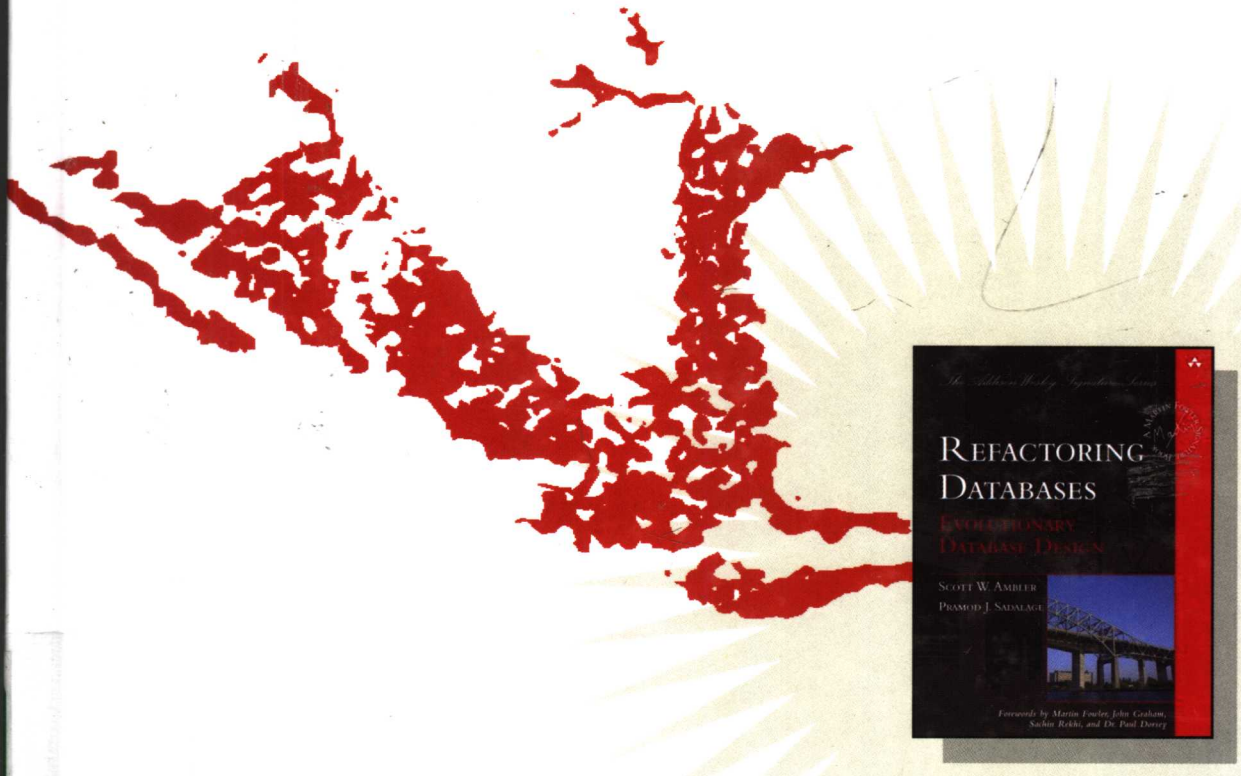
PEARSON
Addison Wesley

典藏原版书苑

Refactoring Databases
Evolutionary Database Design

[美] Scott W. Ambler Pramod J. Sadalage 著

数据库重构(英文版)



荣获2007年第17届Jolt生产力大奖

人民邮电出版社
POSTS & TELECOM PRESS

典藏原版书苑

数据库重构

（英文版）

Refactoring Databases
Evolutionary Database Design

[美] Scott W. Ambler
Pramod J. Sadalage 著

人民邮电出版社
北 京

图书在版编目（CIP）数据

数据库重构/（美）安布勒（Ambler, S. W.），（美）萨达拉戈（Sadalage, P. J.）著.
—北京：人民邮电出版社，2007.6
（典藏原版书苑）

ISBN 978-7-115-15570-2

Ⅰ. 重… Ⅱ. ①安… ②萨… Ⅲ. 数据库系统—程序设计—英文 Ⅳ. TP311.13

中国版本图书馆 CIP 数据核字（2006）第 147794 号

典藏原版书苑
数据库重构（英文版）

◆ 著 [美] Scott W. Ambler Pramod J. Sadalage
责任编辑 付 飞

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京铭成印刷有限公司印刷
新华书店总店北京发行所经销

◆ 开本：800×1000 1/16
印张：23.5
字数：470 千字 2007 年 6 月第 1 版
印数：1－3 000 册 2007 年 6 月北京第 1 次印刷

著作权合同登记号 图字：01-2007-0860 号

ISBN 978-7-115-15570-2/TP

定价：65.00 元

读者服务热线：(010)67132705 印装质量热线：(010)67129223

内容提要

重构技术已经在领域广泛的开发项目中证明了自身的价值——帮助软件专业人士改善系统的设计、可维护性、可扩展性和性能。本书首次披露了为数据库系统专门设计的强大重构技术。

本书展示了如何在不改变语义的情况下，对表结构、数据、存储过程及触发器等略作改动，就可以给数据库设计带来实质上的飞跃。

这是一本内容全面的参考、指南书，全面介绍了数据库重构涉及的每个基本观念，运用完整的实例，带领读者学习从重构简单的孤立数据库应用程序到重构复杂的多应用程序环境的全过程，并讲述了数据库重整的五大主要类别。读者将学会如何运用重构改善数据库结构、数据质量和参照完整性，如何同时对结构和方法进行重整。本书提供了用 Oracle 和 Java 建立的多种实例，并可以方便地转换成 C#、C++、VB.NET 等其他语言或 DB2、SQL Server、MySQL、Sybase 等其他数据库。

运用本书的技术和实例，读者可以减少浪费和重复工作，降低风险和成本，建立能够顺利发展以适应未来需求的数据库系统。

Praise for *Refactoring Databases*

"This groundbreaking book finally reveals why database schemas need not be difficult to change, why data need not be difficult to migrate, and why database professionals need not be overburdened by change requests from customers and developers. Evolutionary design is at the heart of agility. Ambler and Sadalage have now shown the world how to evolve agile databases. Bravo!"
—Joshua Kerievsky, founder, Industrial Logic, Inc.; author, *Refactoring to Patterns*

"This book not only lays out the fundamentals for evolutionary database development, it provides many practical, detailed examples of database refactoring. It is a must read for database practitioners interested in agile development."
—Doug Barry, president, Barry & Associates, Inc.; author of *Web Services and Service-Oriented Architectures: The Savvy Manager's Guide*

"Ambler and Sadalage have taken the bold step of tackling an issue that other writers have found so daunting. Not only have they addressed the theory behind database refactoring, but they have also explained step-by-step processes for doing so in a controlled and thoughtful manner. But what really blew me away were the more than 200 pages of code samples and deep technical details illustrating how to overcome specific database refactoring hurdles. This is not just another introductory book convincing people that an idea is a good one—this is a tutorial and technical reference book that developers and DBAs alike will keep near their computers. Kudos to the brave duo for succeeding where others have failed to even try."
—Kevin Aguanno, senior project manager, IBM Canada Ltd.

"Anybody working on non-greenfield projects will recognize the value that Scott and Pramod bring to the software development life cycle with Refactoring Databases. *The realities of dealing with existing databases is one that is tough to crack. Though much of the challenge can be cultural and progress can be held in limbo by strong-armed DBA tactics, this book shows how technically the refactoring and evolutionary development of a database can indeed be handled in an agile manner. I look forward to dropping off a copy on the desk of the next ornery DBA I run into."*
—Jon Kern

"This book is excellent. It is perfect for the data professional who needs to produce results in the world of agile development and object technology. A well-organized book, it shows the what, why, and how of refactoring databases and associated code. Like the best cookbook, I will use it often when developing and improving databases."
—David R. Haertzen, editor, The Data Management Center, First Place Software, Inc.

"This excellent book brings the agile practice of refactoring into the world of data. It provides pragmatic guidance on both the methodology to refactoring databases within your organization and the details of how to implement individual refactorings. Refactoring Databases *also articulates the importance of developers and DBAs working side by side. It is a must have reference for developers and DBAs alike."*
—Per Kroll, development manager, RUP, IBM; project lead, Eclipse Process Framework

"Scott and Pramod have done for database refactoring what Martin Fowler did for code refactoring. They've put together a coherent set of procedures you can use to improve the quality of your database. If you deal with databases, this book is for you."
—Ken Pugh, author, *Prefactoring*

"It's past time for data people to join the agile ranks, and Ambler and Sadalage are the right persons to lead them. This book should be read by data modelers and administrators, as well as software teams. We have lived in different worlds for too long, and this book will help to remove the barriers dividing us."
—Gary K. Evans, Agile Process evangelist, Evanetics, Inc.

"Evolutionary design and refactoring are already exciting, and with Refactoring Databases *this gets even better. In this book, the authors share with us the techniques and strategies to refactor at the database level. Using these refactorings, database schemas can safely be evolved even after a database has been deployed into production. With this book, database is within reach of any developer."*
—Sven Gorts

"Database refactoring is an important new topic and this book is a pioneering contribution to the community."
—Floyd Marinescu, creator of InfoQ.com and TheServerSide.com; author of *EJB Design Patterns*

About the Authors

Scott W. Ambler is a software process improvement (SPI) consultant living just north of Toronto. He is founder and practice leader of the Agile Modeling (AM) (www.agilemodeling.com), Agile Data (AD) (www.agiledata.org), Enterprise Unified Process (EUP) (www.enterpriseunifiedprocess.com), and Agile Unified Process (AUP) (www.ambysoft.com/unifiedprocess) methodologies. Scott is the (co-)author of several books, including *Agile Modeling* (John Wiley & Sons, 2002), *Agile Database Techniques* (John Wiley & Sons, 2003), *The Object Primer, Third Edition* (Cambridge University Press, 2004), *The Enterprise Unified Process* (Prentice Hall, 2005), and *The Elements of UML 2.0 Style* (Cambridge University Press, 2005). Scott is a contributing editor with *Software Development* magazine (www.sdmagazine.com) and has spoken and keynoted at a wide variety of international conferences, including Software Development, UML World, Object Expo, Java Expo, and Application Development. Scott graduated from the University of Toronto with a Master of Information Science. In his spare time Scott studies the Goju Ryu and Kobudo styles of karate.

Pramod J. Sadalage is a consultant for ThoughtWorks, an enterprise application development and integration company. He first pioneered the practices and processes of evolutionary database design and database refactoring in 1999 while working on a large J2EE application using the Extreme Programming (XP) methodology. Since then, Pramod has applied the practices and processes to many projects. Pramod writes and speaks about database administration on evolutionary projects, the adoption of evolutionary processes with regard to databases, and evolutionary practices' impact upon database administration, in order to make it easy for everyone to use evolutionary design in regards to databases. When he is not working, you can find him spending time with his wife and daughter and trying to improve his running.

Scott:
For Beverley, my lovely new bride.

Pramod:
To the women I love most, Rupali and our daughter, Arula.

Acknowledgments

We want to thank the following people for their input into the development of this book: Doug Barry, Gary Evans, Martin Fowler, Bernard Goodwin, Joshua Graham, Sven Gorts, David Hay, David Haertzen, Michelle Housely, Sriram Narayan, Paul Petralia, Sachin Rekhi, Andy Slocum, Brian Smith, Michael Thurston, Michael Vizdos, and Greg Warren.

In addition, Pramod wants to thank Irfan Shah, Narayan Raman, Anishek Agarwal, and my other teammates who constantly challenged my opinions and taught me a lot about software development. I also want to thank Martin for getting me to write, talk, and generally be active outside of ThoughtWorks; Kent Beck for his encouragement; my colleagues at ThoughtWorks who have helped me in numerous ways and make working fun; my parents Jinappa and Shobha who put a lot of effort in raising me; and Praveen, my brother, who since my childhood days has critiqued and improved the way I write.

Forewords

A decade ago *refactoring* was a word only known to a few people, mostly in the Smalltalk community. It's been wonderful to watch more and more people learn how to use refactoring to modify working code in a disciplined and effective manner. As a result many people now see code refactoring as an essential part of software development.

I live in the world of enterprise applications, and a big part of enterprise application development is working with databases. In my original book on refactoring, I picked out databases as a major problem area in refactoring because refactoring databases introduces a new set of problems. These problems are exacerbated by the sad division that's developed in the enterprise software world where database professionals and software developers are separated by a wall of mutual incomprehension and contempt.

One of the things I like about Scott and Pramod is that, in different ways, they have both worked hard to try and cross this division. Scott's writings on databases have been a consistent attempt to bridge the gap, and his work on object-relational mapping has been a great influence on my own writings on enterprise application architecture. Pramod may be less known, but his impact has been just as great on me. When he started work on a project with me at ThoughtWorks we were told that refactoring of databases was impossible. Pramod rejected that notion, taking some sketchy ideas and turning them into a disciplined program that kept the database schema in constant, but controlled, motion. This freed up the application developers to use evolutionary design in the code, too. Pramod has since taken these techniques to many of our clients, spreading them around our ThoughtWorks colleagues and, at least for us, forever banishing databases from the list of roadblocks to continual design.

This book assembles the lessons of two people who have lived in the no-mans land between applications and data, and presents a guide on how to use refactoring techniques for databases. If you're familiar with refactoring, you'll notice that the major change is that you have to manage continual migration of the data itself, not just change the program and data structures. This book tells you how to do that, backed by the project experience (and scars) that these two have accumulated.

Much though I'm delighted by the appearance of this book, I also hope it's only a first step. After my refactoring book appeared I was delighted to find sophisticated tools appear that automated many refactoring tasks. I hope the

same thing happens with databases, and we begin to see vendors offer tools that make continual migrations of schema and data easier for everyone. Before that happens, this book will help you build your own processes and tools to help; afterward this book will have lasting value as a foundation for using such tools successfully.

—Martin Fowler, series editor; chief scientist, ThoughtWorks

In the years since I first began my career in software development, many aspects of the industry and technology have changed dramatically. What hasn't changed, however, is the fundamental nature of software development. It has never been hard to create software—just get a computer and start churning out code. But it was hard to create good software, and exponentially harder to create great software. This situation hasn't changed today. Today it is easier to create larger and more complex software systems by cobbling together parts from a variety of sources, software development tools have advanced in bounds, and we know a lot more about what works and doesn't work for the process of creating software. Yet most software is still brittle and struggling to achieve acceptable quality levels. Perhaps this is because we are creating larger and more complex systems, or perhaps it is because there are fundamental gaps in the techniques still used. I believe that software development today remains as challenging as ever because of a combination of these two factors. Fortunately, from time to time new technologies and techniques appear that can help. Among these advances, a rare few are have the power to improve greatly our ability to realize the potential envisioned at the start of most projects. The techniques involved in refactoring, along with their associate Agile methodologies, were one of these rare advances. The work contained in this book extends this base in a very important direction.

Refactoring is a controlled technique for safely improving the design of code without changing its behavioral semantics. Anyone can take a chance at improving code, but refactoring brings a discipline of safely making changes (with tests) and leveraging the knowledge accumulated by the software development community (through refactorings). Since Fowler's seminal book on the subject, refactoring has been widely applied, and tools assisting with detection of refactoring candidates and application of refactorings to code have driven widespread adoption. At the data tier of applications, however, refactoring has proven much more difficult to apply. Part of this problem is no doubt cultural, as this book shows, but also there has not been a clear process and set of refactorings applicable to the data tier. This is really unfortunate, since poor design at the data level almost always translates into problems at the higher tiers, typically causing a chain of bad designs in a futile effort to stabilize the shaky foundation. Further, the inability to evolve the data tier, whether due to denial or

fear of change, hampers the ability of all that rests on it to deliver the best software possible. These problems are exactly what make this work so important: we now have a process and catalog for enabling iterative design improvements on in this vital area.

I am very excited to see the publication of this book, and hope that it drives the creation of tools to support the techniques it describes. The software industry is currently in an interesting stage, with the rise of open-source software and the collaborative vehicles it brings. Projects such as the Eclipse Data Tools Platform are natural collection areas for those interested in bringing database refactoring to life in tools. I hope the open-source community will work hard to realize this vision, because the potential payoff is great. Software development will move to the next level of maturity when database refactoring is as common and widely applied as general refactoring itself.

—John Graham, Eclipse Data Tools Platform, Project Management, committee chair, senior staff engineer, Sybase, Inc.

In many ways the data community has missed the entire agile software development revolution. While application developers have embraced refactoring, test-driven development, and other such techniques that encourage iteration as a productive and advantageous approach to software development, data professionals have largely ignored and even insulated themselves from these trends.

This became clear to me early in my career as an application developer at a large financial services institution. At that time I had a cubicle situated right between the development and database teams. What I quickly learned was that although they were only a few feet apart, the culture, practices, and processes of each group were significantly different. A customer request to the development team meant some refactoring, a code check-in, and aggressive acceptance testing. A similar request to the database team meant a formal change request processed through many levels of approval before even the modification of a schema could begin. The burden of the process constantly led to frustrations for both developers and customers but persisted because the database team knew no other way.

But they must learn another way if their businesses are to thrive in today's ever-evolving competitive landscape. The data community must somehow adopt the agile techniques of their developer counterparts.

Refactoring Databases is an invaluable resource that shows data professionals just how they can leap ahead and confidently, safely embrace change. Scott and Pramod show how the improvement in design that results from small, iterative refactorings allow the agile DBA to avoid the mistake of big upfront design and evolve the schema along with the application as they gradually gain a better understanding of customer requirements.

Make no mistake, refactoring databases is hard. Even a simple change like renaming a column cascades throughout a schema, to its objects, persistence frameworks, and application tier, making it seem to the DBA like a very inaccessible technique.

Refactoring Databases outlines a set of prescriptive practices that show the professional DBA exactly how to bring this agile method into the design and development of databases. Scott's and Pramod's attention to the minute details of what it takes to actually implement every database refactoring technique proves that it can be done and paves the way for its widespread adoption.

Thus, I propose a call to action for all data professionals. Read on, embrace change, and spread the word. Database refactoring is key to improving the data community's agility.

—Sachin Rekhi, program manager, Microsoft Corporation

In the world of system development, there are two distinct cultures: the world dominated by object-oriented (OO) developers who live and breathe Java and agile software development, and the relational database world populated by people who appreciate careful engineering and solid relational database design. These two groups speak different languages, attend different conferences, and rarely seem to be on speaking terms with each other. This schism is reflected within IT departments in many organizations. OO developers complain that DBAs are stodgy conservatives, unable to keep up with the rapid pace of change. Database professionals bemoan the idiocy of Java developers who do not have a clue what to do with a database.

Scott Ambler and Pramod Sadalage belong to that rare group of people who straddle both worlds. *Refactoring Databases: Evolutionary Database Design* is about database design written from the perspective of an OO architect. As a result, the book provides value to both OO developers and relational database professionals. It will help OO developers to apply agile code refactoring techniques to the database arena as well as give relational database professionals insight into how OO architects think.

This book includes numerous tips and techniques for improving the quality of database design. It explicitly focuses on how to handle real-world situations where the database already exists but is poorly designed, or when the initial database design failed to produce a good model.

The book succeeds on a number of different levels. First, it can be used as a tactical guide for developers in the trenches. It is also a thought-provoking treatise about how to merge OO and relational thinking. I wish more system architects echoed the sentiments of Ambler and Sadalage in recognizing that a database is more than just a place to put persistent copies of classes.

—Dr. Paul Dorsey, president, Dulcian, Inc.; president, New York Oracle Users Group; chairperson, J2EE SIG

前　言

渐近和敏捷软件开发方法，如极限编程（XP）、Scrum、Rational 统一过程（RUP）、敏捷统一过程（AUP）和特性驱动开发（FDD）等，在过去几年里已经以迅雷不及掩耳之势席卷了 IT 业界。顾名思义，渐近式方法是一种具有迭代性和增量性本质的方法，而敏捷式方法则是一种具有渐近性和高度合作性本质的方法。另外，重构、结对编程、测试驱动设计（TDD）和敏捷模型驱动开发（AMDD）等敏捷技术，还正在进军 IT 组织机构。这些方法和技术不是在象牙塔里构造的，而是经年累月地在基层实践中发展起来的，可谓备受实战的磨砺。简而言之，渐近式和敏捷式方法在实践中表现得极其优良。

在开创性著作《Refactoring》中，Martin Fowler 把重构描述为对源代码进行小的调整，在不改变其语义的情况下改善其设计。换言之，可以在不做破坏和添加的前提下提高软件作品的质量。在该书中，马丁论述到这一观点：既然有可能重构应用程序源代码，那么也有可能对数据库模式进行重构。不过，他也提到，因为数据库具有高度的耦合性，数据库重构存在相当的难度，因此他有意识地把数据库的重构排除在其著作之外。

自 1999 年《Refactoring》出版以来，我们已经找到了重构数据库模式的方法。起初，我们两人是各自单独工作的，只是在“软件开发（www.sdexpo.com）”等会议或是邮件列表上才会和对方不期而遇。后来我们探讨观点、观摩彼此的会议讲座和陈词，很快发现我们的观念和技术是互相重合、高度兼容的。于是我们合力写成此书，与大家分享我们通过重构来使数据库模式渐近的经验和技术。

本书中的全部例子都用 Java、Hibernate 和 Oracle 代码写成。几乎每一个数据库的重构描述中都包含对数据库模式本身进行修改的代码，而对于一些比较有趣的重构事例，我们则显示其 Java 应用程序代码所具有的效果。因为各种数据库创建方式不一，所以如果数据库产品之间存在重要的细微差别，我们也对备选实现方案进行讨论。

在有些例子中，我们讨论备选实现方法，是从运用 Oracle 特性（如 SET、UNUSED、RENAME TO 等命令）进行重构的角度出发的；我们很多代码实例都利用了 Oracle 的 COMMENT ON 特性。其他数据库产品具有其他能够简化数据库重构的特性，优秀的数据库管理员应该对如何利用这些特性有所了解。而以后数据库重构工具则会为我们做到这一点。另外，我们还尽量简化 Java 代码，以方便将其转化为 C#、C++，甚至 Visual Basic 代码。

使用渐近式数据库开发的理由

渐近式数据库开发是一个应运而生的概念。不是要在项目早期即预先设计好数据库模式，而是要在整个项目生命周期中建立数据库模式，以便反映股东们所提出的不断变化的要求。随着项目的进展，要求会随之变化，这是不以人的好恶为转移的。传统的开发方法拒绝承认这一基本事实，而是想方设法“管理变化”，实际上就是阻止变化。而现代开发技术的实践者则是乐于接受变化，依靠技术使自己的工作能够随要求的变化而变化。众多的程序员已经采纳了 TDD、重构、AMDD 等技术，还发展出了新的开发工具对其进行简化。我们做过这类工作，深知自己也需要支持渐近式数据库开发的技术和工具。

采用渐近式方法开发数据库的优势如下。

1. 减少浪费。渐近式，也就是实时方法，可以在需求变化时避免连续性技术所固有的不可避免的浪费。在具体要求、构架、设计上的任何前期投入，都可能会因为后来发现某一要求不再适应需求而付之东流。如果你有能力预先做这一工作，当然也有能力实时做同样的工作。

2. 避免大量的重复工作。在本书第 1 章中，读者将会看到，仍然需要预先做一些建模工作，通盘考虑一些问题，这样如果在项目以后进行中发现有这些的问题而需要就这些问题进行重写就无需查阅以往的细节了。

3. 始终保证系统顺利运行。使用渐近式方法，要定时生成能够顺利运行的软件；即使只是部署到验证环境中，也要确保能够顺利运行。如果你每一两个星期就生成一个能够顺利运行的系统新版本，就会大大降低项目的风险。

4. 始终保证数据库设计有尽可能高的质量。这恰恰是数据库重构的功效所在：每次都可以对模式设计有点滴改善。

5. 与开发者工作兼容。开发者是以渐近方式工作的，如果数据设计人员要在现代开发者团队中发挥力量，就有必要选择以渐近方式工作。

6. 减少整体工作量。以渐近方式工作，只要做当天该做的就可以，无需做多余的工作。

渐近式数据库开发也有一些劣势：

1. 存在文化障碍。许多数据设计人员喜欢遵循连续性方法进行软件开发，往往坚持要在编程之前创建某种形式详细的逻辑及物理数据模型并将该数据模型视作金科玉律，不可变更。现代方法论因为该方法效率过低，风险过高，已将其摈弃，这就把许多数据设计人员打入了冷宫。更糟糕的是，数据界有不少“思想领袖”是在 20 世纪 70 年代和 80 年代学有所成的，错过了 90 年代的对象化革命，因而也就没有获取渐近开发方面经验的机会。虽然世界改变了，他们却似乎并不随之而变。读者会从本书中了解到，对于数据设计人员来说，以渐近（即使不是敏捷）方式工作，不但可行，而且可取。

2. 学习曲线。学习这些新的技术需要花费时日，而将连续思维定式转变为渐近思维习惯，则更要花费时日。

3. 工具支持有待发展。1999 年，《Refactoring》出版之际，没有任何工具支持这一技术。仅仅数年之后，每一种集成开发环境（IDE）都具备了内建的代码重构特性。本书撰写之时，也没有任何数据库重构

工作存在，但笔者将手动实现重构需要的代码全部收录在书中。所幸的是，Eclipse 的数据工具项目（DTP）已在其项目计划书中提到需要在 Eclipse 中开发数据库重构功能，所以工具厂商跟进只是个时间问题。

敏捷开发简介

虽然本书并非专门讨论敏捷软件开发，但数据库重构实际上原本就是敏捷开发者的一项技术。一个过程如果符合“敏捷联盟”（www.agilealliance.org）提出的四个价值标准，即可视其为敏捷过程。这些价值标准讲的是胜过，而不是选择，同时还鼓励重视某一方面的同时，不偏废其他方面。换言之，应该重视右边的观念，但更要重视左边的观念。例如，过程和工具重要，个体和交互更重要。敏捷的四大标准如下：

1. 个体和交互胜过过程和工具。你要考虑的最重要的因素就是人和人如何合作，如果不把这一条理顺，工具和过程即使再好，也毫无用处。

2. 有效的软件胜过面面俱到的文档。软件开发最根本的目标就是要创建能满足股东需要的软件。文档还是有其地位的；文档如果撰写得当，可以描述一个系统创建的过程、原因及其使用方法。

3. 客户合作胜过合同谈判。只有客户才能告诉你他们需要什么。可他们却偏偏并不擅长此道 —— 他们往往不具备准确描述系统的能力，或者是开始说得正确，随后又改变想法。与客户签定合同固然重要，但合同无法代替有效的沟通。成功的 IT 专业人士会与客户紧密合作，努力发掘客户的需要，并在这一过程中向客户进行讲解。

4. 响应变化胜过遵循计划。随着工作在系统上的进展，股东们对于其需要的理解也会发生变化，业务环境改变，基础技术也要改变。变化是软件开发的现实，所以如果要让项目计划、总体方案得以顺利实施，这些计划和方案就必须反映不断变化的环境。

本书导读

本书的绝大部分篇幅（从第六章到第十一章）收录了详细描述每一次重构的参考材料。前五章则描述了渐近式数据库开发 —— 特别是数据库重构 —— 的基本概念和技术，读者应依次阅读这一部分。

- 第一章全面讲述了渐近式开发的基本情况及其支持技术。这一章概述了重构、数据库重构、数据库回归测试、通过 AMDD 方法进行渐近式数据建模、数据库资产的配置管理，以及独立开发者沙箱[1]的必要性。
- 第二章详细探讨了数据库重构背后的概念以及在实践中困难重重的原因。这一章还由浅入深地讲述了在“简单的”单个应用程序环境及在复杂的多应用程序环境中的数据库重构事例。

[1] “沙箱”是功能全备的环境，系统可在其内可创建、测试、运行。

• 第三章详细描述了在简单和复杂环境下重构数据库模式所需的步骤。对于单个应用程序数据库而言，由于对环境控制度较大，所以重构模式所需的工作量要小得多。而在多应用程序环境中，则要支持过渡期，在过渡期数据库要并行支持新旧两种模式，使应用程序开发团队能够将其代码更新、部署到产品中。

• 第四章描述部署数据库重构结果部署到产品中背后的过程。因为几个团队带来的变化必须合并到一起进行测试，也就证明了这一过程在多应用程序环境中尤其具有挑战性。

• 第五章总结了几年来我们在涉及到重构数据库模式时所取得的一些“好经验”。我们有几个想法想要尝试，但一直无法施行，在此一并提出。

关于封面

在“Martin Fowler 签名系列”丛书中，每一本的封面上都有桥梁的照片。这一传统反映了一个事实：Martin 的妻子是土木工程师，在本系列图书出版伊始正在从事桥梁、隧道等的水平投影工作。本书封面的桥梁为加拿大南安大略省伯灵顿湾横跨汉密尔顿河口的詹姆斯·N·艾伦高架公路桥。在这里共有三座桥梁，图片中有其中两座，另外一座东港路升降桥没有收入图片。这个桥梁体系之所以饱含深意，原因有二。最重要的一个原因是反映了增量性的交付方法。升降桥起初担负的是通过这一地区的交通。这一任务原本来由另一座桥来分担，但该桥在 1952 年受船只撞击后垮塌。1958 年，高架公路桥中的第一座，即前面路面上方有金属支架的部分，建成通车，以替代撞倒的大桥。因为高架公路桥是北到多伦多，南到尼亚加拉瀑布的主要通路，交通流量很快超过了通行能力。为满足通行要求，第二座桥，即不带金属支架的，于 1985 年建成通车。增量性的交付在土木工程和软件开发上都具有经济性和可取性。笔者采用这幅照片的第二个原因是斯科特是在安大略的伯灵顿长大的 —— 实际上他就出生在距离高架公路桥北部桥基不远的处的约瑟夫·布兰特医院。斯科特拍摄这张封面照片，用的是 Nikon D70S 相机。