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 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

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

ANr	Bezeichnung
101	SCSI Kabel
102	Hamserver
103	Trinitron
104	Stylus Photo

Lieferant

LNr	Name
1	Koch
2	Schulze
3	Ziehm
4	Wegert

Lieferung

ANr	LNr	Preis
101	1	9.90
101	2	7.90
102	2	69.90
103	3	159.90
103	4	249.90
104	4	259.90

Bild 4.1 Beispieltabellen

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 Beispielanfragen

❑ *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 (→ 3.5):

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

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 ⋈ Lieferung
```

4.5.2 Grundgerüst von Anfragen

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

```
SELECT Projektionsliste
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 *Projektionsliste* 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 Projektionsliste 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-Dialekten 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 Beispieltabelle (→ 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 R

A	B
1	2
3	4
2	7
3	8

L UNION CORRESPONDING R

B
2
3
4

L UNION ALL R

A	B
1	2
1	2
3	4
3	4
3	4
2	7
2	7
3	8
3	4
3	4

L INTERSECT R

A	B
3	4

L INTERSECT CORRESPONDING R

B
2

L INTERSECT ALL R

A	B
3	4
3	4

L EXCEPT R

A	B
1	2

L EXCEPT CORRESPONDING R

B
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 Beispieltabellen.

4.5.6 Anfragen mit Verbundoperationen

Werden in der FROM-Klausel einer Anfrage zwei oder mehr Tabellen durch Kommata 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	`L , R` bzw. `L CROSS JOIN R`
Innerer Verbund	`L [INNER] JOIN R Bedingung`
Linker äußerer Verbund	`L LEFT [OUTER] JOIN R Bedingung`
Rechter äußerer Verbund	`L RIGHT [OUTER] JOIN R Bedingung`
Voller äußerer Verbund	`L FULL [OUTER] JOIN R Bedingung`
Natürlicher innerer Verbund	`L NATURAL [INNER] JOIN R`
Natürlicher linker äußerer Verbund	`L NATURAL LEFT [OUTER] JOIN R`
Natürlicher rechter äußerer Verbund	`L NATURAL RIGHT [OUTER] JOIN R`
Natürlicher voller äußerer Verbund	`L NATURAL FULL [OUTER] JOIN R`

► *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 Beispieltabellen 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 Spaltenduplikate. Ein natürlicher Verbund kann als innerer oder äußerer Verbund formuliert werden.

► *Hinweis:* 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);
```