

The Necessity of Schema Review

Article to be published soon
July 2005

By [Joe Celko](#)

The Programming World

I am going to start with a short history lesson, so be patient with me. In the early days of systems development, programmers were pretty much cowboys. They got a specification, went off somewhere and returned with the code. They did a lot of their own testing and never bothered with much documentation.

In fact, there was smugness in code that other people could not understand. The saying was "If this program was hard to write, then it ought to be hard to read!" But this wasn't really a joke so much as an attitude. Testing and the correctness of code was largely down by the "Wright Brother's School of Engineering" - put it all together, push it off a cliff and see if it flies. If it failed, pick up the pieces and repeat the process until something worked well enough to get by.

By the 1970's this situation was called the Software Crisis in the trade press. The average application development shop had a backlog of over 18 months according to COMPUTERWORLD and it was increasing. Furthermore, that sloppy "cowboy code" was a nightmare to maintain. Over the lifetime of a system, maintenance will account for 80% or more of the total cost.

It was very clear that something had to change. We had to find a way to write code that could be maintained by other people and in which we had some degree of demonstrable correctness.

The solutions lead to Structured Programming, Design and Analysis, to Software Engineering and Software Development Tools. The software cowboys were replaced by software engineers with a repeatable process for producing, testing and maintaining code.

The Database World

The late 1970's saw a fundamental change in data processing. We moved from file systems to databases. Simply put, we found out that having redundant data in assorted file systems was dangerous and expensive. You were never sure where anything was or which copy of it was correct. Dr. Codd's relational model gave data a mathematical foundation, just as software engineering had done for programming.

But there is a serious difference between procedures and data. An error in a procedure is limited to the module in which it appears. An error in a database schema affects every program in the system. Data is in a database to be shared. If the data is good, then everything is fine; but if the data is bad, then everyone is drinking from a poisoned well.

The next obvious question is why do we have bad schemas? A big reason is that it is very easy to write a bad schema. A good schema requires that the design cover all the business rules, be internally consistent, be externally consistent (i.e. agree with the real world) and be maintainable. This is a lot of work. The good news is that once it is done in the database, it does not have to be repeated in hundreds of application programs that run against the schema.

A program can let you know that it has found errors by failing. The languages include exception handling and programmers have learned to add code to report unexpected data values.

But a bad schema can return data, even if the data is wrong. Very often, you cannot tell if the data is right just by looking at the result set. As a simple example, suppose that you want to see all of the male employees and use (sex = "m") as the search criteria.

The query does not tell you that the sex column was declared without any constraints and also holds illegal values that might or might not actually be males in the real world. The query does not tell you that some of your males are taking maternity leave because there was no CHECK constraint to prevent pregnant males, that some of the rows are orphans because there was no Defined Referential Integrity action to remove terminated employees.

You will not discover any of this until you do a full data audit. At that point, you have been using the bad data and now have to clean it up, correct previous reports and whatever else it takes to keep all the systems running.

Why didn't the designer see the flaws and prevent them? The reason is that it is not possible to inspect your own work. It is not that people are evil, lazy or stupid. Once a problem reaches a certain complexity and size, the best that you can consistently do inspecting your own work is to find 80-85% of the errors.

A Culture of Inspection

One of the effects of the Structured Programming Revolution was a culture of inspection and review. It was not enough to follow the rules for writing code. The programmer had to present his code to his peers in a code review. There were tools for formatting the code, checking for proper conventions and so forth. What the other team members did was provide human feedback and judgment to the programmer. The extra eyes also overcame the 80% rule for blind spots in self-inspections.

All the team needed was a simple list of things to look for during the code reviews and inspections. This did more for the quality of the code than any other tools. Capers Jones, one of the early pioneers in Software Engineering research found that "Formal design and code inspections average about 65% in defect removal efficiency. Most forms of testing are less than 30% efficient." (SOFTWARE QUALITY: ANALYSIS AND GUIDELINES FOR SUCCESS by Capers Jones, 1997, ISBN:1850328676). Likewise, the IEEE (Institute of Electrical and Electronics Engineers) found that "Peer reviews of software will catch 60% of defects." in their research.

But you had to build a culture in which the team members did not clash. This led to tools for software metrics, so that some part of the inspection and review process could be done by a machine instead of a person.

That culture of inspection does not really exist in SQL yet for several reasons. Given a typical shop, one person will design a schema by himself with a data modeling tool of some kind. No reviews and no tools to support the rest of the design process. This is back to the old "cowboy coder" days again. So, who is going to do the schema code review with him?

An applications programmer is not a good choice for a reviewer. A SQL schema is more like a specification than a program. Traditionally, analysis and coding were separate job functions and skills. The programmer thinks in terms of "how to do this one task" while a database designer thinks in terms of a logical data model that will hold for all possible tasks, present and future.

It requires time to be a "big picture" person and work at an abstract level one minute then move down to a physical implementation level the next. It takes six years to become a Union Journeyman Carpenter in New York State, not even a Master Carpenter. So why is it a surprise that you cannot be a database architect after reading a book or having a training course?

Another problem is that an SQL schema can be huge and complex. It is simply not possible for a human being to keep track of so many tables and relationships. What happens is that we tend to trust the data modeling tool to get everything right for us.

This is not possible. The best a data modeling tool can do is provide us with an internally consistent schema and perhaps even generate some simple SQL statements. Even then, it is hard for most tools to do better than Third Normal Form (3NF). And they certainly cannot translate a business rule from English into an SQL constraint.

The Data Quality Issue

Having a bad schema is like having a leaky pipe. You do not just keep mopping the floor when the leak causes problems; you must fix the leak or this is going to happen over and over. The "solution" is all too often a patch. The bad data is discovered, and a few UPDATE scripts are written to change the data in place. And perhaps a memo asking programmers to check the data in the applications is sent out.

Trying to keep data correct in applications simply cannot work over the long run. Programmers trying to correct hundreds or thousands of lines of code will eventually make an error. John thinks the specification means (age > 18) and Mary read it is a (age >= 18). Each of their programs passes a code review and goes into production. How would you tell which program provided the data that was inserted, updated, deleted or queried in the schema that all the programs share?

Assume you hire only perfect programmers who write code that never lets bad data in the database, from now until the end of time, it does not work. One person using a query tool or one developer testing on live data can circumvent all the front end protections these mythical perfect programmers put in place.

Constraints and business rules must be at the schema level to work. Yet we are faced with a lack of skilled people. Even if we had the qualified personnel, the increasing complexity of systems will exceed human processing capacity.

A Solution: Automating Schema Inspections

If we cannot easily find data architects and the complexity of a schema is beyond what one or two people can manage at the level of detail needed, then we have to look for automated solutions.

The obvious mistakes are best found by a machine and not a human. Anyone who has tried to live without a spelling checker on his word processor knows this all too well. These include things like:

- Redundant indexes
- CHECK() constraints that are redundant or inconsistent
- Inconsistent DEFAULT values
- Missing Relationships
- Other things that can be fixed on the spot.

Normalization mistakes are best found by a machine and not a human.

Getting an ER diagram from a tool does not mean something is properly normalized.

But code reviews had another feature beyond mere mechanical checking. While you cannot replace human knowledge, you can automate a lot of it.

Fellow programmers would look for violations of company coding standards. For example; a machine has to be told that we say "client_nbr" and not "client_num", "client_no", and so forth.

Certain patterns in a schema are a sign, but not proof, of design errors. A human will see these patterns and try to explain the possible problem to his fellow programmer. This assumes that the reviewer knows all the patterns and can find them in the schema. Human beings can teach each other by explaining, making suggestions and providing examples of code.

Validator

The only tool for automating schema review on the market is CA's Validator (<http://www3.ca.com/Solutions/Collateral.asp?CID=33674>). It provides automated support for the features of a human schema code review list in the previous section, along with customizations for a particular installation.

Suggestions for a complete re-write of code, additional entity and relationship tables will always have to come from human programmers who understand the problem domain. This tool will do 90% or better of the work that you should be doing now, but probably are not doing. I have good reasons to believe that you are not doing these reviews. The Cutter Consortium (www.cutter.com) did a survey and found that 32% of organizations responding said that they release software with too many defects, that 38% of them believe they lack an adequate software quality assurance program and yet a full 27% of them do not conduct any formal quality reviews. Remember that this survey is self-reported, voluntary information. If there are errors in it, they will probably be skewed to make the organization look better than it is.

Where do we do this tool and what happens when we do use it? When schema reviews were manual, they required a lot of people and/or a lot of time. Both time and people are in short supply in most enterprises, so such a review was a major process that had to be scheduled as part of the overall project.

Many organizations would hire outside consultants to help or to perform the reviews. You can expect to pay \$1000 to \$2000 per day plus expenses for a qualified consultant. This is not a guess as to price; the author of this paper has made part of his living doing such reviews.

Culture Changes

When a tool replaces a manual operation, it causes the culture to change. It is more than just doing the same thing faster or cheaper. Consider the automobile. Yes, a car gets you places faster than a horse. But it also determines the architecture of your house and the places you shop. Automobiles changed the way people think about space - it is no longer measured in miles but in driving time. Your whole view of the world has been changed by the automobile.

Development

The most obvious effect of Validator is reduction in the cost of database development. If you can get a review faster and without outside consultants, the project can be done much cheaper. Developers spend about 80% of

development costs on identifying and correcting defects according to NIST (the National Institute of Standards and Technology).

But the real benefit is that schemas will be better with the new tool. In the 1970's, studies for the Department of Defense showed that an error in the design phase of a project cost an order of magnitude more to correct at each step in the classic "Waterfall Model" that it passed thru. Thus, if the error was not detected until deployment, the cost to fix it could be more than the project was worth. This was one reason that so many projects failed.

A good schema benefits the entire system. In my article "It's the code, Stupid!" (DMReview, June 2005; www.dmreview.com) I state and give examples to prove that the main performance problem in the huge majority of database applications is bad SQL code. It is not lack of hardware. It is not network traffic. It is not slow front ends. All of those problems have been solved by technologies that made them commodities instead of major purchases and custom-built items. Going further, I say that most bad SQL code is the result of bad schema design. A well-designed database is fairly easy to write code against. But in a poorly designed schema, you need excessive code to clean up the data on the fly.

And even then, you are not sure if the data is correct. Data Quality is a major issue for any organization for obvious reasons. But today, Data Quality is a major concern because it is required by law and you can go to jail for violations. We have the Sarbanes-Oxley Act (SOX) in the United States for corporations, BASEL II in the Banking Industry, HIPAA in Healthcare, SEC Rule 17a-4, NASD 3010, NASD 3110 in the Securities Industry – the list is getting longer, not shorter. Being able to show auditors that your schemas have been properly reviewed is one more mark on the good side of their check list.

With this tool, a single database developer can review his own work against consistent standards as he codes, whenever he needs to. Errors are detected during development and never get into production systems. It is an old systems engineering principle; the sooner feedback is given, the faster and easier corrections are to make.

Immediate feedback and training also improves the performance of humans. For example, word processors that display spelling errors during the writing process improve the spelling of the users. Another positive cultural change!

Maintenance

Developing a schema is hard, but maintaining a schema is worse. Anyone who has worked with a large database project knows how hard it is to make any changes to the schema.

You probably do not have the advantage of a dedicated staff for each schema in your company. When you have to maintain a schema that is new to you, you have to learn what the intent was before you can even read it. The usual approach has been to get an ER diagram of the schema, stick it on a wall and try to trace the effects of your change by eye with colored markers.

The first problem is that simply following the lines between the boxes on an ER diagram does not tell how far the changes will cascade. Let me give a simple example: we wish to add one more decimal place of precision to a measurement in Table A. Any other table that references this column directly will also need to be changed, then any table that references those tables also need to be changed. This can be a bit of work if the same data element is used directly or in calculations in a lot of places.

The second problem is that you do not see the constraints, defaults, assertions and triggers that use the measurement without directly referencing Table A on an ER diagram. Your new longer, more accurate measurement might still be using a default value meant for the old precision of the data element. It might be checked against an old formula that uses other columns in Table A.

The measurement might appear in another table that has no direct relationship to Table A. The classic example of this is changing the precision of temporal values. A query about events in separate unrelated tables does not show up in a diagram (Example: Did we hire any new mail room clerks on the day we shipped the Widget order to Acme Corporation?)

The third problem is that if the schema had problems before your changes, you are not looking for them. You are there to fix one thing and you have the mindset to look at the big picture.

A schema is like an ecological system; you cannot blindly change just one thing because all things are related. Validator lets you do a "what-if" scenario so you can see how far the effects of a schema change will cascade.

Data Migration and Warehousing

Migration of data from one platform to another is a fact of life. It usually occurs in two forms. The first is simple migration because the data has outgrown the current database or because the enterprise has decided to go with another product or release of the current product. There is no attempt to do any data transformations, just preserve the existing data in the new system.

However, unintended data transformations can happen during the move. If the source data was in error, you want to be sure to correct it during the move. This means you need a review of both the source and destination databases with a stress on being sure that they match.

Data warehousing is the second form of migration. Here there are many data transformations from many sources to get the raw data into aggregations, to remove identifiers that are protected under privacy laws and a host of other considerations.

Data warehouses have a high failure rates and high failure costs. Companies build them because when they do succeed, the payoff can be huge. Wal-Mart has one of the largest Data Warehouses on Earth and it drives their business.

Building a data warehouse is often the first time that a company has ever looked at all of its data in one place and tried to make it compatible. The shock can be overwhelming. Suddenly incompatibilities are exposed. The same data elements have completely different names from database to database. The same name refers to different data elements, measurements and definitions.

A major airline found that they had 16 different definitions of how a "passenger" related to a "ticket" and to a "seat" in various databases when they began their warehouse project. For example, a passenger was a person in a seat including stand-by airline personnel, a person with id who bought a ticket, a ticket purchased in advance by a company without a name, a baby in arms without a ticket or a seat, a filled seat on a single flight, a person with tickets for multiple flight legs, a special meal request without regard to seat number, etc. This situation is not uncommon.

The consistency of each data sources is its own problem. Having a data source that is internally consistent is not enough, it is the just ante required to play the game. The consistency of data among the sources as a whole is a data warehouse problem.

If maintaining and correcting one database was hard, then maintaining multiple data sources increases the burden beyond human limits. Without some kind of software tools, you will fail.

Summary

Databases are ubiquitous today. They are also larger and more complex than the information processing capacity of a human being - or even many human beings. It is not enough to have software tools to design and create databases. You must have tools that validate them, correct them and align them with internal standards. Yet the tools cannot replace human judgment as to what the intent of the database is in the enterprise.

Right now, Validator is the only product on the market that meets these criteria by automating the schema review and validation process, for the entire life cycle of the database schema.