Journal of International Information Management

Volume 6 | Issue 1 Article 5

1997

The Gap Between Theory and Practice: A Database Application Case Study

Martha Myers
Kennesaw State University

Paula Skinner
Kennesaw State University

Follow this and additional works at: http://scholarworks.lib.csusb.edu/jiim

 Part of the Management Information Systems Commons

Recommended Citation

Myers, Martha and Skinner, Paula (1997) "The Gap Between Theory and Practice: A Database Application Case Study," *Journal of International Information Management*: Vol. 6: Iss. 1, Article 5.
Available at: http://scholarworks.lib.csusb.edu/jiim/vol6/iss1/5

This Article is brought to you for free and open access by CSUSB ScholarWorks. It has been accepted for inclusion in Journal of International Information Management by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.

The Gap Between Theory and Practice: A Database Application Case Study

Martha Myers and Paula Skinner
Kennesaw State University

Abstract

The transition from theory to practice is often challenging for students. This paper considers this challenge within the context of implementing a database design in an end-user DBMS. The tradeoffs are examined between adhering to the theory that is taught in the classroom and straying from the theory when faced with deadlines, software applications that will not support the theory, and specific client needs. In the first section, the problems and tradeoffs for developers and system clients are described. In the second section, a specific case study of a student project for a charitable organization is described. The third section highlights the advantages of normalized, relational database structure. The fourth section examines some issues involved in the use of PC-based Database Management Systems (DBMS) packages. The fifth section illustrates the collision between theory and practice with the student-developed project. The sixth and final section summarizes inherent paradoxes in the transition from classroom theory to industry practice. Important questions are raised and some conclusions are drawn.

"How" versus "What"

What is more important, to get the job done or the way the job is done? To computer professionals, the "how" can become more important than the "what." To the client, clearly the "what" is much more important. That is, from the developer's perspective, proper design and procedures are extremely important. The system is a reflection of how well developers can take the client's wishes and transform them into a system that will produce the desired results. In order to produce the result desired by the client, developers must analyze and design a correct specification to meet the stated requirements. For this specification to be precise, developers must take into consideration all that the client wants the system to accomplish. After developers look at the client's desires, they must then take into account the limitations of the selected software tool and convey this to the client. With this step accomplished, developers can work with clients to form the best system possible for the client's needs within the framework of the chosen application. The developer's goal is an accurate design that will not disappoint the client.

From the client's perspective, the most important characteristics are as follows:

It works.
It is predictable.
It is delivered on time.
The cost is within budget.

Each client comes to the table with ideas of how a system should run and what exactly the system should be able to do. Clients do this with little or no concept of how database software works; they only know how they want the system to perform. Thus clients may have little understanding of how a system may be implemented.

In the client's view, the system is composed of menus, input screens and output reports. Clients want these screens and reports to be formatted to conform with their view of the information. Screens are usually designed based on the form used for the collection of data. If the design of this form is effective, then reports may also be based on this same format. If the information from the report needs to be utilized in a way that is different from the original form, then clients will ask for the report format to differ from the original input design. In sum, the developer focuses on data; the client focuses on information.

Case Study

The client for this case study, MUST, Inc., is a nonprofit community service organization providing a wide range of services such as a homeless shelter, food pantries, children's gifts for special occasions, job counseling, and dental checkups. Support for MUST, Inc. comes in the form of contributions from individuals, religious organizations, civic groups, foundations, and area businesses. Contributions include cash donations, donations of goods, and donations of volunteer time. This organization has been highly effective in drawing support from the community and in providing new and needed services. However, in recent years, the need for new services has outpaced incoming resources through donations.

The goal of the student-designed system was to track volunteers and donors for marketing purposes. System requirements included the ability to run on an IBM PC compatible machine running DOS 6.22 or Windows 3.1. Suggested software packages included Fox Pro 2.6 for DOS, dBase IV for DOS, Paradox, and Access. The user interface must be menu-driven or otherwise "fail safe" due to the volunteer nature of many users. In addition, the organization's director needed ad hoc relational query capabilities at a much higher level of sophistication. The time allotted for system development was two quarter sessions - approximately 21 weeks.

After a cost-benefit analysis, Paradox for Windows was selected for application development. After a thorough analysis of the problem and careful design in close work with the client, the student team began to build and populate a fully normalized set of tables to meet the needs specified for this client's organization.

The Case for Normalized, Relational Database Structure

As knowledgeable students of database concepts, the development team felt that a relational database was advantageous over a regular file processing system for the client.

1. In file processing systems, data is separated and isolated. This client needed to have access to data about each donor as well as his business, church and civic affiliations. The model is further complicated

by the fact that each business, church and civic group may be considered a "donor." The data must be accessed and used somehow from each of these separate sets of data as if it were a single set of data. This is very difficult without a relational database.

2. Data in flat file systems is often duplicated. This is a waste of file space but more important, there is the problem of data integrity. Data has integrity only if the data is consistent. When the same data is stored in several places, there is no assurance that any updates or changes have been made in all the correct places. David M. Kroenke states unequivocally that "data integrity problems are serious ... when results are inconsistent, the credibility of the stored data, and even the MIS function itself, comes into question."

3. When using file processing systems, the application that is chosen is dependent on the file format. When changes are made to the file formats, the application programs must be changed too. MUST, Inc. does not have a full time programmer who could make these changes if the client decided a format needed changed.

4. Each client has his own view of how he thinks his data should be extracted from a system. This system must track donors and donations in many diverse ways. For each user to be able to use the data in the way that would be applicable to the task that user needed to perform, this system must have the flexibility to present different views of the data.

5. A relational system can be designed so data from any database in the system can be accessed. Reports or ad hoc queries would make it possible for the client to access information on a donor, the business for which the donor worked, the church the donor attended and any civic groups to which the donor is affiliated.

6. Data independence allows developers the freedom needed to present the data in views most familiar to the client. This increases the likelihood that the developers have produced a system that is useful to the client.

Relational databases were developed for the most part to overcome the limitations of file processing systems. In a file processing system, programs directly access files of stored data. In a database system, programs interface with the DBMS which controls all data access. Developers of a specific application are able to focus on the functionality of the system, rather than maintaining the data. The student team planned and designed a system to contain all the different pieces of information about a single donor and access these in a way that the client required.

Tradeoff's and Pitfalls in PC-based RDBMS

To the student designer, the goal is a system built on a normalized, theory-based model that has few anomalies, little white space and runs in an extremely efficient way. How can developers achieve this in a timely manner in today's market? The choice of software, if carefully analyzed to meet the needs of the client, can be a very time-consuming process in itself. The time alone that it takes to convert tables to third normal form can be tremendous. The implementation differences in the end-user products

(Access, dBase4, Paradox) and professional-level products (Oracle) are substantial. Though professional-level packages will support SQL Queries on as many tables as needed, the students found restrictions on the number of tables that could be linked in a data model on the end-user products. The end-user packages forced a compromise on the carefully constructed normalized design.

End-user DBMS products like Paradox are designed to reach two very diverse audiences. End-users demand simplicity in database creation, simple enough that a novice can design a database for their own use, yet the trained designer needs more. Professional developers need greater control which means a more complex application. To make these products simple enough for anyone to use, power and flexibility are sacrificed. That is, meeting the end-user's need for simplicity and the developer's need for control has led to much compromise in the design of an end-user tool such as Paradox. The compromise is not necessarily the best of both worlds.

If this compromise has such a negative effect on normalized relational database applications, then why bother to normalize? Well-designed Relational Database Management Systems (RDBMS) applications have clear advantages over flat file design, and certainly over voluminous paper files. The key advantage of a normalized relational database is the reduction in data redundancy that ultimately increases data integrity.

Collision Between Theory and Practice in a Student Project

The organization's director led the design of the system. For example, the development team tried to change the menus at the implementation stage to make them simpler. Even though the development team tried to point out that such an approach would appeal to a wide variety of volunteer-users, the client was not interested. The client's view of the system was based on the original menu design; therefore, the client rejected the newer version.

Tables in this system were based on the relational model with a supertype, subtype design (Fig. 1) so that relationships between any individual and any business, church or civic groups could be established. The student development team transformed the tables to third normal form (3NF), rather than Boyce-Codd, fourth normal form, or fifth normal form. Practitioners strongly suggest that 3NF is sufficient for most applications. This should not have been a problem for any truly relational database package.

The first step in implementation was to build the tables (relations). These tables are the foundation for the forms, reports and queries necessary to give the clients access to information that they need or desire. In Paradox the development team found a limitation of tables per form with a one-to-one relationship. If forms were constructed based on a data model with more than three tables, the system gave the error message of "cannot modify this table" when the team tried to edit past the first table in the form. No information could be put into any table unless it was the master table of the form. Was third form normalization too much for this product to handle? Could Paradox adequately address a "real" application?

Given the supertype-subtype structure all tables were dependent on the same key - Profile ID. With this parent/child relationship between tables, it was imperative that referential integrity was established on

the Profile ID field. This was needed to assure that the proper links between the tables held so each donor was linked to the corresponding information. This field was set for auto-increment so that each instance in the database would have a unique ID number. For each instance of Profile ID the client wanted only one set of data identified by that id number so one-to-many relationships were not possible. For example, the key in the profile table was Profile ID, the key in the business table was Profile ID (using the parent key in the child to support supertype, subtype). If the team had designated a foreign key in the business table, the one-to-one relationship would have been destroyed. The team would then have established a one-to-many relationship by the rules of the product allowing one Profile ID to have many businesses attached to it. There was also no second key in the business table to establish referential integrity. So the tables could only link as parent/child. This made a link to any table from the child impossible. When the option of a business having subgroups was needed to possibly keep track of departments in the business that would contribute to the organization, the logical link could not be made from the business table. Further examination by the developers revealed a substantial problem with the way Paradox links two tables.

Like other PC products, Paradox is promoted as both user-friendly and developer-friendly. Also, like other software, Paradox error messages such as "cannot modify this table" lead to much confusion for the typical user. Such a message assumes some technical knowledge of database structure and relational theory. For example, the above error message assumes an understanding of table modification.

Not surprisingly, the manuals that accompany these software packages proved to be of little help. Additional desk references (from third party vendors) were essential. With eight different manuals, the development team thought that the answer to any questions that would come up could be found in one of these manuals. This was an erroneous assumption. Along with very poorly-written answers that did not completely explain a problem, these manuals did not even begin to address many of the questions that the development team had about designing in this product.

Conclusions and Lessons Learned

The path from MIS student to employed, productive MIS professional is not short or straight. "My theory doesn't work, what do I do now? Dare I ask or will they think that they have hired the wrong person?" How can theory be taught so that, when problems arise students are able to handle them and get the job done? According to Dr. Joseph H. Astrachan, Associate Professor of Management and Entrepreneurship at Kennesaw State University's Cole's School of Business, theory plays an important role if used correctly. "Theory is one step better than plain rote memorization. With theory you are memorizing a process for dealing with something and that's fine as long as the problem you're dealing with fits into a theory that is already out there. You must take it one step further, which is using existing theories and existing rote memorization and be able to come up with your own theories."

Theory is not the answer to all questions, but it can be an excellent building block to the answers that are needed. In sum, students must learn how to learn. This is required in order to understand theories that exist and then to use those theories, as well as to expand and build upon this information, and not stop where the theory stops. In order to do this, educators must teach students to think for themselves. When students are confident in the ability to construct new theory, panic will not ensue when a particular

business problem does not fit neatly with text book theory. The prepared student must be able to step back and reformulate a way of dealing with the problem at hand. Are educators sending students out prepared to take theory to this next step? Or are educators overlooking the importance of that long bridge between theory and practice?

The student team applied normalization theory to decompose large tables into smaller tables. The goal was to ensure data integrity and few anomalies or duplication of data. The student team applied the supertype-subtype concept so that the common characteristics of all the tables could be passed from one table (super) to the other (sub). But this theory did not work and the team could not find an existing theory that applied. The student team solution was partial denormalization and the complete dismantling of the supertype-subtype. (Fig. 2) However, it took the team several weeks to decide to take this approach. Why the delay? Why did it take the team so long to understand the dilemma? The students took the theory literally. Instead of using theory as a building block to solve the problem, the students used it as absolute law. So there was resistance to even the consideration of denormalization.

There are many different reasons to denormalize a database. Joe B. Edwards states
"The most often-cited reason for denormalization is to provide acceptable performance for specific applications. In doing this the D.B.A. is essentially concluding that performance and application efficiency are more important to the organization than flexibility, integrity and accessibility of data. Denormalization, in essence, provides improved performance for one type of database access at the expense of others both known and unknown."

The driving force behind the decision to denormalize this system was not improved performance. Rather, it was the delivery of a basic working system on time. Tables in third normal form could not be linked in the database application so that the team could access the information that the client needed in the way that it was needed. When the parent/child relationship was established on the parent key, referential integrity was a problem. When the team looked for technical support on this issue, a variety of conflicting answers surfaced.

Is the resulting system sufficiently robust? Will the system last and be efficient or will there be a phone call in a few months indicating that the system is unusable in its present form? One of the ultimate goals of system design is to be able to shake hands and walk away from a satisfied client. If the tables are denormalized and the original design structure is dismantled to make a database work on one specific software application, are developers decreasing the effective life time of the system? Is there more lost than gained? While developers gain the use of the software package of choice, lost is the database structure that developers value so highly.

In conclusion, this paper presents some conflicts that arise naturally when text book theories on database development do not fit a specific application. It explores the question of how well MIS students are prepared to construct and apply theory to "real" problems in a business setting. It raises important questions concerning system correctness and robustness. These issues have broad application, not only to the MIS field, but also to many other professional fields of study.

Figure 1. Fully normalized database structure

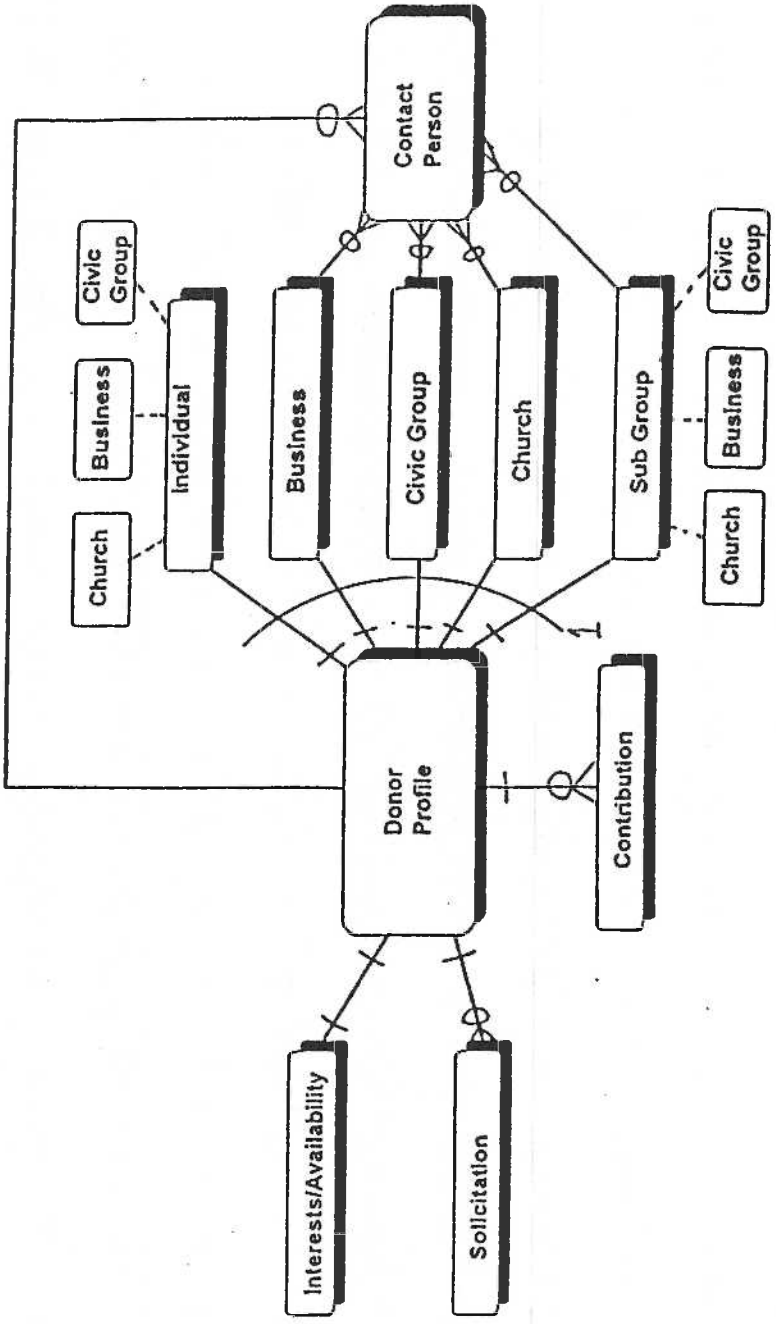
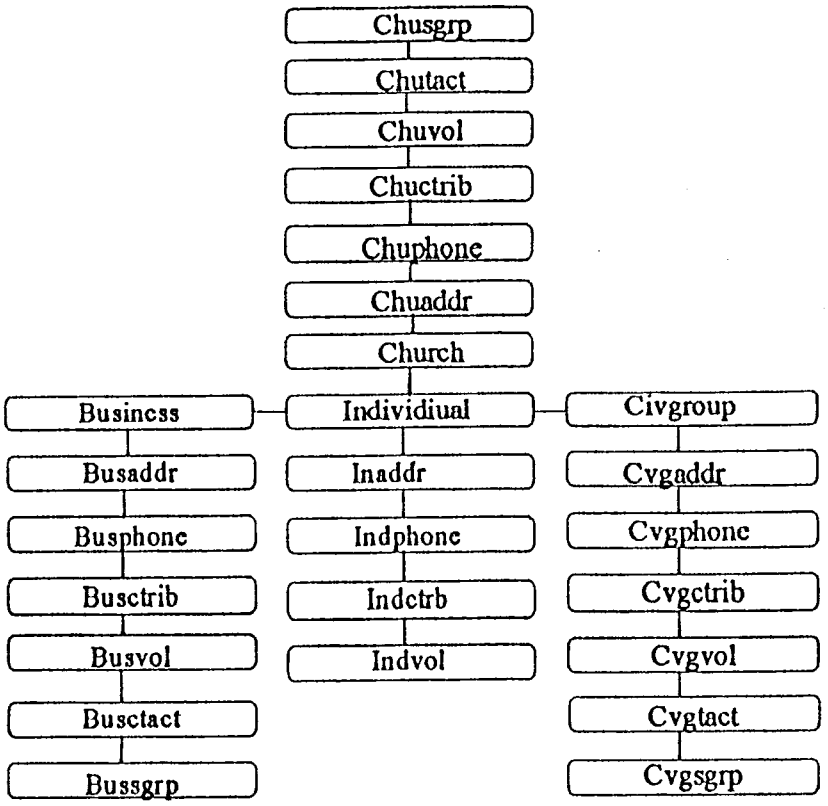


Figure 2. Partially de-normalized database structure



References

Date, C.J. (1990). ***An introduction to database systems***, Volume 1. Reading, MA: Addison Wesley.

Edwards, Joe B. (1990 July) High performance without compromise, ***Datamation.***

Kroenke, David M. (1995). ***Database processing fundamentals, design and implementation***, NY: Macmillan.