

1 Einführung

Datenbanksysteme zählen zu den wichtigsten Komponenten moderner Softwaresysteme. Ausgehend von den klassischen Anwendungsfällen wie Transaktionssysteme im Bankwesen oder betriebliche Informationssysteme haben sich in den letzten Jahren neue Einsatzgebiete eröffnet: von spezialisierten Informationssystemen (Krankenhausinformationssysteme, geografische Informationssysteme) über rechnerunterstützte Ingenieursysteme bis hin zu der Vielzahl neuer Internetanwendungen wie E-Commerce, Portale usw. Mit diesen neuen Anwendungsfeldern sind aber auch eine Reihe spezieller Anforderungen verbunden, die durch ein einziges System bzw. Systemkonzept nicht mehr abzudecken sind. Aus diesem Bedarf heraus wurden daher in den letzten Jahren neben den klassischen relationalen Datenbanksystemen verschiedene neue Datenbankkonzepte entwickelt: objektrelationale Erweiterungen [Sto96], objektorientierte Datenbanksysteme [Heu97, SST97] sowie spezialisierte Lösungen wie XML-Repositories oder Multimediaerweiterungen. Damit verbunden sind aber auch eine Vielzahl von Schnittstellenlösungen, Zugriffsmechanismen und Anfragetechniken, die dem Entwickler den Überblick und demzufolge auch die Entscheidung für eine bestimmte Technologie erschweren.

*Einsatzgebiete von
Datenbanken*

Seit der ersten Veröffentlichung im Jahre 1995 verfolgt die Firma Sun Microsystems das Ziel, mit der Programmiersprache Java den Anspruch einer Internet-Programmiersprache zu erfüllen. So ist Java objektorientiert und plattformneutral, mit weitreichenden Sicherheitsmechanismen ausgestattet und es verfügt über eine Bibliothek, die u.a. Klassen für die Entwicklung verteilter, netzwerkbasierter Anwendungen umfasst. Waren die ersten Java-Programme im Wesentlichen noch kleine Applets, die die bis zu diesem Zeitpunkt eher statischen Web-Inhalte um aktive Elemente erweiterten, so hat sich mit der zunehmenden Komplexität der Internetanwendungen auch schnell der Bedarf nach Mechanismen zur Datenbankanbindung entwickelt.

Heute – knapp fünf Jahre nach der ersten Veröffentlichung der Sprache – stehen eine Vielzahl von Technologien für die Entwicklung von Java-basierten Datenbankanwendungen zur Verfügung. Neben den klassischen Techniken der SQL-Call-Level-Schnittstelle und der SQL-

Spracheinbettung wurden auch Werkzeuge zur Abbildung zwischen Java-Objekten und Relationen und direkte ODBMS-Anbindungen entwickelt. Diese Bandbreite an Techniken in Verbindung mit dem weiteren Ausbau der Java-Plattform, z.B. im Bereich grafischer Benutzerschnittstellen, mobiler Systeme, Multimedia und Kommunikation, prädestiniert Java nicht nur für Internetanwendungen, sondern für nahezu jede Form von Datenbankanwendungen.

Vor diesem Hintergrund wollen wir mit dem vorliegenden Buch einen Überblick zu Java-Technologien für den Zugriff auf und die Arbeit mit Datenbanken geben.

1.1 Inhalt des Buches

Die Entwicklung von Datenbankanwendungen berührt eine ganze Reihe von Fragestellungen, beginnend bei der Auswahl eines geeigneten Datenbanksystems, der Datenmodellierung, der Zugriffsschnittstelle und der Anfragesprache über Aspekte der Architektur, wie z.B. Client-Server, web-basiert, zwei- oder dreistufig, bis hin zur Gestaltung der Benutzerschnittstelle. Die Behandlung aller dieser Aspekte und der dafür geeigneten Technologien würde den Rahmen eines Buches mit Sicherheit sprengen, so dass wir uns in diesem Buch auf die Techniken des Datenbankzugriffs konzentrieren. Wir beschränken uns jedoch nicht auf klassische relationale Datenbanksysteme, sondern betrachten auch die Verwendung objektrelationaler und objektorientierter Erweiterungen, die mit SQL:1999 eingeführt, durch Werkzeuge zur objektrelationalen Abbildung bereitgestellt oder direkt durch objektorientierte Datenbanksysteme implementiert werden. Auf diese Weise soll dem Entwickler der Vergleich und damit auch die Entscheidung für den Einsatz einer bestimmten Technologie erleichtert und dem interessierten Studenten ein Überblick über die wichtigsten Techniken der Datenbankanbindung gegeben werden.

Im Einzelnen ist der Inhalt des Buches wie folgt gegliedert: Im weiteren Verlauf dieses Kapitels werden wir zunächst grundlegende Fragestellungen der Entwicklung von Datenbankanwendungen diskutieren und die in diesem Buch benutzte Beispielanwendung vorstellen.

Kapitel 2 Im Kapitel 2 werden wir die für das Verständnis des Buches wichtigen *Sprachkonstrukte von Java* einführen, wobei wir jedoch Grundkenntnisse sowohl in der Programmierung als auch zu Java voraussetzen.

Kapitel 3 Kapitel 3 behandelt die *Grundlagen relationaler Datenbanksysteme*. Ausgehend vom Relationenmodell geben wir einen kurzen Über-

blick zu SQL und erläutern die wichtigsten Aspekte der Anwendungsentwicklung für RDBMS.

Kapitel 4 beschreibt mit JDBC die *Standardschnittstelle* von Java zu relationalen Datenbanksystemen. Dabei werden wir im Wesentlichen die aktuelle Version 3.0 vorstellen, jedoch auch auf Unterschiede zu den teilweise noch vorherrschenden älteren Versionen hinweisen. Kapitel 4

Ein weiteres Konzept der Anbindung von relationalen Datenbanken ist *Embedded SQL* als direkte Einbettung von SQL-Anweisungen in Java-Code. Diese Technik, die inzwischen auch im Rahmen von SQL:1999 standardisiert ist, wird in Kapitel 5 vorgestellt. Darüber hinaus behandeln wir in diesem Kapitel die Entwicklung *gespeicherter Prozeduren* (Stored Procedures) sowie die Implementierung neuer *SQL-Datentypen* in Java. Kapitel 5

Ein häufig angesprochenes Problem der Kopplung von SQL und einer (objektorientierten) Programmiersprache sind die unterschiedlichen Datenstrukturkonzepte: das Konzept der Relation als Menge von Tupeln bei SQL gegenüber dem Tupel bzw. dem Objekt als grundlegende Datenstruktur einer imperativen Programmiersprache. *Objekt-datenbanken* vermeiden diesen als *Impedance Mismatch* bezeichneten Gegensatz, indem sie die Datenstrukturen objektorientierter Sprachen direkt unterstützen. Mit der ODMG-Spezifikation ist hierfür auch ein Industriestandard verfügbar, der die Anbindung an Java festlegt. Am Beispiel von ODBMS FastObjects und ObjectStore werden wir diese Anbindung in Kapitel 6 beschreiben. Kapitel 6

Ein anderer Ansatz zur Vermeidung des Impedance Mismatch ist die Speicherung von Objekten in Datenbankrelationen. Auf diese Weise können Anwendungen die Vorzüge der Objektorientierung nutzen und gleichzeitig auf die etablierten und stabilen relationalen Datenbanksysteme zurückgreifen. Die transparente *Abbildung zwischen Objekten und Relationen* wird durch Methoden und Werkzeuge unterstützt, die wir im Kapitel 7 am Beispiel von Java Data Objects, einem neuen Standard zur transparenten Persistenz von Java-Objekten mit relationalen und objektorientierten Datenbanksystemen, vorstellen. Kapitel 7

Im Datenbankbereich sind *Transaktionen* die zu synchronisierenden Einheiten der Anwendungsprogrammierung im Mehrbenutzerbetrieb. Während übliche Datenbanksysteme das Transaktionskonzept für nur auf ihnen laufende Anwendungen bereits lokal zufrieden stellend realisieren, wird in Kapitel 8 nach einer Rekapitulation der wichtigsten Transaktionsbegriffe beschrieben, wie insbesondere verteilte Transaktionen auf mehreren Datenquellen mittels Transaktionsmonitoren korrekt realisiert werden können. Kapitel 8

Von Vertretern der Programmiersprachenentwicklung wird das Prinzip der *persistenten Programmiersprachen* als zukunftsweisende Kapitel 9

Alternative zur klassischen Entwicklung von Datenbankanwendungen propagiert. Persistente Programmiersprachen speichern direkt die Zustände von Programmobjekten auf dem persistenten Speicher. In Kapitel 9 wird mit *PJama* eine persistente Programmiersprachenentwicklung basierend auf Java vorgestellt.

Anhang A

Im Anhang A sind die Definitionen für das Schema der Beispielanwendung angegeben, sowohl als SQL-Skript für die relationale Datenbank als auch als Java-Klassen für die Objektdatenbank.

1.2 Architekturen für Datenbankanwendungen

Client-Server-Architektur

Moderne Datenbankanwendungen basieren im Allgemeinen auf Client-Server-Architekturen, die es ermöglichen, dass eine Vielzahl von Clients mit zentral verwalteten Daten und Diensten arbeiten können. Die Vorteile solcher Architekturen sind hinlänglich bekannt: Einerseits können durch die lokal verfügbare Rechenleistung die zentralen Server entlastet werden und andererseits bietet eine zentrale Verwaltung der Daten überhaupt erst die Möglichkeit, Datenbestände zu integrieren und effizient zu verwalten, Redundanzen zu vermeiden sowie auch im Mehrbenutzerbetrieb die Integrität der Daten sicherzustellen.

Anwendungspartitionierung

Neben der Entscheidung für eine konkrete Datenbanktechnologie spielt die Frage der Partitionierung der Anwendung eine wichtige Rolle. So ist festzulegen, wie Funktionen auf Clientanwendung und Datenbankserver aufzuteilen sind. Gerade für internetbasierte Anwendungen kann diese Aufteilung die Performance und Funktionalität des Systems nachhaltig beeinflussen. Grundsätzlich lassen sich für (interaktive) Datenbankanwendungen drei Funktionsgruppen unterscheiden:

- ❑ Präsentation und Benutzerinteraktion,
- ❑ die eigentliche Anwendungslogik
- ❑ sowie die Datenmanagementfunktionalität einschließlich der Anfragebearbeitung und Transaktionskontrolle.

Während die Zuordnung von Präsentations- und Interaktionsfunktionen zum Client sowie der Datenmanagementfunktionen zum Server nahe liegend ist, gibt es für die Verteilung der Anwendungslogik verschiedene Möglichkeiten.

2-Schichten-Architektur

Bei der *2-Schichten-Architektur* wird die Anwendungslogik meist vollständig im Client implementiert (Abb. 1.1). Das Datenbankmanagementsystem (DBMS) übernimmt in diesem Szenario die Rolle des Servers, der ausschließlich Datenbankoperationen (Anfragen und Ände-

rungen) ausführen und Anfrageergebnisse zum Client liefern kann. Client und Server kommunizieren dabei über das DBMS-spezifische Protokoll.

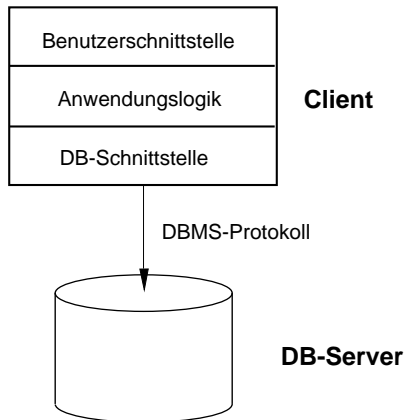


Abbildung 1.1
2-Schichten-
Architektur

Obwohl Anwendungen auf der Basis dieser Architektur vergleichsweise einfach zu entwickeln sind, ergeben sich hierbei eine Reihe von Nachteilen. Zum einen wird durch die Implementierung der Anwendungslogik im Client der Umfang des Codes deutlich vergrößert, was insbesondere bei Applets zu erhöhten Ladezeiten führt. Speziell bei Internetanwendungen besteht das Problem, dass z.B. ein Applet aus Sicherheitsgründen nur zum Host des Webservers Netzwerkverbindungen aufbauen darf und demzufolge das DBMS auch dort installiert sein muss bzw. geeignete Maßnahmen zur Weiterleitung der Aufrufe (z.B. spezielle Connection Manager, die als Konzentrador oder Proxy arbeiten) vorzusehen sind.

Weitere Schwierigkeiten, die sich aus der 2-Schichten-Architektur ergeben, sind

Probleme

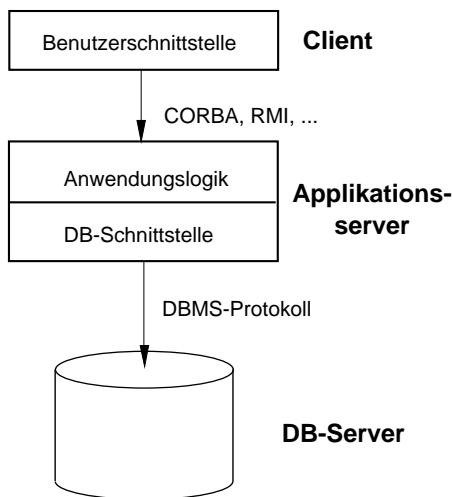
- ❑ eine hohe Netzwerkbelastung durch Operationen, die über vielen Datensätzen ausgeführt werden sollen und daher einen Transport dieser Datensätze zum Client erfordern,
- ❑ eingeschränkte Skalierbarkeit, da das DBMS den Flaschenhals bildet,
- ❑ und nicht zuletzt eine u.U. hohe Wartungs- und Installationskomplexität, wenn z.B. neue Versionen der Software auf alle Clients verteilt werden müssen.

Zum Teil lassen sich diese Probleme durch Verlagerung von Anwendungsfunktionen in die Datenbank in Form von gespeicherten Prozedu-

ren oder durch Applets, die selbstständig von einem Webserver geladen werden, umgehen.

Eine andere Alternative bildet die Verwendung einer *3-Schichten-Architektur*. Hier wird eine zusätzliche Schicht zwischen Client und Server eingeführt, die die eigentliche Anwendungslogik kapselt (Abb. 1.2). Dadurch entsteht ein Applikationsserver, der höherwertige Dienste in Form von Funktionen oder gar Objekten anbietet. Solche Dienste können z.B. das Eintragen eines Kunden, das Anlegen einer Bestellung oder das Ausliefern eines Produktes mit allen damit verbundenen Aktionen sein. Bei einer streng objektorientierten Vorgehensweise lassen sich diese Dienste applikationsspezifischen Objekten, wie Kunden, Bestellungen oder Produkten, zuordnen. Da Clients auf diese *Geschäfts- oder Business-Objekte* ausschließlich über die Methoden zugreifen können, wird sichergestellt, dass die Daten nur in der zulässigen Weise entsprechend der implementierten »Geschäftslogik« manipuliert werden.

Abbildung 1.2
3-Schichten-
Architektur



Für die Kommunikation zwischen Client und Applikationsserver werden im Allgemeinen spezielle Middleware-Lösungen eingesetzt, während Applikationsserver und Datenbankserver weiter über das jeweilige DBMS-Protokoll kommunizieren. Geeignete Mechanismen zum Zugriff auf den Applikationsserver sind speziell im Java/Internet-Umfeld Techniken wie CORBA, Remote Method Invocation (RMI) oder Web-Mechanismen wie HTTP in Verbindung mit Servlets.

Die 3-Schichten-Architektur ist insbesondere dann geeignet, wenn die Anwendungslogik in mehreren verschiedenen Clients genutzt werden soll oder wenn durch die Realisierung der Clients Einschränkungen

gegeben sind, z.B. bei der Implementierung als Applets oder über einfache HTML-Formulare. Auch bei datenintensiven Operationen im Rahmen von Internetanwendungen kann die Einführung eines separaten Applikationsservers, der direkten und unbeschränkten Zugriff auf den Datenbankserver hat, Vorteile bringen. Erkauft werden diese Vorteile durch einen erhöhten Entwicklungsaufwand, wobei jedoch gerade im Java-Umfeld eine Reihe von Technologien und Werkzeugen als Hilfsmittel zur Verfügung stehen. Beispiele hierfür sind die *Enterprise JavaBeans (EJB)*-Technologie, die eine Infrastruktur für serverseitig einsetzbare Komponenten bereitstellt, und *Web-Services*, die eine Inanspruchnahme von (entfernten) Diensten auf Basis von XML und verschiedenen möglichen Transportmedien erlauben.

Enterprise JavaBeans

1.3 Beispielanwendung

In diesem Buch verwenden wir eine durchgängige Beispielanwendung, anhand derer die Nutzung der einzelnen Technologien erklärt wird. Diese Anwendung stellt ein Verwaltungssystem für einen Onlineversand für Bücher dar. Es sind dabei Bücher, Kunden und Bestellungen zu verwalten.

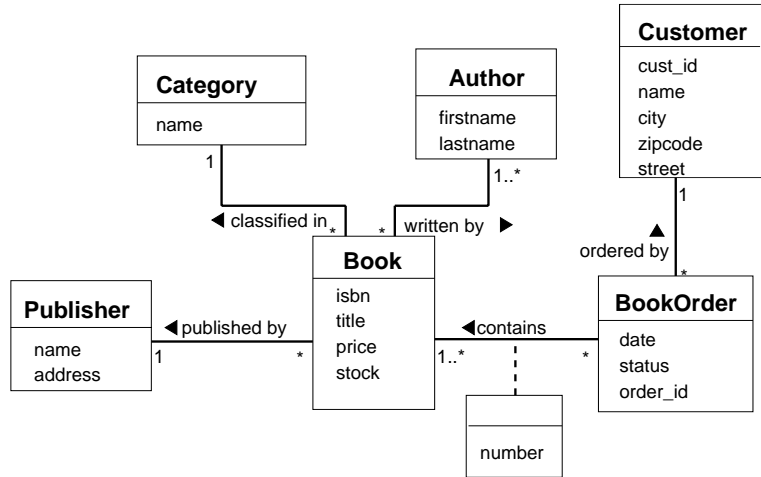
- ❑ Ein *Buch (Book)* ist durch die ISBN, Titel, Autoren, Preis sowie den aktuellen Bestand charakterisiert und einem Sachgebiet (Bellettristik, Informatik usw.) zugeordnet.
- ❑ Ein *Kunde (Customer)* wird durch Namen und Adresse sowie eine Kundennummer beschrieben.
- ❑ Eine *Bestellung (BookOrder)* ist schließlich genau einem Kunden zugeordnet und umfasst eine Menge von Büchern in einer bestimmten Anzahl. Außerdem besitzt jede Bestellung einen Status (offen oder ausgeliefert).

Das Modell dieser Anwendung ist als Klassendiagramm in UML-Notation [FS97, HK99] in Abbildung 1.3 dargestellt. Wir benutzen nur wenige Modellierungskonzepte, so dass das Diagramm auch ohne tiefer gehende UML-Kenntnisse verständlich sein sollte.

Neben Funktionen zur Pflege des Datenbestandes werden noch folgende Dienste benötigt:

- ❑ Recherche nach Büchern über den Titel, die Autoren oder das Sachgebiet,
- ❑ Aufgeben einer Bestellung durch Angabe einer Kundennummer, der gewünschten Bücher sowie der jeweiligen Anzahl,

Abbildung 1.3
Beispielanwendung in
UML



- Auslösen einer Bestellung, wobei zunächst zu prüfen ist, ob die gewünschten Bücher lieferbar sind, und anschließend der Preis zu berechnen ist.

Natürlich erfüllt die hier skizzierte Anwendung nicht die Anforderungen an ein wirkliches Buchhändler- oder Verlagssystem. Wir wollen uns aber im Weiteren auf die wesentlichen Aspekte einer Datenbankanwendung konzentrieren und Problemstellungen der Benutzerschnittstelle, der Realisierung einer Web-Schnittstelle oder der Datensicherheit durch Authentisierung und Verschlüsselung außer Betracht lassen. Hierfür sei auf die entsprechende Literatur zur Entwicklung von Benutzerschnittstellen in Java mit Hilfe von Swing [ELW98], zu Servlets [RS99], zu Enterprise JavaBeans [Mon99] und Middleware-Techniken [OH98] bzw. zu Sicherheitsmechanismen [Knu98] verwiesen.

1.4 Technische Hinweise

Obwohl standardisierte Technologien wie SQL oder ODMG eigentlich eine weitgehend herstellernerneutrale und damit portable Entwicklung von Datenbankanwendungen ermöglichen sollen, müssen dennoch immer wieder herstellerspezifische Besonderheiten und Inkompatibilitäten berücksichtigt werden. Auch stellt die Vielfalt an verfügbaren Plattformen und Systemen eine besondere Herausforderung dar. Wir haben uns daher für dieses Buch hinsichtlich relationaler Datenbanksysteme auf DB2 Universal Database von IBM, Oracle9i von Oracle und als frei verfügbares System auf MySQL sowie als Vertreter von Objektdatenbanken auf FastObjects (früher Poet) und ObjectStore konzentriert.

Trotzdem haben wir versucht, die Beispiele möglichst allgemein nutzbar zu gestalten. Wo dies nicht möglich war, wird explizit darauf hingewiesen.

Sofern notwendig, werden die folgenden Datenbank- und Verbindungsinformationen genutzt:

```
Datenbank: mydb
Server:    antarctica
Benutzer:  tux
Passwort:  pingus
```

*Verbindungs-
informationen*

An vielen Stellen des Buches wird auf Java-Klassen und -Schnittstellen verwiesen, die bei der Entwicklung von Datenbank Anwendungen genutzt werden. Wir haben darauf verzichtet, die vollständigen Schnittstellendefinitionen anzugeben, da die API-Dokumentation bei vielen Java-Entwicklungsumgebungen mitgeliefert wird bzw. vom Java-Webserver von Sun¹ bezogen werden kann. Auch lassen sich diese Dokumente mit einem Web-Browser besser »erforschen« als in der gedruckten Fassung. Stattdessen geben wir an den Stellen, wo es für das Verständnis wichtig ist, die relevanten Methoden der entsprechenden Klasse in der folgenden Form an:

API-Dokumentation

```
java.lang.Object
```

```
boolean equals (Object obj);
```

Hierbei ist **java.lang.Object** der Klassenname und `equals` eine Methode. Wir verzichten dabei auch jeweils auf die Angabe der Sichtbarkeit der Methoden, da alle angegebenen Methoden `public` sind.

¹<http://java.sun.com>