1

Was sind Datenbanken?

Der Leser dieses Buches hat sicher bereits eine Vorstellung davon, was man unter Datenbanken zu verstehen hat. Allerdings wird dieser Begriff im Alltagsdeutsch sehr großzügig benutzt, sodass wir doch mit einer etwas genaueren Charakterisierung dieses Begriffes beginnen wollen.

1.1 Warum Datenbanken?

Daten ist die Pluralform von *Datum*, lateinisch für das Geschriebene, das Gegebene, Wert, Tatsache, Information. Eine Datenbank ist nun eine Bank für Daten, also im übertragenen Sinne ein sicherer Aufbewahrungsort für (wertvolle) Informationen. Diese übertragene Bedeutung einer Art Geldinstitut für Daten gibt uns bereits einige Charakteristika von Datenbanken:

- Eine Datenbank hat die (langfristige) *Aufbewahrung* von Daten als Aufgabe.
- Die *Sicherheit vor Verlusten* ist eine Hauptmotivation, etwas „auf die Bank zu bringen“.
- Eine Bank bietet *Dienstleistungen für mehrere Kunden* an, um effizient arbeiten zu können.

Eine Bank kostet Gebühren, auch eine Datenbank verursacht Kosten, also bringt man natürlich nur Daten auf die Bank, für die sich das tatsächlich lohnt. In einer Datenbank bewahren wir daher operative, sozusagen für einen Betrieb „lebenswichtige“ Daten, sehr große Datenbestände, langfristig vorzuhaltende und von vielen gleichzeitig zu nutzende Daten auf.

Betrachten wir als typisches Beispiel die Daten einer Hotelkette. In einer Datenbank aus obigen Gründen zu speichernde Informationen sind hier Reservierungen von Kunden und Preise sowie Verfügbarkeiten von Zimmern, sowie alle Angaben für die Rechnungslegung. Hingegen würden die Geburtstage und Hochzeitstage der Geschäftsführung nicht unbedingt in einer Datenbank aufzubewahren sein.

Für ein Hotel ist es wichtig, die Rechnungsdaten längere Zeit korrekt zu speichern (Regressansprüche von Gästen, Jahresumsatz), sowie verschiedene Arten von Software auf diese Daten zugreifen zu lassen (Rechnungserstellung, Umsatzberechnung, Vorratshaltung der Hotelküche, Datenaustausch zu Buchungsportalen, Erstellung von Gästeprofilen etc.).

Eine Datenbank hat nun die Aufgabe, die Daten in strukturierter Form zur Verwendung durch mehr als ein Software-System zu speichern. Die obigen Beispiele zeigen unterschiedlichen Zugriffsbedarf: In einem Fall werden die Daten eines Gastes analysiert, im Fall der Vorratshaltung die Nachbestellung von Lebensmitteln ausgelöst. Es ist daher sinnvoll, die Daten in einem Format zu speichern, das für viele Zugriffsarten offen ist. Bei Datenbanken ist insbesondere die Speicherung in *Tabellenform in SQL-Datenbanken* verbreitet.

Die Abbildung von Datenbeständen in eine Tabellenform wird uns in diesem Buch intensiv beschäftigen. Die oben genannten Daten liegen durchaus teilweise bereits in Tabellenform vor. Auf einer Rechnung werden aber auch verschiedene Informationen gemischt, die man in einer Datenbank trennen sollte. Auch macht es Sinn, bestimmte Daten (etwa die Adresse des Gastes oder die Informationen zum Zimmertyp) nur einmal abzuspeichern, um Tippfehler bei der erneuten Eingabe von vornherein zu verhindern. Für unser Beispiel einer Hotelrechnung bedeutet dies, dass wir mehrere Tabellen zur Speicherung nutzen sollten:

- Eine Gästetabelle speichert relevante Informationen des Gastes (Namen, Adresse etc.).
- Analog gibt es Tabellen mit Informationen zu Zimmertypen, Zimmern und – im Falle einer Hotelkette – zu den einzelnen Hotels.
- Weitere Tabellen können Angaben zu reservierten sowie belegten (und damit in Rechnung zu stellenden) Zimmern verwalten.
- Eine weitere Tabelle enthält die Daten einzelner Rechnungen.

Abbildung 1.1 zeigt die Daten des Rechnungsausschnitts in einer derartigen Tabellenform.

Die Einträge in den verschiedenen Tabellen werden über die identifizierenden Nummern miteinander verknüpft. Die Speicherung in Tabellen erscheint bei den ersten Einträgen nicht besonders sinnvoll, da es sich ja jeweils nur um eine einzelne Zeile handelt – wir dürfen aber nicht vergessen, dass wir in diesen

Tabellen die Daten *aller* Gäste und *aller* Rechnungen über mehrere Jahrzehnte speichern wollen.

Datenbanksysteme sind nun Software-Systeme, die derartig strukturierte Datenbestände verwalten. Die folgenden Abschnitte dieses Kapitels sollen diese Systeme, die Anforderungen und deren Eigenschaften genauer charakterisieren.

Gast	Knr	Vorname	Name
	103	Lilo	Pause

Rechnung	RNr	Datum	KNr	Status
	1014	03.07.2020	103	beendet

Zimmer	ZNr	Zimmertyp	Preis
	201	Einzelzimmer	99.00
	202	Doppelzimmer	129.00
	203	Suite	199.00

Übernachtungen	RNr	ZNr	Anzahl_Nächte
	1014	202	4
	1014	203	1

Abbildung 1.1: Rechnungsdaten in Tabellenform

1.2 Datenbanksysteme

Die Rolle der Datenbanksysteme ist eine sehr elementare: Sie sollen andere Softwaresysteme wie Buchhaltungssysteme, Rechnungslegungssysteme, Zimmerverwaltungssysteme, Schnittstellen zu Buchungsportalen und weitere Anwendungssoftware mit standardisierten Informationen unterstützen.

Das Problem der Datenredundanz

Ohne den Einsatz von Datenbanksystemen tritt dabei das Problem der *Datenredundanz* auf. Die Anwendungssoftware verwaltet in diesem Szenario jeweils ihre eigenen Daten in ihren eigenen Dateien, jeweils in eigenen speziellen Formaten. Ein typisches Szenario in unseren Hotels gibt die folgende Auflistung wieder:

- Die Buchhaltung speichert Gäste-, Adressinformationen sowie Reservierungen und Belegungen.
- In der Vorratshaltung der Küche werden Reservierungs- und Belegungszahlen benötigt.
- Das Marketing braucht die Zahlen der Vorreservierungen, außerdem zur Direktwerbung die Gäste- und Adressinformationen und die Gästeprofile.

In diesem Szenario sind die Daten *redundant*, also mehrfach gespeichert. Etwa werden die Gästeadressen und die Reservierungen/Belegungen von mehreren Anwendungen in jeweils eigenen Dateien verwaltet. Die entstehenden Probleme sind Verschwendung von Speicherplatz und „Vergessen“ von lokalen Änderungen, die typisch für das Fehlen einer zentralen, „genormten“ Datenhaltung sind. Ein Ziel der Entwicklung von Datenbanksystemen ist die Beseitigung der Datenredundanz. Die Situation wird in Abbildung 1.2 verdeutlicht.

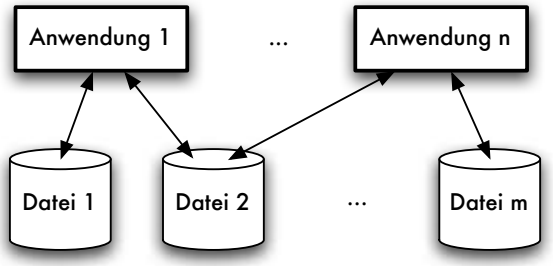

Abbildung 1.2: Zugriff auf Dateien ohne spezielle Verwaltung

Auch in der Anwendungserstellung führt der fehlende Einsatz einer zentralen Datenhaltungskomponente zu erheblichen Defiziten. Die Anwendungsprogrammierer oder auch Endanwender können Anwendungen nicht programmieren bzw. benutzen, ohne

- die interne Darstellung der Daten sowie
- Speichermedien oder Rechner (bei verteilten Systemen)

zu kennen. Dieses Problem wird als fehlende *Datenunabhängigkeit* bezeichnet und in Abschnitt 1.5 noch intensiver diskutiert. Auch ist die Sicherstellung der *Zugriffskontrolle* und der *Datensicherheit* ohne zentrale Datenhaltung nicht gewährleistet.

Vermeidung der Datenredundanz durch Datenbanksysteme

Die obigen Probleme können mit der Datenbanktechnologie gelöst werden. Wir sprechen dann im Gegensatz zur Datenredundanz von einer *Datenintegration*. Das Prinzip der Datenintegration basiert auf folgenden Überlegungen:

Die gesamte Anwendungssoftware arbeitet auf denselben Daten, die in einer zentralen Datenhaltungskomponente verwaltet werden. Der Gesamtbestand der Daten wird nun als *Datenbank* bezeichnet.

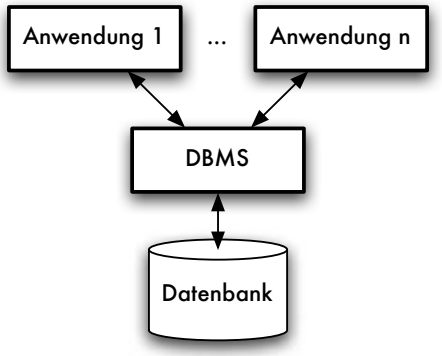

Abbildung 1.3: Datenbankmanagementsysteme

Diese Architekturvorstellung wird in Abbildung 1.3 grafisch verdeutlicht. Eine derartige Datenbank muss natürlich äußerst sorgfältig entworfen und in einer geeigneten Datendefinitionssprache beschrieben werden. Auch andere Probleme im Umgang mit großen Datenbeständen, etwa Fragestellungen der Effizienz, Parallelität, Zugriffskontrolle und Datensicherheit, können mit heutigen kommerziellen Datenbankmanagementsystemen (DBMS) zufriedenstellend gelöst werden. Diese Systeme zeichnen sich durch folgende Eigenschaften aus:

- Datenbanksysteme können große Datenmengen effizient verwalten. Sie bieten benutzergerechte *Anfragesprachen* an, die eine komfortable Anfrageformulierung ohne Rücksichtnahme auf die interne Realisierung der Datenspeicherung ermöglichen. Eine interne *Optimierung* ermöglicht trotzdem einen effizienten Zugriff auf die Datenbestände.
- Viele Benutzer können parallel auf Datenbanken arbeiten. Das *Transaktionskonzept* verhindert hier unerwünschte Nebeneffekte beim Zugriff auf gemeinsam genutzte Daten.
- Die *Datenunabhängigkeit* wird durch ein Drei-Ebenen-Konzept gewährleistet, das eine externe Ebene der Anwendungssicht, eine konzeptuelle Ebene der logischen Gesamtsicht auf den Datenbestand und eine interne Ebene der implementierten Datenstrukturen unterscheidet.
- Zugriffskontrolle (kein unbefugter Zugriff) und Datensicherheit (kein – ungewollter – Datenverlust) werden vom System gewährleistet.

1.3 Anforderungen: Die Codd'schen Regeln

Im Laufe der Jahre hat sich eine Basisfunktionalität herauskristallisiert, die von einem Datenbankmanagementsystem erwartet wird. Bereits Anfang der 80er Jahre hat Edgar F. („Ted“) Codd diese Anforderungen in einer Liste von neun Punkten zusammengefasst, die noch heute ihre Gültigkeit besitzt [Cod82]:

1. Integration

Die Datenintegration erfordert die *einheitliche* Verwaltung *aller* von Anwendungen benötigten Daten. Hier verbirgt sich die Möglichkeit der kontrollierten nicht-redundanten Datenhaltung des gesamten relevanten Datenbestands.

2. Operationen

Auf der Datenbank müssen Operationen möglich sein, die Datenspeicherung, Suchen und Änderungen des Datenbestands ermöglichen.

3. Katalog

Der Katalog, auch „Data Dictionary“ genannt, ermöglicht Zugriffe auf die Datenbeschreibungen der Datenbank.

4. Benutzersichten

Für unterschiedliche Anwendungen sind unterschiedliche Sichten auf den Datenbestand notwendig, sei es in der Auswahl relevanter Daten oder in einer angepassten Strukturierung des Datenbestands. Die Abbildung dieser speziellen Sichten auf den Gesamtdatenbestand muss vom System kontrolliert werden.

5. Konsistenzüberwachung

Die Konsistenzüberwachung, auch als *Integritätssicherung* bekannt, übernimmt die Gewährleistung der Korrektheit von Datenbankinhalten und der korrekten Ausführung von Änderungen, sodass diese die Konsistenz nicht verletzen können.

6. Zugriffskontrolle

Aufgabe der Zugriffskontrolle ist der Ausschluss unautorisierter Zugriffe auf die gespeicherten Daten. Dies umfasst datenschutzrechtlich relevante Aspekte personenbezogener Informationen ebenso wie den Schutz firmenspezifischer Datenbestände vor Werksspionage.

7. Transaktionen

Unter einer Transaktion versteht man eine Zusammenfassung von Datenbank-Änderungen zu Funktionseinheiten, die als Ganzes ausgeführt werden sollen und deren Effekt bei Erfolg permanent in der Datenbank gespeichert werden soll.

8. Synchronisation

Konkurrierende Transaktionen mehrerer Benutzer müssen synchronisiert werden, um gegenseitige Beeinflussungen, etwa versehentliche Schreibkonflikte auf gemeinsam benötigten Datenbeständen, zu vermeiden.

9. Datensicherung

Aufgabe der Datensicherung ist es, die Wiederherstellung von Daten etwa nach Systemfehlern zu ermöglichen.

Basierend auf diesen Anforderungen können wir einige grundlegende Begriffsbildungen vornehmen, die uns im Laufe dieses Buches begleiten werden:

Unter dem Begriff *Datenbankmanagementsystem*, kurz DBMS, verstehen wir die Gesamtheit der Software-Module, die die Verwaltung einer Datenbank übernehmen. Ein *Datenbanksystem*, kurz DBS, ist die Kombination eines DBMS mit einer Datenbank[1].

Diese Begriffsbildung ist für das Verständnis der Datenbankkonzepte essenziell und wird in Tabelle 1.1 zusammengefasst.

Kürzel	**Begriff**	**Erläuterung**
DB	Datenbank	Strukturierter, von einem DBMS verwalteter Datenbestand
DBMS	Datenbankmanagementsystem	Software zur Verwaltung von Datenbanken
DBS	Datenbanksystem	DBMS plus Datenbank(en)

Tabelle 1.1: Begriffsbildungen für Datenbanksysteme

Grundmerkmale von modernen Datenbanksystemen sind die folgenden, die eine direkte Umsetzung der aufgeführten neun Punkte von Codd darstellen:

- DBMS verwalten *persistente* (langfristig zu haltende) Daten, die einzelne Läufe von Anwendungsprogrammen überstehen sollen.

[1] Vereinfachend werden wir im Verlaufe dieses Buches ein Datenbankmanagementsystem auch als Datenbanksystem bezeichnen, wenn es aus dem Kontext ersichtlich ist, dass hier keine konkrete Bindung an eine Datenbank vorliegt.

- Sie haben die Aufgabe, *große* Datenmengen *effizient* zu verwalten.
- DBMS definieren ein *Datenbankmodell*, mit dessen Konzepten alle Daten *einheitlich beschrieben* werden.
- Sie stellen *Operationen und Sprachen* (Datendefinitionssprache, interaktive Anfragesprachen, Datenmanipulationssprachen ...) zur Verfügung. Derartige Sprachen sind *deskriptiv*, verzichten also auf die explizite Angabe von Berechnungsschritten. Die Sprachen sind getrennt von einer Programmiersprache definiert. Für die Speicherung in Tabellenform in relationalen Datenbanken bildet die Sprache *SQL* hierbei den Standard.
- Datenbankmanagementsysteme unterstützen das *Transaktionskonzept* inklusive *Mehrbenutzerkontrolle*: Logisch zusammenhängende Operationen werden zu Transaktionen zusammengefasst, die als atomare (unteilbare) Einheit bearbeitet werden. Auswirkungen von Transaktionen sind langlebig. Transaktionen können parallel durchgeführt werden, wobei sie voneinander isoliert werden.
- Sie unterstützen die Einhaltung des *Datenschutzes*, gewährleisten *Datenintegrität* (Konsistenz) und fördern die *Datensicherheit* durch geeignete Maßnahmen.

1.4 DBMS-Architektur

Abbildung 1.4 zeigt einen Überblick über die prinzipielle Aufteilung eines Datenbankmanagementsystems in Funktionsmodule, angelehnt an eine Aufteilung in drei Abstraktionsebenen. Die *externe Ebene* beschreibt die Sicht, die eine konkrete Anwendung auf die gespeicherten Daten hat. Da mehrere angepasste externe Sichten auf eine Datenbank existieren können, gibt die *konzeptuelle Ebene* eine logische und einheitliche Gesamtsicht auf den Datenbestand. Die *interne Ebene* beschreibt die tatsächliche interne Realisierung der Datenspeicherung.

Die in Abbildung 1.4 gezeigten Komponenten können wie folgt kurz charakterisiert werden:

- **Dateiorganisation:** Definition der Dateiorganisation und Zugriffspfade auf der internen Ebene
- **Datendefinition:** Konzeptuelle Datendefinition (konzeptuelles Schema)
- **Sichtdefinition:** Definition von Benutzersichten (externe Ebene)
- **Masken:** Entwurf von Menüs und Masken für die Benutzerinteraktion

- **Einbettung:** Einbettung von Konstrukten der Datenbanksprache in eine Programmiersprache
- **Anfragen/Updates:** Interaktiver Zugriff auf den Datenbestand
- **DB-Operationen:** Datenbank-Operationen (Anfrage, Änderungen)
- **Optimierer:** Optimierung der Datenbankzugriffe
- **Plattenzugriff:** Plattenzugriffssteuerung bzw. Ansteuerung anderer Speichermedien
- **Auswertung:** Auswertung von Anfragen und Änderungen
- **P1 ... Pn:** Verschiedene Datenbank-Anwendungsprogramme
- **Data Dictionary** (oder auch Datenwörterbuch): Zentraler Katalog aller für die Datenhaltung relevanten Informationen

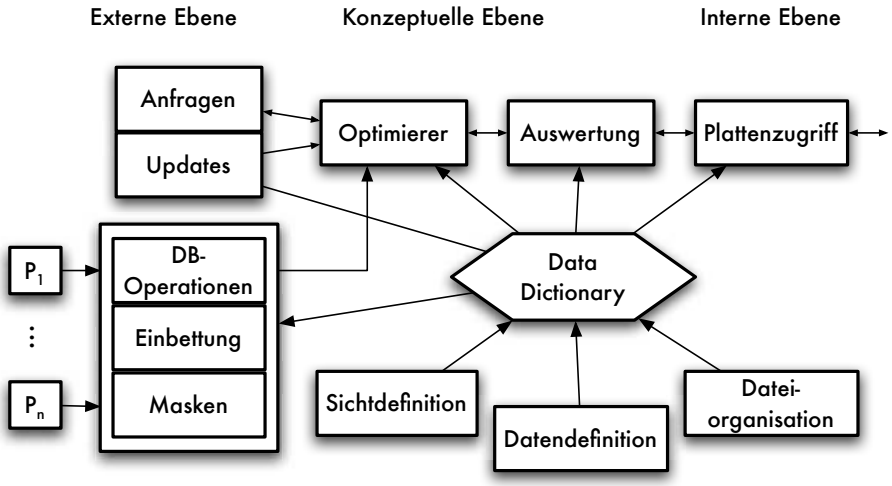

Abbildung 1.4: Vereinfachte Architektur eines DBMS

In den folgenden Abschnitten werden wir einzelne Komponenten kurz erläutern. Hierzu diskutieren wir exemplarisch einige Funktionen, die von einem Datenbankmanagementsystem ausgeführt werden müssen, sowie die zugehörigen datenbankspezifischen Sprachen.

1.5 Datenunabhängigkeit

Ein wesentlicher Aspekt bei Datenbankanwendungen ist die Unterstützung der *Datenunabhängigkeit* durch das Datenbankmanagementsystem. Sowohl Datenbanken als auch Anwendungssysteme haben in der Regel eine lange Lebensdauer, während derer sowohl die Realisierung der Datenspeicherung als auch externe Schnittstellen aus verschiedensten Gründen modifiziert oder erweitert werden. Das Konzept der Datenunabhängigkeit hat das Ziel, eine (oft langlebige) Datenbank von notwendigen Änderungen der Anwendung abzukoppeln (und umgekehrt). Die Datenunabhängigkeit kann in zwei Aspekte aufgeteilt werden:

- Die *Implementierungsunabhängigkeit* oder auch *physische Datenunabhängigkeit* bedeutet, dass die konzeptuelle Sicht auf einen Datenbestand unabhängig von der für die Speicherung der Daten gewählten Datenstruktur besteht.

- Die *Anwendungsunabhängigkeit* oder auch *logische Datenunabhängigkeit* hingegen koppelt die Datenbank von Änderungen und Erweiterungen der Anwendungsschnittstellen ab.

Zur Unterstützung der Datenunabhängigkeit in Datenbanksystemen wurde bereits in den 70er Jahren von der „ANSI/X3/SPARC[2] Study Group on Database Management Systems“ eine *Drei-Ebenen-Schema-Architektur* als Ergebnis einer mehrjährigen Studie vorgeschlagen. ist das Kürzel für die amerikanische Standardisierungsbehörde „American National Standards Institute“. Die dort vorgeschlagene Aufteilung in drei Ebenen ist im Datenbankbereich inzwischen allgemein akzeptiert. Abbildung 1.5 zeigt die diesem ANSI-Vorschlag folgende, prinzipielle Schema-Architektur.

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

- Das *interne Schema* beschreibt die systemspezifische Realisierung der Datenbank, etwa die eingerichteten Zugriffspfade. Die Beschreibung des internen Schemas ist abhängig vom verwendeten Basissystem und der von diesem angebotenen Sprachschnittstelle.

- Das *konzeptuelle Schema* beinhaltet eine implementierungsunabhängige Modellierung der gesamten Datenbank in einem systemunabhängigen Datenmodell, zum Beispiel dem ER-Modell oder dem relationalen Modell. Das konzeptuelle Schema beschreibt die Struktur der Datenbank vollständig.

[2] SPARC steht für Standards Planning and Requirements Committee.

- Basierend auf dem konzeptuellen Schema können mehrere *externe Schemata* definiert werden, die anwendungsspezifische (Teil-)Sichten auf den gesamten Datenbestand festlegen.

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

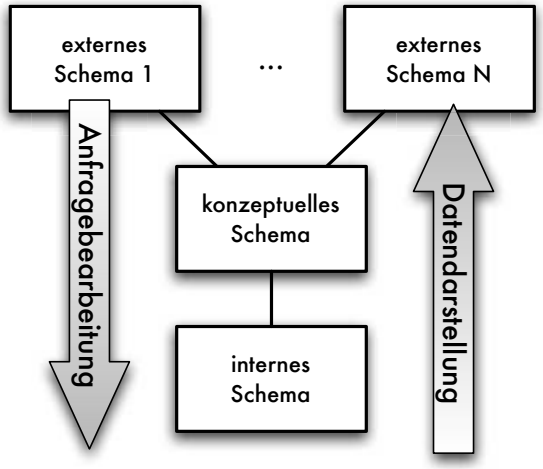

Abbildung 1.5: Drei-Ebenen-Schemaarchitektur für Datenbanken

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

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

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

- Die *Anfragebearbeitung* erfordert eine Übersetzung von Anfragen und Änderungsoperationen, die bezüglich der externen Schemata formuliert wurden, in Operationen auf den internen Datenstrukturen (über den Zwischenschritt der konzeptuellen Ebene).

- Die *Datendarstellung* erfordert eine Transformation in der umgekehrten Richtung: Die internen Datenstrukturen von Anfrageergebnissen müssen derart transformiert werden, dass sie den Beschreibungskonzepten der externen Darstellungen entsprechen.

Ebenen-Architektur am Beispiel

Wir wollen die Basisidee der Drei-Ebenen-Architektur von Datenbankschemata im Folgenden anhand einer kleinen Beispielmodellierung diskutieren. Die modellierte Datenbank enthält Daten über Produkte und deren Lieferanten. Die konzeptuelle Gesamtsicht ist im relationalen Datenbankmodell beschrieben.

Die konzeptuelle Gesamtsicht

Die konzeptuelle Gesamtsicht erfolgt in relationaler Darstellung. Die Datenbank ist in zwei Relationen gespeichert, wie in Abbildung 1.6 dargestellt.

Zimmer

ZimmerId	Zimmertyp	Preis	HotelId → Hotels.HotelId
O-201	Einzelzimmer	99.0	901
O-203	Suite	199.0	901
M-205	Suite	129.0	902

Hotels

HotelId	HotelName
901	Ostseetraum
902	Mecklenburger Hof

Abbildung 1.6: Konzeptuelle Beispieldatenbank in Relationendarstellung

Schlüssel, also identifizierende Attribute in Relationen, werden durch Unterstreichung gekennzeichnet: Die Einträge in diesen Spalten der Tabelle müssen eindeutig sein. Bezüge zwischen Relationen, die sogenannten Fremdschlüssel als Verweise auf Schlüssel, sind in der Beispielrelation mit dem Bezug zu dem Schlüssel einer anderen Relation angegeben.

Externe Sichten

Eine mögliche Anwendungssicht wäre dadurch gegeben, dass die Daten in *einer* Relation dargestellt werden, wobei die `HotelId` ausgeblendet werden soll (wie in Abbildung 1.7 gezeigt).

Diese externe Sicht kann in SQL-Datenbanksystemen einfach durch eine Sicht-Definition (**`create view`**, vgl. Abschnitt 8.1) definiert werden.

Dieses erste Beispiel definiert eine flache Tabelle als Sicht auf andere flache Tabellen, verlässt also den verwendeten Beschreibungsrahmen im gewissen Sinne nicht. Aber auch Sichten in einem anderen Datenbankmodell sind möglich, etwa als eine hierarchisch aufgebaute Relation wie in Abbildung 1.8.

ZimmerId	Zimmertyp	Preis	HotelName
O-201	Einzelzimmer	99.0	Ostseetraum
O-203	Suite	199.0	Ostseetraum
M-205	Suite	129.0	Mecklenburger Hof

Abbildung 1.7: Externe Sicht auf zwei Relationen, dargestellt als eine Relation

Hotels	Zimmer		
	ZimmerId	Bezeichnung	Preis
Ostseetraum	O-201	Einzelzimmer	99.0
	O-203	Suite	199.0
Mecklenburger Hof	M-205	Suite	129.0

Abbildung 1.8: Externe Sicht als hierarchisch aufgebaute Relation

Diese externe Darstellung ist in den meisten SQL-Datenbanken nicht möglich, entspricht aber der hierarchischen Darstellung von Tabellen, wie sie in vielen Anwendungen üblich ist. Eine derartige Datenrepräsentation ist allerdings in *objektrelationalen Datenbanken* möglich, die Sie später noch kennenlernen werden (Kapitel 13).

Interne Darstellung

Für die interne Darstellung kann ein Datenbankmanagementsystem optimierte Datenstrukturen verwenden, etwa eine Baumstruktur über die ZimmerIDs und eine Hash-Tabelle über die Namen der Hotels. Die Verbindung zwischen den beiden Datenstrukturen könnte dadurch erfolgen, dass die Fremdschlüsseleigenschaft des Attributs `HotelId` in die Definition physischer Zeiger umgewandelt wird. Die interne Speicherung von Relationen wird in Abschnitt 10.2 noch erläutert.

1.6 Transaktionen

Ein wichtiges Merkmal von Datenbanksystemen, das für den Einsatz im Mehrbenutzerbetrieb von großer Bedeutung ist, ist die Unterstützung des Transaktionskonzepts. Unter einer *Transaktion* versteht man eine Folge von Datenbankoperationen, die einen konsistenten Datenbankzustand in einen neuen, wiederum konsistenten Zustand überführen und dabei entweder vollständig oder gar nicht ausgeführt werden. Aus Sicht des Anwendungsentwicklers kann eine Transaktion damit als Einheit zur Konsistenzbewahrung angesehen wer-

den. Ein typisches Beispiel für eine Transaktion ist eine Banküberweisung, die aus zwei Datenbankoperationen besteht: die Abbuchung von einem Konto und die Gutschrift des abgebuchten Betrags zum zweiten Konto. Diese Operationsfolge muss entweder ganz oder gar nicht ausgeführt werden: Sollte zwischen beiden Operationen ein Fehler auftreten oder die zweite Operation nicht durchgeführt werden können, so muss auch die Wirkung der ersten Operation rückgängig gemacht werden.

Trotz Einhaltung der semantischen Integrität durch eine einzelne Transaktion können jedoch fehlerhafte Datenbankzustände eintreten, indem sich parallel laufende Transaktionen im Mehrbenutzerbetrieb gegenseitig beeinflussen. Daher wird gefordert, dass Transaktionen das *ACID-Prinzip* einhalten müssen, auf das wir in Abschnitt 10.4 näher eingehen.

Aus Anwender- und Entwicklersicht manifestiert sich die Unterstützung des Transaktionskonzepts durch ein Datenbanksystem insbesondere in speziellen Kommandos zur Transaktionssteuerung. Diese Kommandos definieren das erfolgreiche Abschließen der Transaktion (das sogenannte *Commit*) sowie den Abbruch und damit das Zurücksetzen der Datenbank in den Ausgangszustand der Transaktion (das *Rollback*).

1.7 Konkrete Datenbankmanagementsysteme

Relationale Datenbanksysteme (RDBS) sind seit den 80er Jahren in einer Vielzahl auf dem Markt präsent. Zu den wichtigsten RDBS zählen Oracle, IBM Db2, Microsoft SQL-Server sowie PostgreSQL und MySQL. Gemeinsame Merkmale dieser Systeme sind

- eine Drei-Ebenen-Architektur nach ANSI-SPARC,
- eine einheitliche Datenbanksprache (SQL; Structured Query Language),
- eine Einbettung dieser Sprache in kommerzielle Programmiersprachen,
- diverse Werkzeuge für die Definition, Anfrage und Darstellung von Daten und den Entwurf von Datenbank-Anwendungsprogrammen und der Benutzer-Interaktion sowie
- kontrollierter Mehrbenutzerbetrieb, Zugriffskontrolle und Datensicherheitsmechanismen.

Daneben gibt es gerade im PC-Bereich noch viele Dateiverwaltungssysteme mit tabellarischer Benutzeroberfläche, oder Pseudo-Datenbanksysteme, die nicht alle Eigenschaften eines Datenbanksystems besitzen. Wir bezeichnen sie deshalb als Pseudo-RDBS, weil sie zwar Verwaltungssysteme für strukturierte

Dateien sind, aber nicht die volle Funktionalität eines Datenbanksystems erreichen. MS Access ist ein Datenbankprogramm für Windows, das über eine moderne Benutzeroberfläche verfügt, mit der man bequem und grafisch Anfragen, Änderungen, Sichten, Berichte und Anwendungsmodule erstellen kann. Die grafischen Operationen werden auf eine SQL-Variante abgebildet, die leider nicht dem Standard genügt. Es fehlen einige Konzepte von Datenbanksystemen wie kontrollierter Mehrbenutzerzugriff (für PC-Datenbanksysteme auch nicht so interessant) und eine getrennt einstellbare interne Ebene. Einen Anfrageoptimierer in unserem Sinne bietet MS Access auch nicht. Allerdings ist aus Access heraus der Zugriff auf „echte“ DBMS möglich, sodass die komfortable Benutzerschnittstelle und Entwicklungsumgebung auch für diese Systeme genutzt werden kann.

Im Open-Source-Umfeld haben sich einige (oben schon erwähnte) ernst zu nehmende Lösungen für Datenbanksysteme entwickelt. Hier sind insbesondere MySQL[3] und PostgreSQL[4] zu nennen. MySQL ist aufgrund der Schnelligkeit und des geringen Installations- und Administrationsaufwands speziell bei Entwicklern von Web-Anwendungen beliebt. Gegenüber den „großen“ Systemen weist MySQL jedoch einige Einschränkungen auf. So werden nicht alle SQL-Konstrukte unterstützt, die in diesem Buch beschrieben werden. PostgreSQL entstand aus dem Forschungsprototyp Postgres der University of Berkeley, indem die dort verwendete Anfragesprache durch SQL ersetzt wurde. Ein besonderes Merkmal dieses Systems ist die Unterstützung objektrelationaler Features, wie sie in SQL:1999 und SQL:2003 Einzug in den SQL-Standard gefunden haben.

1.8 Einsatzgebiete und Grenzen

Die klassischen Einsatzgebiete von Datenbanksystemen sind die Verwaltung vieler Objekte, zum Beispiel 150.000 Hotels, 10.000 Gäste und 300 Buchungsvorgänge pro Tag. Die zu verwaltenden Objekte gehören allerdings zu relativ wenigen Objekttypen (in unserem Beispiel die fünf Objekttypen Hotel, Zimmertyp, Zimmer, Rechnung, Gast). Die Objekte eines Objekttyps sind immer gleich strukturiert, so hat etwa jedes Zimmer unserer Datenbank dieselben zu speichernden Angaben (oder in Tabellen-Sprechweise: dieselben Spalten).

In diesem Buch werden die Objekttypen immer durch einfache Tabellenstrukturen beschrieben, die in relationalen Datenbanksystemen auf Basis der Sprache SQL verwendet werden.

Normalerweise sind herkömmliche Datenbanksysteme überfordert mit sehr heterogenen Objekten, die zu vielen unterschiedlichen Objekttypen gehö-

[3] http://www.mysql.com
[4] http://www.postgresql.org

ren, sowie stark strukturierten Objekten. Nur objektorientierte und objektrelationale Datenbanksysteme bieten geeignete Konzepte für stark strukturierte Objekte an (siehe Kapitel 13).

Auch die Verwaltung semistrukturierter Daten bereitet Probleme. Semistrukturiert bedeutet dabei, dass Objekte unterschiedlich strukturierte und auch unstrukturierte Anteile haben können. Multimedia-Datenbanken (siehe auch Kapitel 13), die Texte, Bilder und andere unstrukturierte Informationen verwalten können, eignen sich dafür wie auch NoSQL-Datenbanksysteme (siehe Kapitel 12).

1.9 Beispielanwendungen

In den folgenden Kapiteln werden wir alle Konzepte an zwei einheitlichen Beispielen erläutern. Unsere Beispielszenarien sind dabei

- eine Hotel-Anwendung, in der eine Hotelkette Hotels, Zimmer, Zimmertypen, Gäste, Reservierungen und Buchungen sowie Rechnungen verwalten kann, und
- eine Universitäts-Anwendung, in der Studierende und MitarbeiterInnen der Universität, die Ausleihe von Büchern und Modulprüfungen gepeichert werden.

Wir werden sowohl die konzeptuelle Modellierung im Entity-Relationship-Modell und den Datenbankentwurf, als auch die Umsetzung des Datenbankentwurfs im relationalen Datenbankmodell und die Anfragen an die Datenbank mit SQL an diesen Beispielen demonstrieren.

In den Anhängen A und B sind diese Szenarien im Entity-Relationship-Modell vollständig dargestellt. Weiterhin wird jeweils eine relationale Repräsentation angegeben. Im Universitäts-Beispiel ist diese auch in SQL-Datendefinitionsanweisungen umgesetzt. Weiterhin haben wir für das Universitäts-Beispiel auch vollständige Beispieltabellen angegeben, auf der unsere SQL-Anfragen beispielhaft ausgeführt werden können. An einigen Stellen dieses Buches werden wir diese Anwendungen jedoch gegebenenfalls erweitern, um auf spezielle Probleme einzugehen, die mit diesen einfachen Schemata nicht verdeutlicht werden können.

1.10 Übersicht über die Kapitel des Buches

Dieses Buch gibt eine Einführung in grundlegende Konzepte und Techniken der Datenbanksysteme. Dazu werden zuerst einige wesentliche Prinzipien des rela-

tionalen Datenbankmodells im nachfolgenden Kapitel 2 dargestellt. Anschließend folgt eine Einführung in das Entity-Relationship-Modell, das die Grundlage für den logischen Datenbankentwurf ist (Kapitel 3), und eine Darstellung des Datenbankentwurfsprozesses in Kapitel 4. Um eine redundanzfreie Datenbank zu erreichen, müssen die entworfenen Relationen oft noch weiter normalisiert werden. Der Normalisierungsprozess wird in Kapitel 5 vorgestellt.

Relationale Datenbanksysteme und ihre objektrelationalen Erweiterungen sind heute die Standardtechnik für Datenbankanwendungen. SQL ist hier die dominierende Anfragesprache. Sie enthält Anteile sowohl zur Datendefinition (Kapitel 6), zum Abfragen von Datenbanken (Kapitel 7) sowie Möglichkeiten zur Sichtendefinition und Zugriffskontrolle (Kapitel 8) und zur Vereinbarung von Integritätsbedingungen und Triggern (Kapitel 9).

Einige wesentliche Implementierungsaspekte wollen wir uns in Kapitel 10 anschauen. Da moderne Datenbanksysteme auch komplexe Datenanalysen durchführen können, werden wir uns Themen wie Online-Analysen (OLAP), Data Warehousing und Data Mining in Kapitel 11 anschauen. Gerade für die Nutzung in komplexen Analysen wurden neue Datenbanksystem-Architekturen (column stores: spalten- statt der sonst üblichen zeilenweise Speicherung von Tabellen) und flexiblere Datenbankmodelle in NoSQL-Datenbanksystemen entwickelt. Diese sind Gegenstand von Kapitel 12. Im Ausblick werden wir Lösungen für komplexe Datenbankobjekte, für Multimedia-Daten (wie Bilder) und für die Analyse von Big Data skizzieren (Kapitel 13).

Sollten die Darstellungen dieses Buches einmal nicht ausreichen: Für ein tiefergehendes Verständnis der Konzepte und Sprachen von Datenbanksystemen sei auf den ersten Band des Biberbuches von Saake, Sattler und Heuer verwiesen [SSH18], für eine ausführliche Darstellung der internen Funktionsweise und aller Implementierungsaspekte von Datenbanksystemen auf den dazu passenden zweiten Band [HSS19].

1.11 Übungsaufgaben

Übung 1-1 Nennen Sie grundlegende Eigenschaften eines Datenbanksystems.

Übung 1-2 Erklären Sie die logische und physische Datenunabhängigkeit anhand der 3-Ebenen-Architektur.

Übung 1-3 In welchen Bereichen und für welche Aufgaben sind klassische Datenbanksysteme gut beziehungsweise nicht sinnvoll einsetzbar?

Übung 1-4 Denken Sie an Ihren Alltag: Wo begegnen Ihnen Datenbanksysteme? Sind diese immer auf den ersten Blick zu erkennen oder arbeiten sie im Hintergrund?