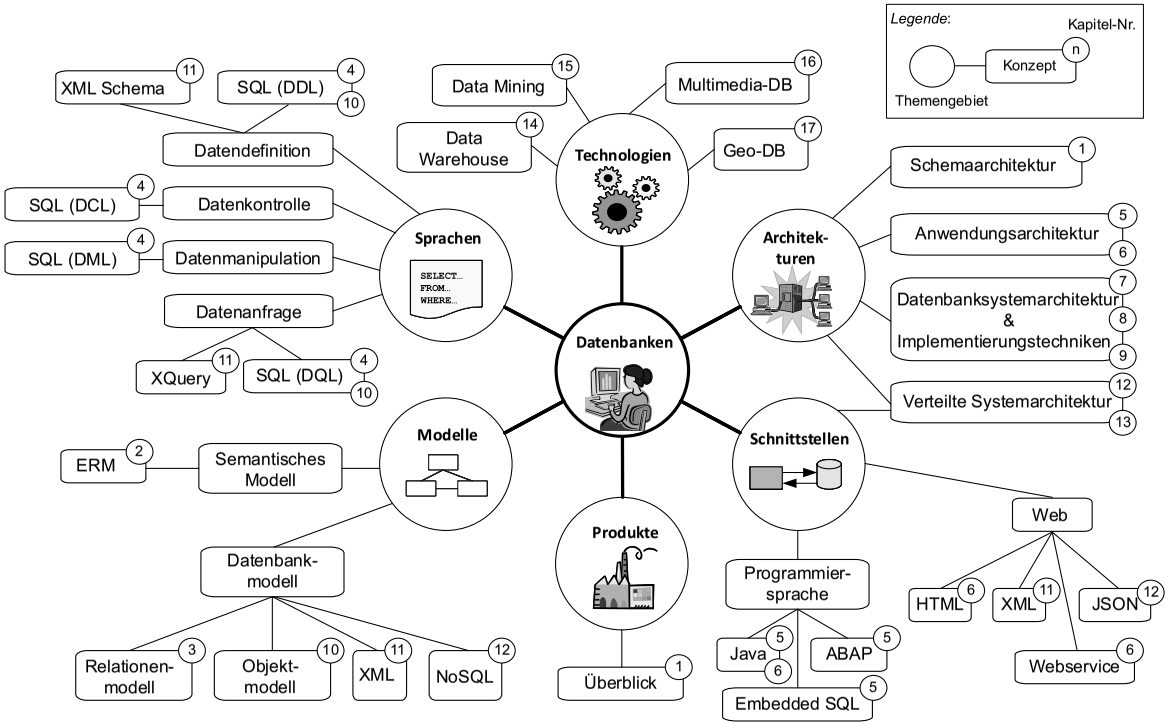
Thomas Kudraß

Taschenbuch Datenbanken

2. Auflage

HANSER


Legende:
Kapitel-Nr.
Konzept
n
Themengebiet
Datenbanken
Sprachen
SELECT...
FROM...
WHERE...
Datendefinition
XML Schema
11
SQL (DDL)
4
10
Datenkontrolle
SQL (DCL)
4
Datenmanipulation
SQL (DML)
4
Datenanfrage
XQuery
11
SQL (DQL)
4
10
Technologien
Data Mining
15
Data Warehouse
14
Multimedia-DB
16
Geo-DB
17
Architek-turen
Schemaarchitektur
1
Anwendungsarchitektur
5
6
Datenbanksystemarchitektur
&
Implementierungstechniken
7
8
9
Verteilte Systemarchitektur
12
13
Schnittstellen
Web
HTML
6
XML
11
JSON
12
Webservice
6
Programmier-sprache
Java
5
6
ABAP
5
Embedded SQL
5
Modelle
ERM
2
Semantisches Modell
Datenbank-modell
Relationen-modell
3
Objekt-modell
10
XML
11
NoSQL
12
Produkte
Überblick
1

1	Datenbanken: Grundlagen und Überblick	19
2	Informationsmodellierung	44
3	Relationales Datenmodell	67
4	Die Datenbanksprache SQL	99
5	Datenbank-Anwendungsprogrammierung	144
6	Datenbanken im Web	189
7	Komponenten eines Datenbankmanagementsystems	227
8	Dateiorganisation und Indexe	269
9	Optimierung von Datenbanken und Leistungsbewertung	287
10	Objektrelationale Datenbanken	322
11	XML und Datenbanken	345
12	NoSQL-Datenbanksysteme	368
13	Verteilte und föderierte Datenbanksysteme	394
14	Data Warehouse	430
15	Data Mining	461
16	Multimedia-Datenbanken	478
17	Geodatenbanken	499

Kudraß
Taschenbuch Datenbanken

Bleiben Sie auf dem Laufenden!

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

www.hanser-fachbuch.de/newsletter

Herausgeber

Prof. Dr. Thomas Kudraß, Hochschule für Technik, Wirtschaft und Kultur Leipzig

Autoren

Prof. Dr. Thomas Brinkhoff
Jade Hochschule Wilhelmshaven/Oldenburg/Elsfleth

Prof. Dr. Olaf Herden
Duale Hochschule Baden-Württemberg Stuttgart, Campus Horb

Prof. Dr. Harm Knolle
Hochschule Bonn-Rhein-Sieg

Prof. Dr. Thomas Kudraß
Hochschule für Technik, Wirtschaft und Kultur Leipzig

Prof. Dr. Klaus Meyer-Wegener
Friedrich-Alexander-Universität Erlangen-Nürnberg

Prof. Dr. Thomas Rakow
Fachhochschule Düsseldorf

Prof. Dr. Norbert Ritter
Universität Hamburg

Prof. Dr. Kai-Uwe Sattler
Technische Universität Ilmenau

Prof. Dr. Petra Sauer
Beuth Hochschule für Technik Berlin

Prof. Dr. Bernhard Schiefer
Hochschule Kaiserslautern, Campus Zweibrücken

Dr. Harald Schöning
Software AG Darmstadt

Prof. Dr. Uta Störl
Hochschule Darmstadt

Dr. Can Türker
Functional Genomics Center Zürich

Taschenbuch Datenbanken

2., neu bearbeitete Auflage

herausgegeben von

Thomas Kudraß

Mit 163 Bildern und 30 Tabellen

Fachbuchverlag Leipzig
im Carl Hanser Verlag

MIX
Papier aus verantwortungsvollen Quellen
FSC® C013736
FSC
www.fsc.org

Alle in diesem Buch enthaltenen Programme, Verfahren und elektronischen Schaltungen wurden nach bestem Wissen erstellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund ist das im vorliegenden Buch enthaltene Programm-Material mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor und Verlag übernehmen infolgedessen keine Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieses Programm-Materials oder Teilen davon entsteht.

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.

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.

ISBN 978-3-446-43508-7
E-Book-ISBN 978-3-446-44026-5

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

© 2015 Carl Hanser Verlag München
Internet: http://www.hanser-fachbuch.de

Lektorat: Franziska Jacob, M. A.
Herstellung: Der Buch*macher*, Arthur Lenner, München
Satz: Satzherstellung Dr. Naake, Brand-Erbisdorf
Coverrealisierung: Stephan Rönigk
Druck und Bindung: Kösel, Krugzell
Printed in Germany

Vorwort

Informationen sind der Rohstoff des Informationszeitalters, in dem wir heute leben. Der größte Teil davon wird auf digitalen Medien gespeichert, wobei Datenbanksysteme zur Verwaltung der Inhalte eine zentrale Rolle spielen. Datenbanken bilden quasi das Rückgrat von Informationssystemen in allen Bereichen – oft gar nicht sichtbar für den Endanwender. So ist es nicht verwunderlich, dass sich das Fach Datenbanken in den letzten Jahren als Kerndisziplin der Informatik etabliert hat, zugleich aber auch immer stärker in die Ausbildung anderer Berufe eingeflossen ist.

Das Taschenbuch Datenbanken erscheint in der bekannten Taschenbuchreihe des Fachbuchverlags Leipzig, von dem auch die Anregung für ein solches Buch kam. Das gesamte Buch ist als Nachschlagewerk für das heutzutage etablierte Wissen im Fach Datenbanken konzipiert. Grundlegende Fakten und Zusammenhänge werden in kompakter und übersichtlicher Form dargestellt. Alle Begriffe sind darüber hinaus in einem umfangreichen Sachwortverzeichnis zu finden. An vielen Stellen gibt es Querverweise und Empfehlungen zu weiterführender Literatur.

Im Taschenbuch werden die wichtigsten theoretischen Grundlagen von Datenbanken behandelt, zugleich aber auch eine Vielzahl von Technologien, die Bestandteil moderner Datenbanksysteme sind. Angesichts des begrenzten Umfangs konnten dabei nicht alle Themen berücksichtigt werden, insbesondere wenn sie noch Gegenstand der Forschung sind.

Das Buch ist in 17 Kapitel gegliedert. Die Kapitel 1 bis 6 dienen vor allem dem Einstieg in das Gebiet und beinhalten Grundlagenwissen über den Datenbankentwurf und die Entwicklung von Datenbankanwendungen. Dabei wird der Datenbanksprache SQL breiter Raum eingeräumt, ebenso der Verbindung von Web und Datenbanken. Die Kapitel 7 bis 9 behandeln die interne Arbeitsweise von Datenbanksystemen, deren Verständnis für die Datenbankoptimierung und -administration notwendig ist. Die Kapitel 10 bis 13 beinhalten Basistechnologien, die für den fortgeschrittenen Datenbanknutzer interessant sind, z. B. objektrelationale und XML-Datenbanken. Zunehmende Bedeutung in Unternehmen erlangt auch die Verteilung und Integration von Datenbanken, so dass diesem Thema ein gesondertes Kapitel gewidmet ist. Die Kapitel 14 bis 17 behandeln Datenbanktechnologien, wie sie in speziellen Informationssystemen genutzt werden. So werden Data Warehouse und Data Mining als Basis entscheidungsunterstützender Systeme dargestellt. Multimediale Daten und Geodaten unterscheiden sich

gegenüber „klassischen“ Daten und werden deshalb in eigenen Kapiteln beschrieben.

Die Idee des Buches besteht darin, vor allem allgemeingültige Konzepte und Technologien darzustellen, nicht aber Produkte einzelner Anbieter. Aufgrund der weitgehenden Standardisierung von Sprachen und Schnittstellen bildet das im Buch präsentierte Wissen eine solide Grundlage zum Verständnis herstellerspezifischer Systeme. Somit eignet es sich für jeden an Datenbanken interessierten Leser: ob Student, Praktiker oder „Quereinsteiger“. Für weiterführende Informationen und Aktualisierungen sei noch auf die zugehörige Webseite verwiesen.

Als Herausgeber möchte ich mich ganz herzlich bei allen Autoren bedanken, die zu den einzelnen Kapiteln ihr spezielles Fachwissen beigetragen haben. Mein besonderer Dank geht an Erika Hotho und Franziska Kaufmann vom Fachbuchverlag Leipzig für die hervorragende Betreuung und die stets angenehme Zusammenarbeit, ebenso an PTP-Berlin für das gelungene Layout. Danken möchte ich auch Hans-Peter Leidhold und Dörte König für Korrekturhinweise zum Manuskript. Gleichfalls ein Dank an Pieter Hauffe für seine Unterstützung bei der technischen Vorbereitung des Buchprojekts.

Ich hoffe, dass das Buch für den Leser zu einer wertvollen Hilfe im Begriffsdschungel der Datenbankwelt wird. Anregungen, Verbesserungsvorschläge oder Kritik werden jederzeit gern entgegengenommen.

Leipzig, im August 2007 Thomas Kudraß

Vorwort zur 2. Auflage

In dieser 2. Auflage wurden die Ausführungen überarbeitet und aktuelle Entwicklungen aus dem Bereich „Big Data“ aufgegriffen. NoSQL-Datenbanken, Column Stores in Data-Warehouse-Systemen und Cloud-Datenbanken sind dabei einige der neuen Themen, die ins Buch aufgenommen wurden. Ein Dank gilt den Rezensenten der Erstauflage, von denen zahlreiche Hinweise berücksichtigt wurden. Ebenso möchte ich mich herzlich für die angenehme Zusammenarbeit bedanken: bei Franziska Jacob vom Fachbuchverlag Leipzig, bei Arthur Lenner und Dr. Steffen Naake, die für Lektorat, Herstellung und Satzlegung verantwortlich waren. Kommentare zu dieser 2. Auflage werden wiederum gern entgegengenommen.

Leipzig, im Januar 2015 Thomas Kudraß

Inhaltsverzeichnis

1 Datenbanken: Grundlagen und Überblick **19**
1.1 *Dateien und Dateisysteme* 19
1.2 *Terminologie* 20
1.3 *Merkmale eines DBMS* 23
1.3.1 Aufgaben eines DBMS 23
1.3.2 Vorteile des Datenbankeinsatzes 24
1.3.3 Nachteile von Datenbanksystemen 25
1.3.4 Produkte 25
1.4 *Architektur eines Datenbanksystems* 26
1.4.1 Architekturen 26
1.4.2 Schemaarchitektur 27
1.4.2.1 Datenbankschema 27
1.4.2.2 Drei-Ebenen-Architektur 28
1.4.3 Datenunabhängigkeit 29
1.5 *Benutzerrollen bei Entwurf und Betrieb von Datenbanken* 30
1.6 *Datenbanken und Informationssysteme* 32
1.7 *Fachgebiet Datenbanken im Überblick* 33
1.7.1 Themenbereiche und Zusammenhänge 33
1.7.2 Einordnung des Fachs innerhalb der Informatik 35
1.8 *Historische Entwicklung* 37
1.8.1 Frühzeit 38
1.8.2 Prärelationale DBMS 38
1.8.3 Durchbruch der relationalen Datenbanken 39
1.8.4 Neue Anwendungsfelder für Datenbanken 40
1.8.5 Neuzeit 40
1.9 *Datenbanken in der Forschung* 41
1.9.1 Fachverbände 41
1.9.2 Aktuelle Forschungstrends 42

2 Informationsmodellierung **44**
2.1 *Datenbankentwurf* 44
2.1.1 Phasenmodell 44
2.1.2 Anforderungsanalyse 45
2.1.3 Konzeptioneller Entwurf 46
2.1.4 Logischer Entwurf 47
2.1.5 Datendefinition 48
2.1.6 Physischer Entwurf 49
2.1.7 Implementierung und Wartung 49
2.2 *Grundlagen des Entity-Relationship-Modells (ERM)* 49
2.2.1 Semantische Datenmodelle 50
2.2.2 Grundkonzepte des klassischen ERM 50
2.2.2.1 Konzepte auf der Instanzebene 51

2.2.2.2 Konzepte auf der Typebene 52
2.2.2.3 Rollenmodellierung 53
2.2.3 Kardinalitäten von Beziehungstypen 54
2.2.3.1 Kardinalitäten im klassischen ERM 54
2.2.3.2 Intervalle und Komplexitätsgrade 54
2.2.4 Existenzabhängigkeit vs. Optionalität 55
2.2.5 Rekursive und *n*-äre Beziehungstypen 56
2.2.5.1 Rekursive Beziehungstypen 56
2.2.5.2 *N*-äre Beziehungstypen 57
2.2.6 Attribute 58
2.2.7 Modellierungsbeispiel 59
2.3 *Erweiterungen des ERM* 60
2.3.1 Erweiterungen bei Attributen 60
2.3.2 Generalisierung und Spezialisierung 61
2.3.3 Aggregation 64
2.3.4 Modellierung zeitlicher Aspekte 65

3 Relationales Datenmodell 67
3.1 *Konzepte und Grundbegriffe des relationalen Datenmodells* 67
3.1.1 Relationen, Tupel, Attribute und Wertebereiche 67
3.1.2 Eigenschaften von Relationen 69
3.2 *Integritätsbedingungen* 70
3.2.1 Grundbegriffe 70
3.2.2 Entitätsintegrität 71
3.2.3 Referenzielle Integrität 71
3.2.3.1 Begriff 71
3.2.3.2 Regeln für Fremdschlüssel 72
3.2.3.3 Gewährleistung der referenziellen Integrität bei kritischen DML-Operationen 73
3.3 *Abbildung des EERM auf das relationale Datenmodell* 75
3.3.1 Problemstellung 75
3.3.2 Abbildungsregeln für Attribute und Entitytypen 75
3.3.3 Abbildungsregeln für Beziehungstypen 76
3.3.4 Abbildungsregeln für die Generalisierung 79
3.4 *Optimierung von Relationen* 80
3.4.1 Problemstellung 80
3.4.2 Anomalien bei DML-Operationen auf Relationen 81
3.4.3 Abhängigkeiten 82
3.4.3.1 Funktionale Abhängigkeiten 82
3.4.3.2 Mehrwertige Abhängigkeiten 83
3.4.4 Verbundtreue und Abhängigkeitstreue 84
3.4.5 Normalformenlehre 85
3.4.5.1 Erste Normalform (1NF) 85
3.4.5.2 Zweite Normalform (2NF) 86
3.4.5.3 Dritte Normalform (3NF) 87
3.4.5.4 Boyce-Codd-Normalform (BCNF) 88
3.4.5.5 Vierte Normalform (4NF) 89

3.4.5.6 Fünfte Normalform (5NF) 90
3.4.5.7 Denormalisierung 92
3.5 *Operationen der Relationenalgebra* 92
3.5.1 Einführung 92
3.5.2 Relationenorientierte Operationen 93
3.5.3 Mengenoperationen 95
3.5.4 Relationenalgebra und relationale Sprachen 96
3.5.5 Relationenkalkül 97

4 Die Datenbanksprache SQL 99
4.1 *Grundkonzepte* 99
4.2 *Historie* 101
4.3 *Spezifikationsdokumente* 102
4.4 *Beispieldatenbank* 103
4.5 *Datenbankanfragen* 103
4.5.1 Einführende Beispielanfragen 104
4.5.2 Grundgerüst von Anfragen 105
4.5.3 Anfragen mit Aggregatfunktionen 107
4.5.4 Anfragen mit Tabellenfunktionen 108
4.5.5 Anfragen mit Mengenoperationen 108
4.5.6 Anfragen mit Verbundoperationen 110
4.5.7 Verschachtelte Anfragen 112
4.5.8 Rekursive Anfragen 114
4.5.9 Anfragen mit sortierter Ausgabe 115
4.6 *Datenmanipulation* 116
4.6.1 Einfügen von Tabellenzeilen 116
4.6.2 Ändern von Tabellenzeilen 117
4.6.3 Löschen von Tabellenzeilen 118
4.6.4 Zusammenführen von Tabellenzeilen 118
4.7 *Datendefinition* 119
4.7.1 SQL-Datentypen 119
4.7.2 Erzeugen und Löschen von Schemata 124
4.7.3 Erzeugen, Ändern und Löschen von Tabellen 124
4.7.4 Erzeugen und Löschen von Domänen 127
4.7.5 Erzeugen und Löschen von Integritätsbedingungen 128
4.7.6 Erzeugen und Löschen von Sichten 130
4.7.7 Erzeugen und Löschen von Routinen 131
4.7.7.1 Prozedurale SQL-Konstrukte 131
4.7.7.2 Erzeugen von SQL-Routinen 135
4.7.7.3 Erzeugen von externen Routinen 136
4.7.7.4 Löschen von Routinen 136
4.7.8 Erzeugen und Löschen von Triggern 137
4.7.9 Erzeugen und Löschen von Sequenzgeneratoren 138
4.8 *Transaktionssteuerung* 138
4.8.1 Transaktionsanweisungen 139
4.8.2 Isolationsebenen 140
4.8.3 Überprüfung von Integritätsbedingungen 141

4.9 *Zugriffskontrolle* 141
4.9.1 Vergabe von Zugriffsrechten 141
4.9.2 Zurücknahme von Zugriffsrechten 143
4.9.3 Erzeugen und Löschen von Rollen 143

5 Datenbank-Anwendungsprogrammierung 144
5.1 *Grundlagen der Datenbank-Anwendungsprogrammierung* 144
5.1.1 Impedance Mismatch 144
5.1.2 Einbettungstechniken 145
5.1.3 Einbettungsarten 145
5.1.3.1 Statische Programmierspracheneinbettung 145
5.1.3.2 Dynamische Programmierspracheneinbettung 146
5.1.4 Architekturansätze 147
5.1.4.1 Fat-Client-Architektur 148
5.1.4.2 Thin-Client-Architektur 148
5.1.4.3 Thin-Client-Architektur mit Applikationsserver 150
5.2 *Embedded SQL* 151
5.2.1 Grundidee und Architektur 151
5.2.2 Syntax 152
5.2.3 Host-Variablen 152
5.2.4 Das Cursor-Konzept 152
5.2.5 Statische Einbettung 154
5.2.6 Dynamische Einbettung 155
5.2.7 Indikatorvariablen 156
5.2.8 SQLJ 156
5.3 *Aufrufschnittstellen* 158
5.3.1 Überblick 158
5.3.2 SQL/CLI 158
5.3.3 JDBC 160
5.3.3.1 Treibertypen 161
5.3.3.2 Verbindung aufbauen 161
5.3.3.3 Verbindung schließen 163
5.3.3.4 Transaktionen 163
5.3.3.5 Leseanweisungen – das ResultSet 164
5.3.3.6 Änderungen 165
5.3.3.7 Zugriff auf Metadaten 167
5.4 *Relationale Datenbankprogrammiersprachen* 168
5.4.1 Datenbankinterne Ansätze 168
5.4.1.1 SQL/PSM 168
5.4.1.2 Benutzerdefinierte Routinen 169
5.4.1.3 Trigger 170
5.4.2 4GL-Programmiersprachen 174
5.4.2.1 Überblick 174
5.4.2.2 Beispiel: ABAP 174
5.5 *Objektrelationale Abbildung* 177
5.5.1 Java Persistence API (JPA) 179
5.5.1.1 Schemaabbildung 180

5.5.1.2 Datenzugriff . . . 181
5.5.1.3 Formulieren von Anfragen . . . 183
5.5.2 JDO – Java Data Objects . . . 184
5.5.3 Entity Beans . . . 188

6 Datenbanken im Web . . . 189
6.1 *Grundlagen des Web* . . . 189
6.2 *Eigenschaften von Webanwendungen* . . . 191
6.2.1 Anforderungen . . . 192
6.2.2 Webseiten . . . 192
6.2.3 Dynamische Webseiten . . . 195
6.2.4 Adressierung . . . 197
6.2.5 Kommunikation . . . 198
6.3 *Datenbankanbindung im Web* . . . 200
6.3.1 Architekturen . . . 201
6.3.2 Programmierung . . . 202
6.3.3 Überblick . . . 204
6.4 *Datenbankanbindung über Datenexport* . . . 206
6.5 *Clientseitige Datenbankanbindung* . . . 206
6.5.1 Skripteinbettung . . . 207
6.5.2 Programmeinbettung . . . 209
6.5.3 Weitere Techniken . . . 212
6.6 *Serverseitige Datenbankanbindung* . . . 212
6.6.1 Webseitengeneratoren . . . 212
6.6.2 Skripteinbettung . . . 215
6.6.3 Programmeinbettung . . . 219
6.6.4 Applikationsserver . . . 221
6.6.5 Webservices . . . 223

7 Komponenten eines Datenbankmanagementsystems . . . 227
7.1 *Architektur von DBMS* . . . 227
7.1.1 Schichtenmodell . . . 227
7.1.2 Prozessarchitektur . . . 229
7.2 *Pufferverwaltung* . . . 230
7.2.1 Notwendigkeit und Aufgabe . . . 231
7.2.2 Speicherzuteilung . . . 232
7.2.3 Seitenersetzung . . . 233
7.3 *Speicher- und Zugriffssystem* . . . 234
7.3.1 Aufgabe . . . 235
7.3.2 Seiten und Sätze . . . 235
7.3.3 Adressierung von Sätzen . . . 237
7.4 *Anfrageprozessor* . . . 238
7.4.1 Basisoperatoren . . . 238
7.4.1.1 Unäre Operatoren . . . 239
7.4.1.2 Binäre Operatoren . . . 241
7.4.2 Anfrageplanung und -optimierung . . . 242
7.4.3 Kosten und Statistiken . . . 247

7.5 *Transaktionsverwaltung* . . . 249
7.5.1 Aufgabe . . . 249
7.5.2 Serialisierbarkeit . . . 251
7.5.3 Sperrverfahren . . . 253
7.5.4 Nicht sperrende Verfahren . . . 257
7.6 *Recovery* . . . 259
7.6.1 Fehlerklassen . . . 259
7.6.2 Recovery-Strategien . . . 260
7.6.3 Logging . . . 261
7.6.4 Wiederanlauf im Fehlerfall . . . 264
7.6.5 Schattenspeicherverfahren . . . 266
7.7 *Datenbanktechniken für moderne Hardware-Architekturen* . . . 266

8 Dateiorganisation und Indexe . . . 269
8.1 *Organisation von Dateien* . . . 269
8.1.1 Formen der Dateiorganisation . . . 269
8.1.2 Dateiorganisationsformen im Vergleich . . . 269
8.1.2.1 Basisoperationen . . . 270
8.1.2.2 Kosten . . . 270
8.2 *Zugriffsstrukturen* . . . 271
8.2.1 Grundlagen von Zugriffsverfahren . . . 271
8.2.2 Eigenschaften von Indexen . . . 272
8.3 *Baumbasierte Verfahren* . . . 274
8.3.1 ISAM-Bäume . . . 274
8.3.2 Balancierte Mehrwegbäume . . . 275
8.3.3 Digitale Bäume . . . 277
8.4 *Hash-Verfahren* . . . 278
8.4.1 Prinzip des Hashing . . . 278
8.4.2 Erweiterbares Hashing . . . 279
8.4.3 Weitere Hash-Verfahren . . . 280
8.5 *Mehrdimensionale Zugriffsverfahren* . . . 281
8.5.1 Mehrdimensionale Baumverfahren . . . 281
8.5.2 Grid-File . . . 281
8.6 *Clusterung und Partitionierung* . . . 283
8.6.1 Clusterung . . . 283
8.6.2 Partitionierung . . . 283
8.7 *Umsetzung in SQL-Systemen* . . . 284
8.7.1 Definition von Tabellen . . . 285
8.7.2 Definition von Indexen . . . 285

9 Optimierung von Datenbanken und Leistungsbewertung . . . 287
9.1 *Motivation der Datenbankoptimierung* . . . 287
9.1.1 Kosten von Datenbankanfragen . . . 288
9.1.2 Optimierungspotenzial . . . 289
9.1.3 Zielbestimmung der Datenbankoptimierung . . . 290
9.2 *Phasen der Datenbankoptimierung* . . . 292

9.3 *Phase 1.1 – Optimierung des Datenbankschemas* 293
9.3.1 Konzeptuelles Schema 294
9.3.1.1 Attribute 294
9.3.1.2 Tabellen 294
9.3.1.3 Redundanz 296
9.3.2 Externes Schema 297
9.3.2.1 Sichten 298
9.3.2.2 Prozedurale SQL-Erweiterungen 298
9.3.3 Internes Schema 298
9.3.3.1 Materialisierte Sichten 298
9.3.3.2 Zugriffspfadstrukturen 299
9.4 *Phase 1.2 – Anwendungsoptimierung* 301
9.4.1 Optimierung von Unternehmensfunktionen 301
9.4.2 Optimierung der Anwendung 301
9.4.3 Optimierungen im Mehrbenutzerbetrieb 302
9.4.4 Formulierung von SQL-Anweisungen 304
9.5 *Phase 2 – Hauptspeicheroptimierung* 306
9.5.1 Gestaltung des Datenbankpuffers 306
9.5.1.1 Komponenten des Datenbankpuffers 307
9.5.1.2 Größe des Datenbankpuffers 308
9.5.1.3 Blockfüllgrad 308
9.5.2 Schreiben des Datenbankpuffers 309
9.5.2.1 Sicherungspunkt-Intervalle 309
9.5.2.2 Protokolldatei 310
9.5.3 Optimierer 310
9.5.3.1 Statistiken 311
9.5.3.2 Planhinweise (Hints) 311
9.6 *Phase 3 – Optimierung der Sekundärspeicherzugriffe* 312
9.6.1 Zusammenspiel mit dem Betriebssystem 312
9.6.2 Verteilung der Eingabe-/Ausgabelast 313
9.6.2.1 Verteilung der Daten- und Indexdateien 313
9.6.2.2 Verteilung der Daten 313
9.6.2.3 Verteilung der Protokolldatei 314
9.6.2.4 RAID-Level 314
9.6.3 Optimierung physischer Speicherstrukturen 315
9.6.3.1 Blockgröße 315
9.6.3.2 Cluster-Techniken 316
9.6.3.3 Kompressions-Techniken 317
9.6.3.4 Reorganisation 317
9.7 *Leistungsbewertung* 318
9.7.1 Transaction Processing Performance Council 319
9.7.2 Vergleichbarkeit der Benchmark-Ergebnisse 320

10 Objektrelationale Datenbanken 322
10.1 *Objektorientierte Konzepte* 322
10.1.1 Objekte 322
10.1.2 Methoden 323

10.1.3 Kapselung . . . 323
10.1.4 Objektidentität . . . 323
10.1.5 Klassen . . . 324
10.1.6 Spezialisierung . . . 325
10.2 *Objektorientierung in Datenbanken* . . . 326
10.3 *Objektrelationale Standard-SQL-Konzepte* . . . 327
10.3.1 Typkonstruktoren . . . 328
10.3.2 Distinct-Typen . . . 330
10.3.3 Strukturierte Typen . . . 330
10.3.4 Methoden . . . 333
10.3.5 Benutzerdefinierte Konstruktoren . . . 335
10.3.6 Benutzerdefinierte Casts . . . 336
10.3.7 Benutzerdefinierte Ordnungen . . . 337
10.3.8 Typisierte Tabellen . . . 338
10.3.9 Typisierte Sichten . . . 339
10.4 *Objektrelationale Anfragen* . . . 341
10.4.1 Anfragen auf Kollektionen . . . 341
10.4.2 Anfragen mit Pfadausdrücken . . . 342
10.4.3 Anfragen mit Methodenaufrufen . . . 342
10.4.4 Anfragen auf flachen Tabellenextensionen . . . 343
10.4.5 Typspezifische Anfragen . . . 343
10.4.6 Anfragen mit temporärer Typanpassung . . . 343

11 XML und Datenbanken . . . 345
11.1 *Überblick über XML* . . . 345
11.1.1 Der XML-Standard und verwandte Standards . . . 345
11.1.2 XML Schema . . . 347
11.2 *Anfragesprachen für XML* . . . 350
11.2.1 Pfadausdrücke (XPath und XQuery) . . . 351
11.2.2 XQuery . . . 352
11.3 *XML und relationale Datenbanksysteme* . . . 356
11.3.1 Relationale Speicherung von XML . . . 356
11.3.2 SQL/XML . . . 358
11.3.3 Realisierung in kommerziellen Systemen . . . 361
11.4 *Reine XML-Datenbanksysteme* . . . 366

12 NoSQL-Datenbanksysteme . . . 368
12.1 *Motivation und Grundbegriffe* . . . 368
12.2 *Klassifikation* . . . 369
12.2.1 Key-Value-Datenbanksysteme . . . 370
12.2.1.1 Datenmodell und Schema . . . 370
12.2.1.2 Anfragen und Datenmanipulation . . . 371
12.2.1.3 Einsatzbereiche und Systeme . . . 371
12.2.2 Dokumentenorientierte Datenbanksysteme . . . 372
12.2.2.1 Datenmodell und Schema . . . 372
12.2.2.2 Anfragen und Datenmanipulation . . . 373
12.2.2.3 Einsatzbereiche und Systeme . . . 374

12.2.3 Column-Family-Datenbanksysteme 374
12.2.3.1 Datenmodell und Schema 374
12.2.3.2 Anfragen und Datenmanipulation 377
12.2.3.3 Einsatzbereiche und Systeme 377
12.2.4 Weitere NoSQL-Datenbanksysteme 378
12.3 *Datenmodellierung* 378
12.3.1 Generelle Aspekte der Datenmodellierung in NoSQL-Datenbanksystemen 378
12.3.2 Datenmodellierung für dokumentenorientierte Datenbanken 380
12.3.3 Datenmodellierung für Column-Family-Datenbanken 382
12.3.3.1 Eingebettete Speicherung in Column-Family-Datenbanksystemen 382
12.3.3.2 Spaltenfamilien 384
12.3.4 Datenmodellierung für Key-Value-Datenbanken 385
12.4 *Anwendungsentwicklung mit NoSQL-Datenbanksystemen* 385
12.4.1 MapReduce 386
12.4.1.1 MapReduce-Prinzip 387
12.4.1.2 MapReduce-Beispiel 388
12.4.1.3 MapReduce-Frameworks 389
12.4.1.4 MapReduce-Trends 389
12.4.2 Schema-Management 389
12.5 *Skalierbarkeit, Verfügbarkeit und Konsistenz* 390
12.6 *Auswahl eines geeigneten Datenbanksystems* 392
12.6.1 Kriterienkatalog 392
12.6.2 Performance 392
12.6.3 Polyglotte Persistenz 393

13 Verteilte und föderierte Datenbanksysteme 394
13.1 *Überblick, Grundbegriffe, Abgrenzung* 394
13.1.1 Verteilte vs. parallele Datenbanksysteme 395
13.1.2 Verteilte vs. föderierte Datenbanksysteme 397
13.2 *Schemaarchitektur und Entwurf verteilter Datenbanksysteme* 400
13.3 *Fragmentierung* 401
13.3.1 Horizontale Fragmentierung 401
13.3.2 Vertikale Fragmentierung 403
13.3.3 Kombinierte Fragmentierung 403
13.4 *Verteilungstransparenz* 404
13.4.1 Vorteile verteilter Datenbanksysteme 404
13.4.2 Transparenzeigenschaften verteilter Datenbanksysteme 404
13.5 *Verteilte Anfrageverarbeitung* 405
13.5.1 Datenlokalisierung 407
13.5.2 Globale Optimierung der Join-Auswertung 409
13.6 *Transaktionsverwaltung in verteilten Datenbanksystemen* 412
13.6.1 Koordination 412
13.6.2 Synchronisation 414
13.6.3 Deadlock-Behandung 415
13.6.4 Synchronisation bei Replikation 416

13.7 *Föderierte Datenbanksysteme* 417
13.7.1 Autonomie und Heterogenität 418
13.7.2 Architektur föderierter DBS 419
13.7.3 Integrationsprozess 420
13.7.4 Anfrageverarbeitung in föderierten DBS 421
13.7.5 Synchronisation in föderierten DBS 422
13.8 *Cloud-Datenbanken* 423
13.8.1 Cloud Data Management und Big Data 423
13.8.2 Das CAP-Theorem 424
13.8.3 Database as a Service (DBaaS) 425
13.8.4 Skalierbarkeit und Verfügbarkeit 426
13.9 *Trends* 428

14 Data Warehouse 430
14.1 *Architektur* 430
14.1.1 Datenquellen 430
14.1.2 Back-End-Bereich 430
14.1.2.1 Monitore 431
14.1.2.2 Extraktionskomponenten 432
14.1.2.3 Transformationskomponente 433
14.1.2.4 Ladekomponente 434
14.1.3 Datenbank 434
14.1.3.1 Data Warehouse 434
14.1.3.2 Data Marts 436
14.1.3.3 Archiv-Datenbank 437
14.1.4 Front-End-Werkzeuge 437
14.1.4.1 Berichts- und Anfragewerkzeuge 437
14.1.4.2 OLAP-Werkzeuge 438
14.1.4.3 Data-Mining-Werkzeuge 438
14.1.4.4 Sonstige Front-End-Werkzeuge 438
14.1.5 Sonstige Werkzeuge 439
14.1.5.1 DWS-Manager 439
14.1.5.2 Metadaten-Repository 439
14.2 *Multidimensionale Datenmodelle* 440
14.2.1 Statische Aspekte 440
14.2.2 Dynamische Aspekte 444
14.3 *Speicherung und Schemagestaltung* 446
14.3.1 Relationale Speicherung 446
14.3.2 Multidimensionale Speicherung 447
14.3.3 Spaltenorientierte Speicherung 448
14.4 *Erweiterung relationaler Datenbanken* 451
14.4.1 Materialisierte Sichten 452
14.4.2 Partitionierung 453
14.4.3 Bitmap-Index 455
14.4.4 SQL-Erweiterungen zum Einfügen 456
14.4.5 Komplexes Gruppieren 457

14.4.6 Star Query . . . 458
14.4.7 Bulk Loader . . . 460

15 Data Mining . . . 461
15.1 *KDD-Prozess* . . . 461
15.2 *Clustering* . . . 462
15.2.1 Definition und Beispiele . . . 462
15.2.2 Anforderungen und Probleme . . . 463
15.2.3 Verfahren . . . 463
15.2.3.1 Partitionierende Verfahren . . . 464
15.2.3.2 Hierarchische Verfahren . . . 465
15.2.3.3 Dichtebasierte Methoden . . . 468
15.3 *Assoziationsanalyse* . . . 468
15.3.1 Definition und Beispiel . . . 468
15.3.2 Anforderungen und Probleme . . . 468
15.3.3 Verfahren . . . 469
15.4 *Klassifikation* . . . 472
15.4.1 Definition und Beispiele . . . 472
15.4.2 Anforderungen . . . 472
15.4.3 Verfahren . . . 473
15.4.3.1 Entscheidungsbaum-Klassifikatoren . . . 473
15.4.3.2 Regelbasierte Klassifikatoren . . . 473
15.4.3.3 Weitere Verfahren . . . 474
15.5 *Anomalieentdeckung* . . . 474
15.5.1 Definition und Beispiele . . . 474
15.5.2 Anforderungen und Probleme . . . 475
15.5.3 Verfahren . . . 475
15.5.3.1 Grafische und statistikbasierte Verfahren . . . 476
15.5.3.2 Distanzbasierte Ansätze . . . 476

16 Multimedia-Datenbanken . . . 478
16.1 *Einführung* . . . 478
16.2 *Mediendaten* . . . 482
16.3 *Suche nach Mediendaten* . . . 485
16.3.1 Textsuche . . . 485
16.3.2 Bildsuche . . . 487
16.3.3 Audiosuche . . . 488
16.3.4 Videosuche . . . 489
16.4 *Mediendatentypen* . . . 490
16.5 *Einbettung in Datenbanksysteme* . . . 494
16.5.1 Schemastrukturen . . . 494
16.5.2 Anfrageformulierung . . . 496
16.6 *Einsatz* . . . 497

17 Geodatenbanken . . . 499
17.1 *Geodaten* . . . 499
17.1.1 Eigenschaften von Geodaten . . . 499
17.1.2 Metadaten . . . 501

17.2 *Datenschemata* 502
17.2.1 Standardisierung 502
17.2.2 ISO 19107 Spatial Schema 502
17.2.3 ISO 19125 Simple Feature Access 504
17.2.3.1 Datenschema 504
17.2.3.2 Datenrepräsentationen 506
17.2.4 ISO/IEC 13249-3 SQL/MM Spatial 507
17.2.5 Räumliche Bezugssysteme 508
17.2.5.1 EPSG-Bezugssysteme 510
17.2.5.2 Lineare Bezugssysteme 510
17.3 *Funktionen* 510
17.3.1 Geometrische Funktionen 511
17.3.2 Topologische Prädikate 512
17.3.2.1 Boolesches Modell 512
17.3.2.2 Dimensionsmodell 513
17.4 *Räumliche Anfragen* 515
17.4.1 Räumliche Basisanfragen 515
17.4.1.1 Räumliche Selektion 515
17.4.1.2 Räumlicher Verbund 516
17.4.1.3 Nächste-Nachbarn-Anfrage 517
17.4.2 Mehrstufige Anfragebearbeitung 517
17.4.3 Approximationen 518
17.5 *Räumliche Indexe* 520
17.5.1 Grundtechniken 520
17.5.1.1 Clipping 520
17.5.1.2 Punkttransformationen 521
17.5.1.3 Raumfüllende Kurven 521
17.5.1.4 Überlappende Blockregionen 522
17.5.2 Quadtrees 523
17.5.3 R-Bäume 525
17.6 *Geodatenbanksysteme* 527

Abkürzungsverzeichnis **529**

Literaturverzeichnis **534**

Sachwortverzeichnis **554**

1 Datenbanken: Grundlagen und Überblick

Thomas Kudraß

In diesem Kapitel werden die wichtigsten Grundbegriffe und Konzepte im Fachgebiet Datenbanken eingeführt und zueinander in Beziehung gesetzt (→ 1.1 bis 1.6). Darauf aufbauend, erfolgt eine Einordnung des Fachgebiets Datenbanken in die Informatik sowie ein Überblick über die zentralen Themen des Faches, die den Gegenstand der Kapitel 2 bis 17 dieses Buches bilden (→ 1.7). Die Historie (→ 1.8) und ein Blick auf aktuelle Trends in der Forschung (→ 1.9) ermöglichen eine zeitliche Einordnung des erreichten Entwicklungsstandes von Datenbanksystemen.

1.1 Dateien und Dateisysteme

Informationen werden durch Daten repräsentiert, die in einem Kontext interpretiert werden. Die Information wird dazu in Zeichen bzw. Zeichenketten codiert, was auf der Grundlage von Regeln erfolgt, die man als Syntax bezeichnet. Die Codierung geschieht auf der Grundlage von Codierregeln.

Daten sind maschinell lesbar und verarbeitbar. Sie beinhalten die ursprünglichen Benutzerdaten sowie andere zusätzliche Elemente, die für eine maschinelle Verarbeitung erforderlich sind.

❑ *Beispiel:* 04275 kann eine Postleitzahl sein und damit zu den Daten gehören. Wenn dazu der Name des Ortes genannt wird, so wird daraus eine Information. Der Wert der Postleitzahl kann durch eine Folge von 5 Zeichen repräsentiert werden: „04275“. Die Codierung dieser Zeichen erfolgt auf der Grundlage des ASCII-Codes. Somit wird diese Zeichenkette hexadezimal als Bytefolge X’3034323735’ repräsentiert.

Man unterscheidet
- strukturierte Daten, z. B. Telefonbücher, Adresslisten,
- unstrukturierte Daten, z. B. Textdokumente, digitale Bilder,
- semistrukturierte Daten, z. B. Produktkataloge, Patientenakten.

Eine **Datei** (file) ist ein Bestand inhaltlich zusammengehöriger Daten, der auf einem Datenträger persistent abgelegt ist. Daten sind *persistent*, wenn sie über die Laufzeit des Programmes hinaus, in dem sie verarbeitet werden, weiter existieren. Anderenfalls spricht man von *transienten* Daten.

Dateien werden in den meisten Betriebssystemen über Dateisysteme verwaltet und können auf unterschiedliche Art und Weise organisiert sein

(→ 7.3, 8.1). Logisch lässt sich eine strukturierte Datei als eine Folge von **Datensätzen** (records) betrachten. Ein Datensatz besteht aus mehreren Feldern (fields), deren Anzahl und Struktur in jedem Datensatz dieser Datei gleich ist.

❑ *Beispiel:* Die Datei `Stadt` besteht aus Datensätzen mit den Feldern `PLZ` und `Ort`: `04275,Leipzig; 64289,Darmstadt; 99425,Weimar`.

Zur Arbeit mit Dateien stehen verschiedene Operationen zur Verfügung:
- Anlegen (create) und Löschen (drop) von Dateien.
- Öffnen (open) und Schließen (close) von Dateien. Das Öffnen einer Datei ist die Voraussetzung, um auf diese zuzugreifen.
- Lesen (read) und Schreiben (write) von Dateien.

Bei der herkömmlichen Dateiverarbeitung definiert und implementiert jeder Benutzer die Dateien, die für eine bestimmte Anwendung benötigt werden, als Teil der Anwendungsentwicklung.

❑ *Beispiel:* So könnte ein Handelsunternehmen die Adressdaten seiner Kunden für Marketingmaßnahmen in einer Datei verwalten. In einer anderen Software zur Abwicklung der Bestellungen werden ebenfalls die Adressen der Kunden erfasst.

Die herkömmliche Arbeit mit Dateien führt zu dem Problem der **Redundanz**, d. h. dem mehrfachen Vorhandensein ein und derselben Information, die in unterschiedlichen Dateien gespeichert werden kann.

Nachteile redundanter Datenhaltung:
- erhöhter Speicherplatzbedarf für die Mehrfachspeicherung.
- erhöhter Aufwand zur Synchronisation von Datenänderungen. Dabei muss ein geänderter Datensatz mit anderen Datensätzen, die die gleiche Information beinhalten, abgeglichen werden.

Redundante Speicherung birgt immer die Gefahr von *Inkonsistenz*, d. h. die widersprüchliche Speicherung gleicher Informationen.

1.2 Terminologie

Eine **Datenbank** (DB) ist eine Sammlung von Daten, die einen Ausschnitt der realen Welt beschreiben.

Der übliche Gebrauch des Begriffs Datenbank wird in vielen Fällen aber eingeschränkt. Demnach hat eine Datenbank folgende Eigenschaften:
- Eine Datenbank stellt Aspekte der realen Welt dar. Dieser Weltausschnitt wird auch als Diskursbereich (universe of discourse) bezeichnet.

- Eine Datenbank ist eine integrierte, d. h. logisch zusammenhängende Sammlung von Daten für einen festgelegten Zweck. Sie wird von einer bestimmten Benutzergruppe in Anwendungen eingesetzt.

Ein **Datenbankmanagementsystem (DBMS)** ist ein Softwaresystem (d. h. eine Sammlung von Programmen), das dem Benutzer das Erstellen und die Pflege einer Datenbank ermöglicht. Dies umfasst die Definition, die Erzeugung und Manipulation von Datenbanken.

Das DBMS legt das *Datenmodell* der Datenbank (*Datenbankmodell*) fest. Die Auswahl des DBMS ist eine kritische Entscheidung. Hierbei ist zu unterscheiden zwischen kommerzieller und freier Software.

► *Hinweis:* Für ein DBMS wird selten auch der Begriff Datenbankverwaltungssystem (DBVS) oder Datenbankbetriebssystem verwendet.

Ein DBMS zusammen mit einer oder mehreren Datenbanken wird als **Datenbanksystem (DBS)** bezeichnet.

Ein **Datenmodell** enthält drei Bestandteile:
- *Datenstrukturen*: Objekte und deren Beziehungen (statische Eigenschaften),
- *Operationen* und Beziehungen zwischen Operationen (dynamische Eigenschaften),
- *Integritätsbedingungen* auf Objekten und Beziehungen. Dies sind Regeln, die die Menge der erlaubten Zustände bzw. Zustandsübergänge definieren. Sie können modellinhärent sein oder explizit definiert werden.

❑ *Beispiel:* Objekte sind dabei etwa Kunden, Artikel, Lieferanten, aber auch Ereignisse wie Bestellungen. Eine Beziehung zwischen diesen Objekten ist beispielsweise die Bestellung von Artikeln durch Kunden. Operationen auf Objekten sind das Erfassen von Neukunden und die Pflege der Artikeldaten. Weiterhin kann man Integritätsbedingungen an Kundenobjekte formulieren, z. B. dass ihre Kundennummer jeweils unterschiedlich ist (Schlüsselbedingung) und dass eine Bestellung genau einem Kunden zugeordnet werden muss.

Ein **Datenbankmodell** ist ein Datenmodell für ein Datenbanksystem. Es bestimmt, auf welche Art und Weise Daten prinzipiell gespeichert werden und wie man die Daten manipulieren kann.

► *Hinweis:* Datenbankmodelle werden im Kontext von Datenbanksystemen oft einfach nur als Datenmodelle bezeichnet.

Ein Datenmodell kann auf Basis eines Datenbankmodells implementiert werden oder selbst als Datenbankmodell dienen.

- *Beispiel 1:* Das mehrdimensionale Datenmodell kann relational in einer Datenbank umgesetzt werden, was als ROLAP (Relational Online Analytical Processing) bezeichnet wird (→ 14.3).

- *Beispiel 2:* XML kann in einer relationalen Datenbank (→ 11.3) oder in einer reinen XML-Datenbank (nativ) gespeichert werden (→ 11.4).

Daneben gibt es abstrakte Datenbankmodelle, die im Datenbankentwurf eingesetzt werden, aber keine Entsprechung in einem konkreten DBMS haben. Solche Modelle werden oft als **semantische Datenmodelle** oder **Informationsmodelle** bezeichnet, ein typisches Beispiel ist das *Entity-Relationship-Modell* (*ERM*) (→ 2).

Die ersten (prärelationalen) Datenbankmodelle waren satzorientiert, das **hierarchische** und das **Netzwerk-Datenmodell**, angelehnt an die Datenstrukturen von kommerziellen Programmiersprachen. Bei hierarchischen Datenmodellen können Datensätze (record types) nur in hierarchischen Beziehungen (1:*N*) stehen, beim Netzwerkmodell auch in vernetzten *M*:*N*-Beziehungen. Das hierarchische Datenmodell erlebt eine Renaissance durch das Aufkommen von Dokument-Datenbanken und die Nutzung von Datenformaten wie XML als Datenmodell.

Das **relationale Datenmodell** (RDM) beruht auf dem Begriff der Relation, der im Wesentlichen ein mathematisches Modell für eine Tabelle beschreibt. Die Daten werden in Form von zweidimensionalen Tabellen verwaltet, die über Schlüssel (Primärschlüssel, Fremdschlüssel) miteinander verknüpft werden können (→ 3). Die meisten in der Praxis eingesetzten Datenbanksysteme beruhen auf einem relationalen DBMS (RDBMS).

Während das relationale Datenmodell eine klare mathematische Grundlage hat, gibt es eine Vielzahl objektorientierter Datenbankmodelle, die sich oft an objektorientierte Programmiersprachen (z. B. C++) angelehnt haben. Demgegenüber haben die großen DBMS-Hersteller ihren relationalen Systemen objektorientierte Eigenschaften hinzugefügt und diese zu **objektrelationalen** Systemen weiterentwickelt.

Nicht-relationale Datenbankmodelle zur flexiblen Speicherung hierarchischer oder unstrukturierter Daten bilden die Grundlage für NoSQL-Datenbanksysteme unterschiedlicher Kategorien (→ 12).

► *Hinweis:* Allgemeine Lehrbücher über Datenbanken sind u. a. /1.1/, /1.2/, /1.4/, /1.5/, /1.9/.

1.3 Merkmale eines DBMS

1.3.1 Aufgaben eines DBMS

Gemäß seiner Definition muss ein DBMS folgende Funktionalitäten bieten:

Integrierte Datenhaltung. Das DBMS ermöglicht die *einheitliche* Verwaltung *aller* von den Anwendungen benötigten Daten. Somit wird jedes logische Datenelement, wie beispielsweise der Name eines Kunden, *an nur einer* Stelle in der Datenbank gespeichert. Dabei muss ein DBMS die Möglichkeit bieten, eine Vielzahl komplexer Beziehungen zwischen den Daten zu definieren sowie zusammenhängende Daten schnell und effizient miteinander zu verknüpfen. In manchen Fällen kann eine *kontrollierte Redundanz* allerdings nützlich sein, um die Effizienz der Verarbeitung von Anfragen zu verbessern (→ 9.3.1.3).

Sprache. Das DBMS stellt an seiner Schnittstelle eine Datenbanksprache (*query language*) für die folgenden Zwecke zur Verfügung:

- Datenanfrage (retrieval),
- Datenmanipulation (Data Manipulation Language, DML),
- Verwaltung der Datenbank (Data Definition Language, DDL),
- Berechtigungssteuerung (Data Control Language, DCL).

Da viele verschiedene Benutzer mit unterschiedlichen technischen Kenntnissen und Anforderungen auf eine Datenbank zugreifen, sollte ein DBMS eine Vielzahl von Benutzeroberflächen bereitstellen. Dazu zählen Anfragesprachen für gelegentliche Benutzer, Programmierschnittstellen und grafische Benutzeroberflächen (*Graphical User Interface, GUI*). Die Möglichkeit eines Webzugriffs ist heute eine Grundvoraussetzung für den Einsatz eines DBMS.

Katalog. Der Katalog (*data dictionary*) ermöglicht Zugriffe auf die Datenbeschreibungen der Datenbank, die auch als Metadaten bezeichnet werden.

❑ *Beispiele:* Eigentümer und Erzeugungszeit von Datenbankobjekten (Tabellen), Beschreibung von Tabellen und ihren Spalten (Datentyp, Länge).

Benutzersichten. Für unterschiedliche Klassen von Benutzern sind verschiedene Sichten (*views*) erforderlich, die bestimmte Ausschnitte aus dem Datenbestand beinhalten oder diesen in einer für die jeweilige Anwendung benötigten Weise strukturieren. Die Sichten sind im externen Schema der Datenbank definiert (→ 1.4.2.2).

Konsistenzkontrolle. Die Konsistenzkontrolle, auch als *Integritätssicherung* bezeichnet, übernimmt die Gewährleistung der Korrektheit von Datenbankinhalten und der korrekten Ausführung von Änderungen. Ein korrek-

ter Datenbankzustand wird durch benutzerdefinierte Integritätsbedingungen (*constraints*) im DBMS definiert, die während der Laufzeit der Anwendungen vom System kontrolliert werden. Daneben wird aber auch die physische Integrität sichergestellt, d. h. die Gewährleistung intakter Speicherstrukturen und Inhalte (Speicherkonsistenz).

Datenzugriffskontrolle. Durch die Festlegung von Regeln kann der unautorisierte Zugriff auf die in der Datenbank gespeicherten Daten verhindert werden (*access control*). Dabei kann es sich um personenbezogene Daten handeln, die datenschutzrechtlich relevant sind, oder um firmenspezifische Datenbestände. Rechte lassen sich auch auf Sichten definieren.

Transaktionen. Mehrere Datenbankänderungen, die logisch eine Einheit bilden, lassen sich zu Transaktionen zusammenfassen, die als Ganzes ausgeführt werden sollen (Atomarität) und deren Effekt bei Erfolg permanent in der Datenbank gespeichert werden soll (Dauerhaftigkeit).

Mehrbenutzerfähigkeit. Konkurrierende Transaktionen mehrerer Benutzer müssen synchronisiert werden, um gegenseitige Beeinflussung, z. B. bei Schreibkonflikten auf gemeinsam genutzten Daten, zu vermeiden (*concurrency control*). Dem Benutzer erscheinen die Daten so, als ob nur eine Anwendung darauf zugreift (Isolation).

Datensicherung. Das DBMS muss in der Lage sein, bei auftretenden Hard- oder Softwarefehlern wieder einen korrekten Datenbankzustand herzustellen (*Recovery*).

1.3.2 Vorteile des Datenbankeinsatzes

Zusätzlich zu den genannten Merkmalen ergibt sich bei Einsatz eines Datenbanksystems eine Reihe von Vorteilen:

Nutzung von Standards. Der Einsatz einer Datenbank erleichtert Einführung und Umsetzung zentraler Standards in der Datenorganisation. Dies umfasst Namen und Formate von Datenelementen, Schlüssel, Fachbegriffe.

Effizienter Datenzugriff. Ein DBMS gebraucht eine Vielzahl komplizierter Techniken zum effizienten Speichern und Wiederauslesen (retrieval) großer Mengen von Daten.

Kürzere Softwareentwicklungszeiten. Ein DBMS bietet viele wichtige Funktionen, die allen Anwendungen, die auf eine Datenbank zugreifen wollen, gemeinsam ist. In Verbindung mit den angebotenen Datenbanksprachen wird somit eine schnellere Anwendungsentwicklung ermöglicht, da der Programmierer von vielen Routineaufgaben entlastet wird.

Hohe Flexibilität. Die Struktur der Datenbank kann bei sich ändernden Anforderungen ohne große Konsequenzen für die bestehenden Daten und

die vorhandenen Anwendungen modifiziert werden (Datenunabhängigkeit, → 1.4.3).

Hohe Verfügbarkeit. Viele transaktionsintensive Anwendungen (z. B. Reservierungssysteme, Online-Banking) haben hohe Verfügbarkeitsanforderungen. Ein DBMS stellt die Datenbank allen Benutzern dank der Synchronisationseigenschaften *gleichzeitig* zur Verfügung. Änderungen werden nach Transaktionsende sofort sichtbar.

Große Wirtschaftlichkeit. Die durch den Einsatz eines DBMS erzwungene Zentralisierung in einem Unternehmen erlaubt die Investition in leistungsstärkere Hardware, statt jede Abteilung mit einem eigenen (schwächeren) Rechner auszustatten. Somit reduziert der Einsatz eines DBMS die Betriebs- und Verwaltungskosten.

1.3.3 Nachteile von Datenbanksystemen

Trotz der Vorteile eines DBMS gibt es auch Situationen, in denen ein solches System unnötig hohe Zusatzkosten im Vergleich zur traditionellen Dateiverarbeitung mit sich bringen würde:

- Hohe Anfangsinvestitionen für Hardware und Datenbanksoftware.
- Ein DBMS ist Allzweck-Software, somit weniger effizient für spezialisierte Anwendungen.
- Bei konkurrierenden Anforderungen kann das Datenbanksystem nur für einen Teil der Anwendungsprogramme optimiert werden.
- Mehrkosten für die Bereitstellung von Datensicherheit, Mehrbenutzer-Synchronisation und Konsistenzkontrolle.
- Hochqualifiziertes Personal erforderlich, z. B. Datenbankdesigner, Datenbankadministrator.
- Verwundbarkeit durch Zentralisierung (Ausweg Verteilung).

Unter bestimmten Umständen ist der Einsatz regulärer Dateien sinnvoll:

- Ein gleichzeitiger Zugriff auf die Daten durch mehrere Benutzer ist nicht erforderlich.
- Für die Anwendung bestehen feste Echtzeitanforderungen, die von einem DBMS nicht ohne weiteres erfüllt werden können.
- Es handelt sich um Daten und Anwendungen, die einfach und wohldefiniert sind und keinen Änderungen unterliegen werden.

1.3.4 Produkte

Tabelle 1.1 gibt einen Überblick über heute verfügbare DBMS verschiedener Hersteller und die zugrunde liegenden Datenbankmodelle, die eine große Vielfalt aufweisen. Ein „*“ in der Hersteller-Spalte bedeutet, dass das jeweilige DBMS von einem Entwicklerteam verantwortet wird.

Tabelle 1.1 DBMS-Produkte und -Hersteller

DBMS	Hersteller	Modell/Charakteristik
Adabas	Software AG	NF^2-Modell (nicht normalisiert)
Caché	InterSystems	multidimensional, „postrelational“
Cassandra	Apache	NoSQL, spaltenorientiert
CouchDB	Apache	NoSQL, dokumentenorientiert
DB2	IBM	objektrelational
db4o	Versant	objektorientiert, leichtgewichtig
Derby	Apache	relational, Java-basiert, leichtgewichtig
Firebird	*	relational, basierend auf InterBase
IMS	IBM	hierarchisch, Mainframe-DBMS
Informix	IBM	objektrelational
InterBase	Embarcadero	relational
MongoDB	MongoDB	NoSQL, dokumentenorientiert
MS Access	Microsoft	relational, Desktop-System
MS SQL Server	Microsoft	objektrelational
MySQL	Oracle	relational
Neo4J	Neo Technology	NoSQL, Graphdatenbank
Oracle	Oracle	objektrelational
PostgreSQL	*	objektrelational
SAP HANA	SAP	In-Memory-Datenbank
SQLite	*	relational, leichtgewichtig
Sybase ASE	Sybase/SAP	relational
Versant	Versant	objektorientiert
Visual FoxPro	Microsoft	relational, Desktop-System
Teradata	Teradata	relational, massiv parallel

1.4 Architektur eines Datenbanksystems

1.4.1 Architekturen

Der Begriff **Architektur** kann bei Datenbankanwendungen einen unterschiedlichen Kontext haben. Hierbei lassen sich nennen:

- *Systemarchitektur* eines DBMS: Es werden die Komponenten eines DBMS beschrieben, die sich in einem Schichtenmodell anordnen lassen. In Standardisierungsvorschlägen werden die Schnittstellen zwischen den DBMS-Komponenten genormt, nicht jedoch diese selbst.

- *Schemaarchitektur* eines DBS: Diese unterscheidet zwischen verschiedenen Abstraktionsebenen, in denen die Daten einer Datenbank repräsentiert werden (→ 1.4.2).
- *Anwendungsarchitektur*: In datenbankbasierten Anwendungen erfolgt die Aufteilung der Funktionalität in verschiedene logische Schichten (tiers), z. B. unter Performance-Gesichtspunkten. Dabei wird die Datenhaltung zumeist einem DBMS zugeordnet (Back-End) sowie das Zusammenwirken der Benutzerkomponenten bei der Abarbeitung einer DB-Anwendung beschrieben.
- *Verteilte Architektur*: Aspekte der Verteilung können alle drei der oben genannten Architekturen betreffen. Sie bilden somit eine eigenständige Dimension von Architektur. So kann ein DBS als verteiltes DBS auf mehreren Knoten eines Netzwerks operieren oder als föderiertes DBS aus mehreren (heterogenen) DBS bestehen.

► *Hinweis:* Architektur und Komponenten eines Datenbanksystems werden vertiefend in /1.3/, /1.6/, /1.7/ beschrieben.

1.4.2 Schemaarchitektur

1.4.2.1 Datenbankschema

Für jedes Datenmodell ist es wichtig, zwischen der Beschreibung der Struktur und dem Inhalt der Datenbank zu unterscheiden. Die strukturelle Beschreibung einer Datenbank auf Basis eines Datenmodells wird als **Datenbankschema** bezeichnet, das im Verlauf des Datenbankentwurfs jeweils in einer bestimmten Sprache (DDL) spezifiziert wird. Ein grafisch dargestelltes Schema wird Schemadiagramm genannt.

❑ *Beispiel:* Für die Speicherung von Lieferantendaten gibt es je nach Datenmodell (Text, Relationenmodell, XML) unterschiedliche Schemadefinitionen (→ Bild 1.1).

Ein Datenbankschema kann spezifisch für eine Anwendung definiert werden. Daneben gibt es aber auch vordefinierte Schemata für bestimmte Kategorien von Anwendungen, auch als *Referenzmodell* bezeichnet. Diese unterliegen einer Standardisierung, erleichtern den Datenaustausch und verkürzen die Entwicklungszeit von DB-Anwendungen.

❑ *Beispiel:* Für geografische Objekte in Geodatenbanken steht das ISO 19107 Spatial Schema als Referenzmodell zur Verfügung (→ 17.2.2).

Die in der Datenbank zu einem bestimmten Zeitpunkt gespeicherten Daten werden als Datenbankzustand bezeichnet. Jeder Datensatz stellt eine **Instanz** dar und entspricht einem Objekt (entity). Mit jeder Veränderung

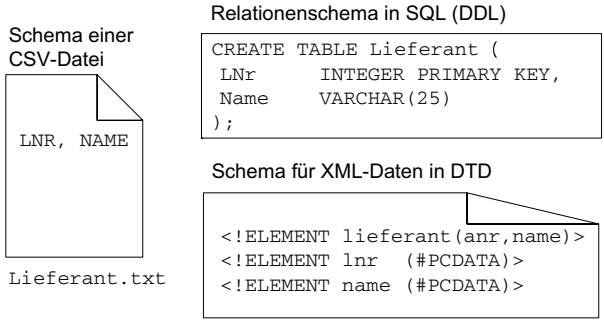


Bild 1.1 Schemadefinitionen im Vergleich

der Datenbank erhalten wir einen anderen Datenbankzustand. Das Schema wird auch als **Intension**, der Datenbankzustand als **Extension** bezeichnet. Die Veränderung des Schemas heißt **Schemaevolution**.

1.4.2.2 Drei-Ebenen-Architektur

Die Drei-Ebenen-Architektur (oder auch Drei-Schema-Architektur) beschreibt den grundlegenden Aufbau eines Datenbanksystems. Die Architektur wurde 1975 vom Standards Planning and Requirements Committee (SPARC) des American National Standards Institute (ANSI) entwickelt und formuliert das Prinzip der *Separation of Concerns* für Datenbanken /1.26/.

Die drei Ebenen umfassen Folgendes:

1. Auf der internen Ebene wird ein **internes Schema** definiert. Das physische Schema beschreibt die Details der physischen Speicherung der Daten (→ 9.3.3) und die Zugriffspfade wie z. B. Indexe (→ 8).
2. Auf der konzeptuellen Ebene wird ein **konzeptuelles Schema** definiert, das die Struktur der *gesamten* Datenbank für alle Benutzer der Datenbank beschreibt. Das konzeptuelle Schema verbirgt die Details der physischen Speicherung und konzentriert sich auf die Beschreibung der Objekte, Datentypen, Beziehungen, Operationen und Constraints. Auf dieser Ebene kann ein von den Speicherstrukturen abstrahierendes *logisches Datenmodell* benutzt werden, z. B. das relationale Datenmodell oder das Entity-Relationship-Modell.
3. Die externe Ebene beinhaltet eine Reihe **externer Schemata** oder **Benutzersichten (views)** basierend auf dem konzeptuellen Schema. Jedes externe Schema beschreibt einen anwendungsspezifischen Ausschnitt aus dem gesamten Datenbestand, der für eine bestimmte Benutzergruppe interessant ist, wobei die übrigen Daten aus der Datenbank

verborgen werden. Auf dieser Ebene kann ebenfalls ein von den Speicherstrukturen abstrahierendes logisches Datenmodell benutzt werden.

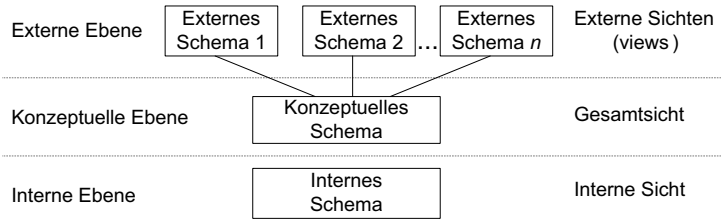


Bild 1.2 Drei-Ebenen-Architektur nach ANSI/SPARC

Zwischen den verschiedenen Schemaebenen müssen Abbildungen (mappings) festgelegt werden, durch die die Transformation von Anfragen und Datendarstellung in zwei Richtungen beschrieben werden kann:

- Die Anfragebearbeitung erfordert eine Übersetzung von Anfragen, die bezüglich der Sichten formuliert werden, in Operationen auf den internen Datenstrukturen (über den Zwischenschritt der konzeptuellen Ebene).
- Ergebnisse einer Anfrage müssen von den internen Datenstrukturen auf der physischen Ebene in das Datenmodell der externen Ebene umgewandelt werden.

Allerdings bedeuten beide Transformationen zusätzlichen Aufwand zur Laufzeit der Programme. Aus diesem Grund wird die volle Drei-Ebenen-Architektur nicht in allen DBMS unterstützt.

1.4.3 Datenunabhängigkeit

Ein wesentlicher Aspekt bei Datenbankanwendungen ist die Unterstützung der Datenunabhängigkeit durch das DBMS. Das Konzept der **Datenunabhängigkeit** hat das Ziel, eine oft langlebige Datenbank von notwendigen Änderungen der Anwendung abzukoppeln und umgekehrt. Die Datenunabhängigkeit drückt die Möglichkeit aus, das Schema auf einer Ebene zu ändern, ohne das Schema der nächsthöheren Ebene modifizieren zu müssen. Datenunabhängigkeit kann auf zwei Arten definiert werden:

Logische Datenunabhängigkeit

Dies besagt, dass das konzeptuelle Schema geändert werden kann, ohne externe Schemata oder Anwendungsprogramme anpassen zu müssen. Nur für die von einer Änderung der Datenbankstruktur betroffenen externen

Schemata müssen die Sichtdefinitionen modifiziert werden. Auf andere Sichten gibt es keine Auswirkungen.

❑ *Beispiel:* Das konzeptuelle Schema beinhaltet Tabellen über Artikel, Lieferanten und deren Lieferungen:

```
Lieferant (LNr,Name,...)
Artikel   (ANr,Bezeichnung,...)
Lieferung (ANr,LNr,Preis,... )
```

Auf der externen Ebene sind verschiedene Sichten denkbar. Zur Analyse der Umsätze könnte man z. B. ermitteln, welcher Artikel wie oft geliefert wurde. Die Sicht wird in einer Tabelle dargestellt: `ArtikelLief (ANr,Anzahl)`. In der Sichtdefinition ist die Berechnung der jeweils gelieferten Anzahl von Artikeln verborgen. Änderungen im konzeptuellen Schema (z. B. die Erweiterung um eine neue Spalte Lieferdatum) haben keine Auswirkung auf die bestehende Sicht.

Physische Datenunabhängigkeit

Dies bedeutet, dass das interne Schema modifiziert werden kann, ohne dass Schemaänderungen auf der konzeptuellen oder externen Ebene erforderlich sind. Änderungen des internen Schemas können z. B. geschehen bei der Reorganisation physischer Dateien oder der Erstellung zusätzlicher Indexe zur Zugriffsbeschleunigung. Wenn die gleichen Daten in der Datenbank bleiben wie zuvor, muss das konzeptuelle Schema nicht angepasst werden.

❑ *Beispiel:* Für die Artikelbezeichnungen wird eine Hash-Tabelle (→ 8.4) als zusätzlicher Suchindex angelegt, die Lieferungs-Datensätze sollen in einer Heap-Datei (→ 8.1) unsortiert abgelegt werden. Für die Tabellen `Artikel` und `Lieferung` ändert sich auf der konzeptuellen Ebene nichts.

1.5 Benutzerrollen bei Entwurf und Betrieb von Datenbanken

Bei Entwurf, Benutzung und Pflege einer großen Datenbank mit ein paar hundert Anwendern sind viele Personen mit unterschiedlichen Rollen beteiligt. Je nach Größe eines Unternehmens kann eine Person mehrere Rollen haben, oder eine Rolle kann auf mehrere Personen verteilt sein.

Endbenutzer sind Personen, die den Zugriff auf die Datenbank zur Erledigung ihrer Aufgaben in einem Unternehmen benötigen. Zu den Aufgaben zählen der Abruf oder die Erzeugung von Berichten (Reports) und die Aktualisierung von Daten. Endbenutzer können in mehrere Kategorien gegliedert werden:

- **Gelegentliche Benutzer** sind sehr häufig auf der Management-Ebene angesiedelt. Sie benötigen ausdrucksstarke, aber einfach zu handhabende Möglichkeiten des Datenzugriffs.

- **„Naive“ oder parametrische Benutzer** greifen auf die Datenbank für immer wiederkehrende Aktivitäten zu; sie verwenden dabei vordefinierte Anfragen und Standardprogramme. Beispiele sind Mitarbeiter an Fahrkartenschaltern oder Online-Banking-Anwender.
- **Professionelle Endbenutzer** haben komplexe Anforderungen und sind mit den Möglichkeiten eines DBMS gut vertraut, was ihnen die Entwicklung spezieller Anwendungen ermöglich. Dazu gehören z. B. Ingenieure, Naturwissenschaftler, Analysten.

Anwendungsprogrammierer implementieren auf Basis einer Spezifikation und mit einem gegebenen Datenbankschema die Applikationen, insbesondere für die naiven oder parametrischen Endbenutzer.

Softwarearchitekten benötigen universelles Wissen über Zusammenhänge und technische Möglichkeiten bei der Entwicklung komplexer Informationssysteme.

Datenbankdesigner ermitteln die Anforderungen der Benutzer, um die zu speichernden Daten festzulegen. Sie kommunizieren mit allen Benutzergruppen und entwickeln für jede von ihnen Sichten, die dann zu einem konzeptuellen Schema zusammengefasst werden. Der Datenbankdesigner ist zusammen mit dem Administrator verantwortlich für die Umsetzung dieses Schemas in einem spezifischen DBMS, ehe die Datenbank tatsächlich realisiert wird.

Der **Datenbankadministrator (DBA)** ist für die Verwaltung von Datenbank und DBMS verantwortlich. Dabei hat er überwiegend operative Aufgaben:

- Vergabe von Zugriffsrechten an Benutzer (Autorisierung),
- Sicherstellung des reibungslosen Betriebs des Datenbanksystems (Verfügbarkeit),
- Regelmäßige Datensicherungen und Archivkopien der Datenbank,
- Installation neuer Versionen des DBMS einschließlich kleinerer Software-Updates (Patches),
- Ansprechpartner für Probleme, die mit dem Datenbanksystem auftreten können, z. B. Sicherheit, Leistungsverhalten (Performance).

DBMS-Entwickler sind Softwareentwickler, die an der Implementierung eines DBMS beteiligt sind. Ein DBMS besteht als ein komplexes Softwaresystem aus vielen Komponenten und Schnittstellen. Hierfür beschäftigt ein DBMS-Hersteller (*vendor*) eine Vielzahl hochspezialisierter Entwickler. Diese haben normalerweise keinen Kontakt mit Endbenutzern der Datenbanksysteme.

1.6 Datenbanken und Informationssysteme

Datenbanksysteme haben sich in vielen Unternehmen zu einem festen Bestandteil der Informationssysteme entwickelt und finden darin ihr Hauptanwendungsgebiet.

> Ein **Informationssystem** (IS) dient der rechnergestützten Erfassung, Speicherung, Verarbeitung, Pflege, Analyse, Verbreitung, Übertragung und Anzeige von Informationen. Es ist ein soziotechnisches System, das menschliche und maschinelle Komponenten (Teilsysteme) für die optimale Bereitstellung von Information und Kommunikation umfasst.

Informationssysteme bestehen aus Hardware (Rechner oder Rechnerverbund), Software (System- und Anwendungssoftware, Middleware) und Daten (meist in einer Datenbank organisiert). Aus der Vielzahl unterschiedlicher Informationssysteme ergibt sich ein breites Einsatzfeld für Datenbanksysteme.

Klassifikation von Informationssystemen

Informationssysteme lassen sich in Bezug auf die benötigte Datenbankfunktionalität in drei Dimensionen charakterisieren: Datentyp, Operation, Suchalgorithmus.

Datentyp. Komplexität der zu verarbeitenden Daten
1. unformatierte Daten: Texte, Bilder, Audio-Files,
2. semistrukturierte Daten: HTML-Dateien, XML- oder JSON-Dokumente,
3. einfach strukturierte Daten: Tabellen,
4. komplex strukturierte Daten: CAD-Entwurfsobjekte, Proteinsequenzdaten.

Operation. Komplexität der Operationen zur Verarbeitung der Daten
1. retrievalorientiert (read-only): überwiegend lesende Transaktionen, die kurz sein können (bei Suche nach einzelnen Datensätzen) oder auch große Datenbestände verarbeiten müssen, z. B. für statistische Anwendungen,
2. updateorientiert: einfache Lese-Schreib-Transaktionen,
3. komplexe, lange Transaktionen: Transaktionen können sehr viele Operationen umfassen.

Suchalgorithmus. Komplexität der Algorithmen zur Gewinnung von Informationen aus der Datenbank
1. einfache Anfragen: exakte Suche, korrektes Ergebnis,

2. retrievalorientierte Anfragen: vage Suche, unvollständiges oder unsicheres Ergebnis,
3. Wissensextraktion (knowledge discovery): Data-Mining-Methoden.

❑ *Beispiele:* für Informationssysteme:

Management-Informationssystem (MIS): Daten in Tabellen – lange Lesetransaktionen – einfache und komplexe Suchalgorithmen

Geoinformationssystem (GIS): unformatierte, einfach und komplex strukturierte Daten – einfache und komplexe Lese- und Schreiboperationen – beliebig komplexe Suchalgorithmen (bis hin zu Spatial Mining)

1.7 Fachgebiet Datenbanken im Überblick

1.7.1 Themenbereiche und Zusammenhänge

Das Fachgebiet Datenbanken lässt sich in sechs Themenbereiche unterteilen, zwischen denen viele Abhängigkeiten bestehen (→ Bild 1.3). Bei der Entwicklung einer Datenbankanwendung wirken alle Bereiche zusammen. Entsprechend ist dieses Buch konzipiert.

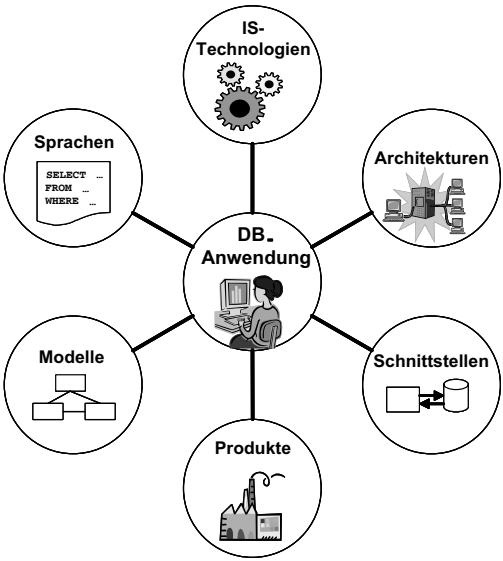


Bild 1.3 Themenbereiche im Fachgebiet Datenbanken

Modelle. Die Modellbildung ist fundamental für den DB-Entwurfsprozess. Dabei werden semantische Modelle (zur Informationsmodellierung) und Datenbankmodelle unterschieden (→ 1.2).

► *Hinweis:* Im Buch erfolgt schwerpunktmäßig die Darstellung des ERM zur Informationsmodellierung (→ 2). Als Datenbankmodell wird das Relationenmodell (→ 3) mit objektorientierten Erweiterungen (→ 10.1) vorgestellt, ergänzend dazu XML (→ 11) und NoSQL-Datenmodelle (→ 12.3). Mehrdimensionale Datenmodelle sind eine besondere Kategorie für OLAP-Systeme (→ 14.2, 14.3). Geodatenmodelle kommen bei GIS zum Einsatz (→ 17.1, 17.2).

Sprachen. Der Funktionsumfang einer Datenbanksprache ist abhängig vom Datenbankmodell, dessen Operationen den Benutzern oder anderen Systemkomponenten außerhalb des DBMS angeboten werden.

► *Hinweis:* Im Buch erfolgt schwerpunktmäßig die Darstellung von SQL als relationale Datenbanksprache (→ 4) zusammen mit den objektrelationalen Erweiterungen (→ 10). Für XML werden XML Schema als Datendefinitionssprache (→ 11.1) sowie XQuery als Anfragesprache (→ 11.2) vorgestellt. Anfragesprachen für NoSQL-Datenbanken befinden sich noch in Entwicklung (→ 12.4).

Bezeichner in Sprachen (z. B. bei Tabellen) werden in allen Kapiteln zur Unterscheidbarkeit gegenüber denen in Modellen (z. B. bei Relationen) im `Letter-Gothic`-Font dargestellt.

Schnittstellen. Die Kommunikation zwischen der Datenbank und höherliegenden Informationssystem-Komponenten erfolgt durch Nutzung von Schnittstellen der Datenbanksprache (zumeist SQL). Dabei kann SQL in eine Umgebungssprache (host language) eingebettet werden oder in dynamische Dokumente (XML, HTML). Durch die Einführung von SQL/PSM (→ 5.4.1) und andere Erweiterungen erreicht SQL mittlerweile die Ausdrucksmächtigkeit von Programmiersprachen, was neue Fragen nach der Softwarearchitektur einer DB-Anwendung aufwirft.

► *Hinweis:* Das Buch behandelt Schnittstellen einer Datenbank zu Programmiersprachen am Beispiel von Java (→ 5) sowie zum Web über Skriptsprachen, dynamisches HTML und Webservices (→ 6). XML-Dokumente können zum Zwecke des Datenaustauschs aus Datenbanken heraus generiert werden (→ 11.3.2).

Architekturen. Die Architektur eines DB-basierten Anwendungssystems umfasst verschiedene Sichtweisen (→ 1.4.1), beeinflusst durch das zugrunde liegende Datenbankmodell, die physische Verteilung der Komponenten (→ 13) sowie die Wahl der Schnittstellen.

► *Hinweis:* Im Buch werden die Komponenten eines DBMS vorgestellt (→ 7) sowie Basismechanismen der Dateiorganisation und Indexierung (→ 8). Diese Kenntnisse sind eine fundamentale Voraussetzung zum Verständnis der Optimierung von großen Datenbanken (→ 9).

IS-Technologien. Die erweiterte Funktionalität eines DBMS umfasst Technologien, die die spezifischen Anforderungen bestimmter Informationssysteme (IS) erfüllen, z. B. Management-Informationssysteme oder Geoinformationssysteme.

► *Hinweis:* Im Buch werden als IS-spezifische Technologien Data Warehousing (→ 14), Data Mining (→ 15), Multimedia-Datenbanken (→ 16) sowie Geodatenbanken (→ 17) vorgestellt. Dazu gehört eine Darstellung von IS-spezifischen Architekturen, Datenmodellen und Sprachen.

Produkte. Produkte von DBMS-Anbietern unterscheiden sich im Datenbankmodell und in den mitgelieferten Werkzeugen. Auch bei gleichem Datenbankmodell weisen sie im Detail Unterschiede auf, z. B. in Form von SQL-Dialekten oder unterschiedlichen Typsystemen.

► *Hinweis:* Einzelne DBMS werden in diesem Buch nicht behandelt, da die gemeinsamen Datenbankkonzepte im Vordergrund stehen. Es erfolgen jedoch exemplarisch Hinweise auf bestimmte DBMS. Aufgrund der ständigen technologischen Weiterentwicklung der DBMS sei auf produktspezifische Literatur (z. B. die Reihe Oracle Press™) verwiesen.

1.7.2 Einordnung des Fachs innerhalb der Informatik

Das Fach Datenbanken besitzt innerhalb der Informatik (und darüber hinaus) eine Vielzahl von Schnittstellen zu anderen Disziplinen /1.8/. Dabei baut es auf diesen auf oder leistet selbst Beiträge zu den weiteren Fachgebieten. Perspektivisch ist eine stärkere *Konvergenz* in vielen Bereichen zu erwarten. Nachfolgend werden einige Zusammenhänge skizziert.

Softwaretechnik. Entwurf und Implementierung einer Datenbank sind Bestandteil des Entwicklungsprozesses eines Softwaresystems, sofern dieses datenbankbasiert ist. Die in der Softwaretechnik beschriebenen Vorgehensmodelle sind auch für den Datenbankentwurf nutzbar, z. B. für die Erhebung der Datenanforderungen (requirements engineeering).

Programmiersprachen. Datenbankmodelle folgen als polymorphe Typsysteme den Konventionen formaler Sprachen. Datenbanksprachen hatten sich zunächst unabhängig von Programmiersprachen entwickelt. Die Erweiterungen von SQL durch Konstrukte von Programmiersprachen und objektorientierte Merkmale lassen jedoch eine deutliche Tendenz zum Zusammenwachsen mit herkömmlichen Programmiersprachen erkennen.

Algorithmen und Datenstrukturen. Sortierverfahren und Indexstrukturen bilden einen wesentlichen Bestandteil der Implementierung eines DBMS auf der physischen Ebene.

Betriebssysteme. Einige Komponenten eines DBMS weisen Gemeinsamkeiten zu Betriebssystemen auf. Dies betrifft z. B. die Dateiverwaltung oder die Synchronisation von parallelen Prozessen.

Verteilte Systeme. Die Verteilung von Datenbanken in Netzwerken hat zahlreiche Auswirkungen auf den Entwurf von Datenbanken und Schnittstellen sowie die Architektur eines DBMS, da neue Aspekte wie Kommunikationskosten, Verfügbarkeit oder Parallelverarbeitung hinzukommen. *Mobile Datenbanken* als moderne Entwicklung sind zusätzlich durch Drahtlosigkeit und räumliche Beweglichkeit gekennzeichnet.

Künstliche Intelligenz. Datenbanken sind nicht nur passive Datenspeicher, sondern können auch zur Repräsentation und Verarbeitung von Regeln (constraints) genutzt werden, oft als Wissensbanken bezeichnet. Die Repräsentation von Wissen in Ontologien spielt eine zunehmende Rolle bei der Integration heterogener Datenbanken sowie als Grundlage von intelligenten Suchmaschinen im Semantic Web. Dabei können Zusammenhänge zwischen Daten mittels RDF (Resource Description Framework) beschrieben und abgefragt werden. Data-Mining-Methoden nutzen Algorithmen zur Mustererkennung in großen Datenbeständen.

Information Retrieval. Retrieval-Techniken (auf Basis von Indexen) sind seit vielen Jahren etabliert und gewinnen durch neue Einsatzfelder für Datenbanken wie Dokumente oder Multimedia an Bedeutung. Dabei etabliert sich *Information Extraction*, die automatisierte Gewinnung von Wissen aus unstrukturierten Informationen, als eigenständiges Gebiet.

Wirtschaftsinformatik. Datenbanken sind die Basis betrieblicher Informationssysteme und als Repository (Speicher) zur Verwaltung von Metadaten über Informationen und Geschäftsprozesse unverzichtbar zur Integration heterogener Systeme in einem Unternehmen. Der Datenbankentwurf ist dabei ein wichtiger Bestandteil der Modellbildung und wird davon beeinflusst.

Datenschutz. Das Bundesdatenschutzgesetz (BDSG) definiert rechtliche Anforderungen zum Schutz persönlicher Daten und formuliert dabei den Grundsatz der Datenvermeidung und Datensparsamkeit. Die Beachtung dieser Rahmenbedingungen ist für den Datenbankentwurf unabdingbar. Hierzu zählt auch der Umgang mit biometrischen Daten (Fingerabdrücke, Iris). Darüber hinaus gewinnen Techniken zum Schutz von Anonymität und Unbeobachtbarkeit (privacy) in Datennetzen wachsende Bedeutung.

1.8 Historische Entwicklung

Folgende Ereignisse sind von besonderer Bedeutung in der Geschichte der Datenbanksysteme und können als **Meilensteine** für die Entwicklung der Datenbanktechnologie angesehen werden:

1889 Erfindung der Lochkarte durch Hermann Hollerith zur schnelleren Abwicklung der Volkszählung in den USA.

1935 Erfindung des Trommelspeichers durch Gustav Tauschek.

1956 Erste Festplatte „IBM 350“ mit einer Kapazität von ca. 5 MB.

1968 IMS, erstes hierarchisches DBMS, geht bei Rockwell in Betrieb.

1970 Veröffentlichung des Relationenmodells durch Edgar F. Codd /1.16/.

1971 Erste Veröffentlichung des offiziellen Netzwerkmodells durch die CODASYL DBTG auf Basis der Arbeiten von Charles Bachman.

1975 Die ANSI-SPARC-Architektur (Drei-Ebenen-Architektur) wird als Referenz für Datenbanksysteme veröffentlicht.

1976 Publikation des Entity-Relationship-Modells (ERM) durch Peter Chen /1.15/.

1976 System R, Prototyp eines relationalen DBMS, wird von IBM vorgestellt (Vorläufer von SQL/DS bzw. DB2).

1979 Oracle, erstes kommerzielles RDBMS, erscheint auf dem Markt.

1982 dbase II, RDBMS für den IBM-PC, ist auf dem Markt erhältlich.

1986 Erster Standard der Datenbanksprache SQL (ANSI).

1989 Das „Object Oriented Database Manifesto“ beschreibt die wesentlichen Anforderungen an objektorientierte Datenbanken /1.12/.

1989 Erfindung des World Wide Web (WWW) durch Tim Berners Lee am CERN Genf.

1995 MySQL wird erstmals im Internet als kostenloses Open-Source-DBMS angeboten.

1998 Veröffentlichung der W3C Recommendation für XML 1.0.

2000 Das *Sloan-Digital-Sky-Survey*-Projekt zur digitalen Erfassung des Himmels gilt als erstes Projekt im Bereich Big Data.

2004 Google Inc. veröffentlicht MapReduce /1.18/, ein Programmiermodell zur Verarbeitung großer Datenmengen.

2009 Entwickler-Treffen über verteilte strukturierte Datenspeicher prägt den Begriff NoSQL für nichtrelationale Web-2.0-Datenbanken.

2014 SAP stellt zusammen mit Technologiepartnern einen neuen Rekord für die weltweit größte Datenbank auf. Das Data Warehouse basiert auf SAP HANA und hat eine Kapazität von 12.1 PB (Petabyte).

Nachfolgend werden die wichtigsten **Entwicklungslinien** skizziert.

1.8.1 Frühzeit

Der Ingenieur Hermann Hollerith gilt als Pionier der modernen Datenverarbeitung. Das von ihm erfundene **Lochkartenverfahren** wurde erstmalig 1890 zur Volkszählung in den USA eingesetzt. Aus der von Hollerith 1896 gegründeten Tabulating Machine Company ging 1924 die IBM hervor. Mit der Entwicklung der ersten Computer im 20. Jahrhundert etablierten sich zunächst Lochkarten als Medium zur langfristigen Speicherung der zu verarbeitenden Daten. Die Entwicklung **elektromagnetischer Speichermedien** für digitale Daten eröffnete in den 1950er und 1960er Jahren dann neue Möglichkeiten

Der erstmalige Gebrauch des Begriffs „data base“ geht zurück auf ein Symposium im Juni 1963 unter dem Titel „Development and Management of a Computer-centered Data Base“, ausgerichtet von der US-Softwarefirma SDC (System Development Corporation) /1.25/. In Deutschland wurde 1967 in einer Fachzeitschrift die Frage thematisiert: „Was ist eine Datenbank?“ /1.23/. Die ersten deutschsprachigen Lehrbücher, die bereits Datenbanken als Grundlage von Informationssystemen beschreiben, erschienen 1970 bzw. 1971 /1.27/, /1.24/. Einen Schwerpunkt bildete in dieser Zeit die Organisation der Daten in Dateien und deren Indexierung, wobei **ISAM** als indexsequenzielle Methode besondere Popularität genoss.

1.8.2 Prärelationale DBMS

Die ersten (*prärelationalen*) Datenbankmanagementsysteme wurden in den 1960er Jahren entwickelt. Ausgangspunkt waren Probleme bei der Verarbeitung herkömmlicher Dateien, die bisher nur anwendungsspezifisch angelegt und verwaltet wurden. DBMS konzipierte man hingegen als Softwareschicht zur Verwaltung der Daten zwischen Betriebssystem (Dateiverwaltung) und Anwendungsprogramm.

Einer der ersten Pioniere auf diesem Gebiet war Charles Bachman, der für die Entwicklung von Datenbanken das Potenzial der damals neuen **Plattenspeicher** mit der Möglichkeit des *Direktzugriffs* erkannte. Die Database Task Group des Konsortiums **CODASYL** (Conference On Data Systems Languages) entwickelte ein Netzwerk-Datenbankmodell, basierend auf den Ideen von Bachman („the programmer as a navigator“) /1.13/. Auf dieser Grundlage wurde eine Vielzahl von kommerziellen DBMS entwickelt,

so z. B. das 1976 eingeführte *UDS* (Universelles Datenbank-System) von Siemens.

Unabhängig davon wurde etwa zur gleichen Zeit auf der Basis des hierarchischen Modells eines der ersten großen DBMS von North American Rockwell implementiert, das später von IBM im System **IMS** zusammen mit der Sprache *DL/I* übernommen wurde.

1.8.3 Durchbruch der relationalen Datenbanken

Einen wesentlichen Fortschritt erzielte seit Ende der 1960er Jahre Edgar F. Codd am IBM Almaden Research Center. Das von ihm 1970 vorgeschlagene **Relationenmodell** /1.16/ überwand die Vermischung von logischen Informationsstrukturen und physischen Zugriffsmechanismen, die für die damals existierenden Datenbankmodelle kennzeichnend war. Über viele Jahre hinweg war das Relationenmodell nur von akademischem Interesse, wohingegen CODASYL und IMS als Systeme für die Praxis galten. Bei IBM wurden die Grundlagen des ersten experimentellen relationalen Datenbanksystems im Projekt **System R** gelegt /1.11/. Darauf aufbauend entwickelten Don D. Chamberlin und Raymond F. Boyce die relationale Datensprache *SEQUEL* (Structured English Query Language), aus der später SQL hervorging. Parallel dazu entstand unter Michael Stonebraker an der Berkeley University das relationale System **Ingres** mit der Abfragesprache *QUEL*.

Oracle (damals noch unter den Firmennamen SDL und RSI) verwertete die Ergebnisse von System R und führte **SQL** zum kommerziellen Erfolg. IBM folgte mit *SQL/DS* und **DB2**. Die relationalen Datenbanksysteme verdrängten seit Beginn der 1980er Jahre die hierarchischen und netzwerkartigen Systeme, und der Großteil der Anwender stellte seine IT auf Datenbanksysteme um. Der Siegeszug der relationalen Systeme lag in der Standardisierung von SQL sowie in deren zusätzlicher Verbreitung auf Personalcomputern begründet. Hier war **dBASE** das erste erfolgreiche Produkt für die PC-Betriebssysteme CP/M und MS-DOS.

Deutsche Forscher leisteten schon in dieser Zeit hervorragende Beiträge. So begründete Hartmut Wedekind das Fach Datenbanken als *wissenschaftliche Disziplin* in Deutschland. Rudolf Bayer gilt (zusammen mit E. McCreight) als der Erfinder des *B-Baums* zur Indexierung von Datensätzen in einem DBMS /1.14/. Theo Härder hatte mit seinen Arbeiten über die *Architektur von DBMS* großen Einfluss auf deren Entwicklung /1.21/. So führte er auch den Begriff *ACID* bei der Definition von Transaktionen ein, basierend auf /1.19/. Bereits im Jahre 1971 brachte die Software AG ein kommerzielles DBMS, *Adabas* (Adaptable Database System), auf den

Markt, das auf invertierten Listen beruhte und noch heute in der Industrie eingesetzt wird.

Datenbanksysteme wurden in den 1980er Jahren überwiegend für *administrativ-betriebswirtschaftliche* Anwendungen eingesetzt. Die Vernetzung von Rechnern führte zu neuen Fragestellungen hinsichtlich verteilter und heterogener Datenbanken, die man bisher ausschließlich zentralisiert betrachtet hatte /1.17/.

1.8.4 Neue Anwendungsfelder für Datenbanken

Ende der 1980er Jahre richtete sich die Aufmerksamkeit auf die Entwicklung von **objektorientierten DBMS** /1.12/. Diese wurde insbesondere durch neuartige Ingenieur-Anwendungen (z. B. CAD) getrieben, auch als Non-Standard-Anwendungen bezeichnet. Sie benötigten die Verwaltung komplexer Datenstrukturen, wofür Relationen ungeeignet erschienen. Einige der dort entwickelten Ideen, z. B. die Erweiterbarkeit des Datenbankmodells um neue Typen, fanden Eingang in die Produkte der relationalen Anbieter, die ihre Systeme um entsprechende Features erweiterten und dafür den Begriff **objektrelational** prägten. Davon profitierten insbesondere *Multimedia-Datenbanken* und *Geodatenbanken*.

Mit der Verbreitung des Internet und des **WWW** ab Mitte der 1990er Jahre entstand der Bedarf, Inhalte von Webseiten dynamisch zu erzeugen. Hierfür wurde eine Vielzahl von Technologien zur Anbindung von Datenbanken entwickelt, die Zahl der Datenbankbenutzer nahm rasant zu.

Management-Informationssysteme als klassische Anwendungsgebiete von Datenbanken erhielten neue Impulse durch die Entwicklung von Data-Warehouse-Technologien und Data-Mining-Algorithmen, wofür auch der Begriff *Business Intelligence* geprägt wurde. Bill Inmon gilt als „Vater“ der **Data Warehouses** /1.22/, Rakesh Agrawal hat mit seinen Arbeiten das Gebiet des **Data Mining** entscheidend beeinflusst /1.10/.

1.8.5 Neuzeit

Während in den 1990er Jahren wenige kommerzielle Hersteller von Datenbank-Software faktisch den Markt beherrschten (Oracle, IBM, Informix und Microsoft mit SQL Server), erlangten in den 2000ern die **Open-Source-Datenbankmanagementsysteme** eine immer größere Bedeutung. Vor allem MySQL erfreute sich wachsender Beliebtheit mit weltweit mehreren Millionen Installationen. Als Reaktion darauf bieten die führenden kommerziellen Hersteller gebührenfreie Versionen ihrer Datenbank-Software an.

Eine weitere Innovation seit der Jahrtausendwende bestand in der Entwicklung von **XML-Datenbanken** durch eine Vielzahl kleiner Anbieter (ähnlich

wie zu den Zeiten der Objektdatenbanken). Mittlerweile wurden etablierte relationale Datenbanksysteme um XML-Funktionalität erweitert. **NoSQL-Datenbanken** mit unterschiedlichen Modellen entstanden als Alternative zu relationalen Systemen als Plattform für Web-2.0-Anwendungen wie Facebook oder Twitter. Diese zeichnen sich durch große Flexibilität und hohe Leistungsfähigkeit im Umgang mit großen Datenmengen durch verteilte Verarbeitung aus.

Parallel dazu eroberten die Datenbanken weitere neue Anwendungsgebiete als Scientific Databases (z. B. in der Biologie) oder als Plattform digitaler Bibliotheken. Große Datenbanken erreichen den Petabyte-Bereich (1 PB = 10^{15} Byte). Für den Umgang mit großen Datenmengen wurden neue Technologien entwickelt, die unter dem Begriff **Big Data** zusammengefasst werden und dabei die Grenzen klassischer Datenbanksysteme überwinden. Datenbanken können dabei auch im Rahmen des so genannten **Cloud Computing** als Dienst angeboten werden.

1.9 Datenbanken in der Forschung

1.9.1 Fachverbände

Datenbankexperten aus Lehre, Forschung und Industrie sind in der Gesellschaft für Informatik (**GI**) im Fachbereich „Datenbanken und Informationssysteme“ (DBIS) /www.gi.de/ organisiert. Der GI-Fachbereich vereinigt verschiedene Fachgruppen (FG), darunter auch die FG Datenbanksysteme. Die FG Datenbanksysteme veranstaltet halbjährlich ein nationales Treffen und alle 2 Jahre die wissenschaftliche Tagung **BTW** (Datenbanken in Business, Technology & Web). Die Zeitschrift „Datenbank-Spektrum“ informiert die Mitglieder der FG regelmäßig über aktuelle Trends in Forschung und Entwicklung. Darüber hinaus gibt es innerhalb der Fachgruppe Arbeitskreise für spezielle oder aktuelle Themen.

Anwender von kommerziellen DBMS können sich Anwendergruppen (user groups) anschließen. In Deutschland ist in diesem Bereich die **DOAG** (Deutsche Oracle-Anwendergruppe) sehr aktiv /www.doag.de/, die auch eine Reihe von Special Interest Groups (SIG) für einzelne Themen sowie Regionalgruppen umfasst.

International ist die **SIGMOD** (Special Interest Group on Management Of Data) innerhalb der amerikanischen **ACM** (Association of Computing Machinery) hervorzuheben /www.sigmod.org/. Die weltweit renommiertesten wissenschaftlichen Datenbank-Tagungen sind die ACM SIGMOD Conference sowie die **VLDB** Conference (Very Large Databases) – beide seit 1975.

1.9.2 Aktuelle Forschungstrends

Jim Gray formuliert in /1.20/ Thesen, die die möglichen Veränderungen von Datenbanksystemen beschreiben. Die darin skizzierten Forschungstrends haben auf die künftige Entwicklung der Datenbanktechnologie revolutionären Einfluss.

Sehr bedeutsam ist die Zusammenführung von Algorithmen und Daten durch die Integration von (objektorientierten) Programmiersprachen mit dem Datenbanksystem zu einem *objektrelationalen* System. Damit zusammenhängend ist jedes DBMS zugleich auch ein *Service*, was große Auswirkungen auf die Anwendungsarchitektur hat. Im Rahmen der Virtualisierung können in einer Cloud Datenbankdienste angeboten werden, die jeweils von unterschiedlichen Mandanten genutzt werden.

DBMS werden zunehmend um Datenwürfel und *OLAP*-Funktionalität erweitert. Alternative Ansätze zur Speicherung und Verarbeitung dünnbesetzter Tabellen mit sehr vielen Spalten gewinnen dabei an Bedeutung (column stores). Darüber hinaus werden Algorithmen des *Data Mining* und des maschinellen Lernens Bestandteil eines DBMS. Dazu gehören z. B. Entscheidungsbäume, neuronale Netze, Zeitreihenanalysen.

Methoden zum Zugriff auf *Text, temporale und räumliche Daten* vergrößern den Funktionsumfang eines DBMS bei Erweiterung des Typsystems. Dies beinhaltet probabilistisches (d. h. unsicheres) Schließen und unscharfe Antworten, wie sie aus dem *Information Retrieval* bekannt sind. DBMS müssen relationale Daten und schwach strukturierte Daten in prinzipiell gleicher Weise behandeln können. So kann ein Datenbanksystem auch auf einem hoch skalierbaren verteilten Dateisystem basieren, das Daten mit geringer oder flexibler Struktur verarbeitet. Methoden zur Extraktion und Integration von Informationen aus unterschiedlichsten Quellen, z. B. Webdaten aus sozialen Netzwerken, gehören seit Jahren zu den Forschungsthemen der Datenbankwelt.

Eine wachsende Menge von Daten wird durch Beobachtung der Umwelt erzeugt (z. B. RFID-Scanner, Monitoring von Patientendaten oder Straßenverkehr). Diese Mengen nennt man auch *Datenströme* (data streams). Hierfür werden spezielle Datenstrommanagementsysteme (DSMS) entwickelt, die eine datengetriebene Verarbeitung der Datenströme ermöglichen, ohne diese persistent in einer herkömmlichen Datenbank zu halten. Hierfür steht auch der Begriff *Complex Event* Processing (*CEP*), der die kontinuierliche und zeitnahe Verarbeitung von Ereignissen beschreibt mit dem Ziel der Erkennung von Situationen durch Kombination vieler Einzelereignisse. *Publish-Subscribe-Systeme* kehren das traditionelle Verhältnis von Anfrage und Daten um, d. h., ankommende Daten werden mit Tausenden oder Mil-

lionen von Anfragen (sog. standing queries) verglichen – umgekehrt zum gewohnten Vorgehen. Diese Technologie stellt eine Weiterentwicklung der Ideen aktiver Datenbanken aus den 1990er Jahren dar. Sie ermöglicht Benachrichtigungsmechanismen bei Anwendungen, die flexibel auf Ereignisse (z. B. Preisänderungen) reagieren müssen. Periphere Geräte (z. B. Disk-Controller, Sensoren) können sich selbst wie eine Datenbank verhalten, werden somit zu *Smart Objects* im Internet der Dinge. Diese Mini-Datenbanken (embedded databases) müssen sich als *adaptierbare* und *autonome* Systeme selbst administrieren, ein Trend, der auch für herkömmliche DBMS gilt.

Die jüngsten Fortschritte bei Hardware und Rechnerarchitektur sind die Basis eines aktuellen Forschungsgebiets in der Datenbank-Szene, *Data Management on New Hardware*. Dabei geht es um neue Ansätze durch moderne Prozessorarchitekturen und Speichermedien als Alternative zur herkömmlichen Festplatte. Prozessoren mit mehreren Kernen (Multicore) ermöglichen neue Algorithmen zur Indexierung und Anfrageverarbeitung durch Parallelisierung. Ebenso erlauben größere Hauptspeicher im Terabyte-Bereich oder neue Medien wie Solid-State-Disks effizientere Implementierungen von Anfrage-Operationen und erweitern die bisherige Systemarchitektur eines DBMS (z. B. als In-Memory-Datenbanken). Im Ergebnis können noch größere Datenmengen noch schneller verarbeitet werden, als dies bisher denkbar war.

Fazit

Das Fachgebiet Datenbanken als eine zentrale Säule der Informatik wird sich auch künftig weiterentwickeln in Richtungen, die durch die Forschung bereits angedeutet sind und durch Hardware-Verbesserungen möglich werden. Davon sind alle Bereiche betroffen: Modellierung, Sprachen, Schnittstellen, Architekturen und Technologien – letztendlich auch die Produkte. Dabei können *alternative Paradigmen* eingeführt werden, z. B. im Verhältnis von Anfragen und Daten.

Der auffälligste Trend besteht hierbei im *Zusammenwachsen* mit anderen Disziplinen der Informatik. So ist eine Konvergenz zwischen Daten und Algorithmen, zwischen exaktem und probabilistischem Schließen bei Anfragen sowie eine einheitliche Repräsentation von unstrukturierten, strukturierten und gemischten Daten zu verzeichnen. Dies macht in Vielem eine Neubestimmung der Position des Faches Datenbanken erforderlich. Die häufigste Prognose lautet hierbei: „Datenbanken werden überall sein, allerdings nahezu unsichtbar.“ Auch in Zukunft werden also Datenbanken den Fortschritt der Informatik entscheidend mitprägen und damit letztendlich auch die Gesellschaft (positiv) verändern.

2 Informationsmodellierung

Petra Sauer

Die Informationsmodellierung beschäftigt sich mit der (formalen) Beschreibung der Datenbestände eines Anwendungsbereiches. Als eine der Kerntätigkeiten des Datenbankentwurfs hat sie entscheidenden Einfluss auf die Qualität einer Datenbank (DB). Als Beschreibungsformalismus werden das Entity-Relationship-Modell (ERM) (→ 2.2) oder der Standard UML (Unified Modelling Language) /2.10/ verwendet.

2.1 Datenbankentwurf

Das Hauptanliegen des Datenbankentwurfs ist die Definition der Struktur der DB. Zur Umsetzung der Drei-Ebenen-Architektur (→ 1.4) gehört die Entwicklung der externen sowie des konzeptuellen und internen Schemas. Als weiteres Entwurfsdokument wird das logische Schema definiert. Die Entwicklung der Schemata unterliegt einer strengen Reihenfolge und führt von einer zunächst abstrakten zu einer zunehmend konkreten Beschreibung der DB. Die Abhängigkeit von der konkreten Realisierung mit einem DBMS wird dabei immer größer: zunächst von dessen Datenmodell (DM), schließlich von System und DB-Sprache. Die Unabhängigkeit der frühen Entwurfsdokumente vom DBMS ermöglicht eine stabile Bezugsbasis für unterschiedliche Implementierungen der DB.

2.1.1 Phasenmodell

Der Entwurf einer DB findet meistens im Zusammenhang mit der Entwicklung eines Informationssystems (→ 1.6) statt und orientiert sich an den Entwurfsschritten des Phasenmodells des Software Engineering. Die Phasen des Datenbankentwurfs sind angepasst an die Bedürfnisse der Datenbankentwicklung: Anforderungsanalyse, konzeptioneller Entwurf, logischer Entwurf, Datendefinition auf Basis einer Datendefinitionssprache (Data Definition Language, DDL), physischer Entwurf, Implementierung und Wartung (→ Bild 2.1).

► *Hinweis:* In diesem Kapitel wird das grundsätzliche Vorgehen bei der Entwicklung einer DB behandelt. In Abhängigkeit von der geplanten Architektur der DB können weitere Phasen hinzukommen. Soll eine verteilte DB entwickelt werden, so ist ein Verteilungsentwurf vor dem logischen Entwurf vorzunehmen (→ 13.2). Ist eine DB auf Basis existierender Datenbestände zu entwerfen, ist eine Integration der vorhandenen Datenbanken vorzunehmen. Hierzu existieren

verschiedene Ansätze wie u. a. die DB-Föderation, die Gegenstand von Kapitel 13 sind.

Wie auch im Software Engineering werden die Phasen nicht streng sequenziell durchlaufen. Der Datenbankentwurf ist ein iterativer Prozess.

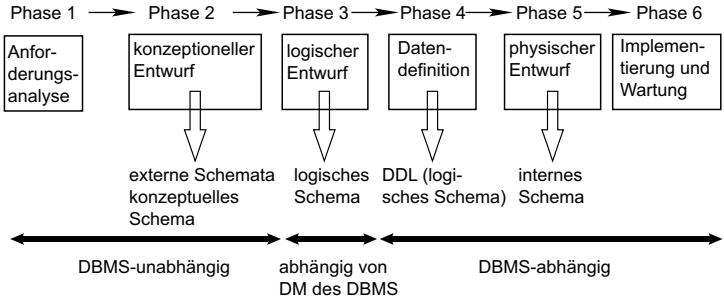


Bild 2.1 Phasenmodell des Datenbankentwurfs

2.1.2 Anforderungsanalyse

Die Anforderungen der potenziellen Benutzer an die DB werden erfasst, analysiert und klassifiziert. Die Beschreibung erfolgt *informal* in Form von Texten, Tabellen etc. Dabei werden Informations- von Bearbeitungsanforderungen unterschieden.

Informationsanforderungen werden im Rahmen der **Datenanalyse** erfasst und beschreiben die Daten unabhängig von der Auswertung.

Der DB-Entwurf beschäftigt sich mit den Informationsanforderungen.

Bearbeitungsanforderungen werden im Rahmen der **Funktionsanalyse** erfasst und beschreiben die Auswertungs- und Bearbeitungserfordernisse auf der DB.

Zu den Bearbeitungsanforderungen gehören Angaben über

- die Art der auszuführenden Bearbeitungsprozesse, wie z. B. Anfragen, Updates, Berichtsgenerierung,
- Quantitäten und Prioritäten der auszuführenden Bearbeitungsprozesse, z. B. Häufigkeit einzelner Prozesse, Datenvolumen, Zuwachsraten, Anforderungen an die Effizienz,
- Sicherheits- und Schutzanforderungen, z. B. Zugriffsrechte und Zugangsbestimmungen.

Der Funktionsentwurf auf Basis der Bearbeitungsanforderungen ist Gegenstand des Software Engineering /1.8/ und wird hier nicht behandelt.

2.1.3 Konzeptioneller Entwurf

Datenbanken stellen ein abstrahiertes Abbild der Realität dar. Da es sich dabei immer nur um Teilbereiche handelt, ist dafür auch der Begriff *Weltausschnitt* (**Universe of Discourse**, **UoD**) gebräuchlich.

❑ *Beispiel:* Weltausschnitte können sein: die Abteilung eines Unternehmens (z. B. Vertrieb), ein komplettes Unternehmen, eine bestimmte Funktion des Unternehmens (z. B. Web-Präsenz).

Auf Basis der Ergebnisse der Anforderungsanalyse erfolgt im konzeptionellen Entwurf eine *erste formale Beschreibung* des abzubildenden Weltausschnitts, auch als Modellierung des **Universe of Discourse** bezeichnet. Diese Beschreibung erfolgt unabhängig von der Realisierung in einem konkreten DBMS und unabhängig von konkreten Anwendungen, um eine stabile Basis für die weiteren Entwurfsphasen zu besitzen. Als Beschreibungsformalismus wird ein abstraktes Datenmodell, wie das ERM eingesetzt. Die Anforderungen der Benutzer werden in formale Schemabeschreibungen überführt. Daraus resultieren das **konzeptuelle** und die **externen Schemata**, Letztere auch als **Sichten** (views) bezeichnet.

Für die Schemaentwicklung in dieser Phase existieren zwei Ansätze:

- Beim **Top-Down-Ansatz** wird zunächst das konzeptuelle Schema modelliert. Externe Schemata werden als nicht notwendig disjunkte Ausschnitte des konzeptuellen Schemas abgeleitet.
- Beim **Bottom-Up-Ansatz** werden die externen Schemata modelliert. Das konzeptuelle Schema resultiert aus deren Integration. Externe Schemata spiegeln die Anforderungen einzelner Anwender wider. Diese können sich überschneiden und Widersprüche enthalten. Eine wichtige Tätigkeit im Rahmen der Schemaintegration ist die Analyse der Schemata, die Erkennung von Konflikten und deren Auflösung.

Bei der **Analyse der Sichten** werden Konflikte zwischen Sichten, Redundanzen und Widersprüchlichkeiten (Inkonsistenzen) aufgedeckt. Konflikte treten u. a. auf:

- als **Namenskonflikt,** bei unbeabsichtigten Namensüberschneidungen oder -abweichungen durch die Verwendung von Synonymen und Homonymen,
- als **Typkonflikt**, indem verschiedene Datenstrukturen für das gleiche Element, basierend auf unterschiedlichen Aussagen von Benutzern zum UoD, modelliert werden,

- als **Bedingungskonflikt**, durch die Verwendung unterschiedlicher Integritätsbedingungen in Sichten,
- als **Strukturkonflikt**, durch die Verwendung von verschiedenen Modellierungsvarianten zu einem Sachverhalt.

Bei der *Schemaintegration* werden diese Konflikte gelöst und Beziehungen zwischen den Sichten identifiziert (→ 13.7.3).

2.1.4 Logischer Entwurf

Auf Basis des konzeptuellen Schemas wird eine *weitere formale Beschreibung* der Datenstrukturen vorgenommen, wobei der Beschreibungsformalismus wechselt. Das logische Schema wird modelliert.

> Das **logische Schema** beschreibt die Datenstrukturen des UoD in Abhängigkeit vom konkreten Datenmodell des DBMS (Datenbankmodell).

Beispielsweise ist in dieser Phase die Entscheidung zu treffen, ob die Realisierung der DB mit einem konventionellen, meist relationalen DBMS (→ 3) oder einem objektorientierten oder objektrelationalen DBMS (→ 10) erfolgen soll. Es werden damit Voreinstellungen hinsichtlich der Modellierungskonzepte des Zielsystems vorgenommen. Die Entscheidung für das konkrete Zielsystem kann später fallen.

Die Entwicklung des logischen Schemas erfolgt in zwei Schritten (→ Bild 2.2):

1. Im ersten Schritt kommt es zur **Transformation** des konzeptuellen Schemas auf ein (initiales) logisches Schema. Wird hier vom ERM auf das relationale Datenmodell abgebildet, kann dieser Schritt weitestgehend automatisiert werden, da ein etabliertes System von **Transformationsregeln** (→ 3.3) existiert.
2. Der zweite Schritt dient der Verbesserung bzw. **Nachbearbeitung des logischen Schemas**. Ein wichtiges Optimierungskriterium ist die Minimierung der Redundanz. Weitere Kriterien, wie schneller Zugriff auf die Daten, können teilweise konkurrieren. Liegt das logische Schema im relationalen Datenmodell vor, erfolgt in diesem Schritt die **Normalisierung** (→ 3.4). Mit Hilfe der Normalisierung können relationale Datenstrukturen so gestaltet werden, dass Einfüge-, Änderungs- und Löschoperationen auf den Datenbeständen ohne Probleme (*Anomalien*) ablaufen. Dazu werden Redundanzen durch Aufteilung von Relationen schrittweise reduziert.

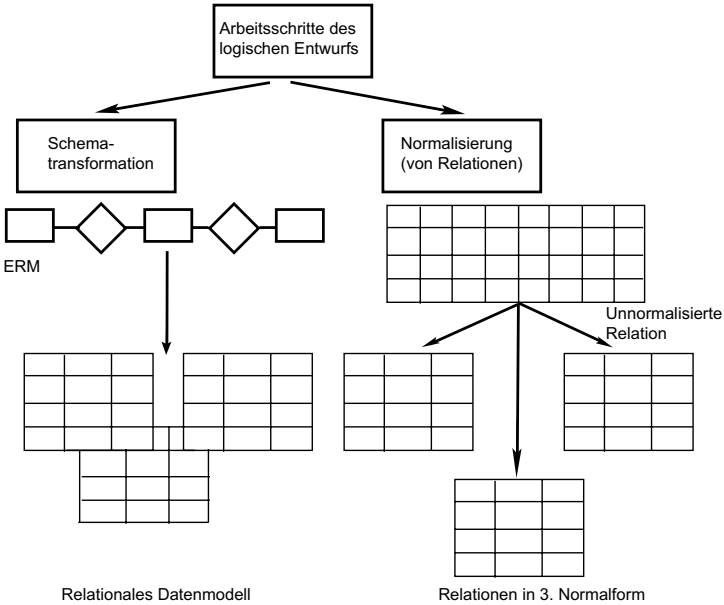


Bild 2.2 Arbeitsschritte des logischen Entwurfs

2.1.5 Datendefinition

In der Datendefinitionsphase werden die entwickelten logischen und externen Schemata mit Hilfe der Datenbanksprache des DBMS beschrieben. Für die **Beschreibung des logischen Schemas** steht die DDL, für die **Beschreibung der Sichten** die Sichtendefinitionssprache (View Definition Language, VDL) als Teil der DDL zur Verfügung.

Mit Hilfe der DDL werden beim relationalen Datenmodell Relationen, Attribute und Wertebereiche definiert. DDL und VDL sind Bestandteile der Sprache **SQL** (→ 4). Obwohl SQL standardisiert ist, existieren Dialekte, die in dieser Phase eingesetzt werden. Die Datendefinitionen sind also *vom DBMS abhängig*.

Neben den Datenstrukturen werden die Integritätsbedingungen mit den Mitteln der Datenbanksprache beschrieben, wie beispielsweise die Primär- und Fremdschlüsselbeziehungen beim relationalen Datenmodell. Ergebnis dieser Phase sind die DB-Schemata für das konkrete DBMS.

2.1.6 Physischer Entwurf

Gegenstand des physischen Entwurfs ist die **Entwicklung des internen Schemas** mit Hilfe der Speicherstruktursprache (Storage Structure Language, SSL) des DBMS. Dazu gehört die Definition der geeigneten Speicherform und möglicher Zugriffspfade für die Daten, z. B. der effektive Zugriff auf die Werte häufig genutzter Attribute über Indexdateien.

Ziel des physischen Entwurfs ist es, eine möglichst hohe Leistungsfähigkeit des DBS zu gewährleisten. Dazu gehören geringe Zugriffszeiten, hohe Durchsatzraten von Transaktionen und die optimale Ausnutzung des Speicherplatzes. In gewisser Weise erfolgt in dieser Phase eine **Optimierung** der DB-Definition (→ 9).

2.1.7 Implementierung und Wartung

Der Entwurfsprozess schließt mit der tatsächlichen **Installation des DBS** in der Umgebung ab, in der der Dauerbetrieb realisiert wird. Hierzu gehört auch die Erfassung oder Generierung der Datenbestände mit Hilfe der Datenmanipulationssprache (Data Manipulation Language, DML) des DBMS. Bei Übernahme aus anderen (DB)-Systemen kommen weitere Migrationsaufgaben zum DB-Entwurf hinzu. Hierzu gehört u. a. der Abgleich der Datenstrukturen sowie die Entwicklung von Konvertierungsprogrammen.

Mit der Inbetriebnahme der DB beginnt die **Wartung**. Das DBS unterliegt einer fortlaufenden Anpassung an neue Anforderungen, wie u. a. strukturelle Veränderungen, Optimierung der physischen Realisierung der DB, Anpassung an neue Systemplattformen, Portierung auf neue DBMS.

2.2 Grundlagen des Entity-Relationship-Modells (ERM)

Das ERM gehört zu den semantischen Datenmodellen. Als abstraktes Datenmodell wird es zum Entwurf des konzeptuellen Schemas sowie zur Modellierung von Benutzersichten in der Phase des konzeptionellen Entwurfs eingesetzt. Es ermöglicht eine **erste formale Beschreibung** des Weltausschnitts, noch unabhängig von Datenmodell und DBMS der Realisierung der DB-Anwendung. Als allgemeiner Beschreibungsformalismus für Datenstrukturen jedweder Anwendung ist es der De-facto-Standard der Informationsmodellierung.

Entwickler des ERM ist Peter P. Chen. Er stellte die grundlegenden Bestandteile des ERM im Jahre 1976 vor /2.2/. Diese ersten Ideen von Chen werden

als das klassische ERM bezeichnet. Teilmengen der in den Folgejahren publizierten Erweiterungen zum ERM werden unter der Bezeichnung EERM (Extended ERM) zusammengefasst /2.1/, /2.13/, /2.14/, /2.15/.

2.2.1 Semantische Datenmodelle

Semantische Datenmodelle wurden in den 70er Jahren entwickelt und gelten als stark vom relationalen Datenmodell (→ 3) beeinflusst. Sie berücksichtigen stärker semantische Aspekte.

Semantische Datenmodelle beschreiben einen Weltausschnitt als Menge von Gegenständen (Objekten), zwischen denen wohldefinierte Beziehungen existieren und die durch Eigenschaften charakterisiert werden.

Ausgewählte Vertreter der semantischen Datenmodelle sind:
- das Relational Model/Tasmania (RM/T) von Codd,
- das Semantic Data Model (SDM) von Hammer/McLeod,
- das Functional Data Model (FDM) von Kerschberg,
- das Entity-Relationship-Modell (ERM) von Chen.

Durchgesetzt hat sich das ERM. Gründe sind u. a.
- die Beschränkung auf einfache und wenige Basiskonzepte,
- die Orientierung auf eine gute grafische Repräsentation,
- keine Beschränkung bez. des zu modellierenden Weltausschnitts, d. h. eine umfassende Anwendbarkeit des Datenmodells,
- die Unabhängigkeit von konkreten Datenmodellen der DBMS.

2.2.2 Grundkonzepte des klassischen ERM

Die grundlegenden *Modellelemente* des klassischen ERM sind das Entity als zu modellierende Informationseinheit, der Entitytyp, die Relationship oder Beziehung, der Relationship- oder Beziehungstyp, das Attribut, der Wert, der Wertebereich sowie die Rolle.

Die grafische Repräsentation des ERM ist das ER-Diagramm. Ein einfaches Beispiel für eine Modellierung im ERM zeigt Bild 2.3. Die Entitytypen Artikel und Lieferant werden über den Beziehungstyp Lieferung assoziiert (→ 2.2.2.2).

Bild 2.3 Beispiel für ein ER-Diagramm: Weltausschnitt „Artikellieferung“

2.2.2.1 Konzepte auf der Instanzebene

Ein **Entity** ist ein abgrenzbarer, selbstständiger Gegenstand, ein Begriff, ein Ding, eine Person oder auch ein Ereignis des zu modellierenden Weltausschnitts.

Einzelne Entities werden im ER-Diagramm nicht dargestellt, sondern stets Mengen von gleichartigen Entities (*Entitymengen, entity sets*), die über den Entitytyp definiert werden.

❑ *Beispiel:* Entities sind in Bezug auf das Beispiel Artikellieferung (→ Bild 2.3) der Lieferant „Koch & Krug“, der Artikel „SCSI-Kabel“ etc.

Zwischen Entities werden Beziehungen oder Relationships definiert.

Eine **Beziehung** ist die logische Verknüpfung von zwei oder mehreren Entities.

❑ *Beispiel:* Beziehungen in Bezug auf das Beispiel sind die logische Verknüpfung „liefert“ zwischen dem konkreten Lieferanten und Artikel. Der Lieferant „Koch & Krug“ liefert den Artikel „SCSI-Kabel“.

Aus den gleichen Gründen, die für Entities gelten, werden auch Beziehungen im ER-Diagramm nicht dargestellt, sondern stets Beziehungsmengen, die über den Beziehungstyp definiert werden. Eine *Beziehungsmenge (relationship set)* ist die Menge aller Beziehungen zwischen den Entities von zwei oder mehreren Entitymengen. In Bild 2.4 wird eine eigene Notation verwendet, um beispielhaft einige Entities und Beziehungen des Weltausschnitts zu visualisieren.

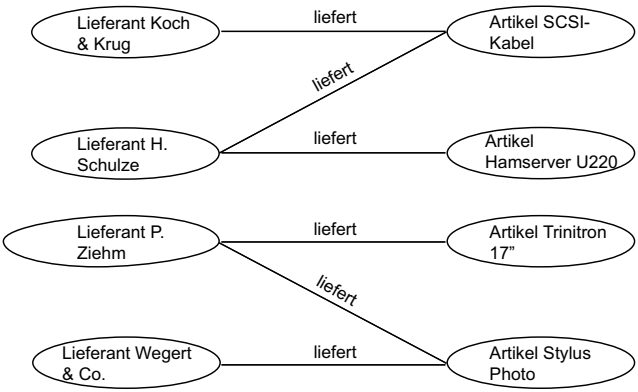


Bild 2.4 Entities und Beziehungen eines Weltausschnitts „Artikellieferung“

Entities und Beziehungen werden über **Werte** (values) näher beschrieben. Wertemengen gehören einem **Wertebereich** (domain) an.

Der Wertebereich wird in der Regel über einen Standard-Datentyp wie z. B. CHAR() für Zeichenketten, INTEGER für ganze Zahlen, DATE für Datumswerte beschrieben.

❑ *Beispiel:* In Bild 2.4 werden die Entities Lieferant über die Werte des Namens repräsentiert, die Entities Artikel über die Werte der Bezeichnung.

2.2.2.2 Konzepte auf der Typebene

Mengen gleichartiger Entities werden über einen **Entitytyp** definiert und im ER-Diagramm als *Rechteck* dargestellt. Kriterium der Gleichartigkeit von Entities sind deren Eigenschaften.

❑ *Beispiel:* Alle Lieferantenobjekte können über den Entitytyp Lieferant definiert werden, wenn sie über gemeinsame Eigenschaften wie Nummer, Name, Ort und Bankverbindung verfügen.

Eine DB enthält endlich viele Entitytypen. Die Entitymengen der Entitytypen müssen nicht disjunkt zueinander sein. Beispielsweise kann eine DB Informationen zu Kunden und Lieferanten beinhalten. Die Entitymenge des Typs Kunde kann Überschneidungen zur Entitymenge des Typs Lieferant aufweisen: Es gibt Kunden, die gleichzeitig Lieferant sind.

Entitytypen werden aus einer textlichen Beschreibung des Weltausschnitts und dabei häufig aus *Substantiven* abgeleitet.

Gleichartige Beziehungen werden über den **Beziehungstyp** definiert und im ER-Diagramm als *Raute* dargestellt. Kriterium für die Gleichartigkeit der Beziehungen sind gleichartige Eigenschaften sowie die Verknüpfung gleichartiger Entities.

❑ *Beispiel:* Die Beziehungen „liefert“ zwischen den Entities der Entitytypen Lieferant und Artikel können über den Beziehungstyp „Lieferung“ definiert werden.

Attribute werden Entitytypen und Beziehungstypen als **Eigenschaften** zugeordnet. Sie erlauben deren Charakterisierung, Identifizierung und Klassifizierung.

❑ *Beispiel:* Attribute des Entitytyps Lieferant sind u. a. Name, Ort und Bankverbindung, Attribute des Beziehungstyps Lieferung sind Menge und Preis.

Attribute werden im ER-Diagramm den Entity- bzw. Beziehungstypen über *Ellipsen* zugeordnet (→ Bild 2.5).

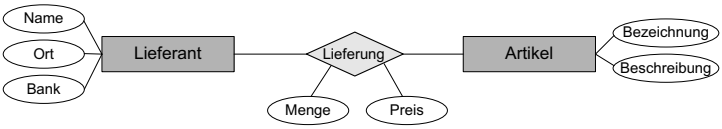


Bild 2.5 ER-Diagramm „Artikellieferung“ mit Attributen

Oftmals werden sie aus Vereinfachungsgründen aus dem Diagramm ausgeblendet und sind textuell den Entity- und Beziehungstypen zugeteilt. Die textuelle Notation ist /1.2/, /1.4/:

$$\textit{Entitytyp}\ (\textit{Attribut}_1\textit{: Datentyp}_1, \ldots, \textit{Attribut}_n\textit{: Datentyp}_n).$$

❑ *Beispiel:* Eine textuelle Notation des Entitytyps Lieferant mit Attributen ist: Lieferant (Name: char(25), Ort: char(50), Bank: char(50)).

2.2.2.3 Rollenmodellierung

Jedes Entity eines Entitytyps erfüllt eine *Funktion* in einer Beziehung. Diese Funktion wird als **Rolle** bezeichnet und über eine Beschriftung der

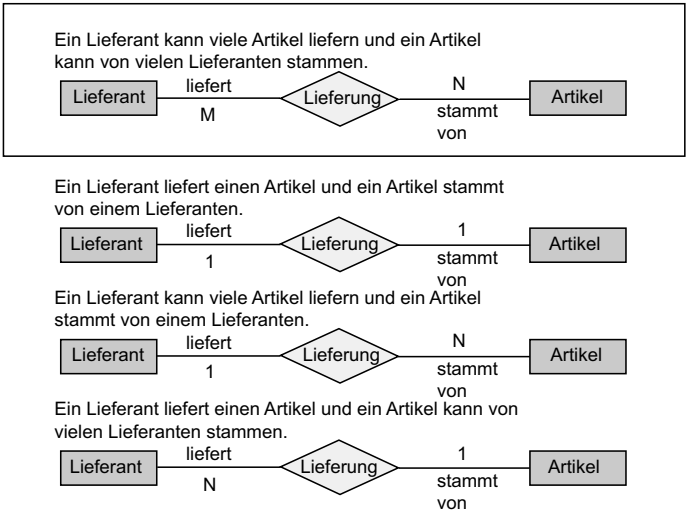


Bild 2.6 Kardinalitäten des Beziehungstyps Lieferung

Kantenenden des Beziehungstyps im ER-Diagramm dargestellt /2.4/, /2.8/, /2.16/.

❑ *Beispiel:* Das Lieferantenobjekt hat in der Beziehung „Lieferung“ zu den Artikelobjekten die Rolle „liefert“, das Artikelobjekt die Rolle „stammt von“ (→ Bild 2.6).

2.2.3 Kardinalitäten von Beziehungstypen

2.2.3.1 Kardinalitäten im klassischen ERM

Beziehungstypen werden auch *quantitativ* beschrieben. Dafür existieren grundsätzlich *zwei Ansätze*, die mit unterschiedlicher Genauigkeit die Quantifizierung des Beziehungstyps möglich machen. Im klassischen ERM wird die mengenmäßige Beschreibung der Beziehungstypen mit der $(1, M, N)$-Notation vorgenommen und die Angabe als *Kardinalität* bezeichnet /2.4/, /2.11/.

> Die **(1, *M*, *N*)-Notation** liefert folgende Aussage: Wie viele Entities eines Entitytyps *können* mit wie vielen Entities eines anderen Entitytyps *maximal* in Beziehung stehen?

Bei der Beschreibung eines konkreten Beziehungstyps wird jede Beziehung in zwei gerichtete Teilbeziehungen zerlegt und getrennt betrachtet. Hierbei können die Rollen eine gute Unterstützung bieten.

❑ *Beispiel:* Für die Betrachtung der Beziehung „Lieferung“ zwischen Lieferant und Artikel resultieren folgende Überlegungen: Ein Lieferantenobjekt steht mit wie vielen Artikelobjekten maximal in Beziehung bzw. wie viele Artikel liefert ein Lieferant? Ein Artikelobjekt steht mit wie vielen Lieferantenobjekten maximal in Beziehung bzw. ein Artikel stammt von wie vielen Lieferanten?

Als mögliche Maxima kommen bei der $(1, M, N)$-Notation in Betracht: 1 und viele (N bzw. M). Daraus ergeben sich als Möglichkeiten der Quantifizierung von Beziehungstypen: $1{:}1$, $1{:}N$, $N{:}1$, $M{:}N$.

Angewendet auf das Beispiel des Beziehungstyps Lieferung sind die Kardinalitäten wie in Bild 2.6 zu interpretieren. Sinnvoll ist hier insbesondere die umrahmte $M{:}N$-Beziehung.

2.2.3.2 Intervalle und Komplexitätsgrade

Zur Quantifizierung wird aktuell in der Regel mit der **(min,max)-Notation** gearbeitet. Diese stellt eine Erweiterung zur $(1, M, N)$-Notation dar:

- Der Wertebereich wird durch die Einbeziehung des Minimums 0 erweitert.

- Kardinalitätsangaben werden als Intervalle über einen minimalen und maximalen Wert realisiert. Die möglichen Wertebereiche sind {0,1} beim Minimum und {1,*} beim Maximum. Das Symbol '*' hat die Bedeutung von „unbegrenzt".
- Die Kardinalitätsangaben fungieren als Teilnahmezahlen eines Entities an einer Beziehung und beschreiben den *Komplexitätsgrad*. Die Angabe des Maximums erfolgt dadurch seitenverkehrt zur $(1, M, N)$-Notation.

Bei Verwendung von **Komplexitätsgraden** wird die Beziehung gedanklich geteilt und folgende Aussage je Hälfte getroffen: Ein Entity eines Entitytyps *muss minimal* und *kann maximal* an wie vielen Beziehungen beteiligt sein?

❑ *Beispiel:* Angewendet auf das Beispiel des Beziehungstyps Lieferung wird nun folgende Aussage möglich: Ein Lieferant muss minimal einen Artikel liefern: Minimum 1. Er ist als Lieferant in der DB nicht zulässig, solange er nicht mindestens einen Artikel geliefert hat. Maximal wird ein Lieferant viele Artikel liefern: Maximum *. Ein Artikel muss nicht unbedingt geliefert werden, da er auch eigenproduziert sein kann: Minimum 0. Ein Artikel kann von vielen Lieferanten stammen: Maximum * (→ Bild 2.7).

► *Hinweis:* Im praktischen Einsatz werden für Intervallangaben oft verkürzte Schreibweisen genutzt: $(0,1) = c$, $(1,1) = 1$, $(1,*) = n$, $(0,*) = cn$. Auch grafische Varianten sind in Nutzung: Krähenfuß-, Bachman- sowie toolspezifische Notationen.

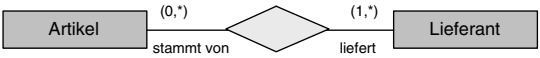


Bild 2.7 Kardinalitäten des Beziehungstyps Lieferung mit (min, max)-Notation

2.2.4 Existenzabhängigkeit vs. Optionalität

Durch die Verwendung der (min, max)-Notation können **Existenzabhängigkeiten** zwischen Entities beschrieben werden. Ein Entity ist von einem anderen existenzabhängig, wenn es nur bestehen kann, solange das andere Entity existiert.

❑ *Beispiel:* Ein Objekt Lieferant ist existenzabhängig von einem Objekt Artikel. Ein Lieferant muss mindestens einen Artikel liefern (→ Bild 2.7).

Die Existenzabhängigkeit wird über das *Minimum 1* der (min, max)-Notation ausgedrückt: