

Datenbanksystemkonzepte und Architektur

Die Architektur von DBMS hat sich von den ersten monolithischen Systemen zu modernen DBMS mit einer Client/Server-Systemarchitektur weiterentwickelt. Diese Weiterentwicklung spiegelt die Trends im Computerbereich wider: Zentrale Großrechner werden durch viele verteilte Workstations und Personalcomputer abgelöst, die über Datennetze miteinander verbunden sind. In einer Client/Server-Architektur wird die Systemfunktionalität auf zwei Systemkomponenten verteilt. Die **Client-Komponente** kommt normalerweise auf einer Benutzer-Workstation oder einem Personalcomputer, dem so genannten Client-Rechner, zur Ausführung. In der Regel werden Anwendungsprogramme und Benutzeroberflächen, die auf die Datenbank zugreifen, auf dem Client-Rechner ausgeführt. Folglich führt der Client-Rechner die Benutzerinteraktionen aus und bietet dem Benutzer in vielen Fällen eine formular- oder menübasierte grafische Benutzeroberfläche (GUI) an. Auf der anderen Seite steht die **Server-Komponente**, die auf dem Server-Rechner ausgeführt wird und für die Datenspeicherung, den Datenzugriff, die Ausführung von Anfragen und andere Funktionen zuständig ist.

Client/Server-Architekturen werden in den Kapiteln 17 und 24 behandelt. Zuerst untersuchen wir weitere Grundkonzepte, die uns ein besseres Verständnis der modernen Datenbankarchitekturen vermitteln, die später in diesem Buch beschrieben werden. In diesem Kapitel werden also die Fachbegriffe und Grundkonzepte behandelt, die im Rest dieses Buchs verwendet werden. Wir beginnen in Abschnitt 2.1 mit der Beschreibung von Datenmodellen und der Definition der Konzepte von Schemata und Instanzen, die für das Verständnis von Datenbanksystemen von grundlegender Bedeutung sind. Danach wird in Abschnitt 2.2 eine aus drei Schichten bestehende DBMS-Architektur sowie die Eigenschaft der Datenunabhängigkeit diskutiert. Dies liefert uns einen Überblick aus Benutzersicht, welche Aufgaben ein DBMS eigentlich übernehmen soll. In Abschnitt 2.3 werden die normalerweise von einem DBMS unterstützten Schnittstellen und Sprachen beschrieben. In Abschnitt 2.4 wird die Softwareumgebung eines Datenbanksystems behandelt. Abschnitt 2.5 präsentiert eine Klassifizierung der verschiedenen Arten von DBMS. Abschnitt 2.6 enthält eine Zusammenfassung des Kapitels.

Das in den Abschnitten 2.4 und 2.5 enthaltene Material beschreibt ausführlich Konzepte, die man als Ergänzung zum einführenden Basismaterial betrachten kann.

2.1 Datenmodelle, Schemas und Instanzen

Ein grundlegendes Merkmal des Datenbankansatzes ist, dass er eine gewisse Datenabstraktion bietet, indem Details der Datenspeicherung, von denen die meisten Datenbanknutzer nichts zu wissen brauchen, verborgen werden. Ein **Datenmodell**, d.h. eine Sammlung von Konzepten, die benutzt werden können, um die Struktur einer Datenbank zu beschreiben, bietet die notwendige Grundlage, um diese Abstraktion zu erreichen.¹ Mit *Struktur einer Datenbank* meinen wir die Datentypen, Beziehungen und Einschränkungen, die für die Daten gelten sollen. Die meisten Datenmodelle beinhalten auch eine Reihe von **Basisoperationen** für die Spezifikation von Anfragen und zur Aktualisierung der Datenbank.

Zusätzlich zu den vom Datenmodell bereitgestellten Basisoperationen werden vermehrt Konzepte zur Spezifikation **dynamischer Aspekte** in das Datenmodell einbezogen, um das **Verhalten** einer Anwendung innerhalb des DBMS zu spezifizieren. Dadurch kann der Datenbankdesigner eine Reihe **benutzerdefinierter Operationen** definieren, die für die Datenbankobjekte zulässig sind.² Ein Beispiel einer benutzerdefinierten Operation ist `COMPUTE_GPA`, die auf ein `STUDENT`-Objekt angewandt werden kann. Andererseits werden generische Operationen zum Einfügen, Löschen, Modifizieren oder zum Zugriff von Objekten oft als *Basisoperationen des Datenmodells* zur Verfügung gestellt. Konzepte für die Spezifikation von Verhalten sind bei objektorientierten Datenmodellen (siehe Kapitel 11 und 12) grundsätzlicher Bestandteil, werden aber durch Erweiterung dieser Modelle auch in traditionelle Datenmodelle integriert. Objektrelationale Modelle (siehe Kapitel 13) erweitern z.B. das traditionelle relationale Modell u.a. um solche Konzepte.

2.1.1 Kategorien von Datenmodellen

In der Vergangenheit wurden zahlreiche Datenmodelle vorgeschlagen, die wir entsprechend der Konzepte, die sie zur Beschreibung der Datenstruktur verwenden, kategorisieren können. **Logische** bzw. **konzeptuelle Datenmodelle** bieten Konzepte zur Beschreibung der realen Welt, wie sie die Benutzer wahrnehmen, während **physische Datenmodelle** Konzepte bieten, welche die Details der Datenspeicherung auf dem Rechner beschreiben. Die von physischen Datenmodellen bereitgestellten Konzepte sind allgemein für Rechnerspezialisten und nicht für den typischen Endbenutzer gedacht. Zwischen diesen beiden Extremen liegt eine Klasse der **Darstellungs-** bzw. **Implementierungsdatenmodelle**, die Konzepte bereitstellen, die Endbenutzer verstehen können, die aber nicht zu weit von der Art und Weise entfernt sind, in der Daten auf dem Rechner organisiert werden. Darstellungsdatenmodelle verbergen einige Details der Datenspeicherung, können aber direkt auf einem Rechtersystem implementiert werden.

Konzeptuelle Datenmodelle verwenden Konzepte wie Entitäten, Attribute und Beziehungen. Eine **Entität** (Entity) stellt ein Objekt oder Konzept aus der realen Welt dar, z.B. einen Angestellten oder Projekte, die in der Datenbank beschrieben werden.

-
1. Manchmal wird der Begriff *Modell* auch verwendet, um eine spezifische Datenbankbeschreibung bzw. ein Schema zu bezeichnen, z.B. »das Marketing-Datenmodell«; wir verwenden diese Interpretation nicht.
 2. Die Einbindung von Konzepten, um Verhalten zu beschreiben, spiegelt den Trend wider, bei dem Datenbank- und Softwareentwurfsaktivitäten vermehrt zu einer einzigen Aktivität kombiniert werden. Die Spezifikation von Verhalten ist traditionell mit Softwareentwurf verbunden.

Ein **Attribut** stellt eine Eigenschaft dar, die die Beschreibung einer Entität weiter ausführt, z.B. Name oder Gehalt des Angestellten. Eine **Beziehung** (Relationship) zwischen zwei oder mehr Entitäten stellt einen Zusammenhang zwischen den Entitäten dar, z.B. eine Arbeitsbeziehung zwischen einem Mitarbeiter und einem Projekt. In Kapitel 3 wird das Entity-Relationship-Modell – ein beliebtes konzeptuelles Datenmodell – beschrieben. In Kapitel 4 werden weitere Datenmodellierungskonzepte, z.B. Generalisierung, Spezialisierung und Kategorien, beschrieben.

Datenmodelle zur Darstellung bzw. Implementierung sind solche Modelle, die vorwiegend in DBMS benutzt werden. Sie umfassen das weit verbreitete **relationale Datenmodell** sowie die so genannten Legacy-Datenmodelle – **Netzwerk- und hierarchische Modelle** –, die früher benutzt wurden. In Teil 2 dieses Buchs wird das relationale Datenmodell mit seinen Operationen und Sprachen beschrieben; dieser Teil enthält auch eine Übersicht über zwei relationale DBMS.³ Der SQL-Standard für relationale Datenbanken wird in Kapitel 8 vorgestellt. **Satzbasierte Datenmodelle** stellen ausschließlich Datensatzstrukturen dar.

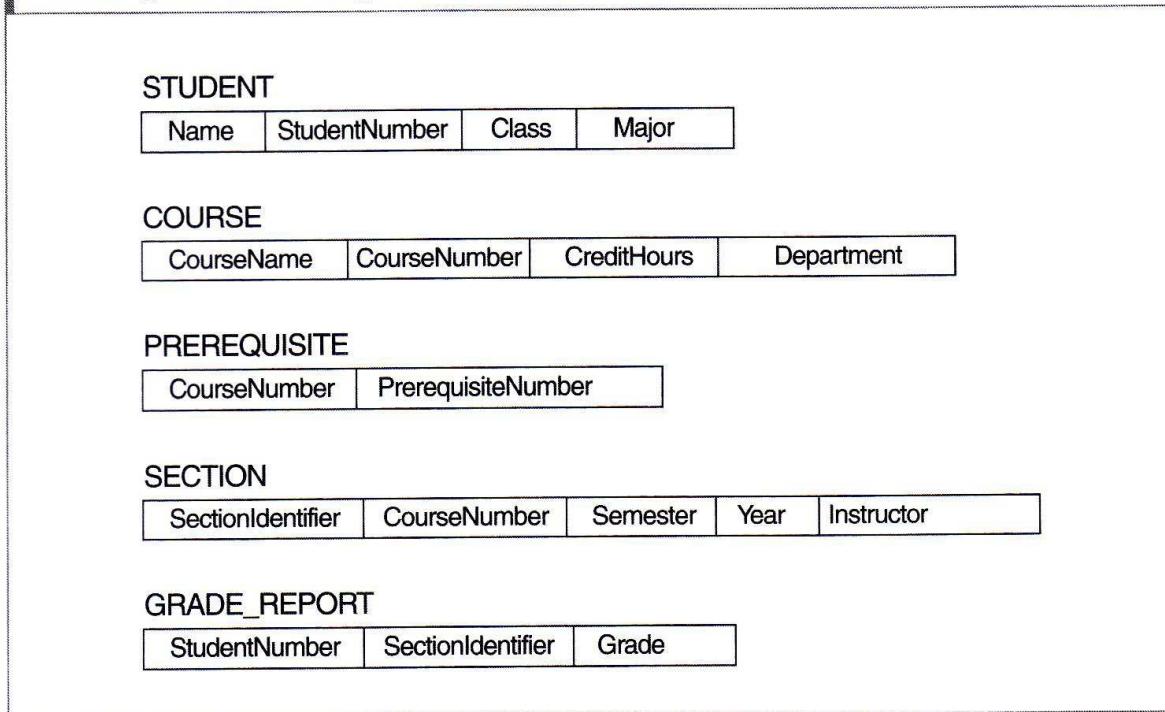
Wir können **Objektdatenmodelle** als neue Familie der Implementierungsdatenmodelle betrachten, die sich näher an konzeptuelle Datenmodelle anlehnen. Wir beschreiben die allgemeinen Merkmale von Objektdatenbanken mit einer Übersicht über zwei Objekt-DBMS in Teil 3. In Kapitel 12 wird der ODMG-Standard beschrieben. Objektdatenmodelle werden auch häufig als höhere konzeptuelle Modelle genutzt, insbesondere im Bereich des Software-Engineering.

Physische Datenmodelle beschreiben, wie die Daten im Rechner durch die Darstellung von Daten, wie Datensatzformate, Datensatzanordnung und Zugriffspfade, gespeichert werden. Ein **Zugriffspfad** ist eine Struktur, durch die sich die Suche nach bestimmten Datensätzen in der Datenbank effizienter gestalten lässt. Physikalische Speichertechniken und Zugriffsstrukturen werden in Kapitel 5 bzw. 6 behandelt.

2.1.2 Schemas, Instanzen und Datenbankzustand

Für jedes Datenmodell ist es wichtig, zwischen der *Beschreibung* der Struktur der Datenbank und dem Inhalt der *Datenbank* zu unterscheiden. Die strukturelle Beschreibung einer Datenbank wird als **Datenbankschema** bezeichnet, das im Verlauf des Datenbankentwurfs spezifiziert wird und sich in der Regel selten ändert.⁴ Die meisten Datenmodelle verwenden festgelegte Konventionen für die grafische Darstellung der Schemas⁵ als Diagramme. Ein grafisch dargestelltes Schema wird als **Schemadiagramm** bezeichnet. Abbildung 2.1 zeigt ein solches Schemadiagramm für die Datenbank aus Abbildung 1.2. Dieses Diagramm beschreibt die Struktur jedes Datensatztyps, jedoch nicht die Instanzen in Form von Datensätzen. Jedes Objekt im Schema, z.B. STUDENT oder COURSE, wird als **Schemakonstrukt** bezeichnet.

-
3. Eine Übersicht über die Netzwerk- und hierarchischen Datenmodelle befindet sich in Anhang C und D. Die vollständigen Kapitel aus der zweiten Auflage dieses Buchs stehen auf unserer Web-Site zur Verfügung.
 4. Schemaänderungen sind normalerweise erforderlich, wenn sich die Anforderungen an die Datenbankanwendung ändern. Neuere Datenbanksysteme beinhalten Operationen für Schemaänderungen; allerdings ist eine Schemaänderung komplizierter als einfache Datenbankaktualisierungen.
 5. In der Datenbankwelt wird eher von *Schemas* als von *Schemata* gesprochen, obwohl *Schemata* die korrekte Pluralform ist.

Abbildung 2.1: Schemadiagramm für die Datenbank von Abbildung 1.2.

Ein Schemadiagramm zeigt nur *bestimmte Aspekte* eines Schemas an, z.B. die Namen von Datensatztypen und Datensatzfeldern und einige Einschränkungsarten. Weitere Aspekte werden im Schemadiagramm nicht spezifiziert. Beispielsweise zeigt Abbildung 2.1 weder den Datentyp jedes Datenfeldes noch die Beziehungen zwischen den verschiedenen Tabellen. Integritätsbedingungen werden in Schemadiagrammen ebenfalls nicht dargestellt. Beispielsweise ist es schwierig, eine Integritätsbedingung der Form »Studenten mit Hauptfach Computerwissenschaften müssen vor dem Ende des Sophomore-Jahres CS1310 belegen« darzustellen.

Die eigentlichen Daten einer Datenbank können sich häufig ändern. Die Datenbank in Abbildung 1.2 ändert sich beispielsweise jedes Mal, wenn ein neuer Student hinzugefügt oder zu einem Studenten eine neue Zensur eingegeben wird. Die in der Datenbank zu einem bestimmten Zeitpunkt gespeicherten Daten werden als **Datenbankzustand** oder **Snapshot** bezeichnet. Jeder Datensatz eines Datenbankzustands stellt ein *aktuelles Vorkommen* (Occurrence) oder eine **Instanz** in der Datenbank dar. In einem Datenbankzustand hat jede Tabelle ihre eigene *aktuelle Menge* von Instanzen. Die Tabelle STUDENT enthält z.B. die einzelnen Studentenentitäten (Datensätze) als Instanzen. Wird der Wert eines Datenfeldes in einem Datensatz geändert oder ein Datensatz eingefügt bzw. gelöscht, geht die Datenbank von einem Zustand in einen anderen über.

Die Unterscheidung zwischen Datenbankschema und Datenbankzustand ist sehr wichtig. Mit der **Definition** einer Datenbank spezifizieren wir ihr Schema. Zu diesem Zeitpunkt befindet sich die betreffende Datenbank im *leeren Zustand* ohne Daten. Dies stellt den *Anfangszustand* der Datenbank dar. Mit jeder Veränderung der Datenbank erhalten wir einen anderen Datenbankzustand. Zu einem beliebigen Zeitpunkt besitzt die Datenbank einen *aktuellen Zustand*.⁶ Das DBMS ist dafür verantwortlich, dass jeder

6. Der aktuelle Zustand wird auch als *aktueller Schnappschuss* der Datenbank bezeichnet.

Zustand der Datenbank ein **gültiger** ist, d.h., der Zustand erfüllt die im Schema definierte Struktur sowie die auf ihr definierten Integritätsbedingungen. Folglich ist die Definition eines korrekten Schemas für das DBMS extrem wichtig und das Schema muss mit höchster Sorgfalt entworfen werden. Das DBMS speichert die Beschreibungen der Schemas und Integritätsbedingungen – beides wird auch als **Metadaten** bezeichnet – im DBMS-Katalog, so dass die DBMS bei Bedarf auf das Schema zugreifen kann. Das Schema wird manchmal auch als **Intension** und der Datenbankzustand als **Extension** bezeichnet.

Obwohl sich das Schema, wie bereits erwähnt, in vielen Fällen selten ändert, muss es dennoch hin und wieder neuen Anwendungsanforderungen angepasst werden. Entscheidet man beispielsweise, dass jeder Datensatz in einer Datei um ein bestimmtes Datenfeld erweitert werden soll, z.B. durch Hinzufügen von DateOfBirth (Geburtsdatum) zum Schema STUDENT in Abbildung 2.1, dann muss das Schema geändert werden. Dieses Fortschreiben des Schemas wird als **Schema-Evolution** bezeichnet. Die meisten modernen DBMS beinhalten Operationen für die Schema-Evolution, die gleichzeitig mit anderen Operationen auf der Datenbank ausgeführt werden können.

2.2 DBMS-Architektur und Datenunabhängigkeit

Die wichtigen Merkmale des Datenbankansatzes, die in Abschnitt 1.3 aufgeführt wurden, sind: (1) Isolierung von Programmen und Daten (Programm/Daten- und Programm/Operationen-Unabhängigkeit), (2) Unterstützung mehrerer Benutzersichten (Views) und (3) Verwendung eines Katalogs zum Speichern der Datenbankbeschreibung (Schema). In diesem Abschnitt beschreiben wir eine Architektur für Datenbanksysteme, die als **Drei-Schichten-Architektur**⁷ bezeichnet wird. Anschließend wird das Konzept der Datenunabhängigkeit behandelt.

2.2.1 Die Drei-Schichten-Architektur

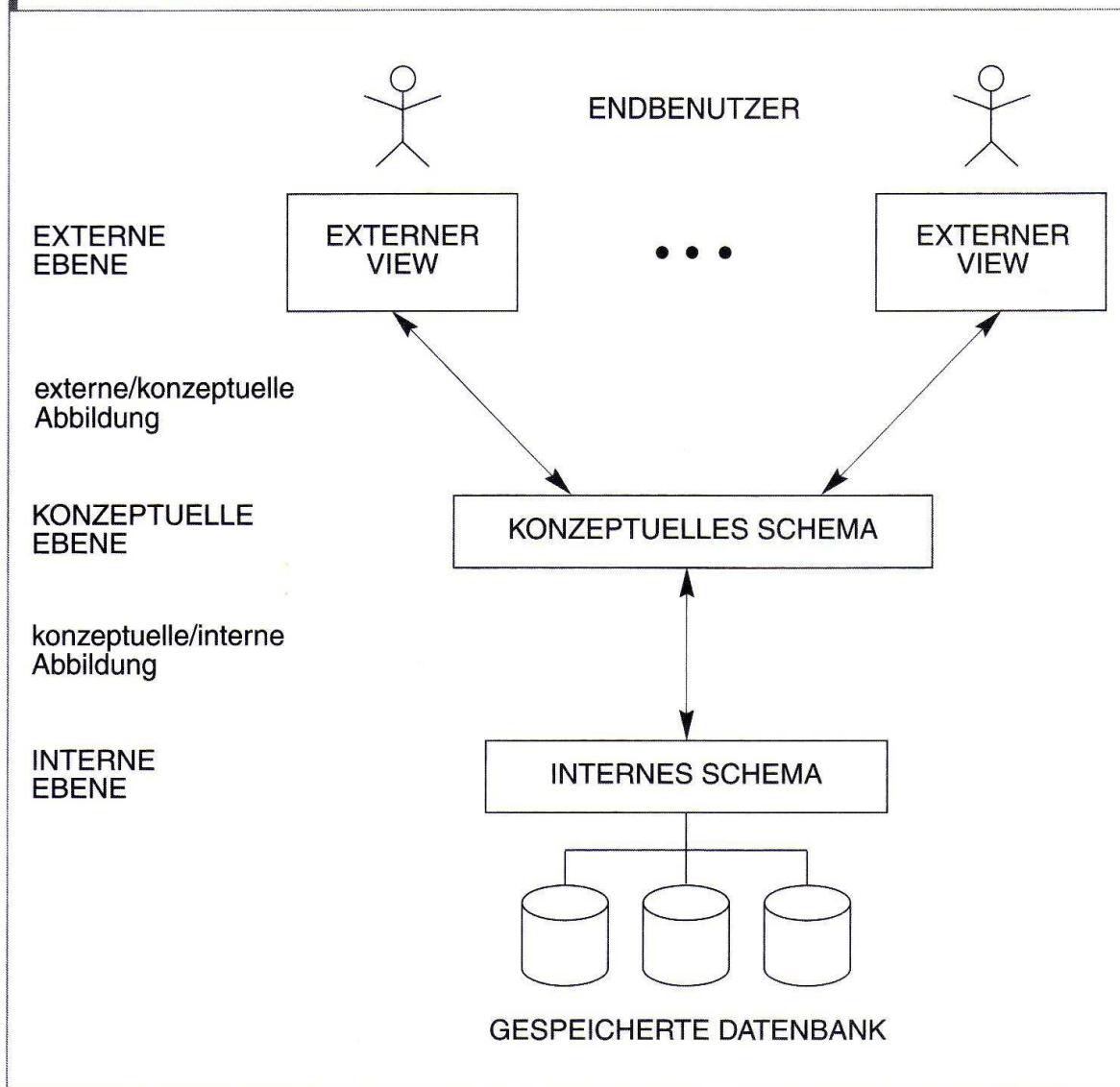
Ziel der in Abbildung 2.2 dargestellten Drei-Schichten-Architektur ist es, die Benutzeranwendungen und die Details der Speicherung (physische Eigenschaften) voneinander zu trennen. Bei dieser Architektur können Schemas auf den folgenden drei Ebenen definiert werden:

1. Auf der **internen Ebene** wird ein **internes Schema** definiert, das die physikalischen Speicherstrukturen der Datenbank beschreibt. Das interne Schema verwendet ein physisches Datenmodell und beschreibt die Details der Datenspeicherung und Zugriffspfade für die Datenbank.
2. Auf der **konzeptuellen Ebene** wird ein **konzeptuelles Schema** definiert, das die Struktur der gesamten Datenbank für alle Nutzer der Datenbank beschreibt. Das konzeptuelle Schema verbirgt die Details der physischen Speicherstrukturen und konzentriert sich auf die Beschreibung von Entitäten, Datentypen, Beziehungen, Benutzeroperationen und Einschränkungen. Auf dieser Ebene kann ein von den Speicherstrukturen abstrahierendes (logisches) Datenmodell benutzt werden.

7. Eine andere Bezeichnung ist die ANSI/SPARC-Architektur nach der Arbeitsgruppe, die sie vorgeschlagen hat (Tsichritzis und Klug, 1978).

3. Die **externe** oder **View-Ebene** beinhaltet eine Reihe **externer Schemas** oder **Benutzersichten (Views)**. Jedes externe Schema beschreibt den Teil der Datenbank, an dem eine bestimmte Benutzergruppe interessiert ist, und verbirgt die übrigen Daten der Datenbank vor dieser Benutzergruppe. Auf dieser Ebene kann ebenfalls ein von den Speicherstrukturen abstrahierendes (logisches) Datenmodell benutzt werden.

Abbildung 2.2: Darstellung der Drei-Schichten-Architektur.



Die Drei-Schichten-Architektur stellt für den Benutzer einen wichtigen Ansatz dar, die unterschiedlichen Schemaebenen in einem Datenbanksystem voneinander zu unterscheiden. Die meisten DBMS trennen die drei Ebenen nicht vollständig, sie unterstützen die Drei-Schichten-Architektur nur bis zu einem gewissen Grad. Bei einigen DBMS sind möglicherweise Details der physischen Ebene im konzeptuellen Schema enthalten. Bei den meisten DBMS, die Benutzersichten unterstützen, werden externe Schemas in dem Datenmodell spezifiziert, das die Strukturen der konzeptuellen Ebene beschreibt. Bei einigen DBMS können unterschiedliche Datenmodelle auf der konzeptuellen und externen Ebene benutzt werden.

Man beachte, dass die drei Schemas lediglich *Beschreibungen* von Daten sind. Die einzigen Daten, die *tatsächlich* existieren, befinden sich auf der physischen Ebene. In einem auf der Drei-Schichten-Architektur basierenden DBMS bezieht sich jede Benutzergruppe nur auf ihr eigenes externes Schema. Folglich muss das DBMS eine Operation auf dem externen Schema in eine Operation auf dem konzeptuellen Schema und anschließend in Operationen auf dem internen Schema für die Verarbeitung der gespeicherten Daten in der Datenbank transformieren. Handelt es sich bei der Operation um einen lesenden Datenbankzugriff, müssen die aus der gespeicherten Datenbank extrahierten Daten auf der externen Sicht des Benutzers möglicherweise umformatiert werden. Der Vorgang der Umwandlung von Operationen und Ergebnissen zwischen den jeweiligen Ebenen wird als **Abbildung (Mapping)** bezeichnet. Diese Abbildungen können zeitaufwendig sein, so dass einige DBMS, insbesondere solche, die für die Unterstützung kleiner Datenbanken ausgelegt wurden, keine externen Sichten unterstützen. Doch sogar in solchen Systemen ist ein gewisser Umfang an Transformationen erforderlich, um Operationen zwischen der konzeptuellen und internen Ebene umzuwandeln.

2.2.2 Datenunabhängigkeit

Die Drei-Schichten-Architektur kann zur Erklärung des Konzepts der **Datenunabhängigkeit** verwendet werden. Der Definition zufolge beschreibt Datenunabhängigkeit die Möglichkeit, das Schema auf einer Ebene ändern, ohne das Schema der nächsthöheren Ebene ändern zu müssen. Datenunabhängigkeit kann auf zwei Arten definiert werden:

1. **Logische Datenunabhängigkeit:** Dies ist die Fähigkeit, das konzeptuelle Schema zu ändern, ohne externe Schemas oder Anwendungsprogramme ändern zu müssen. Wir können beispielsweise das konzeptuelle Schema ändern, um die Datenbank (durch Hinzufügen eines Datensatztyps oder Datenfeldes) zu erweitern oder aber (durch Entfernen eines Datensatztyps oder Datenfeldes) zu reduzieren. Im zweiten Fall sollte sich die Änderung nicht auf solche externe Schemas auswirken, die nur die nicht veränderten Teile des Schemas referenzieren. Beispielsweise sollte sich die Änderung der Datei GRADE_REPORT in Abbildung 1.2 in diejenige aus Abbildung 1.5(a) nicht auf das externe Schema in Abbildung 1.4(a) auswirken. In einem DBMS, das logische Datenunabhängigkeit unterstützt, müssen lediglich die Definitionen von Sichten und die Transformationen (Mappings) geändert werden. Anwendungsprogramme, die auf die Teile des externen Schemas referenzieren, müssen nach einer logischen Reorganisation des konzeptuellen Schemas nach wie vor funktionsfähig sein. Des Weiteren können Änderungen von Integritätsbedingungen im konzeptuellen Schema durchgeführt werden, ohne dass externe Schemas oder Anwendungsprogramme davon betroffen werden.
2. **Physische Datenunabhängigkeit:** Dieser Begriff beschreibt die Eigenschaft, ein internes Schema ändern zu können, ohne die konzeptuellen (oder externen) Schemas ändern zu müssen. Möglicherweise sind Änderungen des internen Schemas nötig, weil einige physische Dateien umorganisiert wurden, z.B. durch Erstellen zusätzlicher Zugriffsstrukturen, um den Zugriff auf die Daten zu beschleunigen. Wenn die gleichen Daten wie zuvor in der Datenbank verbleiben, muss das konzeptuelle Schema in der Regel nicht verändert werden. Die Bereitstellung eines Zugriffspfads für die Verbesserung der Suche von SECTION-Datensätzen (Abbil-

dung 1.2) über »Semester« und »Year« sollte z.B. keine Änderung einer Anfrage wie »Liste alle im Herbst 1998 angebotenen Sections auf« zur Folge haben, auch wenn die Anfrage vom DBMS bei der Verwendung der neuen Zugriffspfade effizienter ausgeführt werden kann.

Wenn wir es mit einem aus mehreren Ebenen bestehenden DBMS zu tun haben, muss sein Katalog um Informationen über die Art, wie Anforderungen und Daten zwischen den verschiedenen Ebenen abzubilden sind, erweitert werden. Das DBMS muss in der Lage sein, diese Transformationen durch Zugriff auf die Abbildungsinformationen im Katalog sicherzustellen. Datenunabhängigkeit wird dann erreicht, wenn eine Schemaänderung auf einer Ebene keine Auswirkungen auf das Schema der nächsthöheren Ebene hat; was sich ändert, ist die Abbildung zwischen den beiden Ebenen. Folglich müssen Anwendungsprogramme, die auf das höherschichtige Schema zugreifen, nicht geändert werden.

Die Drei-Schichten-Architektur unterstützt sowohl physische als auch logische Datenunabhängigkeit. Allerdings bedeuten beide Transformationen während der Übersetzung und Ausführung einer Anfrage oder eines Programms zusätzlichen Aufwand, was zu Ineffizienzen im DBMS führen kann. Aus diesem Grund wird die volle Drei-Schichten-Architektur nicht in allen DBMS unterstützt.

2.3 Datenbanksprachen und -schnittstellen

In Abschnitt 1.4 wurde die Vielfalt der von einem DBMS unterstützten Benutzer vorgestellt. Das DBMS muss für jede Nutzerkategorie entsprechende Sprachenmöglichkeiten und Schnittstellen bereitstellen. In diesem Abschnitt werden die von einem DBMS bereitgestellten Sprachenmöglichkeiten und Schnittstellen sowie die Benutzerkategorien beschrieben, auf die jede einzelne dieser Schnittstellen abzielt.

2.3.1 Datenbanksprachen

Nachdem der Entwurf einer Datenbank fertig gestellt und ein DBMS ausgewählt wurde, um die Datenbank zu implementieren, folgt als erster Schritt die Definition konzeptueller und interner Schemas für die Datenbank und eventuelle Abbildungen zwischen den beiden. In vielen DBMS, in denen keine strikte Trennung der Ebenen erfolgt, benutzen der DBA und der Datenbankdesigner eine so genannte **Datendefinitionssprache (Data Definition Language, DDL)**, um beide Schemas zu definieren. In diesem Fall verfügt das DBMS über einen DDL-Compiler für die Abarbeitung von DDL-Anweisungen, um Beschreibungen der Schemaobjekte zu identifizieren und diese im DBMS-Katalog zu speichern.

In DBMS, in denen klar zwischen der konzeptuellen und der internen Ebene unterschieden wird, kommt die DDL nur für die Spezifikation des konzeptuellen Schemas zum Einsatz. Für die Spezifikation des internen Schemas wird eine **Speicherdefinitionssprache (Storage Definition Language, SDL)** benutzt. Die Mappings zwischen den beiden Schemas können wahlweise in jeder der beiden Sprachen spezifiziert werden. Für eine echte Drei-Schichten-Architektur benötigt man eine dritte, die **View-Definitionssprache (View Definition Language, VDL)**, um Benutzersichten und deren Abbildungen auf das konzeptuelle Schema zu definieren. In den meisten DBMS werden aber die konzeptuellen und externen Schemas mit Hilfe der DDL definiert.

Nachdem die Datenbankschemas übersetzt wurden und die Datenbank mit Daten gefüllt ist, muss den Benutzern die Möglichkeit des Zugriffs auf die Datenbank gegeben werden. Typische Zugriffe sind dabei das Lesen (Retrieval), Einfügen, Löschen und Ändern von Daten. Das DBMS bietet für diese Zwecke eine **DatenmanipulationsSprache (Data Manipulation Language, DML)** an.

In den heutigen DBMS gelten die oben genannten Sprachen meist nicht als *separate* Sprachen; vielmehr wird eine integrierte Sprache zur Verfügung gestellt, die Sprachkonstrukte für die konzeptuelle Schemadefinition, die View-Definition und den Datenzugriff beinhaltet. Die Definition von Speicherstrukturen wird normalerweise separat vorgenommen, um dieses dynamisch für die Feinabstimmung (Tuning) zur Leistungverbesserung des Datenbanksystems zu definieren. Das Tuning wird vorwiegend von DBAs vorgenommen. Ein typisches Beispiel einer umfassenden Datenbanksprache ist die relationale Datenbanksprache SQL (siehe Kapitel 8), die DDL, VDL und DML auf sich vereint sowie Anweisungen für die Spezifikation von Integritätsbedingungen und Schemaevolution umfasst. Die SDL war ein Teil früherer SQL-Versio-nen, wurde aber inzwischen aus der Sprache entfernt.

Bei der DML wird zwischen zwei Arten unterschieden: Eine **nicht prozedurale** DML kann eigenständig benutzt werden, um komplexe Datenbankoperationen präzise und kompakt zu spezifizieren. Viele DBMS ermöglichen die Eingabe von DML-Anweisungen entweder interaktiv an einem Terminal (oder Monitor) oder als Einbettung in eine allgemeine Programmiersprache. Im zweiten Fall müssen DML-Anweisungen im Programm erkannt werden, so dass sie von einem Vorübersetzer (Pre-Compiler) extrahiert und vom DBMS verarbeitet werden können. Eine **prozedurale** DML *muss* in eine allgemeine Programmiersprache eingebettet werden. Dieser Art der DML ruft normalerweise einzelne Datensätze oder Objekte aus der Datenbank ab und verarbeitet sie jeweils getrennt. Folglich müssen hier Programmiersprachenkonstrukte wie beispielsweise Schleifen verwendet werden, um jeden einzelnen Datensatz abzurufen und zu verarbeiten. Aufgrund dieser Eigenschaft wird diese Art der DML-Sprache auch als **satzbasierte DML** bezeichnet. Hochsprachliche DMLs wie beispielsweise SQL können viele Datensätze mit einer einzigen DML-Anweisung verarbeiten; daher werden sie auch als **mengenorientierte DMLs** bezeichnet. Eine Anfrage in einer nicht prozeduralen DML spezifiziert, auf *welche* Daten zuzugreifen ist, und nicht, *wie* auf diese Daten zugegriffen wird. Aus diesem Grund beschreibt man solche Sprachen als **deklarativ**.

Werden DML-Befehle in eine allgemeine Programmiersprache eingebettet, wird diese Sprache als **Wirtssprache (host language)** und die DML als **Daten-Subsprache** bezeichnet.⁸ Wird demgegenüber eine DML der konzeptuellen Ebene einzeln und interaktiv verwendet, wird sie als **Anfragesprache (query language)** bezeichnet. Im Allgemeinen können Zugriffs- und Veränderungsoperationen in einer hochsprachlichen DML als Teil einer Anfragesprache interaktiv benutzt werden.⁹

Gelegentliche Endbenutzer wählen in der Regel eine deklarative Anfragesprache zur Beschreibung ihrer Anfragen, während Programmierer diese eingebettet in eine Programmiersprache verwenden. Für naive und parametrische Benutzer werden

-
8. In Objektdatenbanken bilden Host- und Daten-Subsprache normalerweise eine integrierte Sprache, z.B. C++, mit einigen Erweiterungen für die Unterstützung der Datenbankfunktionalität. Einige relationale Systeme bieten auch integrierte Sprachen, z.B. PL/SQL von Oracle.
 9. Der Bedeutung des Worts zufolge sollte *Anfrage* wirklich nur für den lesenden Zugriff und nicht für Aktualisierungen verwendet werden.

meist **benutzerfreundliche Oberflächen** für die Interaktion mit der Datenbank bereitgestellt. Diese Oberflächen können natürlich auch von gelegentlichen Benutzern verwendet werden, die sich nicht mit dem Erlernen der Details einer deklarativen Anfragesprache befassen wollen. Diese Arten von Oberflächen sind Thema des nächsten Abschnitts.

2.3.2 Benutzeroberflächen für Datenbanksysteme

Ein DBMS kann folgende benutzerfreundliche Oberflächen bereitstellen:

Menügesteuerte Oberflächen zum Durchsuchen der Datenbank Diese Oberflächen präsentieren dem Benutzer Listen mit Optionen, die allgemein als **Menüs** bezeichnet werden. Sie führen den Benutzer durch die Formulierung einer Anfrage. Durch Menüs muss sich der Benutzer keine Befehle oder die Syntax einer Anfragesprache merken. Vielmehr wird die Anfrage schrittweise durch Auswahl von Optionen aus einem Menü, das vom System angezeigt wird, zusammengestellt. Pull-down-Menüs sind seit einigen Jahren eine beliebte Technik für fensterbasierte Benutzeroberflächen. Sie werden meist in Oberflächen verwendet, die es einem Benutzer ermöglichen, den Inhalt einer Datenbank – meist unstrukturiert – zu durchsuchen.

Formularbasierte Oberflächen Diese Oberflächen zeigen jedem Benutzer ein **Formular** an. Der Benutzer kann ein solches Formular ausfüllen, um beispielsweise neue Daten einzufügen. Alternativ werden bestimmte Formulare dazu verwendet, entsprechend der Einträge die Daten aus der Datenbank mit Hilfe des DBMS zu selektieren und anzuzeigen. Formulare werden normalerweise für naive Benutzer als Oberflächen für geplante Transaktionen entworfen und programmiert. Viele DBMS verfügen über **Sprachen zur Definition von Formularen**, d.h. spezielle Sprachen, die den Programmierer bei der Definition solcher Formulare unterstützen. Einige Systeme verfügen über Dienstprogramme (Utilities), die ein Formular definieren, indem sie den Endbenutzer interaktiv ein Musterformular am Bildschirm zusammenstellen lassen.

Grafische Benutzeroberflächen Eine grafische Benutzeroberfläche (GUI) zeigt dem Benutzer normalerweise ein Datenbankschema in Diagrammform an. Der Benutzer kann dann eine Anfrage spezifizieren, indem er das Diagramm verändert. In vielen Fällen verwenden GUIs sowohl Menüs als auch Formulare.

Natürlichsprachliche Oberflächen Diese Oberflächen akzeptieren Anfragen im Klar- text, z.B. in englischer oder einer anderen natürlichen Sprache, und versuchen, sie zu »verstehen«. Eine solche Oberfläche hat meist ihr eigenes »Schema«, das dem konzeptuellen Schema der Datenbank ähnelt. Das Spracherkennungssystem versucht, die eingegebene Anfrage in natürlicher Sprache unter Einbeziehung des Datenbankschemas und eines Wörterbuchs zu interpretieren. Ist die Interpretation erfolgreich, erzeugt das System eine hochsprachliche (deklarative) Anfrage, die der Eingabeanfrage von der Bedeutung her entspricht, und lässt diese durch das DBMS verarbeiten; andernfalls wird ein Dialog gestartet, in dem der Benutzer seine Anfrage erklären muss.

Oberflächen für parametrische Benutzer Parametrische Benutzer, wie beispielsweise Mitarbeiter an Bankschaltern, müssen oft eine kleine Menge von Operationen wiederholt ausführen. Systemanalytiker und Programmierer entwerfen und imple-

mentieren eine spezielle Oberfläche für eine vorher wohl definierte Klasse von Benutzern. Normalerweise wird dabei eine kleine Menge abgekürzter Befehle mit dem Ziel implementiert, die Anzahl der für jede Anfrage erforderlichen Tastenanschläge zu reduzieren. Beispielsweise können Funktionstasten an einem Terminal so programmiert werden, dass sie die verschiedenen Befehle einleiten. Dadurch kann der parametrische Benutzer mit einer geringen Anzahl von Tastenanschlägen auskommen.

Oberflächen für den DBA Die meisten Datenbanksysteme enthalten privilegierte Befehle, die nur vom DBA und seinen Mitarbeitern ausgeführt werden können. Dazu zählen beispielsweise Befehle für das Erstellen von Konten, das Setzen von Systemparametern, das Gewähren von Zugriffsrechten, das Ändern eines Schemas und das Reorganisieren der Speicherstrukturen in einer Datenbank.

2.4 Die Datenbanksystemumgebung

Ein DBMS ist ein komplexes Softwaresystem. In diesem Abschnitt behandeln wir die Softwarekomponenten, aus denen sich ein DBMS zusammensetzt, und die Softwarekomponenten des Betriebssystems, mit dem das DBMS interagiert.

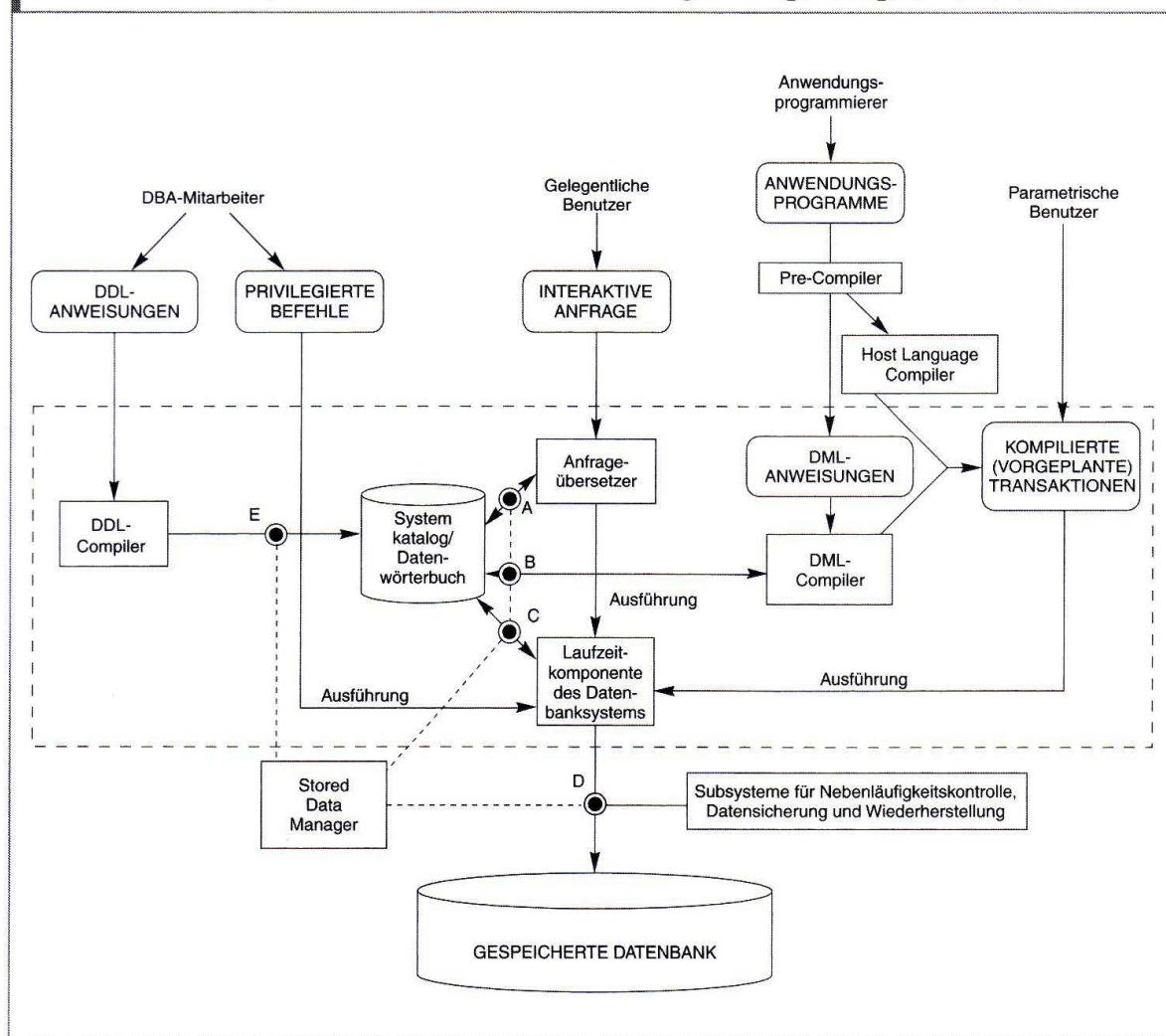
2.4.1 DBMS-Komponentenmodule

Die typischen DBMS-Komponenten sind in Abbildung 2.3 vereinfacht dargestellt. Die Datenbank und die Katalogdaten werden auf Platte, auch Sekundärspeicher genannt, gespeichert. Der Zugriff auf die Platte wird primär vom **Betriebssystem** (Operating System) kontrolliert, das die Lese- und Schreiboperationen auf der Platte ausführt. Der **Storage Manager**, eine Komponente des DBMS, kontrolliert den Zugriff auf die Datenbank und auf die Katalogdaten, die auf dem Sekundärspeicher abgelegt sind. Die gepunkteten Linien und die mit A, B, C, D und E gekennzeichneten Kreise in Abbildung 2.3 stellen Zugriffe auf die Daten unter der Kontrolle des Storage Managers dar. Der Storage Manager kann für die Durchführung des Datentransfers zwischen der Platte und dem Hauptspeicher grundlegende Betriebssystemdienste nutzen, kontrolliert dabei aber weitere Aspekte des Datentransfers, beispielsweise den Zugriff auf Datenpuffer im Hauptspeicher. Befinden sich die Daten im Datenpuffer, können sie von anderen DBMS-Komponenten sowie von Anwendungsprogrammen verarbeitet werden.

Der **DDL-Compiler** verarbeitet Schemadefinitionen, die in der DDL spezifiziert sind, und speichert Beschreibungen des Schemas (Metadaten) im DBMS-Katalog ab. Der Katalog beinhaltet Informationen wie Tabellennamen, Datenattribute, Details zur Speicherung einer jeden Tabelle, Mapping-Informationen zwischen Schemas, Integritätsbedingungen und verschiedene weitere Informationen, die von den DBMS-Komponenten benötigt werden. Bei Bedarf greifen die DBMS-Komponenten auf die benötigten Informationen im Katalog zu.

Die **Laufzeitkomponente des Datenbanksystems** (Runtime Database Processor) führt die Zugriffe auf die Daten zur Ausführungszeit (Laufzeit) einer Anfrage oder Veränderungsoperation aus. Der Zugriff auf die Platte erfolgt durch den Storage Manager. Der **Anfrageübersetzer (Query Compiler)** ist für die Abarbeitung interaktiv eingegebener Anfragen zuständig. Er analysiert und übersetzt bzw. interpretiert eine Anfrage durch die Erzeugung eines Datenzugriffsprogramms einschließlich des Aufrufs der Laufzeitkomponente, die die erzeugten Codes zur Ausführung bringt.

Abbildung 2.3: Typische Komponenten eines DBMS; die gepunkteten Linien stellen Zugriffe dar, die unter der Kontrolle des Storage Manager ausgeführt werden.



Der **Vorübersetzer (Pre-Compiler)** extrahiert DML-Befehle aus einem Anwendungsprogramm, das in einer Wirtssprache (Host Language) geschrieben ist. Diese Befehle werden an den **DML-Compiler** weitergegeben, um sie in Objektcode mit Datenbankzugriffen zu übersetzen. Der Objektcode für die DML-Befehle und der Rest des Programms sind verknüpft und bilden damit eine geplante Transaktion, deren ausführbarer Code Aufrufe der Laufzeitzeitkomponente des DBMS beinhaltet.

Abbildung 2.3 stellt die wichtigsten Komponenten eines typischen DBMS dar. Das DBMS interagiert mit dem Betriebssystem, wenn Plattenzugriffe zum Zugriff auf die Datenbank oder Katalogdaten erforderlich sind. Wird das Rechnersystem von mehreren Benutzern genutzt, regelt das Betriebssystem die Anforderungen für Plattenzugriffe und die DBMS-Verarbeitung sowie weitere Prozesse. Das DBMS verfügt auch über eine Schnittstelle zu Übersetzern (Compiler) für allgemeine Wirtssprachen. Darüber hinaus können benutzerfreundliche Oberflächen für das DBMS bereitgestellt werden, um beispielsweise die verschiedenen Benutzertypen in Abbildung 2.3 bei ihren Anfragen zu unterstützen.

2.4.2 Datenbank-Dienstprogramme

Zusätzlich zu den im vorigen Abschnitt beschriebenen Softwarekomponenten verfügen die meisten DBMS auch über **Datenbank-Dienstprogramme**, die den DBA bei der Verwaltung des Datenbanksystems unterstützen. Die üblichen Dienstprogramme umfassen dabei folgende Funktionen:

1. *Laden einer Datenbank*: Dieses Dienstprogramm wird benutzt, um Datendateien, z.B. Text- oder sequentielle Dateien, in die Datenbank zu laden. Normalerweise werden die Formate der Datendatei (Quelle) und der gewünschten Datenbankdateistruktur (Ziel) im Dienstprogramm spezifiziert, das dann die Daten automatisch umformatiert und in der Datenbank speichert. Mit der Verbreitung von DBMS ist die Überführung von Daten von einem DBMS in ein anderes in vielen Unternehmen heute ein alltäglicher Vorgang. Einige Hersteller bieten Produkte an, die auf der Grundlage der bestehenden Datenbankspeicherbeschreibungen für Datenquellen und Datensenken die entsprechenden Ladeprogramme erzeugen. Solche Werkzeuge werden auch als **Konvertierungswerkzeuge** bezeichnet.
2. *Sichern einer Datenbank (Backup)*: Ein solches Dienstprogramm erzeugt eine Sicherungskopie der Datenbank, meist durch Schreiben der gesamten Datenbank auf Band. Die Sicherungskopie kann dann benutzt werden, um die Datenbank im Fall einer Katastrophe wiederherzustellen. Meist werden auch inkrementelle Datensicherungen durchgeführt, wobei nur Änderungen seit der letzten Datensicherung aufgezeichnet werden. Inkrementelle Datensicherungen sind komplexer, sparen aber Speicherplatz.
3. *Dateireorganisation*: Dieses Dienstprogramm dient der Reorganisation eines Teils einer Datenbank in eine andere Dateiorganisation, um die Leistung zu verbessern.
4. *Leistungsüberwachung*: Dieses Dienstprogramm dient dem DBA dazu, die Datenbanknutzung zu überwachen. Es liefert dem DBA verschiedene Statistiken, die dieser als Grundlage für Entscheidungen nutzt, ob Dateien zur Leistungsverbesserung zu reorganisieren sind.

Darüber hinaus gibt es weitere Dienstprogramme, z.B. zum Sortieren von Dateien, für die Datenkompression, zum Überwachen der Benutzerzugriffe und zum Durchführen weiterer administrativer Funktionen.

2.4.3 Werkzeuge, Anwendungsumgebungen und Kommunikationsdienste

Für die Designer, Benutzer und DBAs einer Datenbank stehen oftmals weitere Werkzeuge zur Verfügung. Beispielsweise werden **CASE-Werkzeuge**¹⁰ in der Entwurfsphase eines Datenbanksystems benutzt. Ein weiteres nützliches Werkzeug, insbesondere für Großunternehmen, ist ein erweitertes **Datenwörterbuch (Data Dictionary oder Data Repository)**. Zusätzlich zur Speicherung von Informationen über Schemas und Integritätsbedingungen umfasst das Data Dictionary weitere Informationen, z.B. Entwurfsentscheidungen, Nutzungsstandards, Beschreibungen von Anwendungsprogrammen und Benutzerinformationen. Ein solches System wird

10. CASE ist zwar die Abkürzung für »Computer Aided Software Engineering«, viele CASE-Werkzeuge werden aber vorwiegend für den Datenbankentwurf benutzt.

auch als **Information-Repository** bezeichnet. Abgesehen vom DBA, der jederzeit bei Bedarf darauf zugreifen kann, haben Benutzer *direkten* Zugriff auf diese Informationen. Ein Datenwörterbuch ist mit dem DBMS-Katalog vergleichbar, beinhaltet aber umfangreichere Informationen. Der Zugriff auf das Data Dictionary erfolgt vorwiegend durch die Benutzer statt durch die DBMS-Software.

In letzter Zeit werden auch **Anwendungsentwicklungsumgebungen** wie das PowerBuilder-System immer beliebter. Diese Systeme bieten eine Umgebung für die Entwicklung von Datenbankanwendungen und umfassen Dienstprogramme, die viele DBMS-Aufgaben vereinfachen, z.B. den Datenbankentwurf, die GUI-Entwicklung, die Formulierung von Anfragen und Veränderungsoperationen sowie die Anwendungsentwicklung.

Ferner benötigt das DBMS eine Schnittstelle zu **Kommunikationssoftware**, die dazu dient, Benutzern den Zugriff auf die Datenbank mit Hilfe von Computer-Terminals, Workstations oder PCs zu ermöglichen. Die Rechner werden dabei untereinander durch Hardware zur Datenkommunikation, wie beispielsweise Telefonleitungen, Wide-Area-Netzwerke, lokale Netzwerke oder Satellitenkommunikationsgeräte verbunden. Viele kommerzielle Datenbanksysteme beinhalten Kommunikationskomponenten, die mit dem DBMS eingesetzt werden können. Ein DBMS mit integrierter Datenkommunikation wird als **DB/DC-System** bezeichnet. Darüber hinaus werden einige dezentrale DBMS physisch auf mehrere Maschinen verteilt. In diesem Fall müssen die Maschinen über Datennetze verbunden werden, wofür meist **lokale Netzwerke (LANs)** zum Einsatz kommen; es eignen sich aber auch andere Arten von Datennetzen.

2.5 Klassifikation von Datenbankmanagementsystemen

Normalerweise werden DBMS anhand verschiedener Kriterien klassifiziert. Das erste Kriterium ist das dem DBMS zugrunde liegende **Datenmodell**. Die beiden in vielen modernen kommerziellen DBMS verwendeten Datenmodelle sind das **relationale Datenmodell** und das **Objektdatenmodell**. Viele ältere und heute überholte Anwendungen laufen immer noch auf Datenbanksystemen, die auf **hierarchischen** und **Netzwerkdatenmodellen** basieren. Die relationalen DBMS werden laufend weiterentwickelt, insbesondere durch Einbindung vieler Konzepte, die in Objektdatenbanksystemen entwickelt wurden. Dies hat zu einer neuen DBMS-Klasse der **objekt-relationalen DBMS** geführt. Folglich können wir DBMS auf der Grundlage des Datenmodells kategorisieren: relational, objektorientiert, objektrelational, hierarchisch, netzwerkbasiert und andere.

Das zweite Kriterium für die Klassifizierung von DBMS ist die vom System unterstützte **Anzahl an Nutzern, die gleichzeitig auf die Datenbank zugreifen können**. **Einbenutzersysteme (Single User System)** unterstützen nur jeweils einen Benutzer und werden vorwiegend auf PCs eingesetzt. **Mehrbenutzersysteme (Multi User Systems)** unterstützen mehrere Benutzer gleichzeitig und stellen heute die Mehrheit der DBMS dar.

Ein drittes Kriterium ist die **Anzahl der Rechner**, auf die sich die Datenbank verteilt. Ein DBMS gilt als **zentral**, wenn die Daten auf einem einzigen Rechner gespeichert sind. Ein zentrales DBMS kann mehrere Benutzer unterstützen, das DBMS und die Datenbank residieren aber vollständig auf einem Rechner. Bei einem **dezentralen**

DBMS (Distributed DBMS, DDBMS) kann die Datenbank und das DBMS auf viele Rechner (sogenannte Rechnerknoten) verteilt sein, die über ein Rechnernetzwerk miteinander verbunden sind. **Homogene DDBMS** nutzen das gleiche DBMS auf allen Rechnerknoten. Ein neuerer Trend ist die Entwicklung von Software für den Zugriff auf mehrere autonome Datenbanken, auf die durch unterschiedliche, d.h. **heterogene DBMS** zugegriffen wird. Diese Heterogenität führt zu einem sogenannten **föderativen DBMS** (oder **Multidatenbanksystem**), bei dem die beteiligten DBMS lose gekoppelt sind und ein gewisses Ausmaß an Autonomie aufweisen. Viele DDBMS basieren auf einer Client/Server-Architektur.

Ein vierter Kriterium sind die **Kosten** eines DBMS. Der Großteil der heute angebotenen DBMS liegt im Preisbereich von 10.000 bis 100.000 Euro. Preisgünstige Einbenutzersysteme für PCs kosten zwischen 100 und 3.000 Euro. Am oberen Ende der Leistungsskala liegen Hochleistungs-DBMS ab 100.000 Euro.

Ferner lässt sich ein DBMS auch auf der Grundlage der unterschiedlichen Arten von **Zugriffspfaden** für die Speicherung von Dateien klassifizieren. Viele DBMS nutzen invertierte Dateistrukturen. Schließlich kann ein DBMS entweder **allgemein** oder **aufgabenspezifisch** sein. Wenn die Leistung ein vorrangiges Entscheidungskriterium ist, kann ein aufgabenspezifisches DBMS für eine bestimmte Anwendung entworfen und entwickelt werden. Ein solches System lässt sich dann aber meist nur durch umfangreiche Änderungen für andere Anwendungen nutzen. Viele Flugreservierungs- und Telefonverzeichnissysteme sind Beispiele von aufgabenspezifischen DBMS, die in der Vergangenheit entwickelt wurden. Diese Systeme gehören zur Kategorie der **Online-Transaktionsverarbeitungssysteme (OLTP-Systeme)**, die generell eine große Zahl gleichzeitiger Transaktionen ohne nennenswerte Verzögerungen ausführen können.

Wir kommen an dieser Stelle nochmals auf das Datenmodell – das wichtigste Kriterium für die Klassifizierung von DBMS – zurück. Das relationale Datenmodell repräsentiert eine Datenbank als Sammlung von Tabellen, wobei jede Tabelle in vielen Fällen in einer getrennten Datei gespeichert wird. Die Datenbank in Abbildung 1.2 ähnelt stark einer relationalen Darstellung. Die meisten relationalen Datenbanksysteme verwenden die Anfragesprache SQL und unterstützen eine begrenzte Anzahl von Benutzersichten. Das relationale Modell, seine Sprachen und Operationen sowie zwei Beispiele kommerzieller Systeme werden in den Kapiteln 7 bis 10 behandelt.

Das Objektdatenmodell definiert eine Datenbank in Bezug auf Objekte und deren Eigenschaften und Operationen. Objekte mit gleicher Struktur und gleichem Verhalten werden zu einer **Klasse zusammengefasst**; Klassen werden in **Hierarchien (in azyklischen Graphen)** organisiert. Die Operationen der einzelnen Klassen werden in Bezug auf vordefinierte Prozeduren spezifiziert, die als **Methoden** bezeichnet werden. Relationale DBMS erweitern ihre Modelle durch die Einbeziehung von Objektdatenbankkonzepten und anderen Fähigkeiten; diese Systeme werden als **objektrelationale** oder **erweiterte relationale DBMSe** bezeichnet und in den Kapiteln 11 bis 13 behandelt.

Das **Netzwerkmodell** repräsentiert Daten als Datensatztypen und stellt auch in begrenzter Form 1:N-Beziehungen dar, die als **Mengentyp** bezeichnet werden. Abbildung 2.4 zeigt ein Netzwerkschemadiagramm für die Datenbank aus Abbildung 1.2; dabei werden Datensatztypen als Rechtecke und Mengentypen als direkte beschriftete Pfeile dargestellt. Das Netzwerkmodell, das auch als CODASYL-DBTG-Modell¹¹

11. CODASYL DBTG ist die Abkürzung für »Computer Data Systems Language Data Base Task Group«, die Arbeitsgruppe, die das Netzwerkmodell und seine Sprache spezifiziert hat.

Ein DBMS, das diese drei Schichten deutlich trennt, benötigt Mappings zwischen den Schemas, um Anforderungen und Ergebnisse von einer Ebene zur nächsten überführen zu können. Die meisten DBMS trennen diese drei Ebenen nicht völlig. Wir haben die Drei-Schichten-Architektur dazu benutzt, um die Konzepte der logischen und physischen Datenunabhängigkeit zu definieren.

Anschließend wurden die wichtigsten Arten von Sprachen und Benutzeroberflächen von DBMS beschrieben. Eine Datendefinitionssprache (DDL) definiert das konzeptuelle Schema der Datenbank. Bei den meisten DBMS definiert die DDL ebenso Benutzersichten und manchmal auch Speicherstrukturen, während in anderen DBMS hierfür getrennte Sprachen (VDL, SDL) existieren. Das DBMS übersetzt alle Schemadefinitionen und speichert ihre Beschreibungen im DBMS-Katalog ab. Eine Datenmanipulationssprache (DML) dient zur Spezifizierung von Datenbankzugriffen und Veränderungsoperationen. Wir unterscheiden zwischen einer deklarativen DML (mengenorientiert, nicht prozedural) und einer prozeduralen DML (satzorientiert). Eine deklarative DML kann in eine Wirtssprache (Host Language) eingebettet oder auch als unabhängige Sprache benutzt werden; in diesem Fall wird sie als »Anfragesprache« bezeichnet.

Des Weiteren wurden verschiedene Arten von Benutzeroberflächen und die jeweils damit zusammenhängenden Benutzertypen beschrieben, ehe die Datenbanksystemumgebung, typische DBMS-Komponenten und DBMS-Dienstprogramme vorgestellt wurden.

Als Letztes wurden DBMS nach verschiedenen Kriterien klassifiziert: Datenmodell, Anzahl der unterstützten Benutzer, Verteilung, Arten der Zugriffspfade und Anwendungsbereiche. Die wichtigste Klassifizierung von DBMS basiert jedoch auf dem verwendeten Datenmodell. In diesem Zusammenhang wurden die wichtigsten Datenmodelle in den heute verfügbaren kommerziellen DBMS kurz vorgestellt.

WIEDERHOLUNGSFRAGEN

1. Definieren Sie die folgenden Begriffe: *Datenmodell, Datenbankschema, Datenbankzustand, internes Schema, konzeptuelles Schema, externes Schema, Datenunabhängigkeit, DDL, DML, SDL, VDL, Anfragesprache, Wirtssprache, Daten-Subsprache, Datenbank-Dienstprogramm, Katalog, Client/Server-Architektur*.
2. Diskutieren Sie die wichtigsten Datenmodelle.
3. Welcher Unterschied besteht zwischen einem Datenbankschema und einem Datenbankzustand?
4. Beschreiben Sie die Drei-Schichten-Architektur. Warum sind Abbildungen zwischen den Ebenen nötig? Wie wird diese Architektur von den verschiedenen Schemadefinitionssprachen unterstützt?
5. Welcher Unterschied besteht zwischen logischer und physischer Datenunabhängigkeit?
6. Welcher Unterschied besteht zwischen prozeduraler und nicht prozeduraler DML?
7. Diskutieren Sie die verschiedenen Arten von benutzerfreundlichen Oberflächen und die Benutzertypen, für die die einzelnen Oberflächen ausgelegt sind.

8. Mit welchen anderen Softwarekomponenten eines Rechners interagiert ein DBMS?
9. Diskutieren Sie die Funktionalität einiger Datenbankdienstprogramme und -werkzeuge.

ÜBUNGEN

10. Überlegen Sie sich verschiedene Benutzertypen für die Datenbank in Abbildung 1.2. Welche Anwendungen benötigt jeder Benutzer? Zu welcher Benutzerkategorie würde jede dieser Anwendungen gehören und welche Benutzeroberfläche wäre jeweils vorteilhaft?
11. Wählen Sie eine Datenbankanwendung aus, mit der Sie vertraut sind. Entwerfen Sie ein Schema und zeigen Sie eine Beispieldatenbank für diese Anwendung unter Verwendung der Notation in den Abbildungen 2.1 und 1.2. Welche zusätzlichen Informationen und Integritätsbedingungen würden Sie vorzugsweise in dem Schema darstellen? Denken Sie an mehrere Benutzer für Ihre Datenbank und entwerfen Sie eine Sicht (View) für jeden einzelnen.

AUSGEWÄHLTE LITERATUR

Viele Lehrbücher über Datenbanken, darunter Date (1995), Silberschatz u. a. (1998), Ramakrishnan (1997), Ullman (1988, 1989) und Abiteboul u. a. (1995) behandeln die verschiedenen Datenbankkonzepte, die hier vorgestellt wurden. Tsichritzis und Lochovsky (1982) ist eines der ersten Lehrbücher über Datenmodelle. Tsichritzis und Klug (1978) und Jardine (1977) präsentieren die Drei-Schichten-Architektur, die erstmals im DBTG-CODASYL-Bericht (1971) und später in einem ANSI-Bericht (American National Standards Institute, 1975) vorgeschlagen wurde. Eine ausführliche Analyse des relationalen Datenmodells mit einigen möglichen Erweiterungen finden sich in Codd (1992). Der für objektorientierte Datenbanken vorgeschlagene Standard wird in Cattell (1997) beschrieben.

Ein gutes Beispiel für Datenbank-Dienstprogramme sind das ETI Extract Toolkit (www.eti.com) und das Datenbank-Verwaltungswerkzeug DB Artisan von Embarcadero Technologies (www.embarcadero.com).