

Edward Sciore



DATABASE DESIGN AND IMPLEMENTATION

A SOFTWARE DEVELOPER'S HANDS-ON GUIDE TO DATABASE SYSTEMS

Combining an eminently readable style with a practical approach, Edward Sciore's *Database Design and Implementation* introduces you to database systems from a software developer's perspective. In its pages, you will learn how to use a database and how to develop one of your own.

Organized according to the components of a database, *Database Design and Implementation* takes you through database systems concepts from low-level disk access all the way to the query planner. Presenting only the essential algorithms and techniques that most clearly illustrate the issues discussed, the book covers such topics as data design and manipulation, integrity and security, memory and record management, metadata, query processing and optimization, integrity and security, indexing, and more.

As you proceed through the chapters and numerous exercises, you'll learn:

- How to build database applications in Java
- JDBC, JPA, XML, and Servlet development
- The internals of a database server
- Sophisticated techniques for indexing, sorting, intelligent buffer usage, and query optimization

Best of all, this book includes a simple but fully functional database system, SimpleDB, that enables you to apply your conceptual knowledge by examining and modifying the code. With numerous hands-on exercises and such tools as SimpleDB, *Database Design and Implementation* will give the self-studier as well as the traditional student an ideal introduction to the world of database systems.

Edward Sciore is an Associate Professor in the Computer Science Department at Boston College. He received his Ph.D. from Princeton University in 1980, and has been studying and teaching about database systems ever since. He is the author of numerous research articles about database systems. His favorite activity, however, is to teach database courses to college students. These teaching experiences, accumulated over a 25-year period, are what led to the writing of this book.



www.wiley.com/college/sciore

ISBN 978-0-471-75716-0

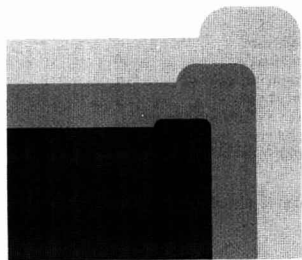


9 0000



9 780471 757160

DATABASE DESIGN AND IMPLEMENTATION



Edward Sciore
Boston College



WILEY

John Wiley & Sons, Inc.

ASSOCIATE PUBLISHER
SENIOR PRODUCTION EDITOR
EXECUTIVE MARKETING MANAGER
SENIOR DESIGNER
COVER PHOTO
COVER DESIGN
EDITORIAL ASSISTANT

Daniel Sayre
Sujin Hong
Christopher Rucl
Kevin Murphy
©Ryan McVay/Getty Images
David Levy
Carolyn Weisman

This book was set in Galliard by Aptara and printed and bound by R.R. Donnelly. The cover was printed by Phoenix Color.

This book is printed on acid free paper. ∞

Copyright © 2009 John Wiley & Sons, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, website www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030-5774, (201) 748-6011, fax (201) 748-6008, website www.wiley.com/go/permissions.

To order books or for customer service, please call 1-800-CALL WILEY (225-5945).

Library of Congress Cataloging-in-Publication Data

Sciore, Edward.

Database design and implementation/Edward Sciore.

p. cm.

Includes index.

ISBN 978-0-471-75716-0 (cloth)

1. Database design. 2. Computer software—Development. I. Title.

QA76.9.D26S38 2008

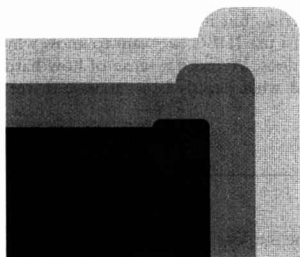
005.1—dc22

2008032213

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Preface



A database system is a common, visible tool in the corporate world—employees frequently interact directly with database systems to submit data or create reports. Database systems are also common, but invisible, as components of software systems. For example, consider an e-commerce website that uses a server-side database to hold customer, product, and sales information. Or consider a GPS navigation system that uses an embedded database to manage the road maps. In both of these examples, the presence of the database system is hidden from the user; the application code performs all of the database interaction.

From the point of view of a software developer, using a database directly is rather mundane, because modern database systems contain sophisticated front ends that make the creation of queries and reports straightforward. On the other hand, the possibility of incorporating database functionality into a software application is exciting, because it opens up a wealth of new and unexplored opportunities.

But what does “incorporating database functionality” mean? A database system provides many things, such as persistence, transactional support, and query processing. Which of these features are needed, and how should they be integrated into the software? Suppose for example that a programmer is asked to modify an existing application, say to add the ability to save state, or to increase reliability, or to improve the efficiency of file access. The programmer is faced with several architectural options. She could:

- purchase a full-featured, general-purpose database system, and then modify the application to connect to the database as a client;
- obtain a more specialized system that contains only the desired features, and whose code can be embedded directly into the application; or
- write the necessary functionality herself.

In order to make the proper choice, the programmer needs to understand what each of these options entail. She needs to know not only what database systems do, but also how they do it, and why.

This text examines database systems from the point of view of the software developer. It covers the traditional database system concepts, but from a systems perspective. This perspective allows us to investigate *why* database systems are the way they are. It is of course important to know how to write queries, but it is equally important to know how they are processed. We don't want to just use JDBC, we want to know why the API contains the classes and methods that it does. We need a sense of how hard it is to write a disk cache or logging facility. And what exactly is a database driver, anyway?

Organization of the Text

The text begins with an introductory chapter on the purpose and features of a database system. The remaining chapters are arranged into four parts:

Part 1 covers the fundamentals of relational databases. It contains five chapters, which respectively examine the concepts of *table*, *relationship*, *query*, *integrity*, and *efficiency*. In particular, Chapter 2 covers tables and their constraints, including keys and foreign keys. Chapter 3 introduces database design using UML diagrams, and normalization using BCNF. Chapter 4 examines both relational algebra and SQL in significant detail. Chapter 5 discusses database features that help ensure that the data remains accurate, namely assertions, triggers, and authorization. And Chapter 6 examines how indexes and materialized views can be used to make queries more efficient.

Part 2 is devoted to how to write a database application using Java. Chapter 7 introduces the client-server paradigm and the premise that application programs can act as clients of a database server. Chapter 8 presents the basics of JDBC, which is the fundamental API for Java programs that interact with a database. Chapter 9 examines how a Java application, by encapsulating its interaction with the database system, is able to provide the concept of *persistent* objects. Chapter 10 considers how XML can be a database-independent representation of data, and how it can be used for exchanging data between database systems. And Chapter 11 considers webserver-based applications that use servlets, and examines how their interaction with a database server compares with that of stand-alone applications.

Part 3 considers the internals of a database server. Each of its eight chapters covers a different database component, starting with the lowest level of abstraction (the disk and file manager) and ending with the highest (the JDBC client interface). The chapter for each component explains the issues and considers possible design decisions. As a

result, the reader can see exactly what services each component provides, and how it interacts with lower-level components to get what it needs. By the end of this part, the reader will have witnessed the gradual development of a simple but completely functional system.

Part 4 contains four chapters on efficient query processing. This part studies the sophisticated techniques and algorithms that can replace the simple design choices described in Part 3. Topics include indexing, sorting, intelligent buffer usage, and query optimization.

The SimpleDB Software

In my experience, most students can grasp conceptual ideas (such as concurrency control, buffer management, and query optimization) much more easily than they can grasp how these ideas are embodied inside a database system. Ideally, a student should write an entire database system as part of his coursework, just as the student would write an entire compiler in a compiler course. However, database systems are much more complex than compilers, so that approach is not practical. My solution was to write a simple but fully functional database system, called *SimpleDB*. Students can apply their conceptual knowledge by examining SimpleDB code and modifying it.

SimpleDB “looks” like a commercial database system, both in its function and structure. Functionally, it is a multi-user, transaction-oriented database server that executes SQL statements and interacts with clients via JDBC. Structurally, it contains the same basic components as a commercial system, with similar APIs. Each component of SimpleDB has a corresponding chapter in the text, which discusses the component’s code and the design decisions behind it.

SimpleDB is a useful educational tool because its code is small, easily readable, and easily modifiable. It omits all unnecessary functionality, implements only a tiny portion of SQL, and uses only the simplest (and often very impractical) algorithms. There consequently are numerous opportunities for students to extend the system with additional features and more efficient algorithms; many of these extensions appear as end-of-chapter exercises.

SimpleDB can be downloaded from the URL www.wiley.com/college/sciore. Details on installing and using SimpleDB are in its distribution file and in Chapter 7. I welcome suggestions for improving the code, as well as reports of any bugs; send email to sciore@bc.edu.

End-of-Chapter Readings

This text is motivated by the following two questions:

- What Java tools and techniques will best help us build an application that uses a database system?
- What functionality do database systems provide, and what algorithms and design decisions will best implement this functionality?

Of course, entire shelves can be filled with books that address one aspect or another of these questions. Since there is no way that a single text could hope to be comprehensive, I

have chosen to present only those algorithms and techniques that most clearly illustrate the issues involved. My overriding goal is to teach the principles behind a technique, even if it means omitting (or reducing) discussion of the most commercially viable version of it. Instead, the end of each chapter contains a “suggested reading” section. Those sections discuss interesting ideas and research directions that went unmentioned in the text, and provide references to relevant web pages, research articles, reference manuals, and books.

Suggested Course Contents

This book is best read sequentially, from cover to cover. However, there is much more material than typically fits into a single one-semester college course; thus some picking and choosing is necessary, depending on the goals of the course and the background of the students. Here are outlines for three possible courses that use this book.

A Usage-Centric Introductory Course

This course corresponds to the traditional first database course, but with a Java slant. The course covers Parts 1–2 in detail.

A System-Oriented Introductory Course

This course assumes that students have had no prior exposure to databases, leaving the teaching of “how to use a database” to another non-prerequisite course. This course lightly covers Chapters 1–2 and skims through Chapter 4, covering only the *select*, *project*, *group-by*, and *product* operators and their corresponding SQL. It skims Chapters 7–8 to establish JDBC competence. It then covers Part 3 in detail, and Chapters 21 and 24 of Part 4.

An Advanced Course in Database Implementation

This course assumes that students have already had a traditional first database course, and are familiar with SQL and relational algebra. The course covers Chapters 7–8 if students are not familiar with JDBC, and then does Parts 3 and 4 in detail. If desired, the instructor can supplement the text with the advanced readings suggested at the end of each chapter.

Text Prerequisites

This text is intended for upper-level undergraduate or beginning graduate courses in Computer Science. It assumes that the reader is comfortable with basic Java programming; for example, it uses the classes in *java.util* extensively, particularly collections and maps. Advanced Java concepts (such as RMI, JDBC, JPA, and servlets) are fully explained in the text.

SimpleDB implements each of its components as a Java package. Many students may have not had to deal with packages before; if so, I am glad that this text can rectify that deficiency.

Acknowledgements

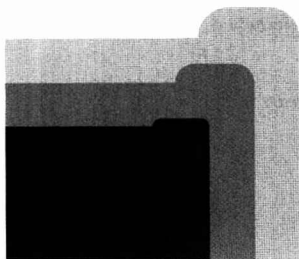
Writing this text was a labor of love. I am grateful to all of the people who supported me in this endeavor. In particular, earlier versions of this text were class-tested by Kate Lowrie at Boston College, Ronnie Ward at Texas A&M University, and Brian Howard at DePauw University. Their enthusiasm and detailed feedback helped to make the book considerably better.

I am also indebted to the reviewers of the text, including:

Hani Abu-Salem - *DePaul University*
 Cindy Chen - *University of Massachusetts, Lowell*
 Evangelos Christidis - *Florida International University*
 Martin H. Davis Jr. - *Wright State University*
 Henry Etlinger - *Rochester Institute of Technology*
 Amrit Goel - *Syracuse University*
 Delbert Hart - *University of Alabama, Huntsville*
 Latifur Khan - *University of Texas, Dallas*
 Andreas Koeller - *Montclair State University*
 Herman Lam - *University of Florida*
 Mohamed F. Mokbel - *University of Minnesota*
 Glen Nuckolls - *University of Texas*
 Iqbal Omar - *Texas A&M University, Kingsville*
 Richard Orwig - *Susquehanna University*
 Hassan Reza - *University of North Dakota*
 Lisa Singh - *Georgetown University*
 Il-Yeol Song - *Drexel University*
 Greg Speegle - *Baylor University*
 Chengyu Sun - *California State University, Los Angeles*
 Ankur Teredesai - *Rochester Institute of Technology*
 James Terwilliger - *Portland State University*
 Traian Truta - *Northern Kentucky University*
 Larry West - *Columbia College*
 Xiong Whang - *California State University, Fullerton*
 Xintao Wu - *University of North Carolina at Charlotte*
 Mohammed J. Zaki - *Rensselaer Polytechnic Institute*
 Lu Zhang - *DePaul University*
 Hairong Zhao - *Purdue University, Calumet*

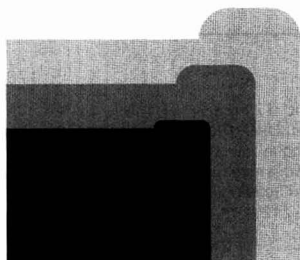
Finally, I feel very fortunate to have the love and unwavering encouragement of my family, especially my wife, Amy, and my children, Leah and Aaron. Amy's presence graces many of the examples in this book. Leah's name also appears in the text, but I'm not going to tell her where.

About the Author



Edward Sciore is an Associate Professor in the Computer Science Department at Boston College. He received his Ph.D. from Princeton University in 1980, and has been studying and teaching about database systems ever since. He is the author of numerous research articles about database systems. His favorite activity, however, is to teach database courses to college students. These teaching experiences, accumulated over a 25-year period, are what led to the writing of this text.

Contents



| | | |
|--|-----------------------------|-----------|
| 1 | | |
| Introduction: | | |
| Why a Database System? | 1 | |
| 1.1 Databases and Database Systems | 1 | |
| 1.2 Record Storage | 4 | |
| 1.3 Multi-User Access | 5 | |
| 1.4 Memory Management | 6 | |
| 1.5 Data Models and Schemas | 6 | |
| 1.6 Physical Data Independence | 9 | |
| 1.7 Logical Data Independence | 10 | |
| 1.8 Chapter Summary | 12 | |
| 1.9 Suggested Reading | 13 | |
| 1.10 Exercises | 13 | |
| 2 | | |
| Data Definition | | 17 |
| 2.1 Tables | 17 | |
| 2.2 Null Values | 18 | |
| 2.3 Keys | 19 | |
| 2.4 Foreign Keys and Referential Integrity | 21 | |
| 2.5 Integrity Constraints | 24 | |
| 2.6 Specifying Tables in SQL | 26 | |
| 2.7 Chapter Summary | 27 | |
| 2.8 Suggested Reading | 29 | |
| 2.9 Exercises | 29 | |
| 3 | | |
| Data Design | | 31 |
| 3.1 Designing Tables Is Difficult | 32 | |
| 3.2 Class Diagrams | 33 | |
| PART 1 | Relational Databases | 15 |

| | |
|---|----|
| 3.3 Transforming Class Diagrams to Tables | 36 |
| 3.4 The Design Process | 39 |
| 3.5 Relationships as Constraints | 54 |
| 3.6 Functional Dependencies and Normalization | 57 |
| 3.7 Chapter Summary | 63 |
| 3.8 Suggested Reading | 64 |
| 3.9 Exercises | 65 |

4 Data Manipulation 70

| | |
|------------------------|-----|
| 4.1 Queries | 70 |
| 4.2 Relational Algebra | 71 |
| 4.3 SQL Queries | 91 |
| 4.4 SQL Updates | 109 |
| 4.5 Views | 111 |
| 4.6 Chapter Summary | 114 |
| 4.7 Suggested Reading | 116 |
| 4.8 Exercises | 117 |

5 Integrity and Security 121

| | |
|---|-----|
| 5.1 The Need for Integrity and Security | 121 |
| 5.2 Assertions | 122 |
| 5.3 Triggers | 124 |
| 5.4 Authorization | 126 |
| 5.5 Mandatory Access Control | 133 |
| 5.6 Chapter Summary | 135 |
| 5.7 Suggested Reading | 136 |
| 5.8 Exercises | 136 |

6 Improving Query Efficiency 139

| | |
|--|-----|
| 6.1 The Virtues of Controlled Redundancy | 139 |
| 6.2 Materialized Views | 141 |
| 6.3 Indexes | 148 |
| 6.4 Chapter Summary | 158 |
| 6.5 Suggested Reading | 159 |
| 6.6 Exercises | 160 |

PART 2 Client-Server Database Systems 163

7 Clients and Servers 165

| | |
|---|-----|
| 7.1 The Data-Sharing Problem | 165 |
| 7.2 Database Clients and Servers | 166 |
| 7.3 The <i>Derby</i> and <i>SimpleDB</i> Database Servers | 168 |
| 7.4 Running Database Clients | 171 |
| 7.5 The <i>Derby</i> <i>ij</i> Client | 173 |
| 7.6 The <i>SimpleDB</i> Version of SQL | 174 |
| 7.7 Chapter Summary | 176 |
| 7.8 Suggested Reading | 176 |
| 7.9 Exercises | 176 |

8 Using JDBC 178

| | |
|-------------------------------|-----|
| 8.1 Basic JDBC | 178 |
| 8.2 Advanced JDBC | 190 |
| 8.3 Computing in Java vs. SQL | 206 |
| 8.4 Chapter Summary | 209 |
| 8.5 Suggested Reading | 211 |
| 8.6 Exercises | 212 |

9 Persistent Java Objects 214

| | |
|---|-----|
| 9.1 The Domain Model and View of a Client Program | 214 |
| 9.2 The Problems with Our Domain Model | 233 |
| 9.3 The Java Persistence Architecture | 237 |
| 9.4 The Java Persistence Query Language | 246 |
| 9.5 Downloading a JPA Implementation | 253 |
| 9.6 Chapter Summary | 253 |
| 9.7 Suggested Reading | 255 |
| 9.8 Exercises | 255 |

10 Data Exchange 257

| | |
|--|-----|
| 10.1 Sharing Database Data with Nonusers | 257 |
| 10.2 Saving Data in an XML Document | 259 |
| 10.3 Restructuring an XML Document | 264 |
| 10.4 Generating XML Data from the Database | 276 |
| 10.5 Chapter Summary | 281 |
| 10.6 Suggested Reading | 282 |
| 10.7 Exercises | 283 |

11**Webserver-Based Database Clients 286**

- 11.1 Types of Database Clients 286
- 11.2 Interacting with a Web Server 288
- 11.3 Basic Servlet Programming 293
- 11.4 Managing Database Connections 301
- 11.5 Configuring a Servlet Container 302
- 11.6 Chapter Summary 305
- 11.7 Suggested Reading 306
- 11.8 Exercises 307

PART 3**Inside the Database Server 309****12****Disk and File Management 313**

- 12.1 Persistent Data Storage 313
- 12.2 The Block-Level Interface to the Disk 324
- 12.3 The File-Level Interface to the Disk 326
- 12.4 The Database System and the OS 330
- 12.5 The SimpleDB File Manager 331
- 12.6 Chapter Summary 339
- 12.7 Suggested Reading 340
- 12.8 Exercises 341

13**Memory Management 345**

- 13.1 Two Principles of Database Memory Management 345
- 13.2 Managing Log Information 347
- 13.3 The SimpleDB Log Manager 349
- 13.4 Managing User Data 356
- 13.5 The SimpleDB Buffer Manager 362
- 13.6 Chapter Summary 372
- 13.7 Suggested Reading 372
- 13.8 Exercises 373

14**Transaction Management 376**

- 14.1 Transactions 377
- 14.2 Using Transactions in SimpleDB 379
- 14.3 Recovery Management 381
- 14.4 Concurrency Management 398
- 14.5 Implementing SimpleDB Transactions 418

- 14.6 Testing Transactions 422
- 14.7 Chapter Summary 424
- 14.8 Suggested Reading 425
- 14.9 Exercises 426

15**Record Management 433**

- 15.1 The Record Manager 433
- 15.2 Implementing a File of Records 440
- 15.3 The SimpleDB Record Manager 446
- 15.4 Chapter Summary 462
- 15.5 Suggested Reading 463
- 15.6 Exercises 464

16**Metadata Management 467**

- 16.1 The Metadata Manager 467
- 16.2 Table Metadata 469
- 16.3 View Metadata 473
- 16.4 Statistical Metadata 475
- 16.5 Index Metadata 480
- 16.6 Implementing the Metadata Manager 484
- 16.7 Chapter Summary 486
- 16.8 Suggested Reading 487
- 16.9 Exercises 487

17**Query Processing 490**

- 17.1 Scans 490
- 17.2 Update Scans 495
- 17.3 Implementing Scans 496
- 17.4 Pipelined Query Processing 504
- 17.5 The Cost of a Scan 506
- 17.6 Plans 511
- 17.7 Predicates 517
- 17.8 Chapter Summary 526
- 17.9 Suggested Reading 527
- 17.10 Exercises 527

18**Parsing 531**

- 18.1 Syntax vs. Semantics 531
- 18.2 Lexical Analysis 532
- 18.3 Implementing the Lexical Analyzer 534

| | | |
|------|------------------------------|-----|
| 18.4 | Grammars | 537 |
| 18.5 | Recursive–Descent Parsers | 540 |
| 18.6 | Adding Actions to the Parser | 542 |
| 18.7 | Chapter Summary | 555 |
| 18.8 | Suggested Reading | 556 |
| 18.9 | Exercises | 556 |

19

Planning 561

| | | |
|------|-----------------------------------|-----|
| 19.1 | The SimpleDB Planner | 561 |
| 19.2 | Verification | 563 |
| 19.3 | Query Planning | 564 |
| 19.4 | Update Planning | 570 |
| 19.5 | Implementing the SimpleDB Planner | 572 |
| 19.6 | Chapter Summary | 575 |
| 19.7 | Suggested Reading | 576 |
| 19.8 | Exercises | 576 |

20

The Database Server 580

| | | |
|------|---|-----|
| 20.1 | Server Databases vs. Embedded Databases | 580 |
| 20.2 | Client–Server Communication | 583 |
| 20.3 | Implementing the Remote Interfaces | 586 |
| 20.4 | Implementing the JDBC Interfaces | 592 |
| 20.5 | Chapter Summary | 595 |
| 20.6 | Suggested Reading | 596 |
| 20.7 | Exercises | 596 |

PART 4 Efficient Query Processing 599

21

Indexing 601

| | | |
|------|--------------------------------------|-----|
| 21.1 | The <i>Index</i> Interface | 601 |
| 21.2 | Static Hash Indexes | 605 |
| 21.3 | Extendable Hash Indexes | 608 |
| 21.4 | B-Tree Indexes | 613 |
| 21.5 | Index-Aware Operator Implementations | 635 |
| 21.6 | Index Update Planning | 641 |
| 21.7 | Chapter Summary | 644 |
| 21.8 | Suggested Reading | 645 |
| 21.9 | Exercises | 646 |

22

Materialization and Sorting 651

| | | |
|------|------------------------------|-----|
| 22.1 | The Value of Materialization | 651 |
| 22.2 | Temporary Tables | 652 |
| 22.3 | Materialization | 653 |
| 22.4 | Sorting | 658 |
| 22.5 | Grouping and Aggregation | 670 |
| 22.6 | Merge Joins | 676 |
| 22.7 | Chapter Summary | 682 |
| 22.8 | Suggested Reading | 683 |
| 22.9 | Exercises | 684 |

23

Effective Buffer Utilization 687

| | | |
|------|---|-----|
| 23.1 | Buffer Usage in Query Plans | 687 |
| 23.2 | Multibuffer Sorting | 688 |
| 23.3 | Multibuffer Product | 691 |
| 23.4 | Implementing the Multibuffer Operations | 692 |
| 23.5 | Hash Joins | 699 |
| 23.6 | Comparing the Join Algorithms | 703 |
| 23.7 | Chapter Summary | 705 |
| 23.8 | Suggested Reading | 706 |
| 23.9 | Exercises | 707 |

24

Query Optimization 710

| | | |
|-------|--|-----|
| 24.1 | Equivalent Query Trees | 711 |
| 24.2 | The Need for Query Optimization | 720 |
| 24.3 | The Structure of a Query Optimizer | 724 |
| 24.4 | Finding the Most Promising Query Tree | 725 |
| 24.5 | Finding the Most Efficient Plan | 737 |
| 24.6 | Combining the Two Stages of Optimization | 739 |
| 24.7 | Merging Query Blocks | 747 |
| 24.8 | Chapter Summary | 748 |
| 24.9 | Suggested Reading | 750 |
| 24.10 | Exercises | 750 |

References 754

Index 757

List of Figures

Chapter 1

- Figure 1-1 Some records for a university database 2
- Figure 1-2 Implementing the STUDENT records in a text file 4
- Figure 1-3 Two ways to retrieve the name of students graduating in 1997 8
- Figure 1-4 The three schema levels 11

Chapter 2

- Figure 2-1 The schema of the university database 18
- Figure 2-2 Foreign keys for the university database 22
- Figure 2-3 An example of referential integrity 23
- Figure 2-4 The SQL specification of the STUDENT table 26

Chapter 3

- Figure 3-1 A class diagram for the university database 34
- Figure 3-2 A class diagram for the example CD database 37
- Figure 3-3 The algorithm to transform a relational schema to a class diagram 37
- Figure 3-4 The algorithm to transform a class diagram to a relational schema 38
- Figure 3-5 Two relational schemas generated from Figure 3-1 39
- Figure 3-6 The requirements specification for the university database 40
- Figure 3-7 Extracting nouns and verbs from the requirements specification 41
- Figure 3-8 A preliminary class diagram based on the nouns and verbs in Figure 3-7 42
- Figure 3-9 A class diagram containing an inadequate relationship 43

- Figure 3-10 Two ways to handle a multi-way relationship 44
- Figure 3-11 The class diagram that results from reifying the inadequate relationships 45
- Figure 3-12 The class diagram that results from removing redundant relationships 46
- Figure 3-13 Reifying a many-many relationship 47
- Figure 3-14 The need to reify a many-many relationship 48
- Figure 3-15 Reifying a weak-weak many-one relationship 49
- Figure 3-16 Reifying a weak one-one relationship 50
- Figure 3-17 Dealing with strong-strong relationships 51
- Figure 3-18 An algorithm to turn a class into an attribute 52
- Figure 3-19 Adding attributes to a class diagram 53
- Figure 3-20 Adding a requirement that cannot be easily enforced 56
- Figure 3-21 Adding the car manufacturer to the database 59
- Figure 3-22 Dealing with an omitted relationship between PROF and COURSE 60
- Figure 3-23 Redundancy that FDs cannot catch 62

Chapter 4

- Figure 4-1 A query tree for Q3 73
- Figure 4-2 A query tree for Q6 74
- Figure 4-3 The output of query Q12 77
- Figure 4-4 The output of query Q13 77
- Figure 4-5 The output of queries Q18-Q20 79
- Figure 4-6 The query tree for Q21 80
- Figure 4-7 The output of query Q22 81
- Figure 4-8 The query tree for Q23 82
- Figure 4-9 The query tree for Q25-Q29 83

- Figure 4-10 The query tree to find the grades Joe received during his graduation year 85
- Figure 4-11 The query tree for Q38–Q40 87
- Figure 4-12 The query tree for Q49–Q51 89
- Figure 4-13 The output of query Q55 90
- Figure 4-14 Numeric types in SQL 93
- Figure 4-15 Some common SQL string functions 95
- Figure 4-16 Some common SQL date/interval functions 96
- Figure 4-17 The result of query Q68 97
- Figure 4-18 A query tree for Q72 98

Chapter 5

- Figure 5-1 The SQL specification of the integrity constraint “No section can have more than 30 students” 123
- Figure 5-2 The SQL specification of the integrity constraint “A student’s graduation year must be at least 1863” 123
- Figure 5-3 The SQL specification of the integrity constraint “Students can take a course at most once” 124
- Figure 5-4 An SQL trigger that logs changes to student grades 125
- Figure 5-5 An SQL trigger that replaces inappropriate graduation years with nulls 126
- Figure 5-6 Some SQL grant statements for the university database 127
- Figure 5-7 Some SQL statements and their required privileges 130
- Figure 5-8 Using a view to limit the power of professors 131
- Figure 5-9 An easy way to abuse authorization 133
- Figure 5-10 What should be the classification of this query’s output table? 134

Chapter 6

- Figure 6-1 The effect of a redundant relationship 140
- Figure 6-2 The value of a redundant *Courseld* field 141

- Figure 6-3 A materialized view that adds *Courseld* to ENROLL 143
- Figure 6-4 A materialized view defined by aggregation 144
- Figure 6-5 An example of denormalization 145
- Figure 6-6 Materialized views that simulate denormalization 146
- Figure 6-7 Two queries that find the courses taken by Joe 147
- Figure 6-8 SQL statements to create indexes 150
- Figure 6-9 The *SID_IDX* and *MAJOR_IDX* indexes for STUDENT 151
- Figure 6-10 Implementing a query with and without an index 152
- Figure 6-11 Implementing another query with and without an index 153
- Figure 6-12 Implementing a range query 155
- Figure 6-13 Implementing a join query 156
- Figure 6-14 Two ways to use indexes to implement a query 157

Chapter 7

- Figure 7-1 Setting the classpath for an installation of Derby 169
- Figure 7-2 An *ij* session to find the two professors who taught the most courses 174
- Figure 7-3 An *ij* session to change the titles of various courses 175

Chapter 8

- Figure 8-1 The API for basic JDBC 179
- Figure 8-2 The JDBC code for the *StudentMajor* client 181
- Figure 8-3 JDBC code for the *ChangeMajor* client 184
- Figure 8-4 Using *ResultSetMetaData* to print the schema of a result set 187
- Figure 8-5 The JDBC code for the *SQLInterpreter* client 189
- Figure 8-6 Part of the API for the class *DriverManager* 190
- Figure 8-7 Connecting to a Derby server using *DriverManager* 191
- Figure 8-8 Creating a data source for a Derby server 192

- Figure 8-9 Code that could behave incorrectly in autocommit mode 194
- Figure 8-10 More code that could behave incorrectly in autocommit mode 194
- Figure 8-11 The *Connection* API for handling transactions explicitly 195
- Figure 8-12 A revision of Figure 8-9 that handles transactions explicitly 196
- Figure 8-13 Two concurrent transactions that could manage to “lose” an update 198
- Figure 8-14 A transaction that could give out more rebates than expected 199
- Figure 8-15 The JDBC code for the *FindMajors* client 201
- Figure 8-16 Revising the *FindMajors* client to use prepared statements 202
- Figure 8-17 Part of the API for *PreparedStatement* 203
- Figure 8-18 Using a prepared statement in a loop 203
- Figure 8-19 Part of the API for *ResultSet* 204
- Figure 8-20 Revising the code of Figure 8-9 205
- Figure 8-21 An alternative (but bad) way to code the *StudentMajor* client 207
- Figure 8-22 A clever way to recode Step 2 of the *FindMajors* client 209
- Figure 9-11 Terminology used by JPA 237
- Figure 9-12 Part of the API for the *EntityManager* interface 238
- Figure 9-13 A JPA client to test the entity manager 239
- Figure 9-14 The JPA version of the *Student* model class 241
- Figure 9-15 A typical JPA persistence descriptor 245
- Figure 9-16 Part of the API that supports JPA queries 246
- Figure 9-17 Queries to retrieve the students who took basket weaving 248
- Figure 9-18 Two JPQL queries that use path expressions 248
- Figure 9-19 A JPQL query that retrieves triples of objects 249
- Figure 9-20 A JPA client to print the name of students graduating in 2005 251
- Figure 9-21 Two ways to find the students in Joe’s calculus section 252

Chapter 9

- Figure 9-1 A code fragment of a more object-oriented version of *FindMajors* 215
- Figure 9-2 The API for the object-relational mapping of the university database 218
- Figure 9-3 Printing the titles of the courses taken by a given student 219
- Figure 9-4 An API for the model class *DatabaseManager* 221
- Figure 9-5 Using the class *DatabaseManager* 221
- Figure 9-6 A screenshot of the *StudentInfo* client 222
- Figure 9-7 The code for the *StudentInfo* view 223
- Figure 9-8 The code for the model class *Student* 226
- Figure 9-9 The code for the DAO class *StudentDAO* 228
- Figure 9-10 The code for the class *DatabaseManager* 231
- Figure 10-1 A possible format for data on graduating students 258
- Figure 10-2 An XML document that represents data on graduating students 260
- Figure 10-3 Revising Figure 10-2 so that the graduation year appears once 261
- Figure 10-4 Using nested XML tags to represent data on graduating students 262
- Figure 10-5 A tree representation of the XML document of Figure 10-4 264
- Figure 10-6 Extracting the student names from Figure 10-4 265
- Figure 10-7 An iterative XSLT program to extract student names 267
- Figure 10-8 A browser-based view of the student data 268
- Figure 10-9 An XSLT program to transform Figure 10-4 into HTML 268
- Figure 10-10 SQL insert commands corresponding to Figure 10-4 270
- Figure 10-11 An XSLT program to transform Figure 10-4 into SQL insert commands 271
- Figure 10-12 An XSLT program to transform Figure 10-4 into Figure 10-2 272