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 is it 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.