10 Datenbanksysteme

In vielen Anwendungen müssen große Datenbestände dauerhaft auf Externspeichern verwaltet werden. Stellen diese Daten eine logische Einheit dar, so spricht man von einer *Datenbank*. Im Gegensatz zu rein privaten Datenbanken, die nur von einem Benutzer gepflegt und genutzt werden, wird in diesem Kapitel der für die Praxis relevantere Fall betrachtet, dass eine Datenbank einen gemeinsamen Datenbestand *mehrerer* Benutzer darstellt. Insbesondere muss der gleichzeitige Zugriff verschiedener Benutzer unterstützt werden. Die Software zur Verwaltung einer Datenbank wird auch als *Datenbankverwaltungssystem* (DBVS) (engl. *database management system*) bezeichnet. Der Begriff *Datenbanksystem* (DBS) wird für die Datenbank und ihr zugehöriges DBVS verwendet.

10.1 Datenbanken und Datenbanksysteme

In den 60er Jahren wurde bereits die Notwendigkeit für spezielle DBVS erkannt. Es war damals üblich, die für eine Anwendung relevanten Daten direkt in den Anwendungsprogrammen zu verwalten. Dies führte insbesondere zu folgenden Problemen:

- Die Daten der verschiedenen Anwendungen hatten verschiedene Datenformate und Datenmodelle, so dass ein Austausch zwischen den Anwendungen nur mit sehr hohem Aufwand zu bewerkstelligen war. Dies führte dazu, dass Daten mehrfach in verschiedenen Datenbeständen vorlagen und in sich nicht konsistent waren.
- Die Wartungskosten der vielen Programme zur Datenverwaltung waren immens hoch. Änderte sich die Schnittstelle zu den Daten, mussten alle Programme geändert, neu übersetzt und getestet werden.
- Die Programme unterstützten keinen Mehrbenutzerbetrieb, da im Allgemeinen nur ein Benutzer zu einem Zeitpunkt auf die Datenbank zugreifen konnte.
- Die Anwendungsprogramme waren äußerst komplex, da insbesondere ein hoher Programmieraufwand für die effiziente Ausführung von Anfragen notwendig war.

Durch die Einführung eines DBVS wird eine einheitliche Schnittstelle für den Zugriff auf die Daten angeboten. Alle Benutzer verwenden zwar noch spezielle Anwendungsprogramme, diese delegieren jedoch alle notwendigen Datenzugriffe über die Schnittstelle an das DBVS. Neben dieser einheitlichen Zugriffsschnittstelle bietet ein DBVS im Vergleich zu Individuallösungen noch weitere wichtige Vorteile:

- Das DBVS verwendet im Wesentlichen drei Abstraktionsebenen zur Repräsentation der Daten. Diese sind in Abb. 10.1 veranschaulicht. Auf der *internen Ebene* wird beschrieben, wie Daten auf den Externspeicher abgebildet werden. Die Gesamtheit dieser Abbildungsvorschriften wird auch als *internes Schema* bezeichnet. Auf der *konzeptuellen Ebene* erfolgt durch ein entsprechendes Schema eine Beschreibung der Gesamtheit aller Daten in der Datenbank. Zusätzlich gibt es noch die *externe Ebene*, wo individuelle Schemata für Benutzergruppen definiert werden können. Diese Schemata sind logische Konstruktionen, die, ähnlich einer Schnittstelle in Java, den Zugriff auf die Daten einschränken.

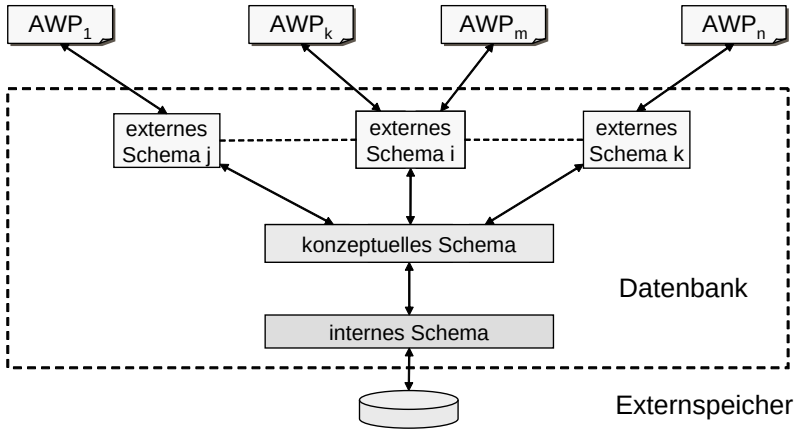

Abb. 10.1: *Drei-Schichten Architektur einer Datenbank*

- Ein Benutzer wird i.A. ein Programm, das *Anwendungsprogramm* (AWP), benutzen, um über ein externes Schema auf die relevanten Daten der Datenbank zuzugreifen. Aufgrund dieser klaren Schichtung führen Änderungen in der Datenbank nicht dazu, dass das Anwendungsprogramm eines Benutzers geändert werden muss. Man spricht dann auch von *Datenunabhängigkeit*. Die Beschreibungen der Schemata werden wiederum als so genannte *Metadaten* in einem Teil der Datenbank (engl. *data dictionary*) gehalten.
- Zusätzlich zu der Möglichkeit, über ein AWP auf die Datenbank zuzugreifen, bieten heutige DBVS eine einfache *ad-hoc Anfragesprache* an, um interaktiv mit der Datenbank zu arbeiten. Der Anforderungskatalog für eine solche Sprache umfasst das Anlegen und Ändern eines Datenbankschemas sowie die Verarbeitung einzelner Objekte in der Datenbank. Die Sprachen bedienen sich vorwiegend deklarativer Konzepte. Dies bedeutet insbesondere, dass bei einer Anfrage nur die Anwortmenge beschrieben wird, nicht aber, wie diese aus dem Datenbestand gewonnen werden soll. Eine wesentliche Aufgabe des DBVS ist es deshalb, aus einer solchen deklarativen Beschreibung einer Antwortmenge einen effizienten Ausführungsplan zu generieren. Dies ist die Aufgabe des *Anfrageoptimierers*, einer der wichtigsten Komponenten eines DBVS.

- DBVS unterstützen einen *Mehrbenutzerbetrieb*, indem intern im DBVS (für den Benutzer unsichtbar) eine Koordinierung der Aktivitäten aller Benutzer stattfindet. Einerseits muss dabei ein hoher Grad an Parallelität erzielt, andererseits stets die Konsistenz der Daten gewährleistet werden.

Im Folgenden wollen wir uns auf Erläuterungen zu den Themenbereichen Datenmodelle, Anfragesprachen, Anwendungsprogrammierung und Mehrbenutzerbetrieb beschränken. Darüber hinaus gibt es eine Reihe interessanter Aspekte von DBVS, die hier unberücksichtigt bleiben. Als Schlagworte seien stellvertretend genannt: Datenschutz, Integritätsbedingungen, Fehlertoleranz. Für die Erläuterung dieser Begriffe und eine Vertiefung der Thematik verweisen wir den Leser auf die entsprechende Fachliteratur, z.B. auf die Bücher von Vossen bzw. von Kemper und Eickler.

10.2 Datenmodelle

Ähnlich wie bei der Erstellung von großen Softwaresystemen wird bei der Entwicklung einer Datenbank zunächst auf Grundlage einer Anforderungsanalyse ein *konzeptionelles Datenmodell* erstellt, das später zur Spezifikation der Datenbank dient. Dieses konzeptionelle Datenmodell ist unabhängig von dem später verwendeten DBVS. Üblicherweise wird es heutzutage auf Basis des so genannten *Entity/Relationship* (kurz:E/R) Modells erstellt, dessen grundlegende Techniken wir hier vorstellen wollen. Im nächsten Schritt wird dann das konzeptionelle Datenmodell in ein Modell des verwendeten DBVS umgesetzt. Aufgrund der marktbeherrschenden Position relationaler DBVS wird im Allgemeinen eine Umsetzung in das relationale Modell erfolgen. Dessen Hauptbestandteile sollen deshalb in einem weiteren Unterkapitel kurz vorgestellt werden.

10.2.1 Entity/Relationship-Modell

Das *E/R-Datenmodell* beschreibt eine Abstraktion der realen Welt durch Verwendung von Entitäten (engl. *entities*) und ihren Beziehungen untereinander (engl. *relationships*). Eine *Entität* entspricht einer Abstraktion eines Gegenstands aus der realen Welt, der sich von anderen Entitäten unterscheidet. Entitäten besitzen Eigenschaften, die im Folgenden als *Attribute* bezeichnet werden. Logisch zusammengehörende Entitäten, die über gleiche Attribute verfügen, werden zu einer *Entitätsmenge* zusammengefasst. Die Strukturbeschreibung einer Entitätsmenge bezeichnet man auch als *Entitätstyp*. Dieser wird durch einen eindeutigen Namen und mit einer zugehörigen Menge von Attributen beschrieben.

Um Entitäten innerhalb der Entitätsmenge eindeutig zu identifizieren, genügt es, eine Teilmenge der zugehörigen Attribute zu betrachten. Eine solche Teilmenge heißt *Schlüsselkandidat*, wenn durch Entfernen eines beliebigen Attributs die Eindeutigkeit nicht erhalten bleibt. Für einen Entitätstyp kann es durchaus mehrere Schlüsselkandidaten geben. Einer dieser Schlüsselkandidaten wird dann zur Identifikation der Entitäten des Typs fest ausgewählt und als *Primärschlüssel* bezeichnet. Man beachte aber, dass Schlüsselkandidaten zeitunabhängige Invarianten sind. Die Eigenschaft kann nicht mit einer derzeit gültigen Entitätsmenge nachgewiesen werden, sondern ist Bestandteil des Entitätstyps. Für jede Entitätsmenge des Typs muss also diese Eigenschaft erfüllt sein.

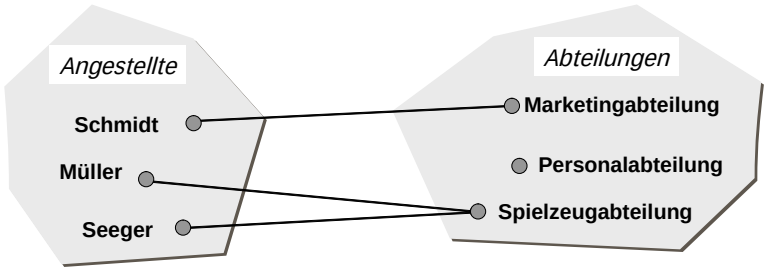

Abb. 10.2: *Entitäten, Entitätsmengen und Beziehungen*

Durch eine *Beziehung* lassen sich Zusammenhänge zwischen verschiedenen Entitäten herstellen. Zusätzlich zu den beteiligten Entitäten kann eine Beziehung optional auch noch eine Menge von Attributen besitzen. Logisch zusammengehörende Beziehungen darf man zu einer *Beziehungsmenge* zusammenfassen. Ein *Beziehungstyp* beschreibt die Struktur einer Beziehungsmenge, indem die Menge der beteiligten Entitätstypen und die Menge der Attribute angegeben werden.

In Abb. 10.2 werden zwei verschiedene Entitätsmengen (`Angestellte` und `Abteilungen`) eines Unternehmens dargestellt. Eine Entität besitzt in diesem einfachen Beispiel als Attribut den Namen des Angestellten bzw. den Namen der Abteilung. Jede Verbindungslinie zwischen den Entitäten veranschaulicht eine Beziehung. So besteht z.B. zwischen der Entität `Müller` und der Entität `Spielzeugabteilung` eine Beziehung.

Beziehungstypen werden nach der Kardinalität der in dem Beziehungstyp auftretenden Entitätstypen klassifiziert. Gehen wir von einem Beziehungstyp mit zwei Entitätstypen A und B aus, so ergeben sich folgende drei Klassen:

- *1:1-Beziehung*: Eine Entität des Typs A steht mit höchstens einer Entität des Typs B in Beziehung und umgekehrt.
- *n:1-Beziehung*: Eine Entität des Typs A steht mit höchstens einer Entität des Typs B in Beziehung, und eine Entität des Typs B kann mit mehreren Entitäten des Typs A in Beziehung stehen.
- *n:m-Beziehung*: Eine Entität des Typs A kann mit mehreren Entitäten des Typs B in Beziehung stehen und umgekehrt.

Ein E/R-Datenmodell lässt sich als Graph (*E/R-Diagramm*) veranschaulichen. Die Knoten des Graphen werden als Rechtecke oder Rauten dargestellt, je nachdem ob sie Entitätstypen oder Beziehungstypen repräsentieren. Zwischen einem Rechteck und einer Raute wird eine Kante gezogen, falls der zugehörige Entitätstyp im entsprechenden Beziehungstyp auftritt. Die Kante besitzt als Markierung die Kardinalität, die der Entitätstyp in dem Beziehungstyp besitzt. Dieser Graph lässt sich noch weiter verfeinern, indem die Attribute durch Ellipsen als eigenständige Knoten veranschaulicht und über eine Kante mit ihrem Entitäts- bzw. Beziehungstyp verbunden werden. Dabei unterstreicht man alle zum Primärschlüssel gehörenden Attribute. Aus Gründen der Übersichtlichkeit stellt man oft nur die wichtigsten Attribute im E/R-Diagramm dar.

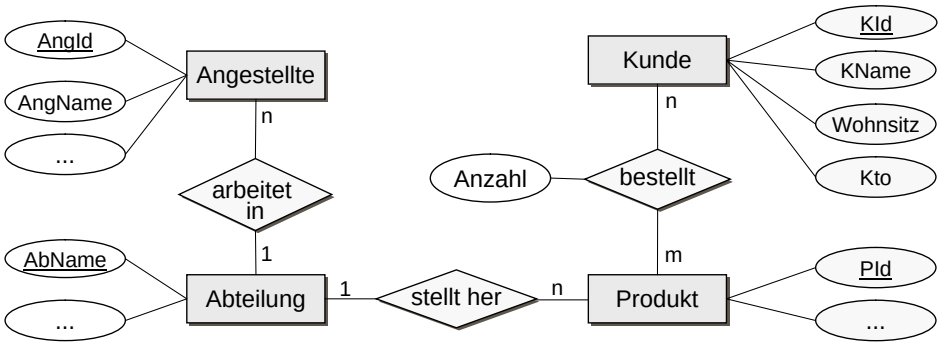

Abb. 10.3: *Beispiel eines E/R-Diagramms*

Ein Beispiel eines E/R-Diagramms zeigt Abb. 10.3. Es beschreibt die „Miniwelt“ eines Industrieunternehmens, in dem als Entitätstypen `Angestellte`, `Abteilung`, `Produkt` und `Kunde` auftreten. Es gibt zwei 1:n Beziehungstypen (`arbeitet in`, `stellt her`) und einen n:m-Beziehungstyp (`bestellt`). Da die Beziehung `bestellt` eine n:m-Beziehung ist, kann ein Kunde verschiedene Produkte bestellen und ein Produkt von verschiedenen Kunden bestellt werden. Bei der n:1-Beziehung `arbeitet in` ist zu erkennen, dass ein Angestellter zu höchstens einer Abteilung gehören darf. Der Entitätstyp `Kunde` besitzt insgesamt vier Attribute: `KId` ist eine eindeutige Kundennummer und wird als Primärschlüssel verwendet, `KName` bezeichnet den Namen, `Wohnsitz` ist der Hauptwohnsitz des Kunden. Da ein Kunde im Allgemeinen noch nicht alle Rechnungen bezahlt hat, wird die Summe seiner Fälligkeiten in dem Attribut `Kto` vermerkt. Wir werden in diesem Kapitel immer wieder auf dieses Beispiel zurückgreifen. Es sei hier ausdrücklich darauf hingewiesen, dass das E/R-Diagramm nur einen Auszug des gesamten E/R-Modells visualisiert und dass deshalb zusätzlich zu dem Diagramm das eigentliche E/R-Modell in einer Dokumentation noch vollständig beschrieben werden muss.

Insgesamt stellt das E/R-Datenmodell die Grundlage für einen guten Datenbankentwurf dar. Die E/R-Modellierung wird auch in anderen Bereichen wie z.B. bei der Entwicklung von Software verwendet. Es wurden vielfältige Erweiterungen des E/R-Datenmodells vorgestellt, z.B. in dem Buch von Vossen, die sich auch in der Praxis gegenüber dem ursprünglichen Modell bewährt haben.

10.2.2 Das Relationale Datenbankmodell

Das *relationale Datenmodell* wurde in seiner ursprünglichen Form 1970 von Codd vorgestellt. Die heute marktbeherrschenden DBVS wie z.B. Oracle, SQL Server, Sybase, Informix und DB2 orientieren sich im Wesentlichen an diesem relationalen Modell. Der folgende Abschnitt beschreibt die Grundlagen, die für das Verständnis der erwähnten Datenbanksysteme von großer Wichtigkeit sind.

10.2.3 Relationen

Der Begriff der *Relation* ist von zentraler Bedeutung innerhalb des relationalen Modells. Zu einer Relation *R* gibt es ein *Relationenschema*, das aus dem Namen der Relation und einer Folge von Attributen besteht. Zu jedem *Attribut* A_i gehört ein *Wertebereich* D_i, für den im Allgemeinen nur atomare Werte wie ganze Zahlen, Fließkommazahlen und Zeichenketten zugelassen sind, nicht aber strukturierte und mengenwertige Daten.

Der Inhalt einer in der Datenbank gespeicherten Relation ist zeitlichen Veränderungen unterworfen. Zu jedem Zeitpunkt hat man daher eine bestimmte *Instanz* I_R von *R* vorliegen. Unter einer Instanz verstehen wir hier eine *Teilmenge* des kartesischen Produkts der zugehörigen Wertebereiche, also das, was man ansonsten in der Mathematik unter einer Relation versteht: $I_R \subseteq D_1 \times ... \times D_n$. Ein Element einer Instanz wird *Tupel* oder *Datensatz* genannt. Eine *Relation* ist daher genau genommen die Menge aller erlaubten Instanzen eines vorgegebenen Relationenschemas. Der Begriff der Relation wird oft aber auch synonym für die (aktuelle) Instanz benutzt.

Der bisherige Begriff der Relation ist für praktische Zwecke zu weit gefasst, da Datensätze zugelassen werden, die in der realen Welt nicht auftreten können. Um die Möglichkeiten der Relationsinstanzen in diesem Sinne einzuschränken, führt man *Integritätsbedingungen* ein, die für alle Instanzen einer Relation erfüllt sein müssen. Ein Beispiel für eine Integritätsbedingung ist der Primärschlüssel einer Relation. Entsprechend dem Begriff des Primärschlüssels im E/R-Datenmodell bezeichnet der *Primärschlüssel* im relationalen Modell eine minimale Menge von Attributen, welche die Datensätze der Relation eindeutig identifiziert. Es darf also keine Instanz einer Relation geben, in der zwei Tupel existieren, die bezüglich ihres Primärschlüssels gleich sind. Primärschlüssel werden im relationalen Modell auch als „Zeiger“ benutzt, die insbesondere zur Modellierung von Beziehungen verwendet werden.

Relationen lassen sich übersichtlich als Tabellen veranschaulichen. Abbildung 10.4 stellt z.B. eine Relation Kunde als Tabelle dar. Die Attribute der Relation werden dabei in einer Kopfzeile aufgeführt. Jede weitere Zeile der Tabelle enthält genau einen Datensatz. Die Repräsentation von Relationen durch Tabellen ist so intuitiv, dass viele DBVS eine entsprechende grafische Benutzerschnittstelle anbieten.

KId	KName	Wohnsitz	Kto
2	Lutz	Darmstadt	0
3	Feldmann	Bremen	–2000
5	Seidl	München	–1000
7	Breitenbach	Darmstadt	0

Abb. 10.4: *Darstellung der Relation* Kunde *als Tabelle*