Objektrelationale Datenbanken

Ein Lehrbuch

von
Can Türker, Gunther Saake

1. Auflage

dpunkt.verlag 2005

Verlag C.H. Beck im Internet:
www.beck.de
ISBN 978 3 89864 190 6

Inhaltsverzeichnis: Objektrelationale Datenbanken – Türker / Saake

1 Einleitung und Motivation

Objektrelationale Datenbanksysteme sind State-of-the-Art! Die aktuellen Produkte führender Datenbanksystemhersteller sowie der aktuelle Datenbankstandard SQL:2003 integrieren objektorientierte Datenmodellkonzepte wie benutzerdefinierte Objekttypen, Methoden und Spezialisierung mit bewährten relationalen SQL-Konzepten, um eine flexible, zuverlässige Plattform für moderne Anwendungen zu schaffen, die Daten aller Art effizient verarbeitet.

Erkauft wird eine solche universelle Infrastruktur durch eine enorme Komplexität der dazugehörigen Sprachvorschläge und der daraus resultierenden uneinheitlichen Realisierung der verschiedenen Datenbanksystemhersteller.

Dieses erste Kapitel rekapituliert grundlegende Begriffe aus dem Gebiet der Datenbanken und stellt die Konzepte objektrelationaler Datenbanken vor.

1.1 Grundlegende Begriffe

Eine *Datenbank* (*DB*) ist eine strukturierte, persistente Ansammlung von Daten, die Fakten und Regeln über eine modellierte Miniwelt repräsentiert. Ein *Datenbankmanagementsystem* (*DBMS*) ist eine Ansammlung von anwendungsunabhängigen Programmen, die das Erzeugen, Ändern und Löschen einer Datenbank ermöglicht. Die Kombination eines DBMS mit einer oder mehreren Datenbanken wird als ein *Datenbanksystem* (*DBS*) bezeichnet. Ein *Datenbankanwendungsprogramm* (kurz *Anwendungsprogramm*; *AP*) ist ein anwendungsspezifisches Programm, das auf Daten einer oder mehrerer Datenbanken zugreift, sie verarbeitet und eventuell Änderungen in die Datenbank(en) zurückschreibt. Abbildung 1.1 zeigt die Drei-Ebenen-Architektur von Datenbanksystemen.

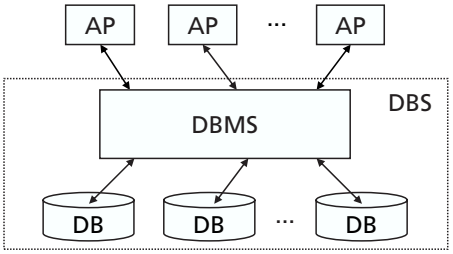

Abbildung 1.1:
Drei-Ebenen-Architektur von Datenbanksystemen

Ein *Datenbankmodell* ist ein System von Konzepten zur Definition und Evolution von Datenbanken. Es legt die Syntax und Semantik von *Datenbankschemata* (Datenbeschreibungen) und *Datenbankoperationen* fest. Anstelle vom Datenbankmodell wird oft auch verkürzt vom *Datenmodell* gesprochen, obwohl dieser Begriff in der deutschsprachigen Literatur auch das Typsystem einer Programmiersprache bezeichnet oder als Synonym für Datenbankschema (Stichwort: »Unternehmensweites Datenmodell«) verwendet wird. Das Datenbankmodell definiert die Menge aller möglichen Datenbankschemata. Eine konkrete Ausprägung des Datenbankmodells beschreibt die modellierte Miniwelt in Form eines Datenbankschemas. Eine Datenbankinstanz ist eine konkrete Ausprägung eines Datenbankschemas. Es korrespondiert mit einem konkreten Zustand der modellierten Miniwelt. Abbildung 1.2 illustriert den Zusammenhang dieser Begriffe.

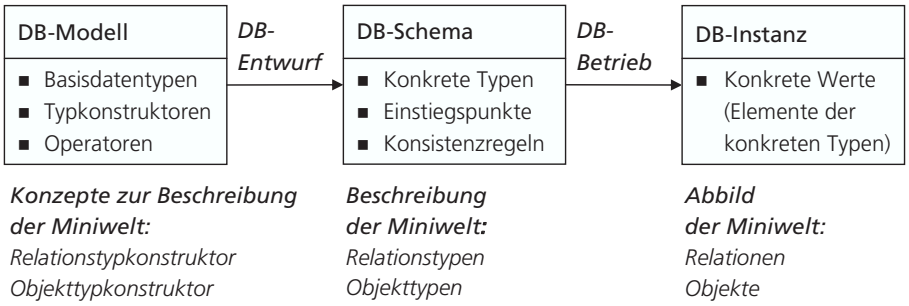

Abbildung 1.2: Grundlegende Begriffe der Datenbankwelt

Das Datenbankmodell bietet neben *Basisdatentypen* eine Reihe von *Typkonstruktoren* zur Beschreibung des Datenbankschemas. Das Datenbankschema legt neben den in der Miniwelt vorkommenden (konkreten) Datentypen auch die Einstiegspunkte in die Datenbank (im Relationenmodell die Relationennamen) fest. Die mit diesen Datentypen assoziierten Operationen bestimmen, wie sich die Ausprägungen der jeweiligen Datentypen, das heißt die konkreten Werte, im Laufe des *Datenbankbetriebs* ändern können.

Im *Relationenmodell* etwa gibt es mit dem *Relationentypkonstruktor* nur einen einzigen Typkonstruktor. Dieser wird genutzt, um die *Relationentypen* zu beschreiben, welche die Strukturen für die Darstellung der Daten in der Datenbank bereitstellen. Das Datenbankschema setzt sich also aus konkreten Relationentypen zusammen. Die Instanziierung dieser Typen führt zu konkreten *Relationen*, aus denen sich der Datenbankzustand zusammensetzt.

Das Schema einer Datenbank wird mit der *Datendefinitionssprache* (DDL) beschrieben. Der Zustand einer Datenbank hingegen wird mit der *Datenmanipulationssprache* (DML) manipuliert. Konkrete Werte können mit der DML erzeugt, geändert und gelöscht werden. Für die Abfrage des Datenbankzustands wird

die *Anfragesprache* eingesetzt. Aus kommerziellen relationalen Datenbanksystemen ist die Sprache SQL nicht nur als standardisierte Anfragesprache, sondern auch als DDL und DML in einem bekannt.

Datenunabhängigkeit

Sowohl Datenbanken als auch Anwendungsprogramme haben in der Regel eine lange Lebensdauer. Während dieser Zeit können sich sowohl die interne Datenspeicherung als auch die externen Zugriffsschnittstellen bedingt durch Fortschritte in der Technologie und/oder durch wachsende bzw. wechselnde Anforderungen ändern. Das Konzept der *Datenunabhängigkeit* fordert deshalb die transparente Entkoppelung von Datenbank und Anwendungen. Zwei Arten der Datenunabhängigkeit werden unterschieden:

- Die *physische Datenunabhängigkeit* bedeutet, dass Änderungen an den internen Speicherstrukturen und Zugriffspfaden für Anwendungen unsichtbar sind. Dies wird durch die Entkoppelung der konzeptionellen Sicht von der internen Datenspeicherung erreicht.
- Die *logische Datenunabhängigkeit* besagt, dass Änderungen an der konzeptionellen Sicht für die Anwendungen unsichtbar sind. Dies wird durch die Bereitstellung von anwendungsspezifischen Sichten auf den gemeinsamen Datenbestand erreicht.

Durch die Einhaltung der in Abbildung 1.3 dargestellten *Drei-Ebenen-Schema-Architektur* wird Datenunabhängigkeit gewährleistet.

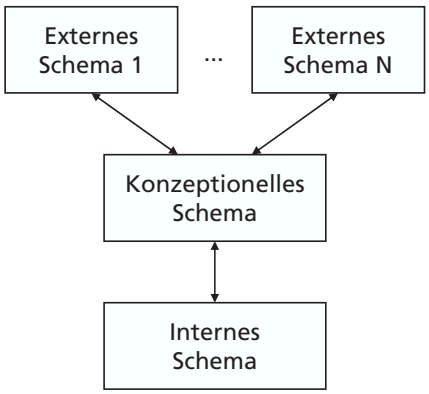

Abbildung 1.3:
Drei-Ebenen-Schema-Architektur

In dieser Architektur wird die Datenbankbeschreibung auf die folgenden Ebenen aufgeteilt:

- Das *konzeptionelle Schema* beschreibt eine von der konkreten internen Realisierung der Datenspeicherung abstrahierende Gesamtsicht auf den gespeicherten Datenbestand.

- Das *interne Schema* legt die zur Implementierung der Datenbank intern genutzten Datenstrukturen fest.
- Die *externen Schemata* definieren anwendungsspezifische Sichten auf die Datenbank. Diese Sichten können als abgeleitete »virtuelle« Datenbanken aufgefasst werden, die auf die Bedürfnisse spezieller Anwendungen bzw. Anwenderklassen zugeschnitten sind.

Die Datenbankzugriffe der Anwender erfolgen im Idealfall über externe Schemata. Das DBMS ist dann für die notwendigen Transformationen von Anfragen, Änderungen und Ergebnisdaten verantwortlich. Je nachdem, welches Datenmodell der Datendefinitionssprache zugrunde liegt, kann diese Abbildung mehr oder weniger komplex werden.

1.2 Aufgaben eines Datenbanksystems

Vereinfachend formuliert handelt es sich bei einem Datenbanksystem um ein *anwendungsunabhängiges* Programm zur effizienten und konsistenten Speicherung, Wiedergewinnung, Verknüpfung und Auswertung von Informationen. Die folgenden »*Neun Codd'schen Regeln*« [Cod82] fassen die grundlegende Funktionalität eines Datenbanksystems kurz zusammen:

1. Das Datenbankmodell ermöglicht eine *integrierte, einheitliche* Verwaltung *aller* von den Anwendungen benötigten Daten.
2. Das Datenbankmodell bietet *Operationen* zum Erzeugen, Ändern, Speichern, Suchen und Entfernen von Daten.
3. Der *Systemkatalog* verwaltet die Datenbeschreibungen und andere systemrelevante Metadaten.
4. *Sichten* stellen anwendungs- bzw. benutzerspezifische Ausschnitte der Datenbank bereit.
5. Die *Integritätskontrolle* überwacht die semantische Korrektheit der Datenbank, das heißt, sie stellt sichert, dass die gespeicherten Fakten den Datenbeschreibungen entsprechen.
6. Die *Zugriffskontrolle* schließt unautorisierte Zugriffe auf die Datenbank aus.
7. *Transaktionen* sind elementare Ausführungseinheiten, die aus einer Folge von Operationen bestehen, deren Effekt bei Erfolg persistent in der Datenbank gespeichert wird.
8. Die *Synchronisation* von Transaktionen im Mehrbenutzerbetrieb dient dazu, einen inkorrekten Informationsfluss beim nebenläufigen Arbeiten auf gemeinsam benutzten Daten auszuschließen.
9. Die *Datensicherung* verhindert den Verlust von Daten auch nach System- und Mediafehlern.

Eine implizite Forderung ist die *Persistenz* der Daten, die über die Lebensdauer einer Anwendungsausführung hinausgeht.

Eine weitere in den Codd'schen Regeln nicht explizit erwähnte Forderung ist die nach einer *deklarativen* Anfragesprache. Eine deklarative Anfragesprache ermöglicht die Ad-hoc-Formulierung von Anfragen und schafft überdies insbesondere die Grundlage für die Optimierbarkeit von Anfragen.

1.3 Akteure in Datenbanksystemen

In einem Datenbanksystem werden entsprechend ihren Aufgaben verschiedene Gruppen von Akteuren unterschieden:

- Der *Datenbankadministrator* ist für alle datenbanksystemseitigen und anwendungsunabhängigen Aspekte einer Datenbank verantwortlich. Er definiert und verwaltet das konzeptionelle und das interne Schema.
- Ein *Anwendungsadministrator* erstellt und wartet ein externes Schema für eine Anwendung. Er leistet dabei die Abbildung zwischen externem und konzeptionellem Schema. Oft sind der Datenbankadministrator und der Anwendungsadministrator ein und dieselbe Person.
- Die *Anwendungsprogrammierer* erstellen auf der Basis von externen Schemata Anwendungsprogramme. Sie realisieren die Abbildungen zwischen Anwendung- und Datenbanksemantik.
- Die *Anwender* sind die Benutzer der Anwendungsprogramme. Sie benötigen normalerweise kein Datenbankwissen. Allerdings können Anwender, die über Datenbankwissen verfügen, über die Datenbankschnittstelle mittels einer (deklarativen) Anfragesprache selbstständig Informationen aus einer Datenbank extrahieren.
- Die *DBMS-Entwickler* sind für die Implementierung des DBMS verantwortlich. Sie stellen somit keine Datenbankbenutzer in engen Sinne dar. Sie spielen dennoch eine wichtige Rolle, da sie durch das Datenbankmodell Funktionalität anbieten, die sonst von den anderen Akteuren umgesetzt werden müsste.

Abbildung 1.4 ordnet die jeweiligen Akteure den verschiedenen Ebenen der Schemaarchitektur zu. Die DBMS-Entwickler stellen das Datenbankmodell bereit und beeinflussen nicht nur alle drei Ebenen der Schemaarchitektur, sondern allen voran die Kluft zwischen Anwendungsprogramm und externem Schema.

Die Aufgabenlast, ein Anwendungssystem zu entwickeln bzw. zu benutzen, verteilt sich je nach Architektur des zugrunde liegenden Datenbanksystems auf diese fünf Gruppen von Personen. Beispielsweise ist die Last für die Anwender geringer, wenn ihnen höhere (semantisch reiche) Schnittstellen angeboten werden, etwa eine Schnittstelle, bei der sie eventuell nur noch Daten eingeben und mit einem Knopfdruck Anfrageergebnisse formatiert erhalten.

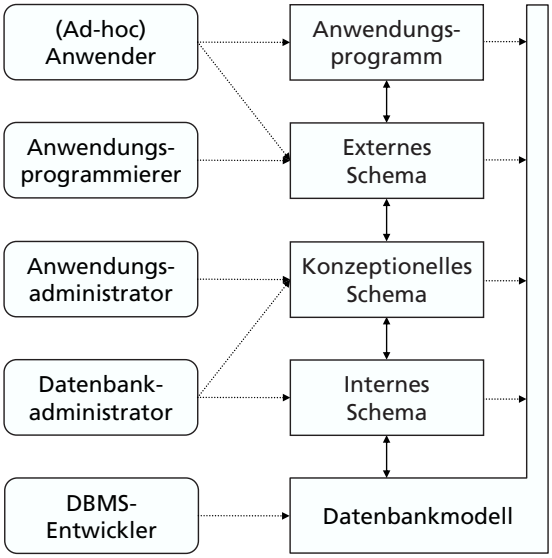

Abbildung 1.4: Akteure in einem Datenbanksystem

Eine höhere Schnittstelle bedeutet in der Regel mehr Arbeit für die Entwickler der darunter liegenden Ebenen. In diesem Beispiel müssen die Anwendungsprogrammierer entsprechende »Knöpfe« und Formatierungen implementieren.

Auf den darunter liegenden Ebenen verhält es sich ähnlich. Die Entwickler eines relationalen Datenbanksystems müssen viel weniger (und zudem einfachere) Funktionalität im Vergleich zu den Entwicklern objektrelationaler Datenbanksysteme implementieren. Von der zusätzlichen Funktionalität eines objektrelationalen Datenbanksystems, also von der höheren Datenbankschnittstelle, können dann die Datenbankadministratoren und insbesondere die Anwendungsprogrammierer profitieren. Bei Letzteren wird die Kluft zwischen Anwendungs- und Datenbankwelt derart verringert, dass einfachere Abbildungen zwischen den Konzepten der Anwendungsprogrammier- und der Datenbanksprache möglich sind. Im Idealfall können sogar die Anwendungsobjekte eins-zu-eins auf Datenbankobjekte (und umgekehrt) abgebildet werden.

1.4 Historie von Datenbanksystemen

Bevor Datenbanksysteme existierten musste jeder Anwendungsprogrammierer nicht nur die anwendungsspezifische Funktionalität implementieren, sondern auch die Funktionalität für die Realisierung der Datenspeicherung. In jedem Anwendungsprogramm mussten immer wieder die Routinen für das Schreiben der Daten in speziell formatierte Dateien implementiert werden, ebenso wie Routinen für das Auslesen der Daten aus den Dateien. Das Ergebnis waren entsprechend »fette« Anwendungen.

Abbildung 1.5 illustriert die Aufgabenverteilung zwischen Anwendungen und Datenverwaltungssystemen. Mit dem Aufkommen von DBMS wurde immer mehr generische, das heißt anwendungsunabhängige Funktionalität von den Anwendungen in das Datenbanksystem verlagert. Je »höher« die Datenbankschnittstelle wird, desto kompakter werden die Anwendungsprogramme.

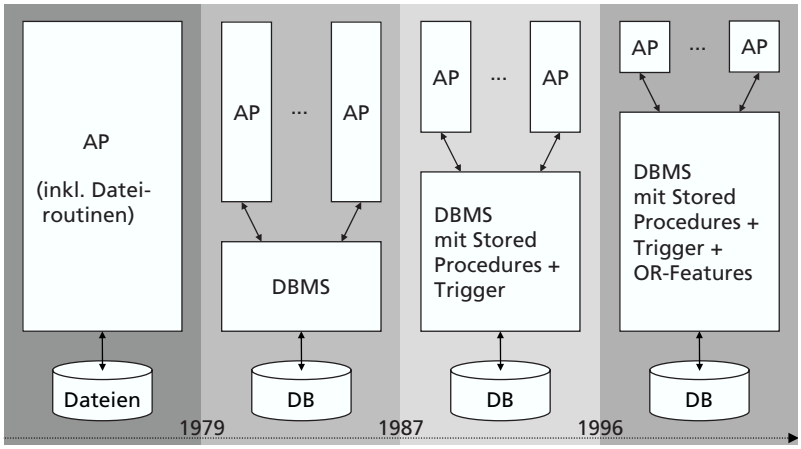

Abbildung 1.5: Aufgabenverteilung im Wandel der Zeit

Zunächst wurde von der Datenspeicherung abstrahiert, indem durch das Datenbanksystem persistente Strukturen mit den zugehörigen Routinen zur Verfügung gestellt wurden. Weiterhin wurde von Einflüssen bei konkurrierendem Zugriff und bei Fehlern abstrahiert. Das Jahr 1979 steht für das Release des ersten kommerziellen *relationalen* Datenbanksystems von der Firma Oracle.

In den darauf folgenden Jahren wurden vor allem Konzepte zur Integration von Anwendungsfunktionalität in die Datenbank entwickelt. Benutzerdefinierte (gespeicherte) Prozeduren und Trigger stellen zwei zentrale Konzepte für die funktionale Erweiterung einer Datenbank dar. Das erste kommerzielle relationale Datenbanksystem, das Trigger unterstützte, war Sybase im Jahre 1987.

Mit dem Aufschwung objektorientierter Programmiersprachen gelangten immer mehr Objektkonzepte ins DBMS. Im Jahr 1996 kamen die ersten kommerziellen *objektrelationalen* DBMS auf den Markt. Zu dieser Zeit war Informix das System mit den meisten objektrelationalen Erweiterungen. Oracle und DB2 warteten noch im selben Jahr mit ihren objektrelationalen Erweiterungen auf.

1.5 Evolution von Datenmodellen

Eng verbunden mit der Historie von Datenbanksystemen ist die Evolution von Datenmodellen. Mit der Entwicklung immer höherer Datenbankschnittstellen

ging die Entwicklung immer komplexerer Datenmodelle einher. Abbildung 1.6 skizziert die gängigsten Datenmodelle und setzt sie anhand ihrer strukturellen und operationalen Komplexität in Beziehung.

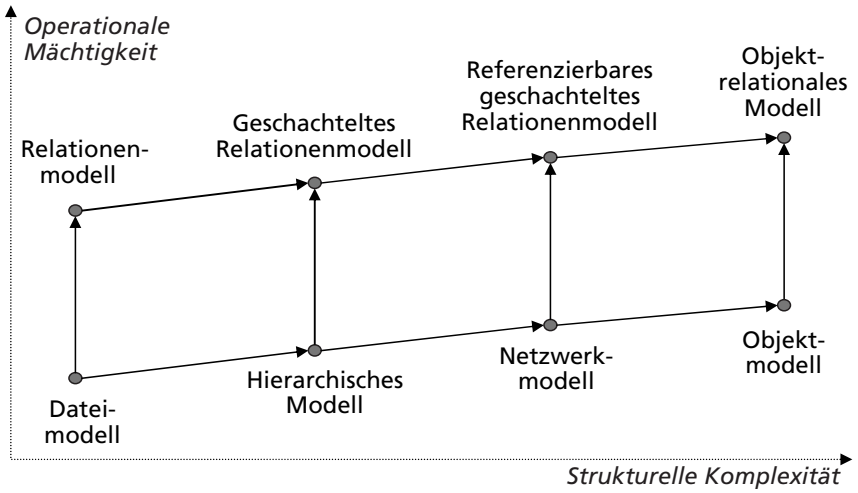

Abbildung 1.6: Evolution von Datenmodellen

Das *Dateimodell* bietet das Konzept einer Datei zur Speicherung von Daten. Eine Datei kann als eine Sequenz von Datensätzen (Tupeln) aufgefasst werden. Die strukturelle Komplexität des Dateimodells entspricht dem einer Menge von Tupeln. Als Operationen stehen primitive Routinen zum Lesen und Schreiben einzelner Datensätze zur Verfügung. Der Anwendungsprogrammierer muss die Strukturierung der Datei genau kennen, um die Datensätze bzw. Teile davon auslesen bzw. schreiben zu können.

Das *Relationenmodell* speichert Daten in Relationen, die Mengen von Tupeln darstellen. Die Tupel bestehen aus Attributen mit atomaren Werten. Die strukturelle Komplexität bleibt im Vergleich zum Dateimodell gleich. Die operationale Komplexität ist jedoch im Relationenmodell wesentlich höher, da die *Relationenalgebra*, welche die Operationen auf Relationen definiert, mengenstatt satzorientiert arbeitet. Mit der Selektion und Projektion werden bestimmte Ausschnitte von Relationen ausgewählt; der (natürliche) Verbund verknüpft mehrere Relationen. Darüber hinaus gibt es weitere Operationen, etwa die Mengenoperationen auf Relationen oder die Umbenennung von Attributen.

Im *hierarchischen Modell* werden die Daten in Form von hierarchisch organisierten Datensätzen gespeichert. Die Eltern-Kind-Beziehung zwischen den Datensätzen wird mit Zeigern umgesetzt. Das Lesen und Schreiben der Daten erfolgt satzweise. Im Vergleich zum Dateimodell ist also eine Navigation entlang der Eltern-Kind-Beziehungen durch das Verfolgen der Zeiger möglich.

Das Modell der *geschachtelten Relationen* (auch unter dem Begriff NF2-Modell bekannt) speichert die Daten in hierarchisch geschachtelten Relationen. Die Werte von Attributen dürfen nun auch Relationen sein. Im Vergleich zum (flachen) Relationenmodell nimmt nicht nur die strukturelle, sondern auch die operationale Komplexität zu. Die Relationenalgebra wird so erweitert, dass alle Operationen auch auf den inneren Relationen möglich sind.

Das *Netzwerkmodell* verallgemeinert das hierarchische Datenmodell dahingehend, dass die gespeicherten Datensätze beliebig in allgemeinen Graphstrukturen verkettet werden. Es können insbesondere mehrere Zeiger auf denselben Datensatz verweisen. Im Vergleich zum hierarchischen Modell werden die Navigationsoperationen entsprechend der verallgemeinerten Struktur erweitert.

Das *Modell der referenzierbaren geschachtelten Relationen* verallgemeinert das Modell der geschachtelten Relationen dahingehend, dass die Relationen beliebig miteinander verkettet werden können. Eine Relation kann somit von mehreren Relationen referenziert werden. Entsprechend dieser strukturellen Verallgemeinerung wird die Relationenalgebra erweitert.

Das *Objektmodell* erweitert das Netzwerkmodell in verschiedener Hinsicht. Datensätze werden durch Objekte ersetzt. Objekte besitzen einen gekapselten Zustand, der nur über die Methoden des Objekts gelesen und geschrieben werden kann. Methoden können beliebig komplexe Funktionen realisieren. Außerdem hat jedes Objekt einen zustandsunabhängigen Objektidentifikator, über den es referenziert werden kann. Ein Objekt kann andere Objekte einbetten oder referenzieren. Polymorphismus wird durch Typvererbung möglich. Ein polymorphes Objekt kann auch in der Gestalt eines Objekts eines Supertyps auftreten.

Das *objektrelationale Modell* kombiniert die Mengenorientierung des Relationenmodells mit den komplexen Strukturen des Objektmodells. Die Daten werden als Relationen von (komplexen) Objekten gespeichert. Objekte können über Referenzen verkettet werden. Darüber hinaus können Relationen in Untermengenbeziehungen zueinander stehen. Das heißt, eine Relation kann eine Subrelation einer anderen Relation sein. Im Vergleich zum rekursiv geschachtelten Relationenmodell kommen Operationen hinzu, die auf Relationenhierarchien arbeiten, etwa Subrelationen ausblenden oder typspezifisch Methoden aufrufen.

1.6 Anwendungen von Datenbanksystemen

Relationale Datenbanksysteme sind sehr erfolgreich in klassischen Anwendungsgebieten, wie etwa der Personaldatenverwaltung, Lagerverwaltung oder Buchhaltung, im Einsatz. Klassische Anwendungsgebiete zeichnen sich durch die Verwaltung großer Mengen von einfach strukturierten Datensätze in einer Mehrbenutzerumgebung aus.

Für den Einsatz in so genannten Nicht-Standardanwendungsgebieten sind relationale Datenbanksysteme hingegen nicht so gut geeignet. Dies liegt vor allem an der höheren Komplexität der Datenobjekte und Operationen von Nicht-Standardanwendungen. Üblicherweise werden hier semantisch reiche Operationen auf komplexen Objekten ausgeführt, die in Spezialisierungs- und Aggregationshierarchien organisiert sind. Diese Operationen können Aufrufe komplexer Methoden und sogar rekursive Funktionen enthalten. Die Dauer der Transaktionen ist in der Regel lang. Eine Konsequenz daraus ist, dass die Forderung nach Atomarität der Transaktionsausführung nicht sinnvoll ist, da sonst viele ausgeführte Schritte zurückgesetzt werden müssten. Folglich sind generelle kleinere Rücksetzeinheiten notwendig. Abbildung 1.7 stellt die Kennzeichen der beiden Anwendungsrichtungen gegenüber.

Eigenschaft	Klassisch	Nicht-Standard
Strukturierung der Objekte	einfach	komplex
Größe der Objekte	klein	eher groß
Größe der Objektmengen	groß	klein bis groß
Art der Operationen	read/write	höhere
Strukturierung der Operationen	flach	geschachtelt
Dauer der Transaktionen	kurz	lang
Ausführung der Transaktionen	atomar	nicht atomar
Zurücksetzen der Transaktionen	vollständig	partiell

Abbildung 1.7: Klassische versus Nicht-Standardanwendungen

Beispiele für komplexe Objekte in Nicht-Standardanwendungen sind geografische Objekte, topografische und thematische Karten, Textformulare und Bildobjekte. Große Objekte ergeben sich beispielsweise aus Auswertungen von Luftbildaufnahmen oder Massenspektrometerexperimenten. Mit der höheren strukturellen Komplexität ist auch eine höhere funktionale Komplexität verbunden. Für geografische Objekte zum Beispiel werden spezielle Prädikate benötigt, die etwa auswerten, ob sich zwei Objekte überschneiden oder angrenzen. Analog werden Funktionen benötigt, welche die Fläche oder Volumen von geografischen Objekten berechnen.

Selbst wenn die Operationen einfach sind, ist deren Unterstützung und Ausführung im Datenbanksystem von zentraler Bedeutung. In diesem Fall ist das Datenbanksystem in der Lage zu entscheiden, ob die Operationen im Datenbankserverprozess oder im Clientprozess auszuführen sind, je nachdem, wie groß die Eingabemenge und die erwartete Ergebnismenge sind. Nehmen wir an, dass es eine Funktion gibt, die zu einem gegebenen Grundstück alle angrenzenden Grundstücke liefert. Ferner soll es in der Datenbank eine Million (großer) Objekte geben, die Grundstücke darstellen. Dann wird es effizienter

sein, die Funktion im Datenbankserver auszuführen, statt eine Million Objekte über das Netzwerk zum Client zu schicken, um dort dann die wenigen angrenzenden Grundstücke zu ermitteln. Die Unterstützung von benutzerdefinierten Funktionen durch das Datenbanksystem bildet also die Grundlage für die Optimierung von Funktionsausführungen.

1.7 Objektrelationale Datenbanksysteme

Objektrelationale Datenbanksysteme sind eine Weiterentwicklung von relationalen Datenbanksystemen. Sie unterstützen eine Reihe von neuen Konzepten, die ihren Ursprung in der Objektorientierung haben. Das Ziel der objektorientierten Erweiterung relationaler Datenbanksysteme ist es, die reife Datenbanktechnologie auf die Verwaltung *beliebiger* Arten von Daten auszudehnen.

Mit der Objektorientierung in Datenbanksystemen wird eine Reihe von Vorteilen verbunden:

- Vereinfachte Abbildung und Darstellung von komplexen Realweltobjekten sowie komplexer Anwendungsfunktionalität.
- Schnellere Entwicklung von Datenbankanwendungen durch Wiederverwendung von in der Datenbank gespeicherter Funktionalität. In diesem Zusammenhang ist auch die Spezialisierung von existierenden Datentypen (Subtypenbildung) zu nennen.
- Besseres Verständnis und einfachere Verwaltung und Anpassung komplexer Anwendungen durch höheren Abstraktionsgrad von Datenbankobjekten. Kapselung unterstützt Anwendungsunabhängigkeit von der internen Repräsentation benutzerdefinierter Datentypen. Änderungen am Code können isoliert und vor allem zentral vorgenommen werden.
- Höhere Konsistenz der Datenbank durch strenge Typisierung und Wiederverwendung. Einige semantisch inkorrekte (sinnlose) Vergleiche und Zuweisungen werden dadurch automatisch ausgeschlossen.
- Bessere Performance durch Ausnutzen der Semantik der Anwendungsobjekte. Das DBMS kann benutzerdefinierte Indexstrukturen und Selektionsfunktionen etwa zur Anfrageoptimierung nutzen.

1.8 Objektrelationale Konzepte

Objektrelationale Konzepte resultieren aus der Synthese von relationalen und objektorientierten Konzepten. Bislang gibt es jedoch keine allgemein anerkannte Definition, welche Konzepte als objektrelational zu bezeichnen sind. Wir orientieren uns hier an der Begriffsbildung und den Konzepten des SQL-

Standards, die jedoch mehr oder weniger stark von denen der aktuellen SQL-Dialekte abweichen. Hier führen wir die wichtigsten Begriffe lediglich kurz ein:

- *Typkonstruktoren,*
- *benutzerdefinierte Datentypen,*
- *Typhierarchien,*
- *typisierte Tabellen,*
- *Tabellenhierarchien,*
- *typisierte Sichten sowie*
- *Sichtenhierarchien.*

Eine detaillierte Diskussion dieser Konzepte folgt in den Kapiteln 3 bis 5.

Typkonstruktoren

Um komplexe Sachverhalte möglichst »einfach« und »natürlich« in einer Datenbank darstellen zu können, muss das zugrunde liegende Datenmodell entsprechende Mittel zur Beschreibung solcher anbieten. Die Grundlage bilden elementare Datentypen, die so genannten Basisdatentypen, die den Vorrat an zulässigen elementaren Werten festlegen. Auf den Basisdatentypen aufbauend definieren konstruierte Datentypen Wertebereiche für komplexe Werte. Die Beschreibung von solchen konstruierten Datentypen erfolgt mit Hilfe von *Typkonstruktoren*, die aus Basisdatentypen (und anderen konstruierten Datentypen) neue konstruierte Datentypen erzeugen. Durch geschachtelte Anwendung der Typkonstruktoren entstehen beliebig komplexe Datentypen.

Aus diversen Datenmodellen sind folgende Typkonstruktoren bekannt:

- *Tupeltypkonstruktor,*
- *Mengentypkonstruktor,*
- *Multimengentypkonstruktor,*
- *Listentypkonstruktor,*
- *Arraytypkonstruktor und*
- *Vereinigungstypkonstruktor.*

Neben der Möglichkeit, komplexe Strukturen beschreiben zu können, muss ein Datenmodell für die konstruierten Datentypen eine Menge von Operationen zum Erzeugen und Manipulieren dieser Strukturen anbieten. Wird zum Beispiel eine Schachtelung des Tupeltypkonstruktors erlaubt, so sind auch Pfadausdrücke zu unterstützen, um entlang der geschachtelten Struktur navigieren zu können.

Benutzerdefinierte Typen

In manchen Anwendungsgebieten gibt es häufig wiederkehrende Datentypen, die man nur einmal (mit den zugehörigen typischen Operationen) definieren

und dann wiederverwenden möchte. Mit dem Konzept der benutzerdefinierten Datentypen bieten objektrelationale Datenbanksysteme ein Konzept zur Erweiterung des Datenmodells um solche anwendungsspezifischen Datentypen.

Im Prinzip wäre es möglich, alle konstruierten Datentypen durch Benennung als benutzerdefinierte Datentypen wiederverwendbar zu speichern. Die Unterstützung für das Anlegen von benutzerdefinierten Datentypen ist jedoch stark abhängig vom zugrunde liegenden Datenmodell. Der SQL-Standard beispielsweise sieht zwei Arten von benutzerdefinierten Datentypen vor: Distinct-Typen und strukturierte Typen.

Distinct-Typen

In SQL können beliebige Wertausdrücke miteinander verglichen bzw. einander zugewiesen werden, solange die Datentypen der Operanden kompatibel sind. Je nach Modellierung der Realwelt sind somit »sinnlose« Vergleiche und Zuweisungen möglich. Beispielsweise kann die Körpergröße einer Person mit seinem Kontostand verglichen werden, da es sich in beiden Fällen um numerische Werte handelt.

Das Konzept der Distinct-Typen wurde daher mit dem Ziel eingeführt, solche semantisch unzulässigen Vergleiche und Zuweisungen durch das *Prinzip der strengen Typisierung* weitgehend auszuschließen. Ein Distinct-Typ stellt eine Kopie eines existierenden Datentyps dar. Sofern der Benutzer keine Cast-Funktionen zur automatischen Typumwandlung zwischen Distinct- und Quelltyp definiert, sind Vergleiche und Zuweisungen zwischen den Instanzen dieser Datentypen ausgeschlossen. Insbesondere sind keine Vergleiche und Zuweisungen zwischen Instanzen unterschiedlicher Distinct-Typen möglich.

Strukturierte Typen und Typhierarchien

Das Konzept der *strukturierten Typen* ist eine der zentralen Konzepte objektrelationaler Datenmodelle. Ein strukturierter Typ entspricht in etwa einem *Objekttyp*, der die *Eigenschaften* (*Attribute*) und das *Verhalten* (*Methoden*) von gleichartigen Datenbankobjekten beschreibt. Den strukturierten Typen liegen folgende Prinzipien der Objektorientierung zugrunde:

- *Strukturierung*: Komplexe Datenstrukturen bilden mit den assoziierten Funktionen semantische Einheiten, die zu einer besseren Modularisierung und Wartbarkeit der Datenbank führen. Fehler können lokalisiert und Funktionsänderungen isoliert vorgenommen werden.
- *Kapselungsprinzip*: Der Zugriff auf ein Objekt erfolgt über die Methoden ihrer Schnittstelle. Die Implementierung der Methoden bleibt nach außen hin verborgen. Die klare Trennung zwischen Schnittstelle und Implementierung von Objektmethoden schafft die Basis für eine transparente Evolution der Implementierung von Objektmethoden.

- *Prinzip der zustandsunabhängigen Objektidentifikation*: Die Entkoppelung der Identifikation der Objekte von den zugrunde liegenden Objektwerten ermöglicht zum einen eine eindeutige, *unveränderliche* Objektreferenz (OID) und zum anderen die Unterscheidung zwischen *identischen* und *gleichen* Datenbankobjekten.
- *Vererbungsprinzip*: Ein *Subtyp* erbt die Attribute und Methoden eines Supertyps. Die geerbten Methoden können *überschrieben* werden. Das *dynamische Binden* der Methoden ermöglicht das objektspezifische Auswählen der Methodenimplementierungen zur Laufzeit.
- *Substituierbarkeitsprinzip*: Ein Objekt eines Subtyps kann überall dort verwendet werden, wo ein Objekt eines zugehörigen Supertyps erwartet wird. Damit können heterogene Kollektionen aufgebaut werden.

Analog zu den Distinct-Typen liegt auch den strukturierten Typen das *Prinzip der strengen Typisierung* zugrunde.

Erweiterte Tupeltabellen

In SQL-Datenbanken werden Daten in Form von Tabellen abgespeichert. Waren bei relationalen SQL-Datenbanken die Tabellenspalten noch auf atomare Werte beschränkt (bedingt durch die erste Normalform), so können die Tabellenspalten mit den objektrelationalen Erweiterungen auch komplexe Werte enthalten. Abbildung 1.8 illustriert dies grafisch.

ANr	Name(Vorname, Nachname)	Hobbys	Bewerbung	Wagen	Foto
INT	ROW(VARCHAR, VARCHAR)	VARCHAR ARRAY[5]	Bewerbung(...)	REF(Auto)	BLOB
1311	ROW('Jim', 'Jones')	ARRAY['Kino', 'Sport']	Bewerbung(...)	X'123282'	X'...'
atomare Spalte	tupelwertige Spalte	arraywertige Spalte	objektwertige Spalte	referenzwertige Spalte	LOB-Spalte

Abbildung 1.8: Tabellen mit »komplexen« Spaltenwerten

Entsprechend den Basisdatentypen und Typkonstruktoren können nun in den Spalten einer Tabelle auch folgende Arten von Werten stehen:

- *Tupel,*
- *Kollektionen (Mengen, Multimengen, Listen, Arrays),*
- *Tabellen (Multimengen von Tupeln),*
- *strukturierte Werte (bzw. eingebettete Objekte),*
- *Referenzen (referenzierte Objekte) und*
- *Large Objects.*

Typisierte Tabellen und Tabellenhierarchien

Zur Speicherung von Daten gibt es in einer objektrelationalen SQL-Datenbank eine zweite Art von Tabellen, die so genannten typisierten Tabellen, deren Zeilentyp durch einen benutzerdefinierten strukturierten Typ festgelegt wird. Eine *typisierte Tabelle* entspricht in etwa dem Konzept einer *Klasse* in objektorientierten Datenbanken. Eine solche Tabelle basiert auf einem strukturierten Typ. Die Zeilen einer typisierten Tabelle repräsentieren Objekte. Typisierte Tabellen werden daher oft auch als *Objekttabellen* bezeichnet. In Standard-SQL ist die erste Spalte einer typisierten Tabelle immer die OID-Spalte (siehe Abbildung 1.9), über welche die Zeilen mittels eines streng typisierten *Referenztyps* referenzierbar werden.

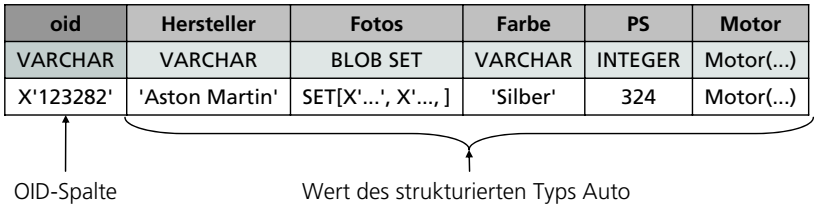

Abbildung 1.9: Typisierte bzw. Objekttabelle

Das Konzept einer *Tabellenhierarchie* ist das objektrelationale Gegenstück einer *Klassenhierarchie* in einer objektorientierten Datenbank. Eine typisierte Tabelle, die als *Subtabelle* einer anderen typisierten Tabelle definiert wird, legt eine Untermengenbeziehung zu der Supertabelle fest. Damit sind alle Zeilen der Subtabelle auch in der Supertabelle enthalten. Dies impliziert, dass die Zeilen einer Subtabelle speziellere Objekte repräsentieren. Der Typ der Objekte einer Subtabelle ist ein Subtyp des Typs der Objekte der Supertabelle.

Typisierte Sichten und Sichtenhierarchien

Das Konzept einer typisierten Sicht ermöglicht die Definition von virtuellen, typisierten Tabellen. Eine *typisierte Sicht* entspricht in etwa einer Objektsicht, auf deren Zeilen die Methoden des zugrunde liegenden strukturierten Typs aufgerufen werden können. Solche Sichten können aus beliebigen SQL-Daten berechnet werden. Somit können objektorientierte Sichten auf relationale Daten geboten werden.

Analog zu Tabellenhierarchien können typisierte Sichten in *Sichtenhierarchien* organisiert werden. Eine typisierte Sicht, die als *Subsicht* einer anderen typisierten Sicht definiert ist, erweitert die »Sicht« der Supersicht. Das heißt, die Supersicht enthält implizit alle Zeilen der Subsicht. Auf diese Weise wird die Untermengenbedingung zwischen Super- und Subsicht durchgesetzt.

Erweiterungspakete

Mit dem Anlegen von benutzerdefinierten Datentypen kann die Funktionalität eines Datenbanksystems erweitert werden. Die Datenbankbenutzer können Instanzen dieser Datentypen erzeugen, manipulieren und abfragen – analog zu den Instanzen der vordefinierten Datentypen.

Was das DBMS noch nicht kann, ist die Steigerung der Effizienz der Operationen auf Instanzen dieser Datentypen. Hierzu fehlt dem DBMS das Wissen über den Inhalt dieser Instanzen. Aus Sicht des DBMS stellen diese Instanzen gekapselte, abstrakte Werte dar.

Um die Auswertung von Anfragen zu beschleunigen, in denen Instanzen der benutzerdefinierten Datentypen vorkommen, muss das DBMS analog zu den vordefinierten Datentypen spezielle Indextypen kennen. Mit anderen Worten, mit dem Anlegen eines benutzerdefinierten Datentyps sollten entsprechende Indextypen mit zugehörigen Indexierungsmethoden implementiert werden. Ebenso sind spezielle Funktionen bereitzustellen, die Auskunft über die Eigenschaften von benutzerdefinierten Routinen geben, etwa über ihre Selektivität. Diese Information ist zum Beispiel von zentraler Bedeutung für die Optimierung von Anfragen, in denen benutzerdefinierte Funktionen vorkommen.

Diese Funktionalität wird üblicherweise vom Datenbanksystementwickler implementiert. Ein SQL-Datenbanksystem zum Beispiel stellt einen Datentyp **INTEGER** mit einer Reihe von Indexstrukturen bereit, die Instanzen dieses Datentyps indexieren können, wie etwa der B-Baum. Bei benutzerdefinierten Datentypen obliegt die Bereitstellung solcher Funktionalität konsequenterweise dem Entwickler des benutzerdefinierten Datentyps.

1.9 Beispielanwendung

In diesem Lehrbuch verwenden wir weitgehend ein durchgängiges Anwendungsbeispiel. Wir nehmen an, dass im Diskursbereich folgende Objekttypen und -beziehungen vorliegen:

- Im System werden Daten über *Kunden*, *Angestellte*, *Bestellungen*, *Artikel* und *Kataloge* verwaltet.
- Über *Kunden* werden folgende Daten gespeichert: *Name*, *Geburtsdatum*, *Anschrift*, *Telefone* (Liste von Telefonnummern), *Kundennummer* und *Kreditlimit*.
  Kunden geben Bestellungen ab.
  Bei Kunden soll es möglich sein, a) die Anzahl ihrer Bestellungen und b) ihre letzte Bestellung zu ermitteln.
- Über *Angestellte* werden folgende Daten verwaltet: *Name*, *Geburtsdatum*, *Anschrift*, *Telefone* (Liste von Telefonnummern), *Angestelltennummer*, *Gehalt* (in Euro) und *Bewerbung* (Unterlagen zum Beispiel in PDF).

Jeder Angestellte kann einen Angestellten als Vorgesetzten haben. Umgekehrt kann ein Angestellter beliebig vielen Angestellten vorgesetzt sein. Einem Angestellten kann im Laufe der Zeit ein neuer Vorgesetzter zugeordnet werden.
Bei Angestellten soll es möglich sein, a) ihr Jahresgehalt mit der Formel *12*Gehalt+500* zu berechnen, b) ihr Gehalt jeweils um einen bestimmten Betrag (um drei Prozent) zu erhöhen und c) ihre Kenntnisse zu ermitteln.

- Es gibt spezielle Angestellte, die *Manager* darstellen. Manager haben zusätzlich ein Bonusgehalt. Bei Managern erfolgt zudem die Berechnung des Jahresgehalts über die Formel *12*Gehalt+Bonus*. Ebenso wird die Funktion zum Erhöhen des Gehalts für Manager überschrieben, so dass deren Gehalt um fünf Prozent steigt.
- Es kann Angestellte und Manager geben, die *gleichzeitig* auch Kunden sind.
- Ein *Artikel* weist folgende Eigenschaften auf: *Artikelnummer*, *Bezeichnung*, *Beschreibung* und *Bildkollektion*.
  Spezielle Artikel sind DVDs und Bücher.
- Bei einer *DVD* sind folgende speziellen Attribute von Interesse: *Region* (Code), *Sprachen*, *Untertitel* (Menge der Sprachen), *Laufzeit* (in Minuten), *Freigabe* (Altersbeschränkung) und *Erscheinungsjahr*.
- Ein *Buch* hat folgende speziellen Attribute: *ISBN*, *Autoren*, *Titel*, *Untertitel*, *Verlag* und *Erscheinungsjahr*.
  Bei Büchern soll es möglich sein, a) die Anzahl der Autoren und b) den Erstautor zu ermitteln.
- Es gibt noch eine dritte spezielle Art von Artikeln, die *Artikel-Sets*, die sich aus mindestens zwei Artikeln zusammensetzen. Artikel-Sets stellen eigenständige Artikel dar. Insbesondere ist ihr Preis unabhängig von den Preisen der zu einem Set zusammengefassten Artikel.
- Ein *Katalog* hat einen *Titel* und ein *Erscheinungsjahr*. Er beinhaltet eine Menge von Artikeln, die zu einem bestimmten *Preis* (in Euro) im Katalog erscheinen. Ein Artikel kann in verschiedenen Katalogen zu unterschiedlichen Preisen angeboten werden.
- Eine *Bestellung* besteht aus einer Liste von Positionen. Jede *Position* bezieht sich auf einen Artikel, der in einer bestimmten *Anzahl* und zu einem bestimmten *(Einzel-)Preis* (in Euro) bestellt wird.

Abbildung 1.10 zeigt eine Umsetzung dieser Anforderungen in einem UML-Schema. An dieser Stelle gehen wir nicht weiter auf die UML-Details ein. Die Konzepte von UML werden in Abschnitt 3.3.3 erläutert. Verschiedene Teile des UML-Schemas werden im Rahmen des Datenbankentwurfs in Kapitel 6 diskutiert.

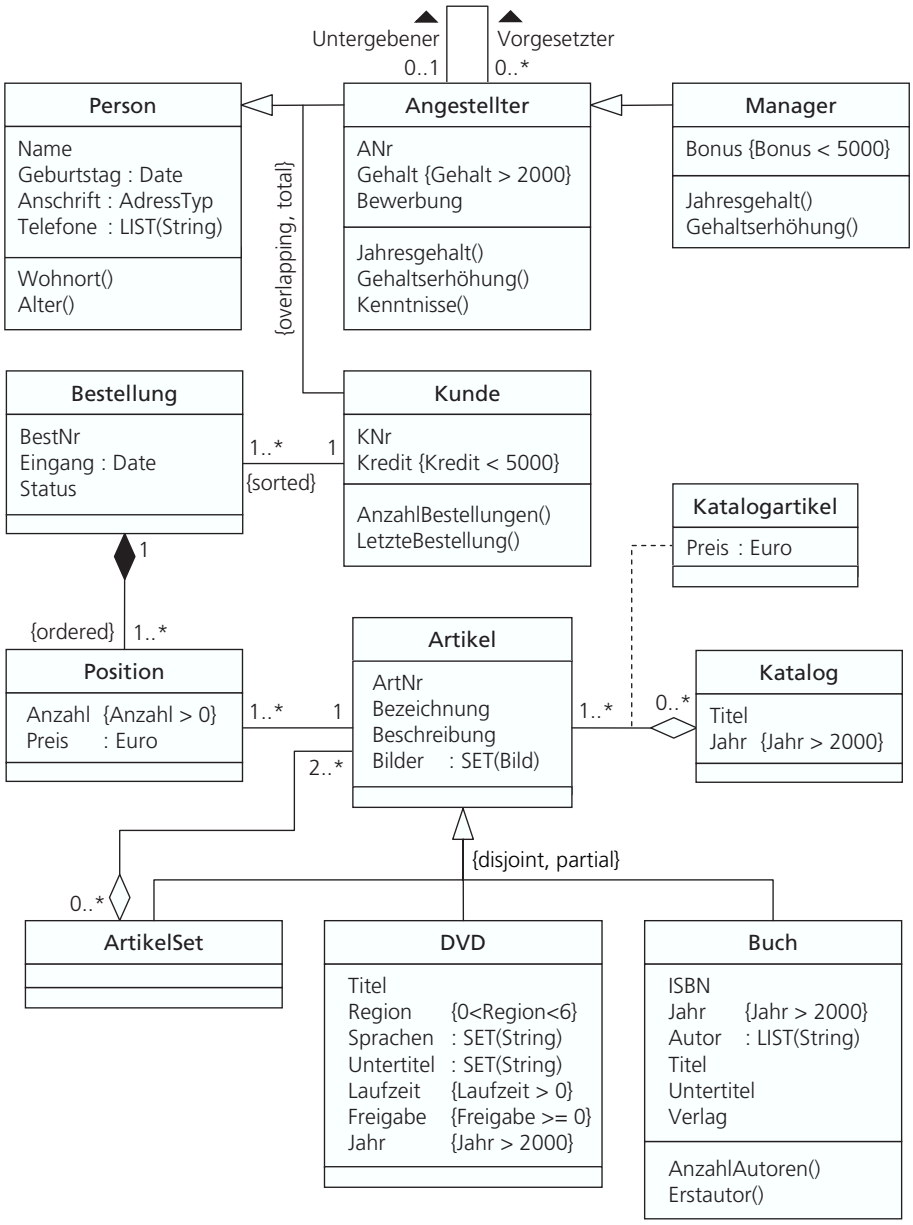

Abbildung 1.10: Beispielmodellierung

1.10 Gliederung des Buches

Das Buch umfasst Kapitel zu folgenden Themen, die für das Verständnis objektrelationaler Datenbanken relevant sind:

- *grundlegende Datenbankarchitekturen,*
- *grundlegende Datenmodellkonzepte,*
- *objektrelationale Konzepte,*
- *objektrelationales SQL,*
- *objektrelationaler Datenbankentwurf,*
- *relationale Umsetzung objektrelationaler Konzepte,*
- *objektrelationale Datenbankanwendungsprogrammierung,*
- *Implementierung objektrelationaler Konzepte,*
- *Erweiterbarkeit objektrelationaler Datenbanken und*
- *XML-Unterstützung objektrelationaler Datenbanken.*

Im Detail behandeln die Kapitel Folgendes:

- Kapitel 2 skizziert die typischen Architekturfragestellungen, wobei jeweils der Bezug zu objektrelationalen Datenbanken hergestellt wird.
- Kapitel 3 zeichnet die Evolution von Datenmodellen nach und führt insbesondere die grundlegenden relationalen sowie objektorientierten Datenmodellkonzepte ein.
- Kapitel 4 stellt die Konzepte objektrelationaler Datenmodelle zusammen. Die Präsentation dieser Konzepte abstrahiert so weit wie möglich von den syntaktischen Eigenarten der Konstrukte aktueller SQL-Sprachen. Bei Konzepten, die in den SQL-Dialekten unterschiedlich interpretiert bzw. umgesetzt werden, stellen wir die verschiedenen Varianten gegenüber.
- Kapitel 5 stellt Standard-SQL vor. Dabei stehen die objektrelationalen Sprachkonstrukte von Standard-SQL im Vordergrund der Präsentation.
- Kapitel 6 thematisiert den Entwurf von objektrelationalen Datenbanken. Hier wird vertieft auf den konzeptionellen Entwurf mittels UML und dessen Abbildung auf objektrelationales SQL eingegangen.
- Kapitel 7 demonstriert, welche Arbeit bei der Datendefinition durch ein objektrelationales Datenmodell abgenommen wird. In diesem Zusammenhang werden auch die Unterschiede zwischen verschiedenen Umsetzungen der objektrelationalen Konzepte bei der Formulierung und Ausführung von Anfragen herausgearbeitet.
- Kapitel 8 stellt mit JDBC und SQLJ zwei standardisierte Datenbankschnittstellen vor, welche die Anbindung von Java-Programmen an SQL-Datenbanken unterstützen. Im Fokus dieses Kapitels steht vor allem die Verwendung der objektrelationalen Erweiterungen in Java-Programmen.
- Kapitel 9 behandelt Implementierungsaspekte objektrelationaler Konzepte. Im Detail wird auf die Speicherung von Large Objects, Kollektionen,

Objektidentifikatoren, strukturierten Werten und Spezialisierungshierarchien eingegangen und zugehörige Indexstrukturen vorgestellt.

- Kapitel 10 geht auf die *Erweiterbarkeit* objektrelationaler Datenbanken ein. Es zeigt insbesondere die Möglichkeiten des SQL-Standards auf, um SQL durch in Java implementierte Klassen und Methoden zu erweitern. Am Beispiel von Informix und Oracle wird demonstriert, wie kommerzielle DBMS den Begriff der Erweiterbarkeit umsetzen.
- Kapitel 11 diskutiert das Zusammenspiel zwischen XML und objektrelationalen Datenbanken. Implizit gibt dieses Kapitel auch eine Antwort auf eine häufig gestellte Frage: »*Wird objektrelationale Datenbanktechnologie in der Praxis überhaupt eingesetzt?*« Am Beispiel von Oracle und DB2 sehen wir, dass zumindest die DBMS-Hersteller selbst ihre objektrelationalen Erweiterungen nutzen, um XML-Funktionalität bereitzustellen.

Das vorliegende Buch ist so gegliedert, dass die einzelnen Kapitel aufeinander aufbauen. Der Stoff dieses Buch ist eher für eine drei- oder mehrstündige Vertiefungsvorlesung über »*Objektrelationale Datenbanken*« ausgelegt. Im Rahmen einer zweistündigen Vorlesung müssten einige Kapitel etwas kürzer gehalten werden. Abbildung 1.11 illustriert die Abhängigkeiten zwischen den verschiedenen Kapiteln.

Das Kapitel 2 kann zum Beispiel kurz gehalten werden, wenn die dort vermittelten Kenntnisse bereits durch eine vorangegangene Vorlesung über »Datenbankgrundlagen« abgedeckt sind. Auch Kapitel 3 kann in einem solchen Fall etwas kompakter erörtert werden. Für das Verständnis objektrelationaler Datenbanken ist es jedoch unerlässlich, dass alle in diesem Kapitel eingeführten Typkonstruktoren mit ihren Operationen und den möglichen Typkompositionsregeln im Detail behandelt werden.

Kapitel 4 hingegen sollte ohne Abstriche ausführlich diskutiert werden. Der Inhalt dieses Kapitels kann dabei auch mit dem Inhalt von Kapitel 5 kombiniert werden. Das heißt, die jeweiligen Konzepte könnten gleich in Standard-SQL-Notation vorgestellt werden. Für Kapitel 5 gilt generell, dass der Fokus auf den Sprachkonstrukten liegen sollte, die für die objektrelationalen Erweiterungen relevant sind (vorausgesetzt, dass SQL-Grundlagen bei den Zuhörenden der Vorlesung vorhanden sind). Für die nachfolgenden Kapitel ist es wichtig, dass diese Sprachkonstrukte im Detail erklärt werden. Zur weiteren Vertiefung der objektrelationalen Sprachkonstrukte von Standard-SQL und vor allem von den SQL-Dialekten der führenden kommerziellen ORDBMS Oracle, DB2, Informix und PostgreSQL sollte das Buch [Tür03] als Begleitlektüre herangezogen werden.

Die Ausführlichkeit, mit der die Kapitel 6 und 7 präsentiert werden, hängt wesentlich von der zur Verfügung stehenden Zeit ab. In einer Kurzversion im Rahmen einer zweistündigen Vorlesung sollten auf jeden Fall die Modellierung mit UML und ihre Abbildung auf ein objektrelationales Schema enthalten sein.

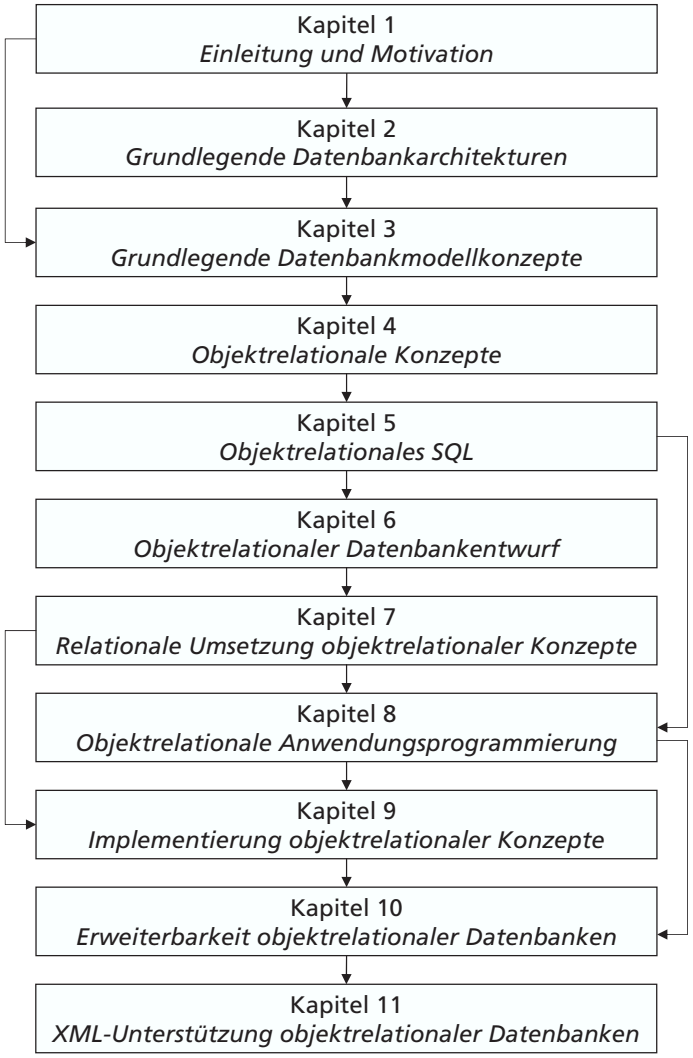

Abbildung 1.11: Gliederung des Buches

Ebenso sollte die relationale Umsetzung der objektrelationalen Konzepte diskutiert werden, um den Zuhörern den Vorteil objektrelationaler Datenbanktechnologie klar vor Augen zu führen. Kapitel 7 ist auch für eine Vorlesung über »Datenbankimplementierung« von besonderer Relevanz.

Kapitel 8 sollte ein unverzichtbarer Teil aller Vorlesungen über objektrelationale Datenbanken sein. Das Wichtige hierbei ist, dass klar gezeigt wird, wie die objektrelationalen Erweiterungen aus bzw. in Anwendungsprogrammen verarbeitet werden können. Hier sollten vor allem die benutzerdefinierten

Typabbildungen diskutiert werden. Dieses Kapitel kann unabhängig von den beiden Kapiteln zuvor behandelt werden. Auch hier sollte das Buch [Tür03] als Begleitlektüre für eine weitere Vertiefung verwendet werden, vor allem wenn es darum geht zu zeigen, wie die kommerziellen ORDBMS Oracle und Informix die objektrelationale Datenbankanwendungsprogrammierung unterstützen.

Kapitel 9 kann in einer zweistündigen Vorlesung eher kurz gehalten werden. Bei einer technisch ausgerichteten Vorlesung hingegen sollte der Inhalt dieses Kapitel vertiefend präsentiert werden. Dieses Kapitel ist auch von generellem Interesse für Vorlesungen über »Datenbankimplementierung«.

Aus Kapitel 10 sollte auf jeden Fall der Abschnitt über die Erweiterung mittels Standard-SQL im Detail besprochen werden. In einer Vorlesung mit einem praktischen Übungsteil sind auch die beiden Abschnitte über die Erweiterung mittels Informix DataBlades und Oracle Data Options von besonderer Relevanz.

Der Inhalt von Kapitel 11 ist im Prinzip optional für eine Vorlesung über »Objektrelationale Datenbanken«. Aus unserer Sicht stellt dieses Kapitel jedoch ein hervorragendes Beispiel für die Verwendung objektrelationaler Datenbanktechnologie dar. Insofern können Teile dieses Kapitel mit dem Inhalt von Kapitel 10 kombiniert werden.

Diverse Kapitel und Abschnitte dieses Buches können natürlich gut auch im Rahmen einer Vorlesung über »Datenbankgrundlagen« verwendet werden. Besonders interessant in diesem Kontext sind die Kapitel 5 bis 8. In diesem Fall würde der Fokus bei Kapitel 5 eher auf den SQL-Grundlagen liegen. Aus Kapitel 6 würde die Modellierung mit UML im Zentrum des Interesses stehen. Aus Kapitel 7 ist die Abbildung von Objektkonzepten auf relationale Entsprechungen fundamental für jeden Datenbankentwurf. Kapitel 8 ist ebenfalls ein Kandidat für eine Grundlagen-Vorlesung. In diesem Fall kann die Behandlung der objektrelationalen Konstrukte vernachlässigt werden, so dass die Java-SQL-Schnittstellen JDBC und SQLJ nur im Kontext relationaler SQL-Datenbanken besprochen werden.

An diversen Stellen im Buch werden wir uns auf konkrete objektrelationale Datenbanksysteme beziehen. Wir werden dabei hauptsächlich auf Systeme fokussieren, die grundlegende objektrelationale Konzepte wie Typkonstruktoren, strukturierte Typen bzw. Objekttypen sowie Tabellen- und Sichtenhierarchien unterstützen. Zu diesen Systemen zählen Oracle, IBM DB2, IBM Informix und PostgreSQL. Andere bekannte (relationale) Datenbanksysteme wie etwa Microsoft SQL Server, Sybase, Ingres oder MySQL finden in diesem Buch keine Beachtung, da sie keine der wesentlichen objektrelationalen Erweiterungen unterstützen. Nischenprodukte wie Caché von InterSystems oder Poet haben wir aufgrund ihrer geringen Bedeutung und der Tatsache, dass sie eigentlich Objektdatenbanksysteme mit einer SQL- bzw. OQL-Schnittstelle darstellen, vernachlässigt.