

HANSER



Leseprobe

Taschenbuch Datenbanken

Herausgegeben von Thomas Kudraß

ISBN (Buch): 978-3-446-43508-7

ISBN (E-Book): 978-3-446-44026-5

Weitere Informationen oder Bestellungen unter

<http://www.hanser-fachbuch.de/978-3-446-43508-7>

sowie im Buchhandel.

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 Begehrtschungel 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 Rezessenten 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	<i>Erweiterungen des ERM</i>	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	<i>Konzepte und Grundbegriffe des relationalen Datenmodells</i>	67
3.1.1	Relationen, Tupel, Attribute und Wertebereiche	67
3.1.2	Eigenschaften von Relationen	69
3.2	<i>Integritätsbedingungen</i>	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	<i>Abbildung des EERM auf das relationale Datenmodell</i>	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	<i>Optimierung von Relationen</i>	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	<i>Operationen der Relationenalgebra</i>	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	<i>Grundkonzepte</i>	99
4.2	<i>Historie</i>	101
4.3	<i>Spezifikationsdokumente</i>	102
4.4	<i>Beispieldatenbank</i>	103
4.5	<i>Datenbankanfragen</i>	103
4.5.1	Einführende Beispieldatenanfragen	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	<i>Datenmanipulation</i>	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	<i>Datendefinition</i>	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	<i>Transaktionssteuerung</i>	138
4.8.1	Transaktionsanweisungen	139
4.8.2	Isolationsebenen	140
4.8.3	Überprüfung von Integritätsbedingungen	141

4.9	<i>Zugriffskontrolle</i>	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	<i>Grundlagen der Datenbank-Anwendungsprogrammierung</i>	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	<i>Embedded SQL</i>	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	<i>Aufrufchnittstellen</i>	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	<i>Relationale Datenbankprogrammiersprachen</i>	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	<i>Objektrelationale Abbildung</i>	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	<i>Grundlagen des Web</i>	189
6.2	<i>Eigenschaften von Webanwendungen</i>	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	<i>Datenbankanbindung im Web</i>	200
6.3.1	Architekturen	201
6.3.2	Programmierung	202
6.3.3	Überblick	204
6.4	<i>Datenbankanbindung über Datenexport</i>	206
6.5	<i>Clientseitige Datenbankanbindung</i>	206
6.5.1	Skripteinbettung	207
6.5.2	Programmeinbettung	209
6.5.3	Weitere Techniken	212
6.6	<i>Serverseitige Datenbankanbindung</i>	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	<i>Architektur von DBMS</i>	227
7.1.1	Schichtenmodell	227
7.1.2	Prozessarchitektur	229
7.2	<i>Pufferverwaltung</i>	230
7.2.1	Notwendigkeit und Aufgabe	231
7.2.2	Speicherzuteilung	232
7.2.3	Seitenersetzung	233
7.3	<i>Speicher- und Zugriffssystem</i>	234
7.3.1	Aufgabe	235
7.3.2	Seiten und Sätze	235
7.3.3	Adressierung von Sätzen	237
7.4	<i>Anfrageprozessor</i>	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	<i>Transaktionsverwaltung</i>	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	<i>Recovery</i>	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	<i>Datenbanktechniken für moderne Hardware-Architekturen</i>	266
8	Dateiorganisation und Indexe	269
8.1	<i>Organisation von Dateien</i>	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	<i>Zugriffsstrukturen</i>	271
8.2.1	Grundlagen von Zugriffsverfahren	271
8.2.2	Eigenschaften von Indexen	272
8.3	<i>Baumbasierte Verfahren</i>	274
8.3.1	ISAM-Bäume	274
8.3.2	Balancierte Mehrwegbäume	275
8.3.3	Digitale Bäume	277
8.4	<i>Hash-Verfahren</i>	278
8.4.1	Prinzip des Hashing	278
8.4.2	Erweiterbares Hashing	279
8.4.3	Weitere Hash-Verfahren	280
8.5	<i>Mehrdimensionale Zugriffsverfahren</i>	281
8.5.1	Mehrdimensionale Baumverfahren	281
8.5.2	Grid-File	281
8.6	<i>Clustering und Partitionierung</i>	283
8.6.1	Clustering	283
8.6.2	Partitionierung	283
8.7	<i>Umsetzung in SQL-Systemen</i>	284
8.7.1	Definition von Tabellen	285
8.7.2	Definition von Indexen	285
9	Optimierung von Datenbanken und Leistungsbewertung	287
9.1	<i>Motivation der Datenbankoptimierung</i>	287
9.1.1	Kosten von Datenbankanfragen	288
9.1.2	Optimierungspotenzial	289
9.1.3	Zielbestimmung der Datenbankoptimierung	290
9.2	<i>Phasen der Datenbankoptimierung</i>	292

9.3	<i>Phase 1.1 – Optimierung des Datenbankschemas</i>	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	<i>Phase 1.2 – Anwendungsoptimierung</i>	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	<i>Phase 2 – Hauptspeicheroptimierung</i>	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	<i>Phase 3 – Optimierung der Sekundärspeicherzugriffe</i>	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	<i>Leistungsbewertung</i>	318
9.7.1	Transaction Processing Performance Council	319
9.7.2	Vergleichbarkeit der Benchmark-Ergebnisse	320
10	Objektrelationale Datenbanken	322
10.1	<i>Objektorientierte Konzepte</i>	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 <i>Objektorientierung in Datenbanken</i>	326
10.3 <i>Objektrelationale Standard-SQL-Konzepte</i>	327
10.3.1 Typkonstrukturen	328
10.3.2 Distinct-Typen	330
10.3.3 Strukturierte Typen	330
10.3.4 Methoden	333
10.3.5 Benutzerdefinierte Konstrukturen	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 <i>Objektrelationale Anfragen</i>	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 <i>Überblick über XML</i>	345
11.1.1 Der XML-Standard und verwandte Standards	345
11.1.2 XML Schema	347
11.2 <i>Anfragesprachen für XML</i>	350
11.2.1 Pfadausdrücke (XPath und XQuery)	351
11.2.2 XQuery	352
11.3 <i>XML und relationale Datenbanksysteme</i>	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 <i>Reine XML-Datenbanksysteme</i>	366
12 NoSQL-Datenbanksysteme	368
12.1 <i>Motivation und Grundbegriffe</i>	368
12.2 <i>Klassifikation</i>	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	<i>Datenmodellierung</i>	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	<i>Anwendungsentwicklung mit NoSQL-Datenbanksystemen</i>	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	<i>Skalierbarkeit, Verfügbarkeit und Konsistenz</i>	390
12.6	<i>Auswahl eines geeigneten Datenbanksystems</i>	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	<i>Überblick, Grundbegriffe, Abgrenzung</i>	394
13.1.1	Verteilte vs. parallele Datenbanksysteme	395
13.1.2	Verteilte vs. föderierte Datenbanksysteme	397
13.2	<i>Schemaarchitektur und Entwurf verteilter Datenbanksysteme</i>	400
13.3	<i>Fragmentierung</i>	401
13.3.1	Horizontale Fragmentierung	401
13.3.2	Vertikale Fragmentierung	403
13.3.3	Kombinierte Fragmentierung	403
13.4	<i>Verteilungstransparenz</i>	404
13.4.1	Vorteile verteilter Datenbanksysteme	404
13.4.2	Transparenzeigenschaften verteilter Datenbanksysteme	404
13.5	<i>Verteilte Anfrageverarbeitung</i>	405
13.5.1	Datenlokalisierung	407
13.5.2	Globale Optimierung der Join-Auswertung	409
13.6	<i>Transaktionsverwaltung in verteilten Datenbanksystemen</i>	412
13.6.1	Koordination	412
13.6.2	Synchronisation	414
13.6.3	Deadlock-Behandlung	415
13.6.4	Synchronisation bei Replikation	416

13.7 <i>Föderierte Datenbanksysteme</i>	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 <i>Cloud-Datenbanken</i>	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 <i>Trends</i>	428
14 Data Warehouse	430
14.1 <i>Architektur</i>	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 <i>Multidimensionale Datenmodelle</i>	440
14.2.1 Statische Aspekte	440
14.2.2 Dynamische Aspekte	444
14.3 <i>Speicherung und Schemagestaltung</i>	446
14.3.1 Relationale Speicherung	446
14.3.2 Multidimensionale Speicherung	447
14.3.3 Spaltenorientierte Speicherung	448
14.4 <i>Erweiterung relationaler Datenbanken</i>	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 <i>KDD-Prozess</i>	461
15.2 <i>Clustering</i>	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 Dichtebaserte Methoden	468
15.3 <i>Assoziationsanalyse</i>	468
15.3.1 Definition und Beispiel	468
15.3.2 Anforderungen und Probleme	468
15.3.3 Verfahren	469
15.4 <i>Klassifikation</i>	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 <i>Anomalieentdeckung</i>	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 <i>Einführung</i>	478
16.2 <i>Mediendaten</i>	482
16.3 <i>Suche nach Mediendaten</i>	485
16.3.1 Textsuche	485
16.3.2 Bildsuche	487
16.3.3 Audiosuche	488
16.3.4 Videosuche	489
16.4 <i>Mediendatentypen</i>	490
16.5 <i>Einbettung in Datenbanksysteme</i>	494
16.5.1 Schemastrukturen	494
16.5.2 Anfrageformulierung	496
16.6 <i>Einsatz</i>	497
17 Geodatenbanken	499
17.1 <i>Geodaten</i>	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

4 Die Datenbanksprache SQL

Can Türker

SQL (Structured Query Language) ist die Datenbanksprache, die von allen bedeutenden (relationalen) Datenbanksystemen unterstützt wird. Dieses Kapitel stellt die zentralen Sprachkonstrukte von Standard-SQL vor. Spezialitäten der herstellerspezifischen SQL-Dialekte werden hier nicht betrachtet. Die objektrelationalen Erweiterungen von SQL sind separat in Kapitel 10 dargestellt.

4.1 Grundkonzepte

SQL beruht auf dem Relationenmodell und der Relationenalgebra (→ 3.5). Das zentrale Konzept von SQL sind jedoch **Tabellen** (*table*) (→ 4.7.3) anstelle von Relationen. Im Gegensatz zu einer Relation im Relationenmodell entspricht eine Tabelle in SQL einer *Multimenge* von Tupeln. Die echte Mengeneigenschaft, d. h. der Ausschluss von Tupelduplikaten, muss (falls gewünscht) durch Definition von Integritätsbedingungen erzwungen werden. In einer SQL-Datenbank bieten Tabellen die einzige Datenstruktur zur Speicherung von Daten. Eine Tabelle besteht aus **Spalten** (*columns*) und **Zeilen** (*rows*). Die Spalten haben einen innerhalb der Tabelle eindeutigen Namen. Die Felder einer Tabelle enthalten Werte aus Wertebereichen, die durch Datentypen festgelegt werden.

SQL bietet **Basisdatentypen** (→ 4.7.1) für numerische, alphanumerische, binäre, Datum-Zeit- und XML-Werte. Jeder Datentyp definiert einen Wertebereich mit zugehörigen Operationen. Das Konzept der **Domänen** (→ 4.7.4) ermöglicht die Definition von benutzerdefinierten Wertebereichen durch Einschränkung existierender Datentypen. Ferner gibt es in SQL die Möglichkeit, **benutzerdefinierte Datentypen** (*user-defined types*, UDT) (→ 10.3.3) zu erzeugen.

Integritätsbedingungen (*constraints*) (→ 4.7.5) erlauben die Einschränkung von Wertebereichen auf semantisch gültige Werte. SQL unterscheidet zwischen *Spalten-* und *Tabellenbedingungen*. Während Erstere eine Bedingung über genau einer Spalte formulieren, können Letztere spaltenübergreifende Bedingungen realisieren.

Eine **Anfrage** (*query*) (→ 4.5) in SQL extrahiert Daten aus der Datenbank und gibt diese in Form einer Tabelle zurück. Etwas genauer formuliert, eine Anfrage nimmt eine oder mehrere Tabellen entgegen und transformiert diese in eine Ergebnistabelle.

Das Konzept einer **Sicht** (*view*) (→ 4.7.6) ermöglicht die Speicherung einer benannten und damit wiederverwendbaren Anfrage. Eine Sicht stellt somit eine Tabelle dar, deren Inhalt nicht materialisiert ist, sondern immer wieder neu zum Zeitpunkt der Anfrage berechnet wird. Sichten bieten eine Reihe von Vorteilen. Sie ermöglichen die Umsetzung einer feingranularen Zugriffskontrolle, unterstützen die *logische Datenunabhängigkeit* (→ 1.4.3) und vereinfachen die Anfrageformulierung.

Mit dem Konzept der **benutzerdefinierten Routinen** (*user-defined routines*, UDR) (→ 4.7.7) bietet SQL die Möglichkeit, Anwendungsfunktionalität in der Datenbank aufrufbar abzulegen. Neben der Wiederverwendbarkeit und damit Vereinfachung der Anwendungsprogrammierung bieten benutzerdefinierte Routinen den Vorteil, dass durch ihre Ausführung auf dem Server die Last des Netzwerkes erheblich reduziert werden kann. Dies trifft vor allem bei Routinen zu, die viele Daten zur Berechnung benötigen, jedoch nur wenige als Ergebnis zurückliefern.

Mit dem Konzept eines **Triggers** (→ 4.7.8) gibt es darüber hinaus die Möglichkeit, automatisch auf vordefinierte Ereignisse in der Datenbank durch das Ausführen benutzerdefinierter Anweisungen zu reagieren. Trigger haben den Vorteil, Anwendungsfunktionalität zentral in der Datenbank zu verwalten und damit die Wartbarkeit des gesamten Anwendungssystems zu erhöhen.

Sequenzgeneratoren (→ 4.7.9) sind Konstrukte, mit denen Werte sequenziell generiert werden können. Dieses Konzept ist besonders nützlich für das Erzeugen von Schlüsselwerten zur eindeutigen Identifikation von Zeilen in einer Tabelle.

Zusammengefasst besteht eine SQL-Datenbank aus einer Reihe von unterschiedlichen Objekten. Nach Standard-SQL werden diese Objekte wie folgt „hierarchisch“ organisiert:

- Kataloge
 - Schemata
 - * Tabellen, Domänen, Integritätsbedingungen, Sichten, Routinen, Trigger, Sequenzgeneratoren, ...

Eine SQL-Datenbank besteht aus einem oder mehreren Katalogen. Jeder **Katalog** umfasst ein oder mehrere Schemata. Ein **Schema** (→ 4.7.2) bildet eine Einheit bestehend aus „zusammengehörenden“ Schemaobjekten. Der eindeutige Zugriff auf ein Schemaobjekt ist mit Hilfe der folgenden Pfadnotation möglich:

`Katalogname.SchemaName.Schemaobjektname`

Für den Zugriff auf ein Schemaobjekt benötigt der Benutzer ein **Zugriffsrecht** (→ 4.9.1). Ein Zugriffsrecht kann vom Besitzer des Schemaobjektes vergeben und wieder entzogen werden.

SQL stellt Sprachkonstrukte zum Arbeiten mit den oben genannten Konzepten bereit, die verschiedenen Teilsprachen von SQL zugeordnet werden:

- Die **Datendefinitionssprache** (Data Definition Language; DDL) dient zum Erzeugen, Ändern und Löschen von Schemata und Schemaobjekten (→ 4.7).
- Die **Datenmanipulationssprache** (Data Manipulation Language; DML) ermöglicht das Erzeugen, Ändern und Löschen von Tabelleninhalten (→ 4.6).
- Die **Datenanfragesprache** (Data Query Language; DQL) dient zur Abfrage der Tabelleninhalte (→ 4.5).
- Die **Datenüberwachungssprache** (Data Control Language; DCL) wird zur Vergabe und Zurücknahme von Zugriffsrechten eingesetzt (→ 4.9).

Für das Erzeugen und Löschen von Katalogen sieht der SQL-Standard keine Sprachkonstrukte vor. Dementsprechend sind die Lösungen der verschiedenen kommerziellen Produkte in diesem Fall sehr unterschiedlich. Das Gleiche gilt für die Steuerung der internen Organisation einer SQL-Datenbank. Hier bieten herstellerspezifische SQL-Dialekte Erweiterungen an, etwa die Unterstützung von Indexen zum Beschleunigen des Datenzugriffs. Der SQL-Standard stellt auch keine Anweisungen für die Benutzerverwaltung bereit.

4.2 Historie

Historisch gesehen sind folgende Daten im Zusammenhang mit der Entwicklung von SQL von Bedeutung:

- 1970 IBM-Forscher Edgar F. Codd /4.12/ stellt das Relationenmodell und die Relationenalgebra vor.
- 1974 Darauf aufbauend entwerfen die IBM-Wissenschaftler Don D. Chamberlin und Raymond F. Boyce /4.11/ mit SEQUEL (Structured English QUERy Language) den „SQL-Urvater“.
- 1976 IBM stellt mit System R /4.10/ einen auf SEQUEL basierenden Prototypen eines relationalen Datenbanksystems vor.
- 1979 Die Firma Relational Software (später in Oracle Corporation umbenannt) bringt mit Oracle das erste kommerzielle SQL-Datenbanksystem auf den Markt.
- 1981 IBM liefert mit SQL/DS ein Konkurrenzprodukt aus.
- 1982 ANSI (American National Standards Institute) beginnt mit der Normierung von SQL.

- 1983 IBM bringt mit DB2 ein weiteres relationales Datenbanksystem auf den Markt.
- 1985 Informix und Ingres stellen ihre Datenbanksysteme auf SQL um.
- 1986 SQL wird ANSI-Norm.
- 1987 SQL wird von der ISO (International Standards Organization) ratifiziert.
- 1989 SQL-89 ist eine revidierte und ergänzte Fassung der ISO-Norm.
- 1992 SQL-92 (auch SQL2 genannt) ist erste gemeinsame ISO/ANSI-Norm.
- 1997 Informix, Oracle und DB2 liefern ihre SQL-Datenbanksysteme mit ersten objektrelationalen Erweiterungen aus.
- 1999 SQL:1999 (auch als SQL-3 bekannt) löst SQL-92 mit vielen neuen Features ab und stellt die erste Norm für objektrelationales SQL dar.
- 2003 SQL:2003 löst SQL:1999 mit relativ geringfügigen Erweiterungen ab, die allerdings das Zusammenspiel von XML und SQL festlegen.
- 2008 SQL:2008 kommt mit wenigen Neuerungen, unter anderem Instead-Of-Trigger und XQuery-Erweiterungen.
- 2011 SQL:2011 führt temporale Features in SQL ein.

4.3 Spezifikationsdokumente

Der aktuelle Standard SQL:2011 umfasst folgende Dokumente:

- *Part 1: Framework (SQL/Framework)* /4.1/ gibt einen Überblick über den kompletten Standard und stellt insbesondere die verschiedenen Sprachebenen vor.
- *Part 2: Foundation (SQL/Foundation)* /4.2/ beschreibt das Datenmodell von SQL und die Sprachkonstrukte zur Datendefinition, -abfrage und -manipulation. Die Einbettung von SQL-Anweisungen in Anwendungsprogrammen wird ebenfalls definiert.
- *Part 3: Call-Level Interfaces (SQL/CLI)* /4.3/ beinhaltet die Definition von niederen Schnittstellen für den Zugriff von Anwendungen auf Datenbanken über Funktionsaufrufe.
- *Part 4: Persistent Stored Modules (SQL/PSM)* /4.4/ normiert die prozedurale Erweiterung von SQL.
- *Part 9: Management of External Data (SQL/MED)* /4.5/ definiert Datentypen und Funktionen, um auf externe Daten zugreifen zu können.
- *Part 10: Object Language Bindings (SQL/OLB)* /4.6/ normiert die Java-Anbindung an SQL-Datenbanken.
- *Part 11: Information and Database Schemata (SQL/Schemata)* /4.7/ enthält die Spezifikation der Metatabellen.

- *Part 13: SQL Routines and Types Using the Java Programming Language (SQL/JRT)* /4.8/ spezifiziert die Registrierung von externen Routinen und Datentypen, die in Java definiert wurden.
- *Part 14: XML-related Specifications (SQL/XML)* /4.9/ normiert die Verknüpfung von SQL und XML. Er definiert den Basisdatentyp XML mit zugehörigen Funktionen und Abbildungen zwischen SQL und XML.

Da die Standardisierungsdokumente aufgrund der zum Teil unübersichtlichen Definitionen, bestehend aus vielen Klauseln und Ausnahmen, nur bedingt als „Lektüre“ zu empfehlen sind, sei hier auf Literatur verwiesen, in der Standard-SQL aufbereitet präsentiert wird: /4.16/, /4.14/, /4.15/, /4.13/.

Im Folgenden werden hauptsächlich die SQL-Konstrukte behandelt, die aus den Parts 2, 4, 13 und 14 stammen.

4.4 Beispieldatenbank

Die einzelnen SQL-Sprachkonstrukte werden, soweit es geht, am Beispiel einer einfachen kleinen SQL-Datenbank illustriert. Bild 4.1 zeigt diese Datenbank, die aus drei Tabellen besteht. Bei Bedarf werden weitere einfache Tabellen zur Demonstration herangezogen.

Artikel		Lieferant		Lieferung		
ANr	Bezeichnung	LNr	Name	ANr	LNr	Preis
101	SCSI Kabel	1	Koch	101	1	9.90
102	Hamserver	2	Schulze	101	2	7.90
103	Trinitron	3	Ziehm	102	2	69.90
104	Stylus Photo	4	Wegert	103	3	159.90
				103	4	249.90
				104	4	259.90

Bild 4.1 Beispieldatenbank

Die Datenbank verwaltet Daten über Artikel und über Lieferanten, die diese Artikel zu einem bestimmten Preis liefern. Hier liegt die Annahme zugrunde, dass die Spalten ANr bzw. LNr jeweils Schlüssel der Tabelle Artikel bzw. Lieferant darstellen, d. h. jeweils eindeutige Werte zur Identifikation der Artikel bzw. Lieferanten aufweisen.

4.5 Datenbankanfragen

In SQL stellt eine Anfrage eine Funktion dar, die eine oder mehrere Eingangstabellen als Input empfängt, Verknüpfungen und Berechnungen darauf ausführt und als Ergebnis eine Tabelle zurückliefert.

4.5.1 Einführende Beispieldatenanfragen

- *Beispiel:* Ausgabe des gesamten Inhalts der Tabelle Lieferant:

```
SELECT *
FROM Lieferant;
```

Das Symbol * ist ein Kürzel für „alle Spalten“ der Eingangstabelle. Soll die Ergebnistabelle nur bestimmte Spalten bzw. Zeilen der Ausgangstabelle enthalten, müssen entsprechende Auswahlkriterien formuliert werden. Die Auswahl der Spalten erfolgt mit der SELECT-Klausel, die Einschränkung auf bestimmte Zeilen der Tabelle geschieht mit der WHERE-Klausel.

- *Beispiel:* Kombination von Projektion und Selektion, die aus der Tabelle Lieferant die Spalte Name von allen Zeilen ausgibt, deren Spalte LNr einen Wert größer oder gleich 3 hat:

```
SELECT Name
FROM Lieferant
WHERE LNr >= 3;
```

Die Ergebnistabelle sieht in dem konkreten Beispiel wie folgt aus:

Name
Ziehm
Wegert

Diese SQL-Anfrage entspricht dem folgenden Relationenalgebraausdruck ($\rightarrow 3.5$):

$$\pi_{\text{Name}}(\sigma_{\text{LNr} \geq 3}(\text{Lieferant}))$$

Das Zusammenführen von Daten aus mehreren Tabellen ist mit Hilfe von *Verbundoperationen* möglich.

- *Beispiel:* Ein natürlicher Verbund verknüpft alle Zeilen der Tabelle Artikel mit den Zeilen der Tabelle Lieferung, wobei nur die Zeilenpaare in die Ergebnistabelle kommen, bei denen die Werte der gleichnamigen Spalten jeweils übereinstimmen:

```
SELECT *
FROM Artikel NATURAL JOIN Lieferung;
```

Die traditionelle Formulierung dieser Anfrage ohne expliziten Verbundoperator lautet wie folgt:

```
SELECT a.ANr, Bezeichnung, LNr, Preis
FROM Artikel a, Lieferung l
WHERE a.ANr = l.ANr;
```

Die Ergebnistabelle sieht in dem konkreten Beispiel so aus:

ANr	Bezeichnung	LNr	Preis
101	SCSI Kabel	1	9.90
101	SCSI Kabel	2	7.90
102	Hamserver	2	69.90
103	Trinitron	3	159.90
103	Trinitron	4	249.90
104	Stylus Photo	4	259.90

In der Relationenalgebra würde die obige SQL-Anfrage wie folgt notiert:

Artikel \bowtie Lieferung

4.5.2 Grundgerüst von Anfragen

Eine Anfrage in SQL ist ein Ausdruck, der sich wie folgt zusammensetzt:

```
SELECT Projektsliste
  FROM Tabellenreferenzliste
 [WHERE Selektionsbedingung]
 [GROUP BY Gruppierungsliste]
 [HAVING Selektionsbedingung]
 [ORDER BY Sortierliste]
```

- *Hinweis:* Sofern nicht explizit anders erwähnt, stehen eckige Klammern in Syntaxbeschreibungen für [optionale Elemente] und geschweifte Klammern für eine {obligatorische Auswahl}, wobei Alternativen durch vertikale Striche | voneinander getrennt werden.

Die einzelnen Klauseln einer SQL-Anfrage haben folgende Aufgaben:

- Die **SELECT-Klausel** beschreibt die Spalten der Ergebnistabelle. Die *Projektsliste* besteht aus Wertausdrücken, welche die jeweiligen Spaltenwerte berechnen. Ein Wertausdruck ermittelt eine Instanz, d. h. den Wert eines beliebigen SQL-Datentyps. Ein trivialer Wertausdruck ist zum Beispiel der Name einer Tabellenspalte. In diesem Fall wird der Wert der entsprechenden Spalte geliefert. Bei komplexeren Wertausdrücken, z. B. der Addition der Werte zweier Spalten, sollten die Ergebnisspalten mit „sprechenden“ Namen versehen werden. Das (Um)Benennen einer Spalte wird wie folgt notiert:

Wertausdruck [[AS] Spaltenname]

- *Hinweis:* Sollen Zeilenduplikate aus der Ergebnistabelle eliminiert werden, muss das Schlüsselwort DISTINCT der Projektsliste vorangestellt werden.

- Die **FROM-Klausel** gibt an, welche Tabellen in die Berechnung der Anfrage eingehen. Die *Tabellenreferenzliste* enthält diese Tabellen. Eine *Tabellenreferenz* steht hierbei für einen Tabellenausdruck. Dieser kann im einfachsten Fall der Name einer Tabelle sein. Die Tabellenreferenz kann aber auch aus einem Verbundausdruck bestehen, der zwei oder mehr Tabellen miteinander verknüpft (→ 4.5.6). Weitere mögliche Tabellenreferenzen sind tabellenwertige Funktionen (→ 4.5.4) und Unteranfragen (→ 4.5.7).

Mit der Angabe eines Tabellennamens wird implizit eine *Korrelationsvariable* (alias) eingeführt, die so heißt wie die Tabelle. Kommt eine Tabelle in einer Anfrage mehrfach vor, ist die Eindeutigkeit der Tabellenreferenz nicht gewährleistet. In diesem Fall müssen *explizite* Korrelationsvariablen mit eindeutigen Namen wie folgt deklariert werden:

```
Tabellenreferenzausdruck
[[AS] Korrelationsvariablenname]
[(Spaltennamensliste)]
```

Mit der optionalen Spaltennamensliste können zudem die Spalten der Korrelationsvariablen (um)benannt werden.

- Die optionale **WHERE-Klausel** spezifiziert eine Bedingung an die Zeilen der Ergebnistabelle. Eine *Selektionsbedingung* definiert einen *booleschen* Wertausdruck. Dieser Wertausdruck kann beliebig komplex sein, insbesondere Unteranfragen enthalten.
- Die optionale **GROUP-BY-Klausel** gibt eine oder mehrere so genannte Gruppierungsspalten an, nach denen die Zeilen der Tabelle aggregiert werden sollen. Die *Gruppierungsliste* spezifiziert die Merkmale, nach welchen gruppiert wird. Ein Gruppierungsmerkmal ist eine Spaltenreferenz. SQL erzeugt für jede Gruppe eine Zeile, wobei die Spalten, die nicht Teil des Gruppierungsmerkmals sind, aggregiert werden. Abgesehen von den aggregierten Spalten, müssen alle in der SELECT-Klausel aufgeführten Spalten auch in der GROUP-BY-Klausel genannt werden.
- Die optionale **HAVING-Klausel** definiert eine Bedingung, die alle aggregierten Zeilen erfüllen müssen. Dieser boolescher Wertausdruck darf sich nur auf Gruppierungsspalten und Aggregatfunktionen beziehen.
- Die optionale **ORDER-BY-Klausel** dient zum Sortieren des Anfrageergebnisses nach einem frei wählbaren Kriterium.

Die Auswertung eines Anfrageausdrucks liefert eine temporäre Tabelle, die zunächst nur im transienten Speicher existiert und ohne weitere Ergänzungen nach dem Ende der Anzeige der Daten verworfen wird. Das Ergebnis einer Anfrage kann jedoch auch in eine neue persistente Tabelle kopiert oder einer existierenden persistenten Tabelle hinzugefügt werden (→ 4.7.3, 4.6.1).

- **Hinweis:** In der Sprechweise der SQL-Norm werden die Begriffe „Anfrageausdruck“ und „Anfragespezifikation“ unterschieden. Eine *Anfragespezifikation* ist ein Anfrageausdruck, der keine ORDER-BY-Klausel enthält. Anfragespezifikationen können mit Hilfe von Mengen- und Verbundoperatoren zu einer komplexeren Spezifikation zusammengesetzt werden (→ 4.5.5, 4.5.6).

4.5.3 Anfragen mit Aggregatfunktionen

Eng verbunden mit der Gruppierung sind Aggregatfunktionen, die eine Gruppe von Spaltenwerten oder allgemeiner eine Menge von Werten auf einen Wert abbilden.

Tabelle 4.1 zeigt die in Standard-SQL fest verankerten Aggregatfunktionen. In den SQL-Dialektken finden sich zusätzliche herstellerspezifische Aggregatfunktionen, wie z. B. MEDIAN oder VARIANCE in Oracle, auf die hier nicht weiter eingegangen wird.

Tabelle 4.1 Aggregatfunktionen

Funktion	liefert
COUNT(X)	die Anzahl der Werte der Spalte X
COUNT(*)	die Anzahl der Zeilen in der Tabelle
SUM(X)	die Summe der Werte der Spalte X
AVG(X)	das arithmetische Mittel der Werte der Spalte X
MIN(X)	den kleinsten Wert der Spalte X
MAX(X)	den größten Wert der Spalte X

Alle Aggregatfunktionen können mit oder ohne Duplikateliminierung aufgerufen werden. Die Syntax hierfür sieht wie folgt aus:

Aggregatfunktion([ALL | DISTINCT] Spaltenreferenz)

Das Schlüsselwort DISTINCT schließt Duplikate aus; die Voreinstellung ALL behält die Duplikate. Allgemein gilt, dass Nullwerte in der Aggregatberechnung nicht berücksichtigt werden.

- **Beispiel:** Anfrage zum Ermitteln aller Artikel, die mindestens zweimal geliefert wurden. Für jeden dieser Artikel werden neben der Artikelnummer die Ge-

samtzahl der jeweiligen Lieferungen sowie der mit diesen Lieferungen erzielte Durchschnittspreis ausgegeben.

```
SELECT ANr AS Artikelnummer,
       COUNT(*) AS AnzahlLieferungen,
       AVG(Preis) AS Durchschnittspreis
  FROM Lieferung
 GROUP BY ANr
 HAVING COUNT(*) >= 2;
```

Diese Anfrage illustriert nicht nur die Verwendung einer Aggregatfunktion, sondern die der GROUP-BY- und HAVING-Klauseln. Angewandt auf die Beispieldatenebene (→ Bild 4.1) zeigt sich folgendes Ergebnis:

ANr	AnzahlLieferungen	Durchschnittspreis
101	2	8.90
103	2	204.90

4.5.4 Anfragen mit Tabellenfunktionen

Eine **Tabellenfunktion** ist eine benutzerdefinierte Funktion, die eine Tabelle zurückliefert. Der Aufruf einer Tabellenfunktion kann mit Hilfe des TABLE-Konstrukts als Tabellenreferenz in einer Anfrage verwendet werden.

- *Beispiel:* Anfrage, die aus dem Ergebnis einer Tabellenfunktionen ausgewählte Spalten ausgibt.

```
SELECT Name, Preis
  FROM TABLE(ArtikelUnter(10));
```

Es sei angenommen, dass die Funktion `ArtikelUnter` (→ 4.7.7.2) eine Tabelle liefert. Dort sind alle Artikel aus der Tabelle `Artikel` aufgeführt, deren Preis einen bestimmten Betrag, der als Eingabeparameter dieser Funktion mitgegeben wird, nicht übersteigt. Dann liefert die obige Anfrage den Namen und den Preis aller Artikel, die weniger als zehn (Euro) kosten.

Die Ergebnistabelle sieht in dem konkreten Beispiel wie folgt aus:

Name	Preis
SCSI Kabel	9.90
SCSI Kabel	7.90

4.5.5 Anfragen mit Mengenoperationen

SQL unterstützt die drei klassischen Mengenoperationen **Vereinigung** (UNION), **Durchschnitt** (INTERSECT) und **Differenz** (EXCEPT) auf Basis von

Tabellen sowohl mit Mengen- als auch mit Multimengensemantik, d. h. sowohl mit als auch ohne Duplikateliminierung (→ 3.5.3; Mengenoperationen in der Relationenalgebra). Die beiden Eingangstabellen müssen paarweise typverträgliche Spalten besitzen, wobei die Spaltennamen nicht notwendigerweise gleich benannt sein müssen. Die Ergebnistabelle übernimmt immer die Spaltennamen der zuerst notierten Tabelle.

Die Syntax für eine Anfrage, die mit Mengenoperatoren kombiniert ist, sieht so aus:

```
Anfragespezifikation
{UNION | INTERSECT | EXCEPT}
[ALL | DISTINCT]
[CORRESPONDING [BY (Spaltennamensliste)]]]
Anfragespezifikation
```

L	A	B
1	2	
1	2	
3	4	
3	4	
3	4	

R	B	C
2	7	
2	7	
3	8	
3	4	
3	4	

L UNION ALL R

A	B
1	2
1	2
3	4
3	4
3	4

L UNION R

A	B
1	2
3	4
2	7
3	8

L UNION CORRESPONDING R

B
2
3
4

L INTERSECT ALL R

A	B
3	4
3	4

L INTERSECT R

A	B
3	4

L INTERSECT CORRESPONDING R

B
2
3
4

L INTERSECT ALL R

A	B
1	2
1	2
3	4

L EXCEPT R

A	B
1	2

L EXCEPT CORRESPONDING R

B
2
4

L EXCEPT ALL R

A	B
1	2
1	2
3	4

Bild 4.2 Demonstration der Mengenoperatoren

Die Voreinstellung `DISTINCT` nimmt eine Duplikateliminierung vor. Ist dies nicht gewünscht, muss der Zusatz `ALL` verwendet werden.

Die optionale `CORRESPONDING`-Klausel sorgt dafür, dass die Mengenoperation nur auf den gleichnamigen bzw. auf den explizit angegebenen Spalten der beiden Eingangstabellen ausgeführt wird.

Bild 4.2 illustriert die Wirkungsweise der verschiedenen Mengenoperatoren an zwei einfachen Beispieldatensätzen.

4.5.6 Anfragen mit Verbundoperationen

Werden in der `FROM`-Klausel einer Anfrage zwei oder mehr Tabellen durch Komma getrennt aufgelistet, verbindet SQL diese Tabellen durch ein **kartesisches Produkt (Kreuzprodukt)**. Die Ergebnistabelle enthält dann die Vereinigung aller Spalten der Eingangstabellen sowie alle Kombinationen der Zeilen aus beiden Eingangstabellen. Unerwünschte Zeilenkombinationen sind durch die Angabe einer entsprechenden `WHERE`-Klausel auszuschließen. Redundante Spalten können über eine Projektionsliste in der `SELECT`-Klausel herausprojiziert oder umbenannt werden.

Neben dem Kreuzprodukt unterstützt SQL eine Reihe von expliziten **Verbundoperatoren** (→ Tabelle 4.2).

Tabelle 4.2 Verbundarten; L und R seien Tabellenausdrücke

Verbundart	Notation
Kreuzprodukt	<code>L , R</code> bzw. <code>L CROSS JOIN R</code>
Innerer Verbund	<code>L [INNER] JOIN R</code> Bedingung
Linker äußerer Verbund	<code>L LEFT [OUTER] JOIN R</code> Bedingung
Rechter äußerer Verbund	<code>L RIGHT [OUTER] JOIN R</code> Bedingung
Voller äußerer Verbund	<code>L FULL [OUTER] JOIN R</code> Bedingung
Natürlicher innerer Verbund	<code>L NATURAL [INNER] JOIN R</code>
Natürlicher linker äußerer Verbund	<code>L NATURAL LEFT [OUTER] JOIN R</code>
Natürlicher rechter äußerer Verbund	<code>L NATURAL RIGHT [OUTER] JOIN R</code>
Natürlicher voller äußerer Verbund	<code>L NATURAL FULL [OUTER] JOIN R</code>

- *Hinweis:* Sowohl beim Verbund mit der `USING`-Klausel als auch beim natürlichen Verbund erscheinen in der Ergebnistabelle zunächst die Verbundspalten und danach die restlichen Spalten der linken bzw. rechten Eingangstabelle.

Die Semantik der verschiedenen Verbundoperationen wird in Bild 4.3 anhand von zwei einfachen Beispieldatensätzen demonstriert (→ 3.5.2; Verbundoperationen in der Relationenalgebra).

L	A	B
1	2	
4	3	

R	B	C
3	8	
5	9	

L, R WHERE L.B = R.B

A	B	B	C
4	3	3	8

L JOIN R USING(B)

B	A	C
3	4	8

L JOIN R ON L.B = R.B

A	B	B	C
4	3	3	8

L NATURAL JOIN R

B	A	C
3	4	8

L LEFT JOIN R ON L.B = R.B

A	B	B	C
4	3	3	8
1	2	-	-

L NATURAL LEFT JOIN R

B	A	C
3	4	8
2	1	-

L RIGHT JOIN R ON L.B = R.B

A	B	B	C
4	3	3	8
-	-	5	9

L NATURAL RIGHT JOIN R

B	A	C
3	4	8
5	-	9

L FULL JOIN R ON L.B = R.B

A	B	B	C
4	3	3	8
1	2	-	-
-	-	5	9

L NATURAL FULL JOIN R

B	A	C
3	4	8
2	1	-
5	-	9

Bild 4.3 Demonstration der Verbundoperatoren

Der **innere Verbund** (*join*) verbindet die passenden Zeilen zweier Tabellen anhand einer Verbundbedingung, die mit einer der beiden folgenden Klauseln angegeben wird:

ON Prädikat

oder

USING (Spaltennamensliste)

Die **ON**-Klausel erlaubt die Angabe beliebiger Verbundbedingungen. Die **USING**-Klausel dagegen führt einen *Gleichheitsverbund* über alle in der Liste aufgeführten Spalten durch. In der Ergebnistabelle sind diese Spalten nur einmal enthalten.

Der **äußere Verbund** (*outer join*) übernimmt je nach Typ – linker, rechter oder voller Verbund – alle Zeilen der äußeren Tabelle, d. h. auch die Zeilen, für die es keine passende Zeile in der jeweils anderen Tabelle gibt. Die Spalten der anderen Tabelle werden in diesem Fall mit Nullwerten aufgefüllt. Der volle äußere Verbund ist die Kombination (Vereinigung) des linken und rechten äußeren Verbundes.

Der **natürliche Verbund** (*natural join*) verbindet die Zeilen der Eingangstabellen über die gleichnamigen Spalten und eliminiert implizit die Spalten-duplikate. Ein natürlicher Verbund kann als innerer oder äußerer Verbund formuliert werden.

- ▶ *Eintrag:* Verbundklauseln aller Arten lassen sich beliebig verketten oder verschachteln, d. h., sowohl die linke als auch die rechte Eingangstabelle können selbst verbundene Tabellen sein. Um die Verbundreihenfolge zu bestimmen, setzt man Klammern um die Verbundklauseln. Ansonsten werden die Verbundklauseln von links nach rechts ausgewertet.

4.5.7 Verschachtelte Anfragen

Unter einer **verschachtelten Anfrage** (*nested query*) versteht man üblicherweise eine Anfrage, deren **WHERE**-Klausel ein Prädikat enthält, das mit Hilfe einer **Unteranfrage** (*subquery*) ausgewertet wird. Je nachdem, ob die Unteranfrage genau eine oder mehrere Zeilen liefert, können hier unterschiedliche Arten von Prädikaten eingesetzt werden. Die skalaren Vergleichsoperatoren $\theta \in \{=, <>, <, <=, >, >=, >\}$ sind nur dann verwendbar, wenn die Unteranfrage eine Tabelle mit maximal einer Zeile ausgibt. Für den Fall, dass die Unteranfrage eine Tabelle mit mehr als einer Zeile liefert, kommen die in Tabelle 4.3 dargestellten Arten von Prädikaten in Betracht.

- *Beispiel:* Gib die Nummer der Artikel aus, die zum niedrigsten Preis geliefert wurden.

```
SELECT ANr
  FROM Lieferung
 WHERE Preis = (SELECT MIN(Preis) FROM Lieferung);
```

Die Unteranfrage liefert hier genau einen Wert, den niedrigsten Preis. Obige Anfrage kann auch mit einer nicht skalaren Unteranfrage formuliert werden.

```
SELECT ANr
  FROM Lieferung
 WHERE Preis <= ALL(SELECT Preis FROM Lieferung);
```