



Gunter  
Saake

Kai-Uwe  
Sattler

Andreas  
Heuer

4. Auflage

A close-up photograph of a beaver's head and shoulders, completely drenched in water. The water is dripping from its thick, brown fur, creating a misty spray around its face. The beaver's eyes are partially closed, and its mouth is slightly open, showing its yellow teeth. The background is a soft, out-of-focus greyish-blue.

# Datenbanken

## Konzepte und Sprachen

**Teil I**

**Kernkonzepte relationaler  
Datenbanken**



## Architekturen von Datenbanksystemen

In diesem Kapitel werden wir die prinzipielle Architektur eines Datenbanksystems vorstellen. Datenbankarchitekturen kann man aus verschiedenen Blickwinkeln betrachten:

- Die **Schemaarchitektur** beschreibt den Zusammenhang zwischen dem konzeptuellen, internen und externen Schema. Außerdem ordnet sie die Datenbank-Anwendungsprogramme in diese Schemata ein.
- Die **Systemarchitektur** beschreibt den Aufbau eines Datenbanksystems aus Komponenten, Bausteinen oder Werkzeugen. In Standardisierungsvorschlägen werden die Schnittstellen zwischen diesen Komponenten genormt, nicht jedoch die Komponenten selbst.
- Die **Anwendungsarchitektur** beschreibt die Einbindung des Datenbanksystems in eine konkrete Applikation, etwa ein Web-Shop, eine ERP-Anwendung oder ein entscheidungsunterstützendes Data-Warehouse-System. Insbesondere wird dabei die Aufteilung der Funktionalität des Gesamtsystems auf die einzelnen Komponenten und deren Verbindung festgelegt.

Die *Schemaarchitektur* beschreibt im Wesentlichen drei Schemata:

- Das *konzeptuelle Schema*, welches das Ergebnis der Datenmodellierung, des Datenbankentwurfs und der Datendefinition ist. Diese drei Bereiche sind Thema der nächsten Kapitel.

- Das *interne Schema* legt die Dateiorganisationen und Zugriffspfade für das konzeptuelle Schema fest. Die Realisierung des internen Schemas ist Gegenstand des zweiten Bandes [SHS05].
- Das *externe Schema* ist das Ergebnis der Sichtdefinition und legt Benutzer-sichten auf das globale, konzeptuelle Schema fest. In der Regel handelt es sich nicht nur um ein externes Schema, sondern um mehrere anwendungsspezifische externe Schemata. Sichten werden in Kapitel 14 behandelt.
- Die *Anwendungsprogramme* sind das Ergebnis der Datenbankanwendungsprogrammierung und arbeiten idealerweise auf den externen Schemata.

Die *Systemarchitektur* enthält die folgenden Arten von Komponenten, die in einem Datenbanksystem nötig sind:

- Die *Definitionskomponenten* zur Datendefinition auf der konzeptuellen Ebene, zur Definition der Dateiorganisation auf der internen Ebene und zur Sichtdefinition auf der externen Ebene.
- Die *Programmierkomponenten* zur Datenbankprogrammierung mit Datenbankoperationen, die in herkömmliche Programmiersprachen eingebettet werden.
- Die *Benutzerkomponenten* wie beispielsweise erstellte Datenbankanwendungsprogramme, interaktive Anfrage- und Änderungswerkzeuge.
- Die *Transformationskomponenten* zur Optimierung, Auswertung und Platzenzugriffsteuerung für Datenbankoperationen und zur Transformation der Ergebnisdaten von der internen in die externe (Benutzer-)Darstellung.

Die *Anwendungsarchitektur* wird das Abarbeiten eines Datenbankanwendungsprogramms genauer vorstellen und außerdem eine anwendungsbezogene Sicht auf die Werkzeuge geben, mit denen man eine Datenbankumgebung für ein spezielles Problem erstellen kann.

## 2.1 Schemaarchitektur und Datenunabhängigkeit

Ein wesentlicher Aspekt bei Datenbankanwendungen ist die Unterstützung der *Datenunabhängigkeit* durch das Datenbankmanagementsystem. Sowohl Datenbanken als auch Anwendungssysteme haben in der Regel eine lange Lebensdauer, während der sowohl die Realisierung der Datenspeicherung als auch externe Schnittstellen aus verschiedensten Gründen modifiziert oder erweitert

werden. Das Konzept der Datenunabhängigkeit hat das Ziel, eine (oft langlebige) Datenbank von notwendigen Änderungen der Anwendung abzukoppeln (und umgekehrt).

Die Datenunabhängigkeit kann in zwei Aspekte aufgeteilt werden:

- Die *Implementierungsunabhängigkeit* oder auch *physische Datenunabhängigkeit* bedeutet, dass die konzeptuelle Sicht auf einen Datenbestand unabhängig von der für die Speicherung der Daten gewählten Datenstruktur besteht.
- Die *Anwendungsunabhängigkeit* oder auch *logische Datenunabhängigkeit* hingegen koppelt die Datenbank von Änderungen und Erweiterungen der Anwendungsschnittstellen ab.

Zur Unterstützung der Datenunabhängigkeit in Datenbanksystemen wurde bereits in den 70er Jahren von der ANSI/X3/SPARC<sup>1</sup> Study Group on Database Management Systems eine *Drei-Ebenen-Schemaarchitektur* als Ergebnis einer mehrjährigen Studie vorgeschlagen [Dat86a, TK78, LD87]. ANSI ist das Kürzel für die amerikanische Standardisierungsbehörde American National Standards Institute. Die dort vorgeschlagene Aufteilung in drei Ebenen ist im Datenbankbereich inzwischen allgemein akzeptiert. Abbildung 2.1 zeigt die diesem ANSI-Vorschlag folgende, prinzipielle Schemaarchitektur.

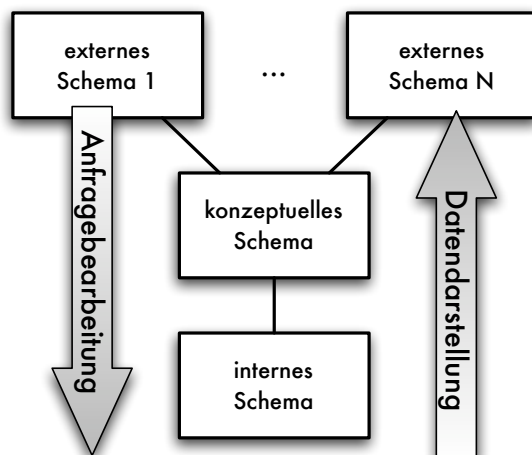


Abbildung 2.1: Drei-Ebenen-Schemaarchitektur für Datenbanken

<sup>1</sup>SPARC steht für Standards Planning and Requirements Committee.

Die ANSI-Schemaarchitektur teilt ein Datenbankschema in drei aufeinander aufbauende Ebenen auf. Von unten nach oben werden die folgenden Ebenen vorgeschlagen:

- Das *interne Schema* beschreibt die systemspezifische Realisierung der Datenbank, etwa die eingerichteten Zugriffspfade. Die Beschreibung des internen Schemas ist abhängig vom verwendeten Basissystem und der von diesem angebotenen Sprachschnittstelle.
- Das *konzeptuelle Schema* beinhaltet eine implementierungsunabhängige Modellierung der gesamten Datenbank in einem systemunabhängigen Datenmodell, zum Beispiel dem ER-Modell oder dem relationalen Modell. Das konzeptuelle Schema beschreibt die Struktur der Datenbank vollständig.
- Basierend auf dem konzeptuellen Schema können mehrere *externe Schemata* definiert werden, die anwendungsspezifische (Teil-)Sichten auf den gesamten Datenbestand festlegen.

Oft beschreiben externe Sichten einen anwendungsspezifischen Ausschnitt des konzeptuellen Schemas unter Benutzung desselben Datenmodells. Es ist aber auch möglich, unterschiedliche Datenbankmodelle für verschiedene externe Schemata zu verwenden.

Die Sprachmittel und typischen Konzepte auf den verschiedenen Ebenen werden im Folgenden exemplarisch anhand einer kleinen Beispielanwendung vorgestellt.

Zwischen den verschiedenen Schemaebenen müssen Abbildungen festgelegt werden, die die Transformation von Datenbankzuständen, Anfragen und Änderungstransaktionen zwischen den Ebenen ermöglichen. Da diese Transformationen vom Datenbankmanagementsystem durchgeführt werden, müssen diese Abbildungen in einer formalen Beschreibungssprache mit festgelegter Semantik notiert werden.

Die Aufgabe der Abbildungen zwischen den Ebenen kann in zwei Problembereiche aufgeteilt werden:

- Die *Anfragebearbeitung* erfordert eine Übersetzung von Anfragen und Änderungsoperationen, die bezüglich der externen Schemata formuliert wurden, in Operationen auf den internen Datenstrukturen (über den Zwischenschritt der konzeptuellen Ebene).
- Die *Datendarstellung* erfordert eine Transformation in der umgekehrten Richtung: Die internen Datenstrukturen von Anfrageergebnissen müssen derart transformiert werden, dass sie den Beschreibungskonzepten der externen Darstellungen entsprechen.

## Ebenenarchitektur am Beispiel

Wir wollen die Basisidee der Drei-Ebenenarchitektur von Datenbankschemata im Folgenden wieder anhand einer kleinen Beispielmodellierung zu unserer Weindatenbank diskutieren. Die konzeptuelle Gesamtsicht ist im relationalen Datenbankmodell beschrieben.

### Die konzeptuelle Gesamtsicht

Die konzeptuelle Gesamtsicht erfolgt in relationaler Darstellung. Die Datenbank ist in zwei Relationen gespeichert, wie in Abbildung 2.2 dargestellt.

WEINE	WeinID	Name	Farbe	Jahrgang	Weingut → ERZEUGER
	1042	La Rose Grand Cru	Rot	1998	Château La Rose
	2168	Creek Shiraz	Rot	2003	Creek
	3456	Zinfandel	Rot	2004	Helena
	2171	Pinot Noir	Rot	2001	Creek
	3478	Pinot Noir	Rot	1999	Helena
	4711	Riesling Reserve	Weiß	1999	Müller
	4961	Chardonnay	Weiß	2002	Bighorn

ERZEUGER	Weingut	Anbaugebiet	Region
	Creek	Barossa Valley	South Australia
	Helena	Napa Valley	Kalifornien
	Château La Rose	Saint-Emilion	Bordeaux
	Château La Pointe	Pomerol	Bordeaux
	Müller	Rheingau	Hessen
	Bighorn	Napa Valley	Kalifornien

Abbildung 2.2: Konzeptuelle Beispieldatenbank in Relationendarstellung

Schlüssel, also identifizierende Attribute in Relationen, werden durch Unterstreichung gekennzeichnet. Bezüge zwischen Relationen, die sogenannten Fremdschlüssel als Verweise auf Schlüssel, sind in der Beispielrelation mit dem Bezug zu dem Schlüssel einer anderen Relation angegeben.

### Externe Sichten

Eine mögliche Anwendungssicht wäre dadurch gegeben, dass die Daten in *einer* Relation dargestellt werden, wobei die Attribute WeinID und Jahrgang ausgeblendet werden sollen (wie in Abbildung 2.3 gezeigt).

Diese externe Sicht kann in SQL-Datenbanksystemen einfach durch eine View-Definition (vgl. Kapitel 14) realisiert werden.

Dieses erste Beispiel definiert eine flache Tabelle als Sicht auf andere flache Tabellen, verlässt also den verwendeten Beschreibungsrahmen im gewissen Sinne nicht. Aber auch Sichten in einem anderen Datenbankmodell sind möglich, etwa als eine hierarchisch aufgebaute Relation wie in Abbildung 2.4.

Name	Farbe	Weingut	Anbaugebiet	Region
La Rose Grand Cru	Rot	Château La Rose	Saint-Emilion	Bordeaux
Creek Shiraz	Rot	Creek	Barossa Valley	South Australia
Zinfandel	Rot	Helena	Napa Valley	Kalifornien
Pinot Noir	Rot	Creek	Barossa Valley	South Australia
Pinot Noir	Rot	Helena	Napa Valley	Kalifornien
Riesling Reserve	Weiß	Müller	Rheingau	Hessen
Chardonnay	Weiß	Bighorn	Napa Valley	Kalifornien

Abbildung 2.3: Externe Sicht auf zwei Relationen, dargestellt als eine Relation

Region	Anbaugebiet	Weingut	Wein	
			Name	Farbe
South Australia	Barossa Valley	Creek	Creek Shiraz	Rot
			Pinot Noir	Rot
Kalifornien	Napa Valley	Helena	Zinfandel	Rot
			Pinot Noir	Rot
		Bighorn	Chardonnay	Weiß
Bordeaux	Saint-Emilion	Château La Rose	La Rose Grand Cru	Rot
	Pomerol	Château La Pointe		
Hessen	Rheingau	Müller	Riesling Reserve	Weiß

Abbildung 2.4: Externe Sicht als hierarchisch aufgebaute Relation

Diese externe Darstellung ist in den meisten SQL-Datenbanken nicht möglich, entspricht aber der hierarchischen Darstellung von Tabellen, wie sie in vielen Anwendungen üblich ist. Eine derartige Datenrepräsentation ist allerdings in *objektrelationalen Datenbanken* möglich, die wir später noch kennenlernen werden.

### Interne Darstellung

Für die interne Darstellung kann ein Datenbankmanagementsystem optimierte Datenstrukturen verwenden. Abbildung 2.5 zeigt eine mögliche Variante: die ERZEUGER-Tupel sind über das Attribut Weingut als Schlüssel in einem Mehrwegbaum organisiert. Die Datensätze selbst sind zusammen mit den zugehörigen Tupeln der Relation WEINE auf Blöcken gespeichert.

Eine derartige Organisation wird auch als *Clusterspeicherung* bezeichnet und zeigt, wie stark die interne Realisierung von der konzeptuellen Darstellung abweichen kann.

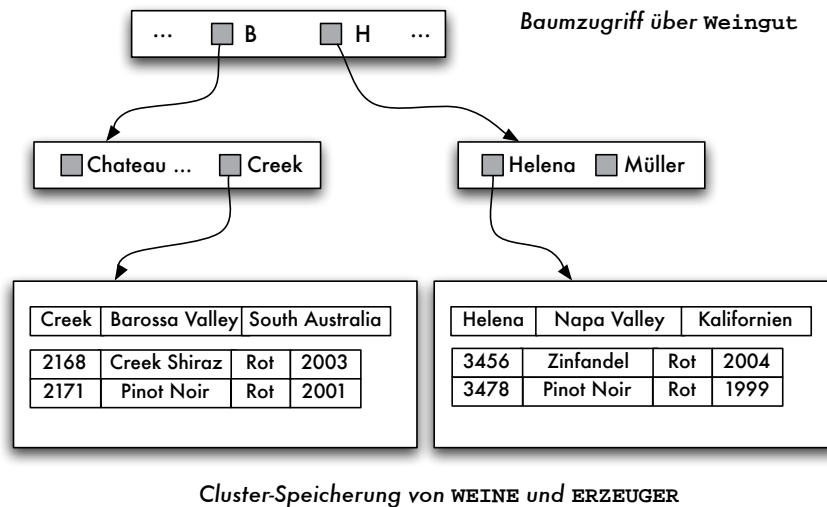


Abbildung 2.5: Interne Realisierung durch Baumzugriffsstruktur und Clusterung

## 2.2 Systemarchitekturen

Systemarchitekturen beschreiben die Komponenten eines Datenbanksystems. Es gibt zwei wichtige Architekturvorschläge, die in diesem Abschnitt vorgestellt werden sollen:

- Die ANSI-SPARC-Architektur als detaillierte Version unserer etwas groben Drei-Ebenen-Architektur.
- Die Fünf-Schichten-Architektur als detaillierte Version der Transformationskomponenten der Drei-Ebenen-Architektur.

Nach diesen beiden Architekturvorschlägen werden wir auf die Architekturen konkreter Datenbanksysteme und Pseudo-Datenbanksysteme eingehen.

### 2.2.1 ANSI-SPARC-Architektur

Im ANSI-SPARC-Normvorschlag wurde neben der Drei-Ebenen-Schemaarchitektur auch eine Drei-Ebenen-Systemarchitektur vorgestellt. Im Wesentlichen entspricht die Architektur unserer vereinfachten Architektur aus dem letzten Kapitel, die wir in Abbildung 2.6 noch einmal aufführen.

Der endgültige Vorschlag stammt aus dem Jahre 1978 und verfeinert die grundlegende Architektur um

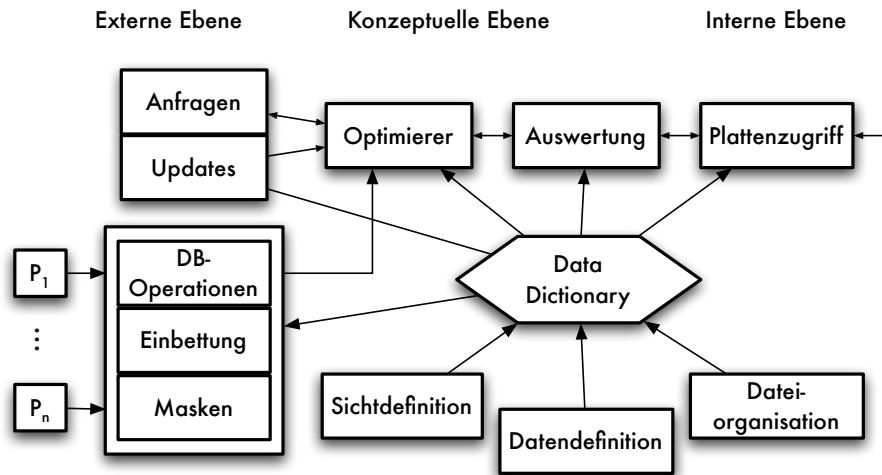


Abbildung 2.6: Vereinfachte Architektur eines DBMS

- eine detailliertere interne Ebene, insbesondere mit Berücksichtigung der diversen Betriebssystemkomponenten,
- weitere interaktive und Programmierkomponenten auf der externen Ebene wie etwa Berichtgeneratoren und
- eine genaue Bezeichnung und Normierung der Schnittstellen zwischen den einzelnen Komponenten.

Eine genauere Darstellung entnehme man der Originalliteratur [TK78, Dat86a] oder [LD87]. Die in Abbildung 2.6 aufgeführten Komponenten kann man folgendermaßen klassifizieren (siehe auch Abbildung 2.7):

- Die *Definitionskomponenten* bieten Datenbank-, System- und Anwendungsadministratoren die Möglichkeit zur Datendefinition, Definition der Dateiformen und Zugriffspfade sowie Sichtdefinition.
- Die *Programmierkomponenten* beinhalten eine vollständige Entwicklungsumgebung in einer höheren Programmiersprache, einer 4GL<sup>2</sup> oder einer graphischen Sprache, die Datenbankoperationen und in den meisten Fällen auch Werkzeuge zur Definition von Menüs, Masken und anderen Primitive einer graphischen Benutzeroberfläche integriert.

<sup>2</sup>4th Generation Language, dt. Programmiersprache der vierten Generation

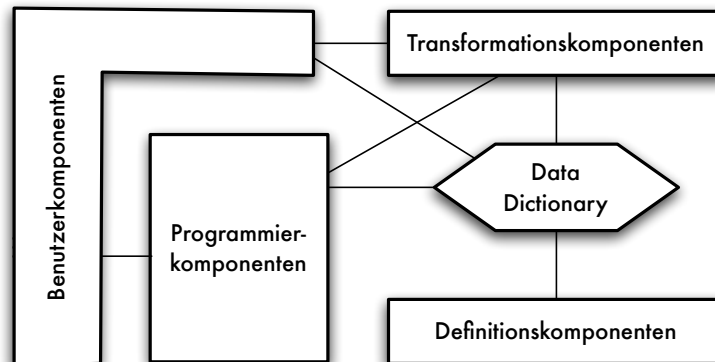


Abbildung 2.7: Klassifikation von Komponenten eines DBMS

- Die *Benutzerkomponenten* umfassen die interaktiven Anfrage- und Änderungs- (oder Update-)Werkzeuge für anspruchsvolle Laien und die vorgefertigten Datenbankanwendungsprogramme für den unbedarften Benutzer („Parametric User“, die in der Abbildung mit  $P_1$  bis  $P_n$  bezeichnet werden).
- Die *Transformationskomponenten* wandeln Anfrage- und Änderungsoperationen schrittweise über Optimierung und Auswertung in Plattenzugriffsoperationen um. Umgekehrt werden die in Blöcken der Platte organisierten Bytes in die externe Benutzerdarstellung (im Relationenmodell: Tabellen) transformiert.
- Zentraler Kern des ganzen Systems ist das *Data Dictionary* (der *Schema-katalog* oder das *Datenwörterbuch*), das die Daten aus den Definitionskomponenten aufnimmt und die Programmier-, Benutzer- und Transformationskomponenten mit diesen Informationen versorgt.

Gerade die Transformationskomponenten sind in der Drei-Ebenen-Architektur noch etwas ungenau beschrieben. Die folgende Fünf-Schichten-Architektur wird die schrittweise Transformation von Operationen und Daten genauer darstellen.

### 2.2.2 Fünf-Schichten-Architektur

Nach Ideen von Senko [Sen73] wurde als Weiterentwicklung von Härder [Här87] im Rahmen des IBM-Prototyps *System R* die folgende Systemarchitektur eingeführt.

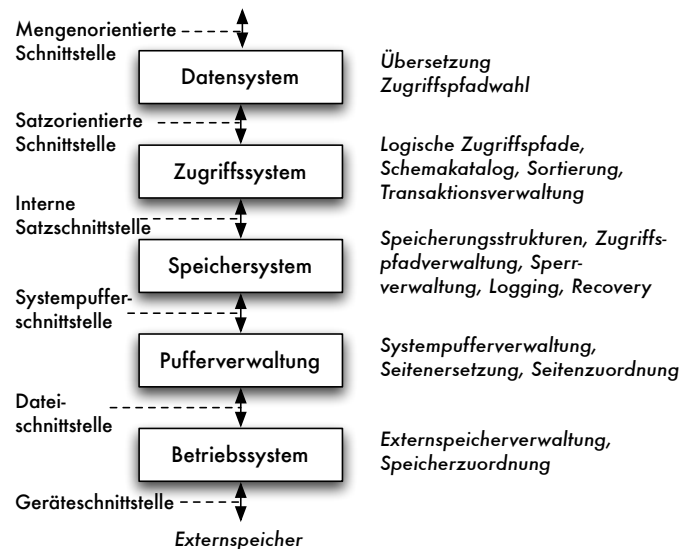


Abbildung 2.8: Funktionsorientierte Sicht auf die Fünf-Schichten-Architektur

Die *Fünf-Schichten-Architektur* basiert auf einer genaueren Beschreibung der in einem Datenbankmanagementsystem enthaltenen Transformationskomponenten. Diese realisiert eine schrittweise Transformation von Anfragen und Änderungen von der abstrakten Datenbankmodellebene bis hinunter zu Zugriffen auf die Speichermedien. Abbildung 2.8 zeigt die einzelnen Transformationskomponenten mit den zugehörigen Aufgaben sowie die zwischen den Komponenten geltenden Schnittstellen. Die Aufgaben der höheren Komponenten sind dort jeweils unterteilt in Aufgaben der Anfragetransformation (links) sowie der Datensicherung (rechts).

Die *mengenorientierte Schnittstelle* realisiert eine deklarative Datenmanipulationssprache auf Tabellen, Sichten und Zeilen einer Tabelle. Eine typische Sprache für diese Abstraktionsebene ist SQL mit mengenorientiertem Zugriff auf Relationen.

Die Anweisungen der MOS werden durch das *Datensystem* auf die *satzorientierte Schnittstelle SOS* umgesetzt. Die SOS realisiert einen navigierenden Zugriff auf einer internen Darstellung der Relationen. Manipulierte Objekte der SOS sind typisierte Datensätze und interne Relationen (geordnete Listen von Datensätzen mit Duplikaten) sowie logische Zugriffspfade, die sogenannten *Indexe*, und temporäre Zugriffsstrukturen, die *Scans*. Aufgaben des Datensystems sind die Übersetzung und Optimierung etwa von SQL-Anfragen auf die

SOS unter Ausnutzen der Zugriffspfade sowie die Realisierung der Zugriffs- und Integritätskontrolle.

Das *Zugriffssystem* übernimmt die Transformation auf die *interne Schnittstelle ISS*. Hier werden interne Tupel einheitlich verwaltet, also ohne Typisierung aufgrund unterschiedlicher Relationstypen wie in der SOS. Auf der ISS werden die Speicherstrukturen der Zugriffspfade implementiert, etwa konkrete Operationen auf B\*-Bäumen und Hashtabellen. Neben der Umsetzung der SOS auf diese implementierungsnähere Darstellung realisiert das Zugriffssystem Operationen wie die Sortierung und den Mehrbenutzerbetrieb mit Transaktionen.

Das *Speichersystem* hat die Aufgabe, die Datenstrukturen und Operationen der ISS auf internen Seiten eines virtuellen linearen Adressraums zu realisieren. Dieser interne Adressraum wird durch die Operationen der *Systempufferschnittstelle* manipuliert. Typische Objekte sind interne Seiten und Seitenadressen, zugehörige Operationen sind etwa Freigeben und Bereitstellen von Seiten. Neben den typischen Operationen zur Verwaltung eines internen Seitenpuffers mit Seitenwechselstrategien realisiert das Speichersystem die Sperrverwaltung für den Mehrbenutzerbetrieb sowie das Schreiben des Logbuchs für das Recovery.

Die *Pufferverwaltung* bildet die internen Seiten auf die Blöcke der *Dateischnittstelle DS* des Betriebssystems ab, das die Externspeicherverwaltung übernimmt. Die Umsetzung der Operationen der Dateischnittstelle auf die *Geräteschnittstelle* erfolgt nun nicht mehr durch Komponenten des DBMS, sondern durch das Betriebssystem.

Die Fünf-Schichten-Architektur ist nur *ein* Vorschlag für eine Aufteilung in Transformationsschritte, der auf den ursprünglichen Prototyp-Entwicklungen für relationale DBMS basiert. Die Architektur kann etwa verkürzt werden, indem Zugriffssystem und Speichersystem in einer Komponente zusammengefasst werden. Einige ältere Datenbankmodelle, aber auch einige moderne objektorientierte DBMS, bieten keine mengenorientierte Schnittstelle an und überlassen deren Aufgaben dem Anwendungsprogrammierer. Auch können einige Aufgaben der tieferen Ebenen alternativ auf den höheren Ebenen realisiert werden; ein Beispiel wäre die Realisierung eines *Objektpuffers* im Zugriffssystem anstelle eines Seitenpuffers.

Auch die Zuordnung der Datensicherungsmaßnahmen zu den Ebenen ist nicht zwingend vorgegeben. Die Sperrverwaltung kann zum Beispiel auf höheren Ebenen angesiedelt werden, während alternativ die Zugriffskontrolle implementierungsnäher modelliert werden könnte.

Abschließend soll noch bemerkt werden, dass alle in diesem Unterabschnitt angesprochenen Komponenten der Fünf-Schichten-Architektur im zweiten Band [SHS05] dieses Lehrbuches weitaus ausführlicher behandelt werden: Der zweite Band widmet sich ausschließlich diesen Implementierungstechniken von Datenbanksystemen, während sich der vorliegende Band eher auf den Ent-

wurf von Datenbankanwendungen und die Benutzung von Datenbanksystemen konzentriert.

### 2.2.3 Konkrete Systemarchitekturen

In diesem Abschnitt wollen wir die Umsetzung der im vorigen Abschnitt beschriebenen idealisierten Systemarchitektur in konkreten Systemen kommerzieller Hersteller vorstellen. Wir beschränken uns dabei auf relationale Datenbankmanagementsysteme (RDBMS), die einerseits heute den Markt dominieren und andererseits in den aktuellen Versionen auch Erweiterungen über das Relationenmodell hinaus bieten, etwa zur Verwaltung objektorientierter Daten, von XML-Dokumenten oder Multimedia-Daten.

Gegenwärtig sind eine ganze Reihe von RDBMS auf dem Markt verfügbar. Zu den wichtigsten Systemen zählen die kommerziellen Vertreter wie IBM DB2 V.9.1, Oracle11g, Microsoft SQL Server 2005, Ingres 2006, Sybase Adaptive Server Enterprise sowie die Open-Source-Systeme PostgreSQL, MySQL und Firebird. Gemeinsame Merkmale dieser Systeme sind:

- eine Drei-Ebenen-Architektur nach ANSI-SPARC,
- eine einheitliche Datenbanksprache (Structured Query Language, kurz SQL),
- eine Einbettung dieser Sprache in Programmiersprachen wie C/C++ oder Java,
- diverse Werkzeuge für die Definition, Anfrage und Darstellung von Daten und den Entwurf von Datenbankanwendungsprogrammen und der Benutzerinteraktion sowie
- kontrollierter Mehrbenutzerbetrieb, Zugriffskontrolle und Datensicherheitsmechanismen.

Das erste relationale Datenbanksystem war der im kalifornischen Entwicklungslabor Anfang der 70er Jahre entstandene Forschungsprototyp System R. Dieses System umfasste ca. 80.000 Zeilen PL/1-, PL/S- und Assembler-Code und kam mit einer Codegröße von etwa 1,2 MByte aus! Als Datenbanksprache wurde SEQUEL unterstützt – eine Sprache, die nachhaltig die Standardisierung von SQL beeinflusst hat. Aus den Erfahrungen mit diesem System hat IBM später zwei kommerzielle Systeme entwickelt:

- *DB2* zunächst für IBM-Rechner unter MVS, inzwischen aber auch für Windows und Unix bzw. Linux,
- *SQL/DS* (SQL/Data System) für IBM-Rechner unter DOS/VSE oder VM/CMS.

Abbildung 2.9 zeigt die Systemarchitektur von DB2, die auf einem Client-Server-Ansatz basiert. Hierbei kommunizieren die Clients über geeignete Netzwerkprotokolle oder gemeinsamen Speicher mit dem Datenbankserver. Dieser besteht aus mehreren Prozessen bzw. Agenten, die auch als Engine Dispatchable Units (EDU) bezeichnet werden. Der erste Kommunikationspartner für eine Clientanwendung ist dabei der Listener-Prozess, der eingehende Verbindungsanforderungen an den Koordinationsagenten weiterleitet. Dieser führt alle Datenbankankorderungen im Auftrag der jeweiligen Anwendung aus. Im Fall einer Parallelverarbeitung auf einer geeigneten Hardwareplattform werden die parallel auszuführenden Teilanfragen noch an spezielle Subagenten delegiert, die in einem Pool verwaltet werden.

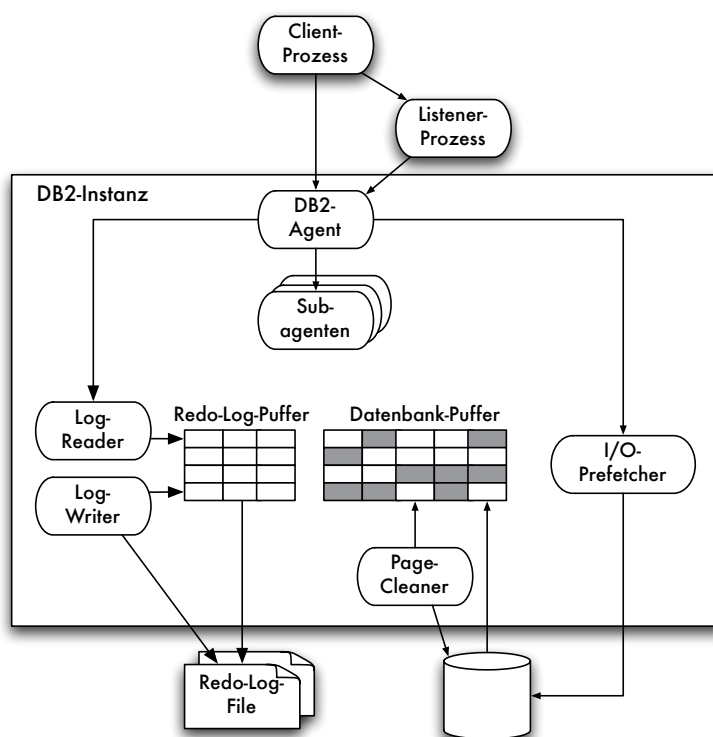


Abbildung 2.9: Architektur von IBM DB2

Zu den eigentlichen Datenbankprozessen gehören der (in Abbildung 2.9 nicht dargestellte) System Controller-Prozess zur Steuerung aller anderen Funktionen sowie die folgenden Prozesse, die pro DB2-Instanz laufen:

- Der I/O-Prefetcher übernimmt das Vorablesen von Seiten in den Puffer.
- Der Page-Cleaner-Prozess ist für das Zurückschreiben von modifizierten Seiten aus dem Puffer auf die Festplatte verantwortlich.
- Der Log-Reader-Prozess behandelt die Logeinträge für die Transaktionsverwaltung und das Recovery im Falle des Abbruchs einer Transaktion.
- Der Log-Writer-Prozess ist für das Schreiben der Log-Einträge in das Logbuch verantwortlich.
- Die Erkennung der Verklemmung von Transaktionen durch gegenseitiges Sperren übernimmt ein weiterer Prozess.

Darüber hinaus gibt es noch weitere Prozesse, etwa zur Archivierung von Logdateien, zur Überwachung der laufenden Prozesse auf abnormale Beendigung sowie zur parallelen und verteilten Verarbeitung. Alle serverseitigen Prozesse kommunizieren über einen gemeinsamen Speicherbereich – dem Database Global Memory.

Fast parallel zu den Arbeiten am System R wurde an der University of California in Berkeley (UCB) das System Ingres entwickelt. Die Originaldatenbanksprache in Ingres war QUEL (QUERy Language). Durch die Standardisierung von SQL wurde SQL als zweite Datenbanksprache in Ingres eingeführt. Ingres hatte sich vom Universitätsprototypen zwischenzeitlich zu einem Technologieführer bei relationalen Datenbanksystemen entwickelt, beispielsweise im Bereich der Optimierung von Anfragen und bei der Integration neuer Konzepte wie benutzerdefinierter Datentypen und Regeln. Aus dem Universitätsprototypen *University INGRES* entstand ab 1980 ein Produkt der Firma Relational Technology. Später wurde diese Firma zur INGRES Inc., wurde dann von dem Softwarehersteller ASK und schließlich von CA (Computer Associates) übernommen. Heute wird Ingres von der Ingres Corp. weiterentwickelt, ist aber als Open-Source-Version verfügbar. Auch Ingres ist für diverse Rechner und Betriebssystemplattformen zu haben, insbesondere für den Unix-Bereich.

*Oracle* (von der Oracle Corporation, früher Relational Software Corporation) wird vom Hersteller als Marktführer bei relationalen Datenbanksystemen bezeichnet. Oracle gibt es für fast alle Plattformen vom PC bis zum Großrechner und für fast alle Betriebssysteme von Windows bis zu herstellerspezifischen Betriebssystemen. Auch hier ist die Datenbanksprache SQL.

Eine weitere wichtige Rolle spielt Sybase sowohl bezüglich des aktuellen Systems Adaptive Server Enterprise als auch aufgrund der „Verwandtschaft“ zu Microsoft SQL Server. Sybase war eines der ersten Systeme für die Unix-Plattform. Ende der 80er Jahre war Microsoft auf der Suche nach einem SQL-System für die Windows NT-Plattform und übernahm dazu die Codebasis von Sybase. Dies erklärt auch die Ähnlichkeit beider Produkte, etwa in

Form der Unterstützung der Datenbanksprache Transact-SQL, einer prozeduralen Erweiterung von SQL. Seit Mitte der 90er Jahre werden beide Produkte jedoch unabhängig voneinander entwickelt. Eine Besonderheit der aktuellen SQL-Server-Version von Microsoft ist speziell die Unterstützung der .NET-Umgebung.

Ein DBMS, das für die Verwaltung besonders großer Datenmengen bis in den Petabyte-Bereich ausgerichtet ist, ist Teradata von NCR. Hierbei handelt es sich um ein paralleles DBMS, das auf Systemen mit mehreren tausend Knoten bzw. Prozessoren eingesetzt werden kann und diese Knoten zur parallelen Verarbeitung komplexer SQL-Anfrage nutzt. Damit ist Teradata speziell für den Data-Warehouse-Bereich geeignet.

Neben den „großen“ serverbasierten DBMS gibt es noch eine Reihe weiterer Datenbanksysteme, die als einbettbare Lösung (*Embedded Database*, d.h. in eine Applikation integriert, wie etwa SQLite oder Berkeley DB) oder als Desktoplösung für den Ein-Benutzer-Betrieb ausgelegt sind. Diese Systeme erfüllen in der Regel nicht alle Anforderungen an ein DBMS und werden daher oft auch als Pseudo-DBS bezeichnet. Der bekannteste Vertreter ist sicher Microsoft Access als Teil der Office-Suite. MS Access besteht aus zwei Komponenten: der graphischen Benutzeroberfläche und dem Datenbankkern – der sogenannten Jet Engine. Die graphische Benutzeroberfläche umfasst neben der graphischen Anfrageschnittstelle QBE (siehe auch Abschnitt 11.2) eine vollständige Entwicklungsumgebung für Datenbankanwendungen mit Formularen (Forms), Reports und der Möglichkeit, Anwendungscode in *Visual Basic for Applications* zu implementieren. Alle diese Objekte werden komplett in der Access-Datenbank abgelegt. Die Jet Engine ist eine Sammlung von Bibliotheken (DLLs) zur Manipulation von Access-Datenbankdateien sowie einer objektorientierten Schnittstelle für Basic-Programme (*Data Access Objects – DAO*). Obwohl diese Jet Engine Techniken zur Verarbeitung von SQL-Anfragen zur Sicherstellung der Datenintegrität sowie zur Synchronisation von Transaktionen bereitstellt, ist sie nur für kleinere Anwendungen mit wenigen Nutzern geeignet. So gibt es bei der Verarbeiten im Netzwerk mit mehreren Nutzern Performanzeinschränkungen. Ebenso fehlen Recoverymaßnahmen sowie leistungsfähige Techniken zur Anfrageoptimierung. Allerdings können die graphische Benutzeroberfläche und die in Access erstellten Anwendungen über ODBC (siehe Kapitel 13) auch auf serverbasierte SQL-Systeme wie SQL Server oder Oracle zugreifen.

Ein vergleichbares System ist FileMaker, das insbesondere auf der Apple-Plattform verbreitet ist.

## 2.3 Anwendungsarchitekturen

Vielen Datenbankanwendungen liegt heutzutage meist eine Client-Server-Architektur zugrunde. Bei dieser Softwarearchitektur nimmt ein Dienstneh-

mer (ein sogenannter Client) die Dienste eines Dienstbringers (Servers) zur Erfüllung seiner Aufgaben in Anspruch. Hierzu sendet er eine Anforderung an den Server, der diese bearbeitet und eventuelle Ergebnisse zurücksendet (Abbildung 2.10). Für jeden Server ist bekannt, welche Dienste er erbringen kann. Die Interaktion zwischen Client und Server wird dabei durch ein Protokoll geregelt. Charakteristisch ist weiterhin die Asymmetrie der Rollenverteilung, wobei diese jedoch nicht unbedingt starr ist. So kann ein Client aus einer Beziehung durchaus die Rolle eines Servers in einer anderen Beziehung einnehmen.

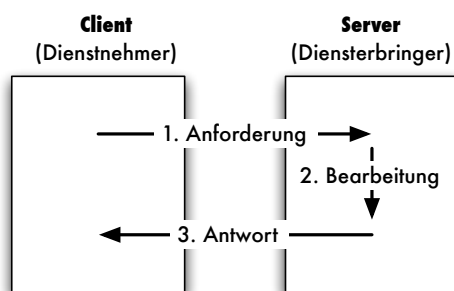


Abbildung 2.10: Client-Server-Modell

Für Datenbankanwendungen ist dieses Modell nahezu ideal. So erfordert ein Mehrbenutzerbetrieb auf einem gemeinsamen Datenbestand eine zentrale Kontrollinstanz, andererseits bietet die heute verfügbare Hardwarelandschaft mit den leistungsfähigen PCs geeignete Frontends für eine benutzergerechte Präsentation und Manipulation der Daten. Das DBMS bildet in diesem Szenario den Server, der als Dienst die Ausführung von Datenbankoperationen wie Anfragen, Änderungen oder Transaktionskommandos anbietet. Die Clients, die häufig auf der Basis von PCs realisiert sind, können Daten als Ergebnis von Anfragen anfordern und in geeigneter Weise präsentieren bzw. weiterverarbeiten. So lassen sich eine Vielzahl von Clients mit einem gemeinsamen Datenbankservers einsetzen. Durch die zentrale Verwaltung der Daten werden Redundanzen und Inkonsistenzen speziell auch im Mehrbenutzerbetrieb vermieden und der Administrationsaufwand gesenkt. Im Gegensatz zu Zentralrechneranwendungen auf Mainframe-Basis ermöglicht jedoch die Verarbeitungskapazität der Clients, die Ressourcen des Servers bei lokaler Verarbeitung oder Aufbereitung der Daten zu entlasten.

Die Trennung in Dienstnehmer und Dienstbringer liefert natürlich nur einen groben Hinweis zur Aufteilung der Funktionalität. So existieren verschiedene Möglichkeiten zur Realisierung einer Client-Server-basierten Datenbankanwendung. Grundsätzlich lassen sich dabei drei Funktionsgruppen unterscheiden:

- Präsentation und Benutzerinteraktion,
- die eigentliche Anwendungslogik sowie
- die Datenmanagementfunktionalität einschließlich der Anfragebearbeitung und Transaktionskontrolle.

Die Zerlegung einer Anwendung und die Zuordnung der Funktionen zu Client bzw. Server kann – unter Beibehaltung der Reihenfolge – im Prinzip an jeder Stelle auch innerhalb einer Gruppe erfolgen. So kann z.B. ein Teil der Präsentationsfunktionalität vom Server erbracht werden, etwa in einer Webanwendung, die Diagramme als Grafiken vorberechnet und zum Webbrowser sendet. Der entgegengesetzte Fall wird durch sogenannte Seitenserver repräsentiert, die ausschließlich Speicherseiten verwalten und auf Anforderung zum Client senden. Eine solche Architektur wird beispielsweise in einigen Objekt-datenbanksystemen verwendet.

Der häufigste Fall ist jedoch die Zuordnung der Präsentations- und Interaktionsfunktionen zum Client sowie der Datenmanagementfunktionen zum Server. Zur Verteilung der eigentlichen Anwendungslogik bieten moderne DBMS zwei Varianten. So können die Anwendungsfunktionen einerseits vollständig im Client implementiert werden. In diesem Fall bietet der Datenbankserver nur die Möglichkeit der Ausführung von SQL-Operationen, d.h. als Schnittstelle dient hier SQL. Mit der zweiten Variante werden Teile der Anwendungsfunktionalität durch gespeicherte Prozeduren realisiert, die vom Datenbankserver verwaltet und auch ausgeführt werden. Bei beiden Ansätzen kommunizieren Client und Server über ein DBMS-spezifisches Protokoll. Da hierbei die drei Funktionsgruppen auf zwei Komponenten aufgeteilt werden, spricht man auch von einer *Zwei-Schichten-Architektur* (Abbildung 2.11(a)).

Obwohl diese Architektur sehr häufig zum Einsatz kommt, weist sie dennoch einige Nachteile auf:

- eine hohe Netzwerkbelastung durch Operationen, die auf vielen Datensätzen ausgeführt werden sollen und daher einen Transport dieser Datensätze zum Client erfordern,
- eine eingeschränkte Skalierbarkeit, da das DBMS den „Flaschenhals“ bildet und
- nicht zuletzt einen hohen Wartungsaufwand, da bei Änderungen der Anwendungsfunktionen diese auf allen Clients aktualisiert werden müssen.

Teilweise können diese Nachteile jedoch durch den Einsatz von gespeicherten Prozeduren vermieden werden.

Eine Alternative ist die Verwendung einer *Drei-Schichten-Architektur* (Abbildung 2.11(b)). Hierbei wird die Anwendungslogik durch eine eigene Schicht

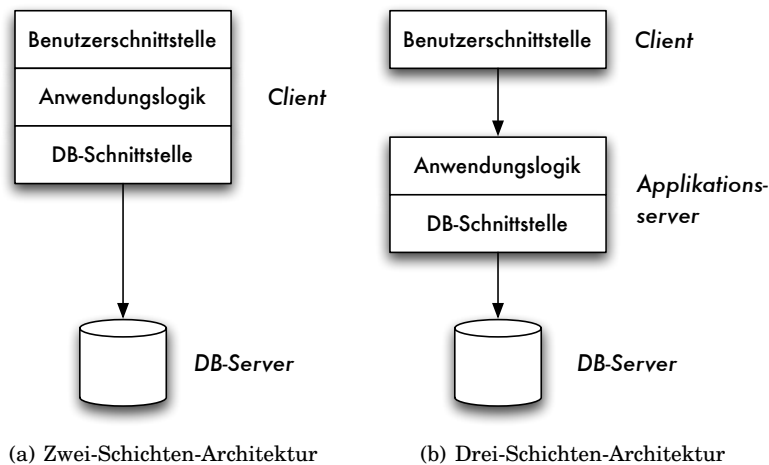


Abbildung 2.11: Anwendungsarchitekturen im Vergleich

realisiert. So entsteht ein Applikationsserver, der semantisch höhere Dienste bereitstellt. Dienste dieser Art können z.B. das Eintragen eines Kunden, das Anlegen einer Bestellung oder das Ausliefern eines Produktes mit allen damit verbundenen Aktionen sein. Bei einer streng objektorientierten Vorgehensweise lassen sich diese Dienste applikationsspezifischen Objekten wie Kunden, Bestellungen oder Produkten zuordnen. Da Clients auf diese *Geschäfts- oder Businessobjekte* ausschließlich über Objektmethoden oder Dienste zugreifen können, wird sichergestellt, dass die Daten nur in der zulässigen Weise entsprechend der implementierten Geschäftslogik manipuliert werden.

Zur Kommunikation zwischen Client und Applikationsserver werden häufig spezielle Middleware-Lösungen (z.B. CORBA oder RMI) oder auch Webmechanismen wie HTTP eingesetzt. Eine aktuelle Technologie sind *Web Services*: Hierbei werden die Dienste in XML beschrieben und auch die Nachrichten für Dienstaufwurf und -antwort sind in XML kodiert und werden über SOAP bzw. XML-RPC ausgetauscht. Web Services erlauben eine einfache Interoperabilität über System- und Programmiersprachengrenzen hinweg, sind aber aufgrund des höheren Kommunikationsaufwandes für zeitkritische Anwendungen weniger geeignet.

Applikationsserver stellen eine integrierte Umgebung für das Zusammenstellen und Ausführen der Anwendungsdienste bereit. Konkrete Beispiele hierfür sind insbesondere die auf Java Enterprise Edition (Java EE) basierenden Umgebungen JBoss von Red Hat, IBM WebSphere oder Apples WebObjects. Java EE-basierte Applikationsserver sind in sogenannte Container unterteilt, die eine Laufzeitumgebung für bestimmte Objekte darstellen. Eine wichtige Form

sind EJB-Container für Enterprise Java Beans (EJB) – die Geschäftskomponenten der Java-Plattform. Der Container arbeitet dabei als Vermittler zwischen Client und Datenbank, indem die Dienstanforderungen (Benutzereingaben) vom Client entgegen genommen werden und an die entsprechenden EJB-Komponenten weitergeleitet werden. Die Verwaltung der Daten der Komponenten (d.h. das Speichern in der Datenbank bzw. das Wiederherstellen aus der Datenbank) kann wahlweise vom Container übernommen oder vom Entwickler von Hand programmiert werden. Die Kommunikation mit dem DBMS erfolgt in beiden Fällen über die DBMS-eigene Schnittstelle (z.B. SQL). Darüber hinaus bietet ein Applikationsserver noch weitere Dienste wie etwa zur Transaktionsverwaltung, zur Kommunikation und auch zur Entwicklungs- und Installationsunterstützung (das sogenannte Deployment).

Eine Drei-Schichten-Architektur besitzt im Vergleich zum Zwei-Schichten-Ansatz vor allem dann Vorteile, wenn die Anwendungslogik für mehrere, eventuell auch verschiedene Clients oder Anwendungssysteme genutzt werden soll. Allerdings steht dem ein erhöhter Entwicklungsaufwand entgegen.

Die sogenannte *Serviceorientierte Architektur (SOA)* ist eine Weiterentwicklung dieser Idee und zielt auf eine verbesserte Unterstützung von Geschäftsprozessen ab, indem die Dienste (Services) auch von verschiedenen voneinander unabhängigen Diensterbringern (Softwaresystemen) angeboten werden können.

## 2.4 Zusammenfassung

Die Architektur von Datenbanksystemen kann aus verschiedenen Blickwinkeln betrachtet werden: als Schemaarchitektur mit den Zusammenhängen zwischen den verschiedenen Schemaebenen, als Systemarchitektur mit den Komponenten eines DBMS sowie als Anwendungsarchitektur mit der Anbindung von Anwendungsprogrammen. Diese verschiedenen Sichtweisen haben wir in diesem Kapitel kurz vorgestellt. Die wichtigsten der damit verbundenen Begriffe sind in Tabelle 2.1 noch einmal zusammengefasst.

## 2.5 Vertiefende Literatur

Architekturen von Datenbankmanagementsystemen werden in diesem Buch nur kurz angerissen. Eine ausführlichere Behandlung, auch von weiteren Architekturprinzipien wie Client-Server-Architekturen, verteilten und föderierten DBMS kann in [SHS05] nachgelesen werden.

Die ANSI-SPARC-Architektur wird in [TK78, Dat86a] und [LD87] erläutert. Dabei ist [TK78] die Originalarbeit, in der Schnittstellen, Sprachen und

<b>Begriff</b>	<b>Informale Bedeutung</b>
Drei-Ebenen-Schemaarchitektur	Aufteilung eines DB-Schemas in drei Ebenen (intern, konzeptuell, extern)
internes Schema	Beschreibung der systemspezifischen Realisierung einer DB
konzeptuelles Schema	implementierungsunabhängige Modellierung der DB
externes Schema	Modellierung der anwendungsspezifischen Sicht auf der DB
Datenunabhängigkeit	Entkopplung der Daten von Anwendungsprogrammen
ANSI-SPARC-Architektur	Normungsvorschlag für Systemarchitektur von DBMS
Fünf-Schichten-Architektur	Systemarchitektur für DBMS zur Beschreibung der Transformationskomponenten

*Tabelle 2.1: Wichtige Begriffe zu Datenbankarchitekturen*

Verantwortlichkeiten definiert werden. Die Fünf-Schichten-Architektur, die auf Ideen von Senko [Sen73] zurückgeht, wird im Buch von Härder [Här87] ausführlich erläutert. Speziell der Schichtenarchitektur von DBMS und deren Entwicklung hinsichtlich neuer Systemkonzepte widmet sich auch der zweiteilige Artikel von Härder [Här05b, Här05a].

Eine ausführlichere Diskussion von Architekturkonzepten für DBMS findet sich im Datenbankhandbuch [LS87] und in den Büchern von Härder und Rahm [HR01] sowie von Saake, Heuer und Sattler [SHS05].

Über relationale Datenbanksysteme gibt es eine ganze Reihe von Spezialbüchern, wie beispielsweise [Cha99] für DB2, [CHRS98, Lon05] für Oracle oder [Pet06] für MS SQL Server. Aktuelle Informationen zu diesen und weiteren Systemen sind aber auch in den Onlinedokumentationen auf den Websites der Hersteller zu finden.

Mehrschichtige Architekturen, J2EE und Applikationsserver werden u.a. in [MB06] behandelt. Einen guten Überblick zu serviceorientierten Architekturen gibt [Mel07].

## 2.6 Übungsaufgaben

**Übung 2-1** Geben Sie analog zu den Beispielen über Weine jetzt auch für Bibliotheksdaten ein konzeptuelles Schema, zwei unterschiedliche externe Sichten und eine interne Realisierung an.

**Übung 2-2** Vergleichen Sie die Drei-Ebenen- und die Fünf-Schichten-Architektur. Welche Schichten lassen sich welchen Ebenen zuordnen?

**Übung 2-3** Informieren Sie sich über frei verfügbare DBMS wie MySQL, Derby und Berkeley DB: Ermitteln Sie ihre Architektur und ihre Systemkomponenten und vergleichen Sie diese mit der Drei-Ebenen-Architektur. Fragestellungen sind etwa:

- Gibt es einen Optimierer?
- Gibt es Dateiorganisationsformen und Zugriffspfade?
- Gibt es eine Datenbanksprache, die in eine herkömmliche Programmiersprache eingebettet werden kann?