

Kapitel 1

Einführung

Datenbanksysteme und Datenbanken sind heutzutage in allen Bereichen zu finden, in denen Computer auftreten. Eine stetig wachsende Zahl von neuen Anwendungsgebieten (insbesondere von Nicht-Standard-Anwendungen) und daraus resultierende neue Anforderungen an Datenbanksysteme haben bis heute zu einer geradezu stürmischen Entwicklung der Datenbanktechnologie geführt. Ziel dieses Buches ist es, wesentliche Elemente dieser Technologie zu vermitteln und zu zeigen, wie Datenbanksysteme aufgebaut sind, aus welchen modularen Komponenten sie bestehen und wie diese realisiert werden können.

In diesem einleitenden Kapitel beschäftigen wir uns mit den Aufgaben und der Architektur von Datenbanksystemen sowie den Anforderungen bezüglich ihrer Implementierung. Abschnitt 1.1 beschreibt zunächst kurz die Nachteile traditioneller Datenorganisation in Dateisystemen, die letztendlich zur Entwicklung von Datenbanksystemen führten. Danach erläutern wir das Konzept des Datenbanksystems. Wir führen die wichtigste Terminologie ein und klären Begriffe wie „Datenbank“, „Datenbanksystem“, „Datenbankmanagementsystem“ und „Datenmodell“. Abschnitt 1.2 umreißt das breite Spektrum der Anforderungen an Datenbanksysteme, insbesondere aus Implementierungssicht. Abschnitt 1.3 stellt ein 3-Ebenen-Modell vor, das heute weitgehend als Grundlage für den Aufbau von Datenbanksystemen dient. Die physische Ebene als Bestandteil des 3-Ebenen-Modells steht im Mittelpunkt unserer Betrachtungen. Abschnitt 1.4 betrachtet DBMS aus der Sicht des Software-Engineering und beschreibt ihren Aufbau anhand einer hierarchischen und modularen Systemarchitektur. Abschnitt 1.5 beschreibt zusätzliche Komponenten in Form von Werkzeugen und Hilfsprogrammen.

1.1 Konzept des Datenbanksystems

Das Wesen traditioneller Datenverwaltung in von Betriebssystemen unterstützten Dateisystemen ist, dass jeder Anwendungsprogrammierer diejenigen Dateien definiert, die er für seine Anwendung braucht, unabhängig und vielleicht sogar ohne Kenntnis der Dateien von Anwendungen anderer Programmierer. Der Dateiaufbau ist unmittelbar an die jeweilige Verarbeitung angepasst, und dementsprechend ist

die Datei auch physisch abgespeichert. Die traditionelle Datenverwaltung mittels Dateisystemen führt im Wesentlichen zu folgenden schwerwiegenden Problemen:

- ❑ *Redundanz (redundancy)*. Die anwendungsspezifische Gestaltung von Dateien führt zu wiederholtem Auftreten von gleichen Daten in verschiedenen Dateien. Die dadurch herbeigeführte Redundanz hat insbesondere bei Änderungen Speicherverschwendung und erhöhten Verwaltungs- und Verarbeitungsaufwand zur Folge und wird in der Regel nicht zentral kontrolliert. Hieraus ergeben sich die folgenden, weiteren Probleme.
- ❑ *Inkonsistenz (inconsistency)*. Die Konsistenz der Daten, d.h. die logische Übereinstimmung der Dateiinhalte, kann nur schwer aufrechterhalten werden. Bei der Änderung eines Datums müssen alle Dateien geändert und angepasst werden, die dieses Datum enthalten. Ferner müssen diese Änderungen so aufeinander abgestimmt werden, dass nicht verschiedene Programme zum selben Zeitpunkt auf unterschiedliche Werte desselben Datums zugreifen können.
- ❑ *Daten-Programm-Abhängigkeit*. Ein weiteres Problem ergibt sich durch die sehr enge Abhängigkeit zwischen Anwendungsprogramm und Datenorganisation; das Anwendungsprogramm hat nämlich einen direkten Zugang zu den Daten einer Datei. Ändert sich der Aufbau einer Datei oder ihre Organisationsform, so müssen alle darauf basierenden Programme geändert werden. Umgekehrt kann eine Erweiterung der Funktionalität eines Anwendungsprogramms neue Anforderungen an den Dateiaufbau stellen und eine Restrukturierung von Dateien erfordern.
- ❑ *Inflexibilität*. Da die Daten nicht anwendungsneutral und in ihrer Gesamtheit, sondern ausschließlich anwendungsspezifisch gesehen werden, erweist sich die Realisierung neuer Anwendungen sowie die Auswertung vorhandener Daten als problematisch. Dies gilt insbesondere für Anwendungen und Auswertungen, die Daten aus verschiedenen Dateien benötigen würden. Mangelnde Anpassungsfähigkeit ist daher ein wesentliches Kennzeichen der traditionellen Datenorganisation.

Datenbanksysteme fielen daher nicht vom Himmel, sondern entstanden aus den erkannten Nachteilen traditioneller Datenorganisation sowie aus den zunehmenden Anforderungen an die Verwaltung und Analyse großer Datenbestände. Während bei der traditionellen Datenorganisation die Anwendungsprogramme im Vordergrund standen und die Daten mehr als deren Anhängsel betrachtet wurden, änderte sich nun die Sichtweise. Die Daten selbst stehen nun im Mittelpunkt der Betrachtung und werden als eigenständig und wesentlich angesehen. Die Daten werden einmal definiert und für alle Benutzer zentral und als integriertes Ganzes verwaltet.

Dies ist der entscheidende Schritt zum Konzept des Datenbanksystems, das wir im Folgenden beschreiben werden. Beginnen wollen wir mit dem Begriff der Datenbank, für den sich eine präzise Beschreibung allerdings schwerlich finden lässt. Folgende Charakterisierung trifft jedoch das Wesentliche:

Eine *Datenbank*¹ (*database*), kurz *DB*, ist eine integrierte und strukturierte Sammlung persistenter Daten, die allen Benutzern eines Anwendungsbereichs als gemeinsame und verlässliche Basis aktueller Information dient.

Das Attribut *integriert* (*integrated data*) bedeutet, dass eine Datenbank eine vereinheitlichende und anwendungsneutrale Gesamtsicht auf die interessierenden Daten bietet, die den natürlichen Gegebenheiten und Zusammenhängen der Anwendungswelt entspricht. Insbesondere sind die Daten also nicht danach angeordnet, wie einzelne Anwendungen sie benötigen. Das Attribut *strukturiert* (*structured data*) besagt, dass eine Datenbank keine zufällige Zusammenstellung von Daten darstellt, sondern dass logisch kohärente Informationseinheiten identifizierbar sind, deren zugehörige Daten (möglichst) redundanzfrei gespeichert werden. Das Attribut *persistente* (*persistent data*) beschreibt die Eigenschaft, dass die Daten in der Datenbank dauerhaft auf externen Speichermedien verfügbar sein sollen. Diese Daten unterscheiden sich also zum Beispiel von flüchtigen Ein- und Ausgabedaten. Die Daten in der Datenbank bilden eine *gemeinsame, geteilte Basis* (*shared data*) für alle Benutzer und Anwendungsprogramme zu jeweils eigenen Zwecken. Auf gleiche Daten kann sogar von verschiedenen Benutzern gleichzeitig zugegriffen werden (*concurrent access*), wobei dafür gesorgt werden muss, dass sich diese nicht gegenseitig stören. Das Attribut *verlässlich* (*reliable data*) besagt, dass trotz etwaiger Systemabstürze oder Versuche des unautorisierten Zugriffs stets für die Sicherheit der gespeicherten Information gesorgt werden muss.

Alle Anwendungsprogramme und Benutzer arbeiten also auf einem gemeinsamen Datenbestand; sie greifen nun aber nicht mehr direkt auf die abgespeicherten Daten zu, sondern erhalten die gewünschten Daten durch das sogenannte Datenbankmanagementsystem. Dadurch wird erreicht, dass ihnen Betriebssystem- und Hardwaredetails verborgen bleiben.

Ein *Datenbankmanagementsystem* (*database management system*), kurz *DBMS*, ist ein All-Zweck-Softwaresystem, das den Benutzer bei der Definition, Konstruktion und Manipulation von Datenbanken für verschiedene Anwendungen applikationsneutral und effizient unterstützt.

Zwischen der physischen Datenbank und seinen Benutzern liegt also eine Software-schicht, die aus einer Menge von Programmen zur Verwaltung und zum Zugriff auf die Daten in der Datenbank besteht. Die Definition einer Datenbank umfasst die Spezifikation der Typen der Daten, die in der Datenbank gespeichert werden sollen, mit einer entsprechenden Beschreibung jedes Datentyps. Ferner umfasst sie die Angabe von Strukturen für die Speicherung von Werten dieser Datentypen. Die Konstruktion der Datenbank beinhaltet den Prozess der Speicherung der Daten auf einem externen Speichermedium, das von dem DBMS kontrolliert wird. Die Mani-

¹ Manchmal wird neben dem Begriff der Datenbank auch der Begriff der *Datenbasis* verwendet. Wir unterscheiden nicht zwischen diesen beiden Begriffen.

pulation einer Datenbank umfasst solche Funktionen wie das Stellen von Anfragen (*queries*) zum Auffinden (*retrieval*) spezieller Daten, das Aktualisieren (*update*) der Datenbank und die Generierung von Berichten über die Daten. Die Aufgaben eines DBMS werden wir ausführlich in Abschnitt 1.2 beschreiben.

Ein *Datenbanksystem* (*database system*), kurz *DBS*, fasst die beiden Komponenten Datenbanksystem und Datenbank zusammen: $DBS = DBMS + DB$.

Die Kernaufgabe eines DBS besteht also darin, große Mengen von strukturierten Informationen entgegenzunehmen, effizient zu speichern und zu verwalten sowie auf Anforderung bereitzustellen. Selbstverständlich kann dasselbe DBMS (in einer oder mehreren Kopien) mehrere Datenbanken verwalten und damit mehrere DBS bilden. Wo Zweifel ausgeschlossen sind, werden wir trotzdem, wie dies auch allgemein üblich ist, den Begriff DBS synonym zu DBMS verwenden. Bild 1.1 zeigt eine stark vereinfachte Sicht eines Datenbanksystems.

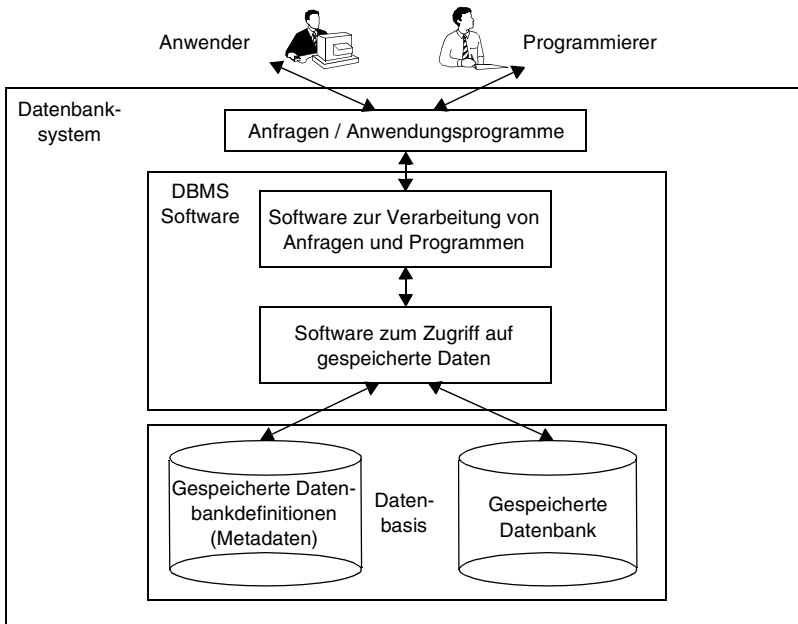


Bild 1.1. Eine stark vereinfachte Sicht eines Datenbanksystems

Jedem Datenbanksystem liegt ein abstraktes Datenmodell zugrunde, das dem Benutzer eine bestimmte Sicht auf die Daten der Datenbank bietet.

Ein *Datenmodell* (*data model*) ist ein mathematischer Formalismus, der aus einer Notation zur Beschreibung der interessierenden Daten und aus einer Menge von Operationen zur Manipulation dieser Daten besteht.

Ein solches Datenmodell erlaubt es, die Struktur einer Datenbank – hierunter verstehen wir die Datentypen, Beziehungen und Bedingungen, die auf den Daten gelten sollen – angemessen zu beschreiben und dem Benutzer Daten in verständlicheren Begriffen zur Verfügung zu stellen. Daten kann man auf verschiedenen Abstraktionsebenen betrachten, wie wir in Abschnitt 1.3 erkennen werden. Wichtige Datenmodelle für DBS sind das relationale und das objekt-orientierte Datenmodell sowie das objekt-relationale Datenmodell, das zunehmend an Popularität gewinnt, indem es versucht, die positiven Eigenschaften des relationalen und des objekt-orientierten Modells miteinander zu verbinden. Ältere Modelle sind das hierarchische Modell und das Netzwerkmodell.

1.2 Anforderungen an Datenbanksysteme

Moderne Datenbanksysteme bieten ein breites Spektrum von Funktionen an und haben eine Vielzahl von Anforderungen zu erfüllen, die sich in der Architektur und somit in der Implementierung solcher Systeme widerspiegeln (siehe Abschnitt 1.4). Einerseits werden die Nachteile der traditionellen Datenorganisation behoben, aber darüberhinaus werden eine ganze Anzahl weiterer Funktionen für den Benutzer sichtbar oder unsichtbar bereitgestellt. Alle im Folgenden beschriebenen Anforderungen werden mehr oder minder in heutigen Datenbanksystemen realisiert und stellen somit gleichsam die Vorteile solcher Systeme dar. Wir betrachten diese Anforderungen insbesondere aus dem Blickwinkel der Implementierung.

- ❑ *Datenunabhängigkeit (data independence)*. Anwendungsprogramme sollten so unabhängig wie möglich von den Einzelheiten der Datenrepräsentation und -speicherung sein. Das DBMS sorgt hierzu für eine abstrakte Sicht auf die Daten (siehe auch Abschnitt 1.3).
- ❑ *Effizienter Datenzugriff*. Ein DBMS verwendet eine Vielzahl von ausgeklügelten Techniken zur effizienten Speicherung von und zum effizienten Zugriff auf Daten. Dies ist insbesondere dann von Bedeutung, wenn die Datenmenge so groß ist, dass sie nicht im Hauptspeicher gehalten werden kann, sondern auf einem externen Medium gespeichert werden muss. Beabsichtigen wir zum Beispiel, in einer Datenbank, die Tausende von Artikeln gespeichert hat, anhand der Artikelnummer nach einem bestimmten Artikel zu suchen, ist es sehr kostenintensiv, den gesamten Datenbestand zu durchlaufen und jeden Datensatz mit der gesuchten Artikelnummer zu vergleichen. Effizienter ist es, einen *Index* einer *Indexstruktur (index structure)* über den Artikelnummern zu benutzen, der einen direkten Zugriff auf den gesuchten Artikel erlaubt.
- ❑ *Gemeinsame Datenbasis*. Alle jetzigen und zukünftigen Anwendungsprogramme und Benutzer greifen auf einen gemeinsamen Datenbestand zu (siehe auch Abschnitt 1.1).

- ❑ *Nebenläufiger Datenzugriff (concurrent access)*. Ein DBMS ermöglicht es, dass auf gleiche Daten von verschiedenen Benutzern quasi gleichzeitig zugegriffen werden kann und dass jedem Benutzer der (fälschliche) Eindruck eines exklusiven Zugriffs auf diese Daten vermittelt wird. Versuchen zum Beispiel zwei Benutzer, das gleiche Datum gleichzeitig zu verändern, kann eine Änderung verloren gehen, weil sie durch den Wert der anderen Änderung überschrieben werden kann. DBMS benutzen das Konzept der *Transaktionen (transactions)*, um nebenläufigen Datenzugriff zu steuern (*Synchronisation*) und dafür zu sorgen, dass sich zwei Zugriffe nicht gegenseitig beeinflussen. Eine Transaktion ist eine konsistenzhaltende Operation, die nach Ausführung die Datenbank in einem konsistenten Zustand zurücklässt, wenn diese vor Beginn der Transaktion ebenfalls konsistent war.
- ❑ *Fehlende oder kontrollierte Redundanz (redundancy)*. Die Nachteile der traditionellen Datenorganisation werden in DBS durch eine integrierte Sicht der Daten, die Kopien desselben Datums nicht erlaubt, vermieden. Allerdings wird in gewissem Rahmen eine begrenzte und vom DBMS *kontrollierte Redundanz (controlled redundancy)* zugelassen, zum Beispiel um die logischen Beziehungen zwischen Daten aufrechtzuerhalten oder um die Leistungsfähigkeit oder *Performance* zu verbessern. Für konsistente Änderungen auf verschiedenen Kopien des gleichen Datums ist dann das DBMS verantwortlich.
- ❑ *Konsistenz der Daten (consistency)*. Fehlende Redundanz der Daten bedingt ihre Konsistenz. Wenn ein Datum nur einmal in der Datenbank auftritt, muss jede Änderung eines Wertes nur einmal durchgeführt werden, und alle Benutzer haben sofortigen Zugriff auf diesen neuen Wert. Bei kontrollierter Redundanz muss das DBMS sicherstellen, dass die Datenbank aus der Sicht des Benutzers niemals inkonsistent wird. Dies geschieht durch automatisches Propagieren der Änderungen (*propagating updates*) eines Datums zu all seinen Kopien.
- ❑ *Integrität der Daten (integrity)*. Integrität bedeutet ganz allgemein Korrektheit und Vollständigkeit der Daten. Selbst bei einer redundanzfreien Datenbank können die abgespeicherten Daten semantisch falsch sein. Zum Beispiel könnte die Datenbank die wöchentliche Arbeitszeit eines Angestellten mit 400 statt 40 Stunden ausweisen oder ihn der nicht existierenden Abteilung D17 zuordnen. *Integritätsbedingungen* oder *-regeln (integrity constraints)* sind ein Mittel, um solche Integritätsverletzungen zu vermeiden. Beispielsweise können solche Regeln festlegen, dass ein Angestellter zwischen 10 und 40 Stunden wöchentlich arbeitet und dass es die Abteilungen D1 bis D7 gibt. Das DBMS muss diese Bedingungen dann beim Einfügen, Ändern und Löschen von Daten überprüfen. Inkonsistenz ist ein Spezialfall mangelnder Integrität.
- ❑ *Datensicherheit (security)*. Datensicherheit bezeichnet den Schutz der Datenbank vor unautorisiertem Zugriff. Beispielsweise sollte ein Personalsachbearbeiter einer Bank, der für Gehaltsabrechnungen verantwortlich ist, nur die

Daten der Bankangestellten aber nicht die Daten der Kundenkonten einsehen können, d.h. er erhält eine bestimmte *Sicht (view)* auf die Daten. Dies erfordert eine *Zugriffskontrolle (access control)* mit Authentisierung und Verschlüsselung als möglichen Mechanismen zur Sicherung. Authentisierung bedeutet, dass Sicherheitsregeln, Sicherheitsverfahren oder zusätzliche Kennwörter für jede Zugriffsart (Lesen, Einfügen, Löschen, Ändern usw.) die Zugriffslaubnis eines Benutzers auf bestimmte, sensitive Daten überprüfen. Zusätzlich kann das DBMS die Daten vor der Abspeicherung verschlüsseln. Wenn dann ein autorisierter Benutzer auf die entsprechenden Daten zugreifen möchte, werden sie von ihm unbemerkt automatisch entschlüsselt und zur Verfügung gestellt. Daten, auf die unautorisiert zugegriffen wird, erscheinen in verschlüsselter Form.

- ❑ *Bereitstellung von Backup- und Recovery-Verfahren.* Ein DBMS verfügt über Mechanismen, um Benutzer vor den Auswirkungen von Systemfehlern zu schützen. Meist nachts werden Sicherungskopien der Datenbank auf Bändern vorgenommen (*backup*). Während des Tagesablaufs wird diese Maßnahme gewöhnlich durch ein Protokoll der durchgeführten Änderungen ergänzt. Wird die Datenbank modifiziert, erfolgt ein Protokolleintrag. Bei einem Systemabsturz werden die Bänder und das Protokoll dazu benutzt, in der Datenbank den zuletzt aktuellen Zustand automatisch wiederherzustellen (*recovery*).
- ❑ *Stellen von Anfragen.* In der traditionellen Datenorganisation muss eine Anfrage an den Datenbestand stets mittels eines eigenen Anwendungsprogramms gestellt werden. Dies ist extrem inflexibel. DBMS stellen eine *Anfragesprache (query language)* zur Verfügung, die es dem Benutzer erlaubt, ad hoc am Bildschirm mittels der Tastatur eine Anfrage zu stellen und unmittelbar eine Antwort zu erhalten. Ferner kann eine solche Anfragesprache auch in eine Programmiersprache eingebettet sein. Eine wichtige Anforderung an das DBMS ist nun, eine solche Anfrage möglichst effizient zu verarbeiten (*query processing*), d.h. eine lexikalische Analyse (*query scanning*) und Syntaxanalyse (*query parsing*) der Anfrage durchzuführen, die Anfrage zu optimieren (*query optimization*) und effizient auszuführen (*query execution*).
- ❑ *Bereitstellung verschiedenartiger Benutzerschnittstellen.* Weil viele Arten von Benutzern mit unterschiedlichem inhaltlichen und technischen Wissen eine Datenbank nutzen, muss ein DBMS verschiedenartige Benutzerschnittstellen zur Verfügung stellen. Mögliche Schnittstellen sind Anfragesprachen für gelegentliche Benutzer, Programmierschnittstellen für Anwendungsprogrammierer, menügesteuerte Schnittstellen für naive Anwender, fenster- und graphikorientierte Benutzeroberflächen, *Datendefinitionssprachen (data definition languages)*, *Datenmanipulationssprachen (data manipulation languages)* usw.
- ❑ *Flexibilität.* Hierunter ist einerseits die Anforderung zu verstehen, dass die Struktur einer Datenbank auf bequeme Weise geändert werden kann, ohne existierende Anwendungsprogramme verändern zu müssen. Zum Beispiel

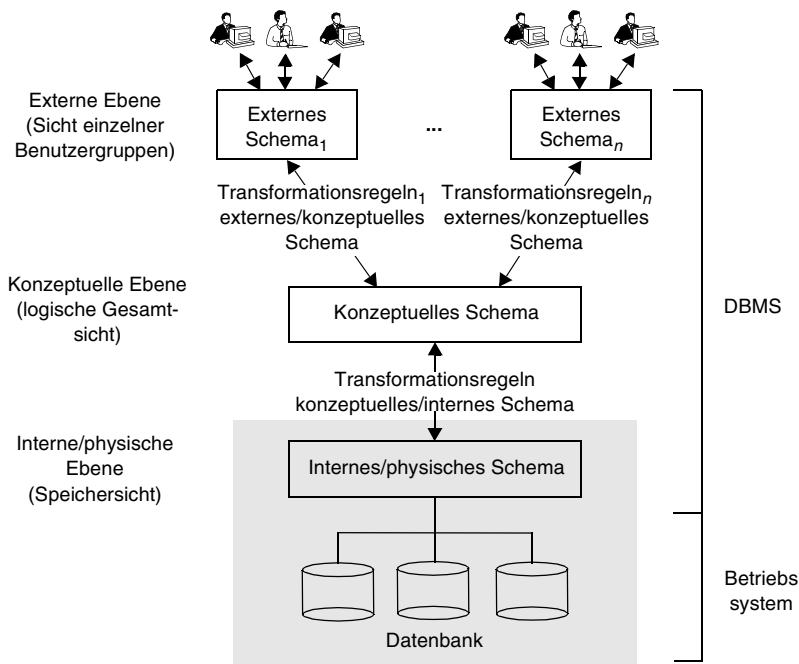
sollte es möglich sein, Datensätze um ein neues Feld zu erweitern oder eine neue Kollektion von Daten in die Datenbank einzufügen. Ferner besteht die Anforderung, auf einfache Weise Daten nach anderen Gesichtspunkten als bisher vorgesehen auswerten zu können.

- *Schnellere Entwicklung von neuen Anwendungen.* Ein DBMS bietet wichtige Funktionen an, die von vielen auf die Daten der Datenbank zugreifenden Anwendungen benötigt werden. Zum Beispiel können komplexe Anfragen leicht durch eine Anfragesprache beantwortet werden; nebenläufiger Zugriff und Fehlerbehandlung bei einem Systemabsturz werden vom DBMS unterstützt. Alle diese Funktionen erlauben eine schnelle Entwicklung von Anwendungen. Darüberhinaus sind Anwendungen, die auf Funktionen des DBMS aufbauen, meist robuster, weil viele Aufgaben bereits vom DBMS übernommen werden und nicht mehr von der Anwendung implementiert werden müssen.

1.3 Das 3-Ebenen-Modell

Die Nachteile traditioneller Datenorganisation sowie die in Abschnitt 1.2 beschriebenen Anforderungen führten 1975 zum Vorschlag eines 3-Ebenen-Modells, einer abstrakten Architektur für DBS, durch das ANSI/SPARC-Standardisierungskomitee. Dieses Modell ist heute im Wesentlichen Grundlage aller modernen DBS. Sein erklärtes Ziel ist es, dem Benutzer eine abstrakte Sicht auf die von ihm benötigten Daten zu geben und diese Sicht von der konkreten, physischen Sicht zu trennen. Das Modell besteht aus drei Abstraktionsebenen (Bild 1.2):

- Die *physische/interne Ebene* (*physicall/internal level*) basiert auf einem *physischen/internen Schema* (*physicall/internal schema*), das die physischen und persistenten Speicherstrukturen der Datenbank beschreibt. Dieses Schema benutzt ein *physisches Datenmodell* und beschreibt für die Datenbank alle Implementierungseinzelheiten über den Aufbau der Daten, die Datenspeicherung und die Zugriffspfade. Hier wird also beschrieben, *wie* Daten gespeichert werden.
- Die *konzeptuelle Ebene* (*conceptual level*) beruht auf einem *konzeptuellen Schema* (*conceptual schema*), das mittels eines vom DBMS bereitgestellten Datenmodells (z.B. relationales Modell) die logische Struktur der gesamten Datenbank für eine Gemeinschaft von Benutzern erfasst. Das konzeptuelle Schema ist eine globale Beschreibung der Datenbank, die Details der physischen Speicherstrukturen verbirgt, von ihnen abstrahiert und sich darauf konzentriert, Entitäten, Datentypen, Beziehungen und Integritätsbedingungen zu beschreiben. Hier wird also modelliert, *was* für Daten gespeichert werden.
- Die *externe Ebene* (*external level, view level*) umfasst eine Anzahl von *externen Schemata* (*external schema*) oder *Benutzersichten* (*user view*). Jedes externe Schema beschreibt die Datenbanksicht einer Gruppe von Datenbank-

**Bild 1.2.** Das 3-Ebenen-Modell

nutzen. Jede Sicht beschreibt typischerweise den Teil der Datenbank, für den sich eine bestimmte Benutzergruppe interessiert (und auf den sie auch zugreifen darf) und verbirgt den Rest der Datenbank vor ihr.

Transformationsregeln (mappings) definieren die Beziehungen zwischen jeweils zwei aufeinanderfolgenden Ebenen. Die *Transformationsregeln konzeptuelles/internes Schema* beschreiben, wie für jedes Objekt des konzeptuellen Schemas die Information aus den physisch abgespeicherten Sätzen, Feldern usw. des internen Schemas erhalten werden kann. Änderungen des internen Schemas wirken sich nur auf die Transformationsregeln, aber nicht auf das konzeptuelle Schema aus. In den *Transformationsregeln externes/konzeptuelles Schema* wird angegeben, wie eine spezielle externe Sicht mit dem konzeptuellen Schema zusammenhängt, d.h. welchen Ausschnitt des konzeptuellen Schemas eine bestimmte Benutzergruppe sieht. Daten über die Schemata aller drei Ebenen sowie über die Transformationsregeln (sog. *Meta-Daten*) werden in einem *Systemkatalog* (*system catalog*, *data dictionary*) persistent abgespeichert.

Ein wichtiger Vorteil des 3-Ebenen-Modells ist die *Datenunabhängigkeit* (*data independence*). Dies bedeutet, dass höhere Ebenen des Modells unbeeinflusst von Änderungen auf niedrigeren Ebenen bleiben. Zwei Arten von Datenunabhängigkeit werden unterschieden: *Logische Datenunabhängigkeit* (*logical data independence*)

bedeutet, dass sich Änderungen des konzeptuellen Schemas (z.B. Informationen über neue Typen von Entitäten, weitere Informationen über existierende Entitäten) nicht auf externe Schemata (z.B. existierende Anwendungsprogramme) auswirken. *Physische Datenunabhängigkeit* (*physical data independence*) beinhaltet, dass Änderungen des internen Schemas (z.B. Wechsel von einer Zugriffsstruktur zu einer effizienteren, Benutzung anderer Datenstrukturen, Austausch von Algorithmen) keine Auswirkungen auf das konzeptuelle (und externe) Schema haben.

Auf die externe und konzeptuelle Ebene gehen wir im Folgenden nicht weiter ein, da im Mittelpunkt unserer Betrachtungen die physische Ebene steht (Bild 1.2). Letztere beschäftigt sich also mit der physischen Implementierung einer Datenbank. Sie bedient sich dabei elementarer Betriebssystemmethoden, um Daten auf externen Speichermedien auszulagern. Statistische Informationen über die Häufigkeit und Art von Zugriffen auf Entitäten, über Werteverteilungen von Attributen sowie eine Anzahl weiterer Faktoren beeinflussen die verwendeten Strukturen und Methoden zum Aufbau der Datenbank. Globales Ziel ist es, eine physische Datenorganisation zu entwerfen, so dass die im konzeptuellen Schema beschriebenen Objekte auf die physische Ebene abbildbar sind und die Aufgaben aller Benutzer insgesamt „gut“ und effizient erfüllen. Fragestellungen, die das physische Schema erfassen muss und die später genauer betrachtet werden, sind z.B. die Repräsentation von Attributwerten, der Aufbau gespeicherter Datensätze, Zugriffsmethoden auf Datensätze, zusätzliche Zugriffspfade (Indexe). Vom physischen Schema hängt also wesentlich die Leistungsfähigkeit des gesamten Datenbanksystems ab.

1.4 Softwarearchitektur eines DBMS

Bedingt durch eine Vielzahl von Anforderungen (siehe Abschnitt 1.2) sind heutige DBMS kompliziert aufgebaute Softwareprodukte. DBMS bieten einerseits nach oben hin eine Schnittstelle zum Endbenutzer und andererseits nach unten hin eine Schnittstelle zum Betriebssystem und zur Hardware an (Bild 1.1). In diesem Abschnitt befassen wir uns mit der systematischen und modularen Zerlegung dieser Softwareprodukte in handhabbare Komponenten sowie mit deren Wechselwirkungen und Schnittstellen untereinander, d.h. wir beschreiben die prinzipielle Systemarchitektur von Datenbanksystemen. Wir betrachten also ein *Modell* eines DBMS, das so allgemein gehalten ist, dass es weitgehend der Realität entspricht. Konkrete Architekturen und die dort verwendete Terminologie können sich natürlich in Einzelheiten unterscheiden.

Bild 1.3 zeigt die grundlegenden Komponenten eines typischen DBMS. Die DBMS-Software ist hierarchisch in Schichten organisiert (wir sprechen daher auch von einer *Schichtenarchitektur*), wobei jede Schicht auf der direkt unter ihr liegenden Schicht aufbaut und bestimmte Objekte und Operatoren der niedrigeren Schicht mittels einer von dieser bereitgestellten Schnittstelle zur eigenen Realisierung benutzen bzw. aufrufen kann. Die interne Struktur der Objekte und die Implemen-

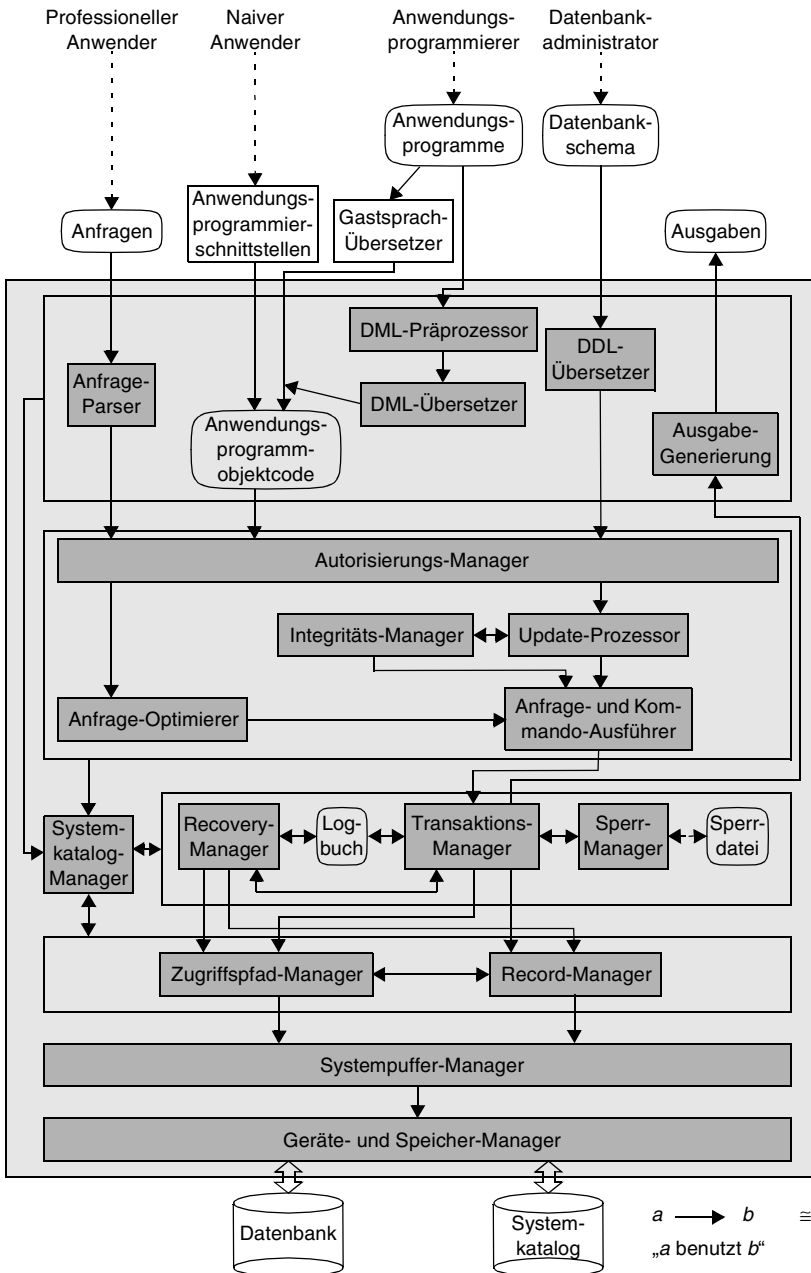


Bild 1.3. Komponenten eines Datenbanksystems

tierung der Operatoren der niedrigeren Schicht bleibt der darüberliegenden Schicht verborgen.

Die unterste Schicht bildet der *Geräte- und Speicher-Manager* (*disk space manager, data manager*), dem die Verwaltung der dem DBMS zur Verfügung stehenden Hardware-Betriebsmittel obliegt und der zwischen externem Speichermedium und dem Hauptspeicher alle physischen Zugriffe auf die Datenbank und auf den Systemkatalog ausführt. Insbesondere ist diese Schicht also für die Bereitstellung und Freigabe von Speicherplatz zur Darstellung von Daten auf externen Speichermedien und für die physische Lokalisierung eines angeforderten Datums verantwortlich. Die von dieser Schicht angebotenen Objekte sind *Dateien* (*files*) und *Blöcke* (*blocks*), die von höheren Schichten allokiert, deallokiert, gelesen und geschrieben werden können. Von Gerätecharakteristika wie Art des Speichermediums, Zylinderanzahl, Spüranzahl, Spurlänge usw. wird abstrahiert. Die Realisierung dieser Komponente erfolgt entweder durch das darunter liegende Betriebssystem oder durch ein spezialisiertes System, das auf die besonderen Bedürfnisse eines DBMS gezielt eingeht. Ein solch spezialisiertes System kann dann beispielsweise versuchen, Datenteile, auf die meistens als Einheit zugegriffen wird, benachbart (geclustert) auf einer Festplatte abzuspeichern. Dieses Vorgehen minimiert die Suchzeit, weil die gesamte Einheit in einem Lesevorgang erhalten werden kann.

Oberhalb dieser Schicht liegt der *Systempuffer-Manager* (*buffer manager*), der den verfügbaren Hauptspeicherbereich (den „Puffer“) in eine Folge von *Seiten* (*pages*) oder *Rahmen* (*frames*) unterteilt. In der Regel wird eine Seite auf einen Block abgebildet, so dass Lesen und Schreiben einer Seite nur einen Plattenzugriff erfordern. Auf Leseanforderung werden Seiten von einem externen Speichermedium in den Puffer gebracht, auf Schreib Anforderung Seiten des Puffers auf einem persistenten Medium gesichert. Nach oben hin stellt dieser Manager *Segmente* (*segments*) mit sichtbaren Seitengrenzen als lineare Adressräume im Systempuffer zur Verfügung. Dadurch erfolgt die konzeptionelle Trennung von Segment und Datei sowie Seite und Block.

Die nächst höhere Schicht wird durch den Zugriffspfad-Manager und den Record-Manager gebildet. Der *Zugriffspfad-Manager* (*access path manager*) verwaltet eine Anzahl von externen Datenstrukturen zur Abspeicherung von und zum Zugriff auf Kollektionen von Datensätzen. Beispiele solcher Strukturen sind sequentielle Speicherungsformen sowie Indexstrukturen wie der B*-Baum. Operationen beinhalten das Erzeugen und Löschen dieser Speicherstrukturen sowie das effiziente Speichern, Auffinden, Ändern und Löschen von Datensätzen innerhalb dieser Strukturen. Der *Record-Manager* (*record manager*) übernimmt alle Aufgaben, die sich mit der internen Darstellung eines *logischen* Datensatzes (z.B. eines Tupels oder Objekts) beschäftigen. Operationen erlauben den Aufbau und die Zerlegung eines *internen* Datensatzes (d.h. eines Satzes der internen Ebene) und den Zugriff auf dessen Komponenten. Nach unten hin werden interne Datensätze und physische Zugriffspfade auf Seiten von Segmenten abgebildet.

Der Datensicherungs- oder Recovery-Manager, der Transaktions-Manager sowie der Sperr-Manager befinden sich in der nächst höheren Schicht. Im Allgemeinen steht eine Datenbank nicht nur einem Benutzer exklusiv, sondern mehreren Benutzern gleichzeitig zur Verfügung. Eine Folge von Befehlen, die nur in ihrer Gesamtheit und ansonsten überhaupt nicht ausgeführt wird, wird als *Transaktion* (*transaction*) bezeichnet. Für einen Mehrbenutzerbetrieb ergibt sich dann das zu lösende Problem der *Synchronisation* (quasi-) parallel ablaufender Transaktionen. Für die Verwaltung von Transaktionen ist der *Transaktions-Manager* (*transaction manager*) zuständig, der nach außen dem Benutzer die Datenbank als ein für ihn exklusiv verfügbares Betriebsmittel erscheinen lässt. Jedoch ist es intern im Allgemeinen nicht sinnvoll, alle zu einem bestimmten Zeitpunkt aktuellen Transaktionen sequentiell ablaufen zu lassen bzw. zu verarbeiten. Dies würde die Blockierung eines kurz andauernden Auftrags durch einen länger andauernden Auftrag zur Folge haben, obwohl beide vielleicht sogar mit disjunkten Teilen der Datenbank arbeiten. Daher werden die einzelnen Transaktionen im Allgemeinen zeitlich verzahnt ausgeführt. Diese Verzahnung ist allerdings nicht frei von Problemen und kann zu Konflikten führen. Daher sind spezielle Kontrollstrategien notwendig (*concurrency control*).

Eine Transaktion setzt vor ihrer Ausführung gemäß einem geeigneten *Sperrprotokoll* (*Sperren* (*locks*)) auf die von ihr exklusiv benötigten Datenbankobjekte und Betriebsmittel und gibt sie am Ende wieder frei. Der *Sperr-Manager* (*lock manager*) verwaltet Anforderungen auf Sperren und gewährt Sperren auf Datenbankobjekte, wenn sie verfügbar werden. Sperrinformationen werden in einer *Sperr-Datei* (*lock file*) abgespeichert.

Bei der Ausführung einer Transaktion kann es passieren, dass der Transaktions-Manager feststellt, dass die aktuell bearbeitete Transaktion nicht erfolgreich beendet werden kann. In diesem Fall übergibt er sie dem *Datensicherungs- oder Recovery-Manager* (*recovery manager*), dessen Aufgabe es ist, die Datenbank in den Zustand vor dem Start der Transaktion zurückzusetzen. Hierzu muss er alle Änderungen, die die Transaktion an der Datenbank bereits vorgenommen hat, wieder rückgängig machen. Dies geschieht mit Hilfe des *Log-Buchs*, welches unter anderem derartige Änderungen protokolliert. Der Recovery-Manager ist auch dann zuständig, wenn das System einen Software- oder Hardware-Fehler erkennt oder wenn es „zusammenbricht“. Er ist dann für den Wiederanlauf des Datenbanksystems verantwortlich und muss die Datenbank in einen konsistenten Zustand (z.B. mit Hilfe von Sicherungskopien) zurückversetzen; alle „verlorengegangenen“ Transaktionen sind vollständig neu zu starten.

Die nächst höhere Schicht wird von mehreren Komponenten gebildet und umfasst den Autorisierungs-Manager, den Integritäts-Manager, den Update-Prozessor, den Anfrage-Optimierer und den Anfrage- und Kommando-Ausführer. Benutzeranfragen (*queries*) und Änderungskommandos liegen an der oberen Schnittstelle dieser Schicht in einer „internen“ Form vor (vergleichbar mit dem von einem Compiler von der einen an die nächste Übersetzungsphase übergebenen Zwischencode). Diese Form kann z.B. für relationale Anfragen ein Syntaxbaum (Operatorbaum) sein, dessen Blätter die Operanden und dessen innere Knoten die auf diesen auszu-

führenden Operationen repräsentieren. Für Anfragen und Kommandos führt der *Autorisierungs-Manager* (*authorization manager*) eine *Zugriffskontrolle* durch. Diese ermittelt, ob der Benutzer auf die in der Anfrage oder im Kommando vorkommenden Daten überhaupt zugreifen darf. Informationen darüber, wer auf welche Daten zugreifen darf, sind in Autorisierungstabellen abgelegt, die Bestandteil des Systemkatalogs sind.

Benutzeranfragen und Änderungskommandos werden danach unterschiedlich gehandhabt. Bei Änderungen (*updates*), die vom *Update-Prozessor* bearbeitet werden, werden mit Hilfe des *Integritäts-Managers* (*integrity manager*) sogenannte *Integritätsbedingungen* (*integrity constraints*) überprüft, die die semantische Korrektheit der Datenbank überwachen. Beispiele solcher Bedingungen sind „Gehälter sind stets größer als 0“ oder „Kundennummern müssen eindeutig sein“. Integritätsbedingungen werden bei der Definition des konzeptuellen Schemas festgelegt und zur Laufzeit vom DBMS automatisch ohne Einwirkung des Benutzers überwacht. Die interne Zwischenform des Kommandos wird hierzu geeignet erweitert. Anfragen werden häufig aus Ausführungssicht unnötig kompliziert vom Benutzer formuliert. Daher wird eine Anfrage an den *Anfrage-Optimierer* (*query optimizer*) übergeben, der die interne Zwischenform so verändert, dass sie einer effizienter ausführbaren Formulierung entspricht, ohne jedoch das Ergebnis zu verändern. Der Update-Prozessor für Kommandos und der *Anfrage-Ausführer* (*query executor*) für optimierte Anfragen in interner Form erstellen dann Ausführungsprogramme bzw. *Zugriffspläne* (*execution plans*), für die Code erzeugt wird. Hierzu benutzen beide Komponenten Implementierungen von Zugriffsstrukturen und -operatoren des DBMS.

Die oberste Ebene besteht ebenfalls aus mehreren Komponenten und bietet nach oben hin eine mengenorientierte Schnittstelle an. Naive, d.h. ungeübte oder gelegentliche Benutzer verwenden *Anwendungsschnittstellen* (*application interfaces*), die bereits in permanenten *Anwendungsprogrammobjectcode* (*application program object code*) übersetzt sind, zur Kommunikation mit der Datenbank. Der *DDL-Übersetzer* (*DDL compiler*) verarbeitet Schemadefinitionen, die in einer *Datendefinitionssprache* (*data definition language, DDL*) spezifiziert sind, und speichert die Beschreibungen der Schemata im Systemkatalog. Geübte, professionelle Benutzer stellen interaktiv und ad hoc *Anfragen* (*queries*) an das System. Die Formulierung von Anfragen erfolgt mittels einer deskriptiven, nicht-prozeduralen Hochsprache, die *Anfragesprache* (*query language*) (z.B. SQL) genannt wird. Charakteristisch für solche Sprachen ist, dass sie das zu lösende Problem, aber nicht den Lösungsweg beschreiben und dass das DBMS stets Mengen von Datenobjekten liefert, die die Anfrage erfüllen. Anfragen werden mit Hilfe des *Anfrage-Parsers* (*query parser*) einer lexikalischen Analyse und einer Syntaxanalyse unterzogen und bei Korrektheit in eine äquivalente, interne, aber effizientere Form, z.B. einem Syntaxbaum, überführt. *Anwendungsprogrammierer* (*application programmer*) interagieren mit dem System durch Kommandos einer *Datenmanipulationssprache* (*data manipulation language, DML*), die in ein Programm einer *Gastsprache* (*host language*) eingebettet werden. Die Syntax einer DML ist gewöhnlich sehr verschieden von der

Syntax der Gastsprache. Der *DML-Präprozessor* (*DML preprocessor*) filtert die besonders markierten DML-Kommandos aus dem Anwendungsprogramm heraus und führt sie dem *DML-Übersetzer* (*DML compiler*) zu, der sie in Objektcode übersetzt. Die Objektcodes für die DML-Kommandos und das restliche Programm, das durch den Gastsprachübersetzer (*host language compiler*) übersetzt wird, werden danach gebunden.

Der *Systemkatalog-Manager* (*system catalog manager, dictionary manager*) steht zu fast allen Komponenten des DBMS in Verbindung und verwaltet den *Systemkatalog* (*system catalog, data dictionary*). Der Systemkatalog ist wie die Datenbank auf einem externen Speichermedium abgelegt und enthält *Meta-Daten* (*meta data*) („Daten über Daten“) über die Struktur der Datenbank. Beispiele für Metadaten sind Beschreibungen der Daten, Angaben zu deren Beziehungen untereinander, Beschreibungen der Programme, Konsistenzbedingungen, Angaben über Zugriffsbefugnisse und Speicherdetails.

1.5 Weitere Komponenten eines Datenbanksystems

Zusätzlich zu den oben beschriebenen Softwarekomponenten gibt es eine Reihe von unterschiedlichen *Werkzeugen* (*tools*), die die Entwicklung von Anwendungen auf Datenbanken erleichtern, und *Hilfsprogrammen* (*utilities*), die den Datenbankadministrator bei seiner Arbeit unterstützen. Werkzeuge gibt es nicht nur für den professionellen Anwendungsprogrammierer, sondern auch für den Endbenutzer, dem es ermöglicht wird, Anwendungen zu erzeugen, ohne selbst konventionell programmieren zu müssen. Beispiele sind *Abfragesysteme*, die es auch dem Nicht-Fachmann erlauben, ad hoc Anfragen an das System zu stellen, *Reportgeneratoren*, die formatierte Berichte erzeugen und Berechnungen auf Daten ausführen, Spreadsheets, Werkzeuge zur Erstellung von Geschäftsgraphiken, Werkzeuge für den Datenbankentwurf und CASE-Werkzeuge für den Entwurf von Datenbankanwendungen.

Beispiele für wichtige Hilfsprogramme sind die folgenden. Eine *Import-Funktion* erlaubt es, in einem genau spezifizierten Format vorliegende Dateien (z.B. Textdateien oder sequentielle Dateien) in eine Datenbank einzuladen. Hierdurch können auf recht einfache Weise gleich oder ähnlich strukturierte Massendaten in Objekte der Datenbank überführt werden. Die *Export-Funktion* erzeugt eine Sicherungskopie der Datenbank in einem gewünschten, genau spezifizierten Format. Dies ist für Recovery-Maßnahmen bei einem Systemabsturz von großem Nutzen. Ferner kann diese Funktion zur Archivierung von Daten und zum Austausch von Daten mit anderen DBMS benutzt werden. Ein Programm zur *Dateireorganisation* ermöglicht es, den physischen Aufbau einer Datei der Datenbank umzustrukturieren, um eine größere Performance und Effizienz zu erreichen. Ein Programm zur *Leistungsüberwachung* kontrolliert die Auslastung des DBMS und liefert statistische Daten zur Entscheidung, ob und welche Maßnahmen ergriffen werden müssen, um die Performance des Systems zu steigern.

1.6 Aufgaben

Aufgabe 1.1: Zwei Sachbearbeiter einer Universitätsverwaltung seien mit unterschiedlichen Aufgaben betraut. Sachbearbeiter *A* im Prüfungsamt ist für die Verwaltung der Leistungsdaten der Studenten zuständig. Er erfasst für jeden Studenten neben privaten Daten die Ergebnisse der mündlichen Prüfungen, Klausuren, Seminare usw. Sachbearbeiter *B* im Studentensekretariat verwaltet neben privaten Daten die belegten Kurse/Vorlesungen der Studenten.

Skizzieren Sie für dieses Szenario eine traditionelle Datenorganisation, indem Sie davon ausgehen, dass jeder Sachbearbeiter eine, auf die jeweilige Anwendung zugeschnittene Datei verwendet, und erläutern Sie beispielhaft die in Abschnitt 1.1 genannten Nachteile der traditionellen Datenorganisation.

Aufgabe 1.2: Überlegen und beschreiben Sie, welche Nachteile Datenbanksysteme haben können.

1.7 Literaturhinweise

Zu Datenbanksystemen allgemein gibt es eine Vielzahl von Büchern, von denen hier nur einige erwähnt werden können. Diese Bücher decken weitestgehend den Gesamtbereich der Datenbanktechnologie ab und behandeln teilweise auch Architektur- und Implementierungsaspekte.

Sehr gute englischsprachige Darstellungen finden sich in den Büchern von Elmasri & Navathe (2000) und von Silberschatz, Korth & Sudarshan (2002). Ein weiteres Buch, das auch verschiedene Nichtstandarddatenbanksysteme beschreibt, stammt von Ramakrishnan (1997). „Klassiker“ der englischsprachigen Datenbankliteratur sind sicherlich das Werk von Date (1995), das weltweit wohl das am meisten verkaufte und eher praktisch orientierte Buch in diesem Bereich ist, und das zweibändige Werk von Ullman (1988, 1989), dessen Gewichtung mehr auf den theoretischen Grundlagen von Datenbanksystemen liegt.

Eine sehr gute und allgemeine, deutschsprachige Einführung in Datenbanksysteme bietet das Buch von Kemper & Eickler (1999), das auch Implementierungsaspekte behandelt. Des Weiteren ist das Buch von Heuer & Saake (2000) zu nennen, das sich auf Datenbankkonzepte und -sprachen konzentriert und auch neuere Entwicklungstrends betrachtet, aber auf die Beschreibung von Implementierungskonzepten gänzlich verzichtet.

Der Schwerpunkt in allen genannten Büchern liegt auf relationalen Datenbanksystemen als der aktuellen, marktbeherrschenden Datenbanktechnologie. Überblickhaft behandelt werden allerdings auch neuere Technologien wie objekt-orientierte, objekt-relationale, wissensbasierte, deduktive, geometrische, temporale, statistische und wissenschaftliche Datenbanksysteme, Entscheidungsunterstützungssysteme,

Netzwerkdatenbanksysteme, Hauptspeicherdatenbanken, geographische Informationssysteme, Bild- und Video-Datenbanken, Multimedia-Datenbanken und Textdatenbanken.

Bücher, die sich ausdrücklich und ausführlich mit Implementierungskonzepten für Datenbanksysteme beschäftigen, sind selten. Eine sehr gute englischsprachige Darstellung ist das Buch von Garcia-Molina, Ullman & Widom (2000). Deutschsprachige Ausführungen finden sich in den Büchern von Saake & Heuer (1999) und Härder & Rahm (1999). Letzteres ist aus dem bekannten "Datenbank-Handbuch" von Lockemann & Schmidt (1987) entstanden.

Weiterhin gibt es auch einige wichtige Fachzeitschriften zum Thema Datenbanken wie z.B.

ACM Transactions on Database Systems (TODS)

ACM SIGMOD Record

IEEE Transactions on Knowledge and Data Engineering (TKDE)
Information Systems

The VLDB Journal

Die wichtigsten regelmäßigen Konferenzen sind

ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS)

ACM SIGMOD International Conference on Management of Data (SIGMOD)

Datenbanksysteme für Büro, Technik, Wissenschaft (BTW)

European Conference on Database Technology (EDBT)

International Conference on Data Engineering (ICDE)

International Conference on Very Large Databases (VLDB)

Das 3-Ebenen-Modell wurde 1975 in einem Zwischenbericht (ANSI 1975) und 1978 in einem Endbericht (Tsichritzis & Klug (1978)) von der ANSI/SPARC-Arbeitsgruppe aufgestellt und beschrieben. Ziel der Arbeitsgruppe war es, Standardisierungsvorschläge für geeignete Bereiche der Datenbanktechnologie zu entwickeln. Die Arbeitsgruppe kam zum Schluss, dass Schnittstellen die einzige Komponente eines Datenbanksystems sind, die für eine Standardisierung geeignet sind. Daher wurde eine verallgemeinerte Datenbankarchitektur entworfen, die die Bedeutung solcher Schnittstellen betont. Trotz seines Alters hat dieses Modell immer noch Bestand. Beschreibungen dieses Modells befinden sich natürlich auch in allen oben genannten Lehrbüchern.

Das 3-Ebenen-Modell betont insbesondere die Aspekte der Datenunabhängigkeit, der Integration von Datenbeständen und der benutzerspezifischen Sicht auf Teile der Datenbank. Ein anderes Modell, die sogenannte *Fünf-Schichten-Architektur* (Lockemann & Schmidt (1987), Härder & Rahm (1999)), setzt andere Schwerpunkte und betont den Schichten- und den Schnittstellenaspekt. Es ähnelt ein wenig der in Abschnitt 1.4 beschriebenen Softwarearchitektur eines DBMS. Ein Datenbanksystem wird in fünf Schichten zerlegt, wobei eine Schicht mittels einer Schnitt-

stelle der direkt über ihr liegenden Schicht Objekte und Operatoren zu deren Realisierung zur Verfügung stellt. Die Struktur der Objekte und die Realisierung der Operatoren sind nach außen hin nicht sichtbar. Die fünf Schichten sind (1) Externspeicherverwaltung, (2) Systempufferverwaltung, (3) Record-Manager, Zugriffspfadverwaltung, Sperrverwaltung, Recovery-Komponente, (4) Systemkatalog, Transaktionsverwaltung und (5) Zugriffspfadoptimierung, Zugriffskontrolle, Integritätskontrolle. Sechs Schnittstellen werden unterschieden: (1) die Geräteschnittstelle, (2) die Dateischnittstelle, (3) die Systempufferschnittstelle, (4) die interne Satzschnittstelle, (5) die satzorientierte Schnittstelle und (6) die mengenorientierte Schnittstelle.

Implementierungskonzepte für Datenbanksysteme

Schneider, M.

2004, XVI, 334 S., Softcover

ISBN: 978-3-540-41962-4