

Database Design and Management: “Getting It Right the First Time”

MAY 2007

Michelle A. Poolet

MOUNT VERNON DATA SYSTEMS LLC

Table of Contents

Executive Summary

SECTION 1	2
The Challenge: The “Right” Design	

SECTION 2	2
The Opportunity: Identifying Problems Where They Exist	

Bad Record Design Lives Forever

Poor Application Performance Is Caused By Flaws in Design Implementation

Restricted Design Restricts Business Operations

SECTION 3	6
The Benefits of Building It Right	

SECTION 4	7
Conclusions	

SECTION 5	7
About the Author	

ABOUT CA	Back Cover
----------	-------------------

Executive Summary

Challenge

In the world of databases, “getting it right” is a phrase filled with ambiguity, for who can say what “right” truly is? Frankly, it’s a lot easier to spot “bad” or “not right” database schemas than it is to confirm that a database schema is “right”. While schema design is the foundation of a well-performing database, you are missing opportunities if schema design doesn’t also take database performance, administration and backup/recovery into consideration.

Opportunity

If you haven’t done your homework or researched the corporate mission and business requirements — if you don’t understand the real-world database management processes that tie database design to performance tuning, administration and backup/recovery, then your database project is headed down the path to failure. Integrating the best schema design that you know how to do, along with best practices in, for example, performance management and properly tuned and deployed SQL code will go a very long way toward ensuring your database environment is functioning at optimal levels. It also ensures that your data architects, DBAs, database performance team and data center managers and even business intelligence and applications teams are aligned and working together efficiently.

Benefits

Database modeling and design is at the core of a healthy database management and performance foundation. Getting it right the first time will enhance the bottom line in a variety of ways. It will:

- Eliminate haphazard approaches to maintaining data.
- Minimize maintenance and upkeep of the database schema.
- Optimize data retrieval for graphs and reports.
- Ensure data accuracy and reliability.
- Support corporate goals and evolving business initiatives, allowing the company to respond more quickly and precisely to new business opportunities and competitive threats.

SECTION 1

The Challenge: The “Right” Design

In the world of database design, “getting it right” is a phrase filled with ambiguity, for who can say what “right” truly is? “Right,” when a legacy system was first created, might not be adequate in today’s rapidly-moving, Internet-based world. To complicate matters, there might not be a way to make “wrong” work in such a way that it’s effectively “right” for today’s situation.

Frankly, it’s a lot easier to spot “bad” or “wrong” database schemas than it is to confirm that a database is “right”. “Right” means that the database is working fine, productivity is high, performance is optimal, the corporate mission is being supported, the organization has the flexibility and underlying foundation, as expressed in the database schema, to move in directions that it needs to move in.

“Wrong” can mean everything from operational constraints that are inhibiting corporate growth and development to partial transactions being applied to the database. When I’m called in to investigate a “wrong” condition, I try to get a holistic view of the situation. While schema design is the foundation of a well-performing database, I would be missing opportunities if I didn’t take database performance, administration and backup/recovery and disaster recovery schemes into consideration.

Each project and situation is going to be a little different from anything that you’ve experienced before, so when starting on a new project, do your homework. Research the corporate goals, strategic business initiatives and underlying business requirements. Understand how database management processes tie the database design to performance tuning and administration and even backup and recovery. Integrate the best schema design that you can after researching current best practices in performance management; ensure that the deployed SQL code is properly tuned and have a plan to keep it maintained. This will go a very long way toward insuring that the database environment you’re working with is functioning at an optimal level. In the process, try to ensure that the data architects, the DBAs and the data center managers are aligned and working together efficiently.

SECTION 2

The Opportunity: Identifying Problems Where They Exist

Each of the following situations is an example of a condition that could have been avoided if the individuals involved with designing and implementing the database had done it right in the first place.

Bad Record Design Lives Forever

Bad record layouts have been with us since the dawn of computing. Records that contain too many fields about too many different aspects of the data, records that when fully populated are longer than the database block size, records that contain repeating groups and that try to combine both detail and summary data, such as “Store Identifier,” “Monday Sales,” “Tuesday Sales,” “Wednesday Sales,” etc. – all these conditions will quickly cause database performance to wither and data integrity to fall by the wayside.

A few years after I opened my business as a database consultant, and shortly after I had co-authored a book on the new, upcoming version of a popular database management system, a client contacted me to consult on a database performance tuning situation. The database under investigation was a production system. It contained an inventory of all the products and services that had been and were being sold, everyone who had purchased any of these products and/or services, a history of each installation, the configuration of each sale, etc. Performance on this database, which was the only major database on the server, was definitely sub-standard, even after upgrading the server to a high-end, multiprocessor box. Management wanted answers to two questions. First, would the database perform better after upgrading it to the new version of the database management system? Second, should they invest in an expensive hardware upgrade to address the performance issues?

Interestingly, they didn't want to hear that the right course of action was to get the data properly normalized, skip the hardware upgrade and take a slow or measured approach to moving to the new version of the database management system. Redesigning the production database schema was not in their option list, but it was the right answer.

With the exception of a few lookup tables, such as State and Customer Type, everything — all data pertaining to a customer, including everything that customer had ever purchased — was packed into a single record. This central table of their production system was hitting against the block size limitation of the database management system. They were looking at upgrading to the new version because the new version had a much longer block size. They were getting ready to add more columns to this out-of-control table, but they had run out of room because of the block size limitation.

Obviously, someone didn't understand the concept of data normalization, or even of vertical partitioning and one-to-one relationships. Using a relational database management system as a glorified spreadsheet is not the most efficient way to manage your data.

Clearly, in a situation like this, no amount of powerful hardware, increase in the block size or any number of additional indexes was going to help these folks out of their performance quandary. An increased block size, added to this company's propensity to treat their relational database management system as though it were a flat-file server, would only exacerbate the poor performance they were experiencing. Physical I/O is arguably the most expensive operation in a computer, and there's nothing that can overcome reading and writing a single record with each physical or logical I/O. Had these folks been monitoring their system, they would have seen exceedingly high activity to and from the hard disks and thrashing — high levels of reads and writes to the page file, even under light to moderate transaction load. These conditions are symptoms of problems with the physical I/O and the logical I/O, respectively.

You need to attack problems like this at the source, even if that means redesigning the production database schema. Using best practices of data normalization, divide the data into sets of related tables based on meaning and functionality. Enforce inter-table relationships using DRI, declarative referential integrity, and index accordingly. If necessary, create views that simulate the old record layout to support the existing applications that request services from the database while you're rewriting these applications to call something other than "select *."

If your initial record and table design is lacking, then you need to take some serious steps to remedy the situation. Bad record design is a prime example of not getting it right at the foundation level. The remedy, as with any faulty foundation, is to re-design and re-build.

Poor Application Performance Is Caused By Flaws in Design Implementation

Sometimes, even with a properly-normalized database, conditions are such that you will still encounter performance problems. One such case was a company that was having a terrible time trying to increase productivity without increasing headcount.

One key to increasing productivity per worker is to document and then automate processes. You can do process analysis; you can identify those manual processes that can be automated. However, if you're working with an inflexible legacy system, one for which there is no source code, and if you're trying to integrate new automated processes into such a system, you may be running up against a brick wall. In the case that I'm describing, which represents a legacy system that had been created years before, there was no source code or documentation. Any previous attempts to modify the schema had caused the custom application that was core to the business to fail.

Their immediate need was the invoicing process. There was so much data stored in the invoicing subsystem of the database that it took 36 hours to run the invoicing routine which ran every ten days. How do you shorten the amount of time that it takes to run invoicing when you can't touch either the database schema or the invoicing application?

The answer lay in the data content. This database schema was fairly well designed. While there were a few places in the schema that might have benefited from some design tweaking, in general the schema was solid. The flaw in this design was its implementation; there was no built-in method to rid the database of old data, and the weight of old data was slowing down processing. No amount of indexing seemed to help. There was just too much old data stored in the database.

Even the best-designed database will stagger under the load of too much data if the queries that call the data are not properly constructed or don't use index structures wisely. In this case, the invoicing application was reading through the entire set of invoicing data before selecting those accounts that required further processing. As a result, every time invoicing was run, many millions of unneeded records were being read. Any attempt to change the invoicing system was met with defeat — the company that crafted the custom application had left no source code behind, and had hard-wired application access to the database.

In situations like this, when you cannot modify legacy code or even the legacy schema, you need to approach the solution from a very low level. Use a technique like horizontal data partitioning to physically separate current data from aged data. Your biggest challenges will be to determine what constitutes "aged" data, and then to develop algorithms that successfully identify the aged data, so that you can move it into its own table or onto its own archive database. For this task you'll need the input of the business users in the organization. They understand how to identify "aged" data, and what constitutes a current data set.

Once you have separated the aged data from the current data, you'll want to create a set of scheduled jobs, using these partitioning algorithms, so that periodically, as data becomes aged, you can move it to the aged data set. This will keep the current invoice data set lean and mean, with the end result that the invoicing routine will run swiftly. In this case, the time required to run the invoicing routine was reduced to just 3 hours from 36 hours.

Restricted Design Restricts Business Operations

In order to get it right the first time, you have to understand the business requirements. Such was not the case on a project that involved a complete redesign of a production database that supported an Internet retail organization. In this case, the company had four Internet retail websites with a plan to increase this number and possibly evolve the business to a franchise-based operation. The four retail sites all relied on a shared database and application code base. Due to how the database schema was constructed, early database design decisions were hampering organizational growth.

The database schema, designed by a third party, had served the company well to this point but failed to take into consideration the going-forward business plan. Not being technical people, the management group never recognized these design shortcomings which led to hard-coded limitations on the number of stores and the type of services that this company could offer. Without fundamentally restructuring the underlying database and rewriting the code base, there was no way to expand the features and functionality of this database system to support business growth initiatives.

For this kind of situation, you should create a checklist for yourself of things to do and things to consider. First on the checklist is to identify the business requirements, both current and future. You'll want to incorporate these into the new database schema. Enlist the help of end users to ensure that your database design will be able to support their needs and expectations. In this case, since this was a retail database, we studied the behavior of real-life shoppers in a brick-and-mortar store, and translated that into the database schema, adding features that were unique to an online shopping experience.

Second on the checklist is to look at the physical implementation, specifically the way the database files have been placed on disk. Using a tool to provide proactive monitoring and management of the database is necessary. In this case, I used an operational profiling tool to test performance and query response both before and after the physical restructuring. My findings before restructuring were not encouraging. The original database had all objects stored in a single file group. This database included all types of data — tabular (numbers, short character strings, dates/times) and text/image (long product descriptions and LOTS of images), and that single file group was located on one physical/logical disk. This configuration adversely impacted performance and degraded the user experience for the Internet shoppers. When restructuring a database like this — one whose content is in great part composed of binary images and extraordinarily long strings of text — it's a best practice to separate the text and image data from the tabular data. Store the text and image data in one set of tables, and then use relationships to link each image or long text string to its associated tabular record(s). You'll also want to separate the system (catalog) tables from the image and tabular data. You will want to place each one of these three types of tables in its own file group or table space. Then, you will want to locate each file group on its own spindle. Using these simple principles of physical design, you'll significantly enhance database performance.

Third on the checklist is indexing for performance. Whenever you have a one-to-many relationship between two tables, you'll have a foreign key reference in the table on the many side of the relationship. In most database systems, the primary key is automatically indexed; that's not always the case with the foreign key. To ensure best performance on joins, make sure that every foreign key in the database is indexed. The next candidate for indexing is any column that will be used for sorting — any column that will consistently be used in an “order by” clause of a SQL query. You'll also want to index columns that will be used for restricting the returned data set, such as those that consistently appear in “where” clauses.

It's no easy task to determine, document and implement business requirements. It takes a lot of cross-functional collaboration, great attention to detail, and a validation of facts and assumptions in order to do it right. If you want to get it right the first time, define and adhere to a disciplined approach to gather and share the right information. Attention to detail while maintaining the big picture in your head is a must for good database design and getting it “right”.

SECTION 3

The Benefits of Building It Right

The benefits of building it right are many. Just as you would never consider building a house without thoughtful design and planning, so should you approach database design. Picture the building architect who is designing to your specifications; you'll be asked questions like “How will you use this?” “What are your future plans?” “What do you want this to look like?” “How do you want it to work?” Asking these same questions when you're building a database will give you the same usability and “livability” that you'll get out of a well-designed house.

- **The benefits of a good foundation:** once the blueprint is drawn and the plans are finished, it's time to start building the databases. You would never consider building a house unless it was underlain by a solid foundation, so the first thing you do is pour the foundation. The foundation to your applications and systems is the database. To ensure that your database will be a good foundation for the superstructure of applications and programs that rely on it, build it right in the beginning.
- **The benefits of good record design:** a well-normalized database makes it easier to maintain the data. In a well-normalized database, when you have to make a data modification you can make it once, make it right and make it fast. You'll experience better performance on data retrieval because, in a well-normalized database, the relatively short record length allows for many records to be read from or written to the hard disk in a single physical I/O operation. Virtual (in-memory) operations are enhanced, as well, because of the shorter record length; more records fit on a single page, therefore more records can be moved or processed in a single pass. Resource-intensive tasks such as disk I/O for the data in storage or for the page file will be greatly reduced. To optimize database operations and response time, practice good record design from the beginning.

- **The benefits of implementing the design correctly:** when you're implementing a design, any flaw can have negative long-term repercussions. Back to our house-building analogy, a flaw in the electrical wiring when the house is being built could result in an electrical short which is expensive to address later or could lead to a disastrous event. You can have a perfectly-formed design, but if you implement it incorrectly, months or years down the road you'll find yourself applying remedies to symptoms or focusing on more superficial problems — not the root cause of problems — as you try to compensate for the implementation flaw that you introduced in the beginning. To avoid haphazard code and techniques, implement it right in the beginning.
- **The benefits of determining, documenting and implementing business requirements:** If your database doesn't support the business requirements, it will be — at best — a disappointment from the start, and at worst, could cost you your job. To avoid having to deal with a database that just doesn't support the requirements, make sure that you understand what those requirements are, and not just today's requirements. Ask about, document, and confirm the business requirements for tomorrow and next year and the year after, and the year after that. Only by understanding where the company plans to be in the future can you get the design right for today's operations.

SECTION 4

Conclusion

Building a database the right way means the database is working fine, productivity is high, performance is optimal, corporate initiatives are well supported and the organization has the flexibility and underlying foundation, as expressed in the database schema, to move in directions that it needs to move in.

"Getting it right" means building on a solid foundation, and working from a good design. The real world situations described in this paper all emanate from poor design and/or lack of a proper foundation. Make sure that the next project you start has both a solid design and a proper foundation. Get it right in the first place.

SECTION 6

About the Author



Michelle Poollet
Mount Vernon Data Systems, LLC

Michelle Poollet, MCIS, University of Denver, Microsoft MCP SQL Server, Neverfail NCIE, is a co-founder and President of Mount Vernon Data Systems, a database consulting company that specializes in database systems, data modeling, data protection, information technology high availability, disaster recovery and business continuity strategies, courseware development, education and training. She has refactored and re-engineered existing production databases, designed, developed and implemented custom database solutions, authored books on Microsoft Access and SQL Server, and has served as technical and contributing editor for Prentice-Hall Publishers, MacMillan Computer Books, WindowsNT Magazine and SQL Server Magazine. She is currently a regular columnist for the SQL Server Magazine, is Senior Adjunct Faculty at the University of Denver, is a Certified Professional for Microsoft's premier database management system, SQL Server, and is a Certified Implementation Engineer for the Neverfail Group's high-availability disaster recovery business continuity solution.

CA, one of the world's largest information technology (IT) management software companies, unifies and simplifies complex IT management across the enterprise for greater business results. With our Enterprise IT Management vision, solutions and expertise, we help customers effectively govern, manage and secure IT.

WP05DBMDDE MP316480507