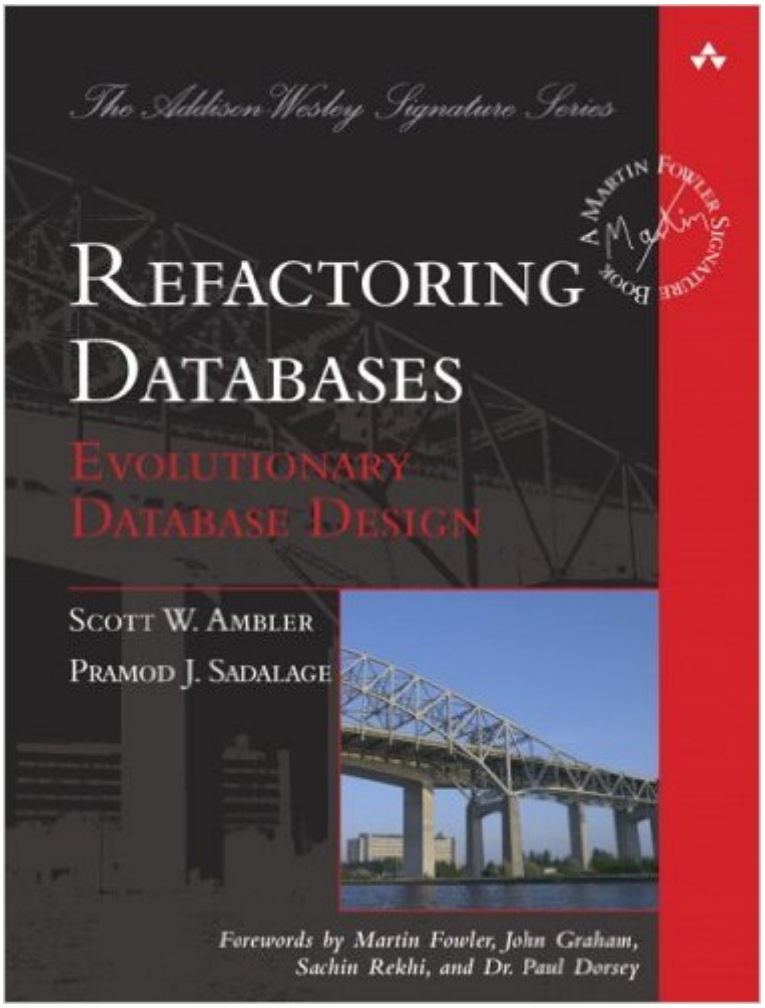
Refactoring Databases: Evolutionary Database Design



Synopsis

Refactoring has proven its value in a wide range of development projectsâ ”helping software professionals improve system designs, maintainability, extensibility, and performance. Now, for the first time, leading agile methodologist Scott Ambler and renowned consultant Pramodkumar Sadalage introduce powerful refactoring techniques specifically designed for database systems. Ambler and Sadalage demonstrate how small changes to table structures, data, stored procedures, and triggers can significantly enhance virtually any database designâ ”without changing semantics. Youâ ™ll learn how to evolve database schemas in step with source codeâ ”and become far more effective in projects relying on iterative, agile methodologies. This comprehensive guide and reference helps you overcome the practical obstacles to refactoring real-world databases by covering every fundamental concept underlying database refactoring. Using start-to-finish examples, the authors walk you through refactoring simple standalone database applications as well as sophisticated multi-application scenarios. Youâ ™ll master every task involved in refactoring database schemas, and discover best practices for deploying refactorings in even the most complex production environments. The second half of this book systematically covers five major categories of database refactorings. Youâ ™ll learn how to use refactoring to enhance database structure, data quality, and referential integrity; and how to refactor both architectures and methods. This book provides an extensive set of examples built with Oracle and Java and easily adaptable for other languages, such as C#, C++, or VB.NET, and other databases, such as DB2, SQL Server, MySQL, and Sybase. Using this bookâ ™s techniques and examples, you can reduce waste, rework, risk, and costâ ”and build database systems capable of evolving smoothly, far into the future.

Book Information

Hardcover: 384 pages

Publisher: Addison-Wesley Professional; 1 edition (March 13, 2006)

Language: English

ISBN-10: 0321293533

ISBN-13: 978-0321293534

Product Dimensions: 7.2 x 1.2 x 9.5 inches

Shipping Weight: 1.9 pounds

Average Customer Review: 4.3 out of 5 starsÂ Â See all reviewsÂ (27 customer reviews)

Best Sellers Rank: #500,274 in Books (See Top 100 in Books) #86 inÂ Books > Computers & Technology > Programming > Software Design, Testing & Engineering > Structured Design #189

inÂ Books > Textbooks > Computer Science > Object-Oriented Software Design #613 inÂ Books > Textbooks > Computer Science > Database Storage & Design

Customer Reviews

This is an excellent book that, in my opinion, serves two purposes. First, it is a compendium of well thought-out ways to evolve a database design. Each refactoring includes descriptions of why you might make this change, tradeoffs to consider before making it, how to update the schema, how to migrate the data, and how applications that access the data will need to change. Some of the refactorings are simple ones that even the most change-resistant DBAs will have used in the past ("Add index"). Most others (such as "Merge tables" or "Replace LOB with Table") are ones many conventional thinking DBAs avoid, even to the detriment of the applications their databases support.This brings me to the second purpose of this book. Many DBAs view their jobs as protectors of the data. While that is admirable, they sometimes forget that they are part of a software development team whose job is to provide value to the organization through the development of new (and enhancement of existing) applications. One of the best DBAs I ever worked with viewed himself as a "Data Valet." He said his job was to make sure the data was presented to applications when and where they wanted and to protect the doors from getting dinged while under his care. Through its first five chapters and then the refactorings that follow, this book will help DBAs expand their view of their role in the organization from one of simply protecting data to one of enhancing the value of data to the organization.This book is one that you'll keep on your reference shelf for many years to come. Highly recommended.

In software development a 'refactoring' is a change that improves code quality without changing functionality. Refactoring helps keep an application maintainable over its life-cycle as requirements evolve, and is particularly of interest to those adopting modern 'agile' methodologies. This book comprises five general chapters on database refactoring - about 70 pages - followed by a 200 page catalog of various refactorings. The refactorings are classified as 'structural', 'data quality', 'referential integrity', 'architectural' and 'methods'. An additional chapter catalogs 'transformations', more on which in a moment. Each catalog entry uses a template including 'Motivation', 'Tradeoffs', 'Schema Update Mechanics', 'Data-Migration Mechanics' and 'Access Program Update Mechanics'. The 'Mechanics' sections include example code fragments for Oracle, JDBC and Hibernate.Several of the structural refactorings are just simple database schema changes: rename/drop column/table/view. Adding is not really a refactoring so add column/table/view were cataloged as

’transformations’ - changes that do affect the application, a distinction that appears to me a little clumsy. Some structural refactorings are more interesting: merge/split columns/tables, move column, introduce/remove surrogate key, introduce calculated column, introduce associative table.The data quality refactorings include introduce/drop default values, null or check constraints, standardize codes, formats and data types, use consistent keys and lookup tables. Most of these are common best practices, seeing them cataloged as refactorings didn’t yield me any new insights. Only replacing type codes with flags was of special interest.Referential integrity refactorings include the obvious add/drop foreign keys with optional cascading delete, but also using triggers to create a change history and hard vs. soft deletes. Using before and after delete triggers to implement soft deletes is probably the best example in the book.Architectural refactorings include using CRUD methods (ie. stored procedures or functions to select/insert/update/delete records), query functions that return cursor refs, interchanging methods and views, implementing methods in stored procedures, using materialized views and using local mirror tables vs. remote ’official’ data sources. All these are common design techniques and the discussion of motivation and tradeoffs is particularly relevant.The final section on method refactorings is more abbreviated and covers typical code refactorings. These qualify for inclusion only because databases include stored procedures, but they have nothing to do with schema evolution.An important aspect of this book is that the catalog of refactorings is presented in the context of evolutionary database development described in the first five chapters: this approach emphasises an iterative approach, automated regression testing, configuration control of schema objects and easy availability of personalized application database environments for developers. Refactorings and transformations are intended to be applied one by one, and an automated regression test suite used to maintain confidence that a change does not introduce an application defect. Change control and a change tracking mechanism are essential to manage the application of schema changes to integration, QA and production environments.What do I like about this book? The catalog of refactorings is thorough (some might say pedantic) which makes it a good learning tool for new database developers and DBAs, and as a shared reference for communicating on larger projects and in larger organizations. Experienced DBAs working on smaller projects are less likely to find it useful.What don’t I like? Relatively little is provided about the tools required to make regular refactoring practical, the authors simply state that these are being worked on. utPLSQL is not mentioned at all. The discussion on tracking changes is thin (but check out the LiquiBase project on Sourceforge). No guidance is provided on how you might use Ant to build and maintain developer database environments. Little is covered on the tough topic of building and maintaining test data sets. A final pet peeve: no discussion of refactoring

across multiple schemas shared by an application suite.In summary this book sketches out some important ideas but much work remains to be done. The catalog takes a number of established techniques and best practices and places them in a new framework which at least provides value to some for now.

This book is a much needed exploration on the subject. It tries to categorize those operations that developers and DBAs do on a database, who, for various reasons, must address a specific problem, need. I only gave it three stars because it is somewhat insufficient. It also doesn't make much of a distinction between a database in development and one in production. In the latter case, it is really difficult to make changes when there is already a data structure in place with data, being updated constantly by users, and plans to migrate to a different data model while a system is in production is really not for the faint of heart. What I am really loooking for is a thick book of bad designs that, for various reasons (unclear or evolving requirements, political), a database model presents itself with a bunch of problems, real problems and not just theoretical, and the ways a DBA and developers came about after a big pow-wow on how to solve it.

Series) Soft Computing: Integrating Evolutionary, Neural, and Fuzzy Systems Information Processing with Evolutionary Algorithms: From Industrial Applications to Academic Speculations (Advanced Information and Knowledge Processing)