Refactoring Databases: A Strategy for Improving Performance and Data

Improving an organization's existing database code often seems an exercise in futility. But database professionals can improve existing assets by using the refactoring concept, which allows the slow and steady improvement of code through incremental changes.

The concept originated within the Smalltalk programming community as a means to "modify working code in a disciplined and effective manner." The goal is making the code easier to both understand and modify. Refactoring has been embraced by programmers and database professionals precisely because of its capacity for iterative, continual improvement.

"Refactoring is absolutely the single greatest, most important tool in your toolbox," stated Nina Zakharenko in her PyCon 2015 presentation, "Technical Debt: The Code Monster in Your Closet." "It's a systematic way to change the code without changing functionality, while improving design and readability." She advocates it as a clearer way of recoding without breaking the existing functionality of the project.

Why Refactor a Database?

In addition to continuous improvement, refactoring can be used to make safe fixes to legacy databases. Both the data and the database (including production databases) can be safely improved over time. Refactoring can help resolve various data quality problems.

Refactoring also enables users to move toward an evolutionary development style. They can begin working with newer methods for database design, such as agile development, extreme programming, or the agile or rational unified processes.

The Challenges of Refactoring

Refactoring databases presents technology professionals with a unique set of challenges. The program and data structures must change and the data is always being migrated. As explained by refactoring expert Martin Fowler on his website:

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….If you're familiar with refactoring you'll notice that the major change is that not just do you have to change program and data structures, you also have to manage continual migration of the data itself.

Scott W. Ambler, writing on his Agile Data blog, notes:

A database refactoring is a simple change to a database schema that improves its design while retaining both its behavioral and informational semantics.…An interesting thing to note is that a database refactoring is conceptually more difficult than a code refactoring; code refactorings only need to maintain behavioral semantics while database refactorings also must maintain informational semantics.

There are many different possible types of database refactoring. These include structural changes—drop table, merge tables, and rename or replace column—method refactoring, architectural refactorings, or data quality refactoring. In the structural change Split Column, a user replaces a single table column with two or more other columns. Or as another example, you may wish to change getPersons()to getPeople()in your code. This is a refactoring process called Rename Method. You cannot change a single instance of this operation. You must change it each and every time it appears. When the changes are complete, the database still works as it once did. The code is now refactored.

Many of these types of refactoring are explained in detail by Ambler in A Catalog of Database Refactorings. Ambler contends that, although refactoring may sound like a simple process, "and sometimes it is," in reality "database refactoring is incredibly difficult in practice when cultural issues are taken into account."

Some professionals may be tempted to add functionality when they are refactoring. This impulse needs to be ignored. The purpose of refactoring is to improve the existing code, not to add code.

Refactoring should not be done in a vacuum. Test as you go is mandatory when refactoring, says Zakharenko. Staff can, depending on the size of the group, rotate in and out of the project or else allocate a small percentage of their workday to refactoring as part of a larger goal of eliminating technical debt.

Improving Database Performance

Database refactoring is but one method for improving database performance. For some other recommendations, be sure to download a copy of the whitepaper The 5S Approach to Improving Database Performance, and read our blog for additional helpful information for database administrators.

If you would like assistance implementing and integrating any of these ideas into your operations, please contact Datavail to discuss a custom solution designed for your enterprise.

About the Author: Eric Russo
Vice President and Practice Leader of SQL Server and MySQL Client Services, Datavail

Experienced in the management of large DBA teams involving offshore staff, Eric specializes in ITIL certification, management of local and remote staff, project coordination, database administration, change management, server OS administration, programming and scripting. Eric's years of experience as a DBA programmer, ITIL implementer, and change control manager allow him to run high-performance DBA teams providing exceptional customer support. In his off time Eric enjoys spending time with his family, snowboarding, karate, running and computer games.