The Schema-Independent Database UI

A Proposed Holy Grail and Some Suggestions

Eirik Bakke
MIT CSAIL
ebakke@mit.edu

Edward Benson
MIT CSAIL
eob@mit.edu

ABSTRACT

If you have ever encountered a piece of highly domain-specific business software, you may have noticed that it was largely a graphical front-end to some relational database. You may also, in fact, have avoided using the system at all—studies show that information workers prefer to dump their data into spreadsheets, a general and more familiar tool which, unfortunately, is poorly suited for many standard database tasks. It is time that we stop streamlining the process of creating a new application for every schema, and that we instead develop the visual query languages that will let end-users access the full power of relational database management systems from a simple and unified interface. Once information workers can create, manage, and query real databases with the same ease as they routinely manipulate spreadsheets today, they will never return to their schema-dependent, consultant-made, and oddly-colored Microsoft Access applications.

1. A WAR STORY

In his younger days (well, before undergrad), one of the authors worked as an administrative assistant for one of Norway's 400 public schools of music and arts. To keep track of students, teachers, lessons, and rental instruments, the school had licensed a special-purpose database application, originally developed in FileMaker Pro[1] and later rewritten in 4th Dimension[2]. See Figure 1. The software was made and sold by a six-person consulting firm started by a former band director and FileMaker whiz on the other side of the country, and was constantly under development. A couple of times per year, the consultants would fly out to our individual schools to train us in the use of new features as well as collect feedback for further development. A common kind of request would be adding another field to an entity type in the database; for instance, we might ask if we could "get a checkbox in the 'Student' dialog to indicate whether their parents had signed the audio recording release form yet." The consultants were generally very helpful, and sure enough, three months later there would be a patch released with the new field in the database. One day, however, they balked. "Sorry. There is no more room in the dialog box."

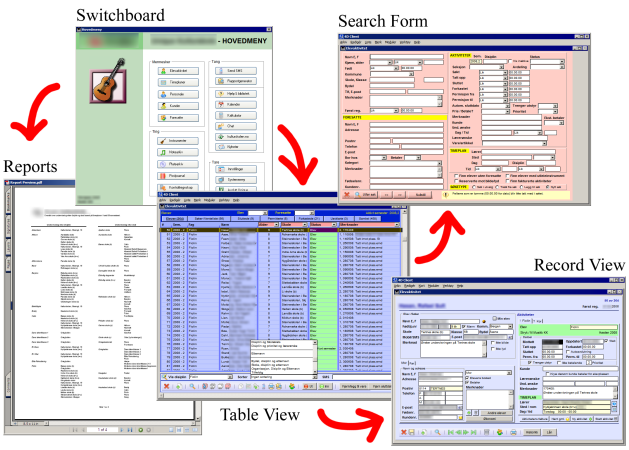

Figure 1: The stereotypical user interface of a FileMaker-style database application. Screenshots from an administration system for public Norwegian music schools.

2. TAILORED DATABASE APPLICATIONS

The story above, which could be retold for thousands of organizations worldwide, illustrates a number of points about database applications "in the wild." First, there is an infinite number of database schemas which may seem perfectly obscure to the world at large, but which may be of great value to a small number of people or organizations (e.g. schemas that deal with the intricacies of running a public Norwegian music school). Second, it takes a lot of effort, and a lot of attention to detail, to implement database applications for production use in such environments. In fact, it may take a small start-up—and we know of companies, such as Visma[3], that make a living out of acquiring said small start-ups after the they begin to reach market saturation. Third, there inevitably ends up being a gap between users and developers, and the schema of the database is hard to change. Moreover, users are no longer fully in control of their data; on one occasion, continuing the story above, we paid the consultants NOK 10,000 (about $2,000) to migrate

[1] http://www.filemaker.com
[2] http://www.4d.com
[3] http://www.visma.com

5^{th} *Biennial Conference on Innovative Data Systems Research (CIDR '11)* January 9-12, 2011, Asilomar, California, USA.

a "record first created at [some date]" field from the old FileMaker database to the newer 4th Dimension one on "only" a week's notice. Fourth, tailor-made database applications require their users to undergo training and learning periods which may be expensive in terms of both the consultants' and the end-users' time. Not to mention customer support: software that is under constant development is unlikely to be free of critical bugs, technical or usability-related. This is true even, and maybe especially so, for large organizations; while a larger number of potential target users may well decrease the marginal cost of custom software development, total training and support costs increase all the more.

A few other things are worth mentioning. Tailor-made database applications, like the one in the story, do very little other than provide basic *CRUD* (Create, Read, Update, Delete) facilities on their underlying relational databases. While the applications often include certain special features hard-coded for the schema in question (for instance, our system could send text messages to students based on their "cell phone number" field), these are typically not the main reason for using the application. Rather, people use database applications because they need to manage many different entities (students, teachers, lessons, instruments... or parts, suppliers, and plants for that matter) and the relationships between them. In other words, these are exactly the kind of tasks that relational databases were made for handling well.

Last, tailor-made applications seldom reach anywhere near the same level of maturity as more general-purpose ones. Features that are taken for granted in other applications may never get implemented in a tailor-made application, simply because the development time would not be justified for the size of the user base. In the music school system example, the developers found the table widget available in 4th Dimension inadequate for their needs, and decided to roll their own (presumably due to the limitations inherent in two-dimensional table views, see our later discussion). It was only after two years that they released a patch to allow the mouse wheel to be used to scroll the table up and down.

3. SPREADSHEETS

With all the problems associated with tailor-made database applications, is there an alternative? Currently, only "sort of."

If given a choice, information workers would rather dump their data into a spreadsheet than use some odd database application, even at substantial pain. Spreadsheet users "shun enterprise solutions" [7] and "do not appear inclined to use other software packages for their tasks, even if these packages might be more suitable" [3]. One survey shows that "sorting and database facilities" are the most commonly used spreadsheet features, with 70% of business professionals using them on a frequent or occasional basis [6]. In contrast, less than half use "tabulation and summary measures such as averages/totals"—one of the original design goals of the original VisiCalc spreadsheet. In fact, one of the most frequent uses of spreadsheets may be as a pseudo-database.

Spreadsheets have some great properties. Their interfaces are extremely mature and afford a large range of streamlined facilities for working with any data that can be arranged in a grid of cells, including multiple selection, copy/paste, find/replace, undo/redo, inserting and deleting, extending data values, sorting and filtering on arbitrary fields, navigating and selecting cells with the keyboard, and so on. They have hundreds of millions of users worldwide that are already trained in their use, and a relatively large portion of these are power users. And, of course, there are no development costs.

Unfortunately, spreadsheets lack features essential to database applications, including joins, views, forms, report generation, multiple users, and so on. In effect, spreadsheets are great for single-table databases shared between only a few users, but become very painful to use as tasks scale out in various dimensions.

4. THE HOLY GRAIL

Spreadsheet software is the quintessential example of a *general-purpose data manipulation tool*, and the fact that people use it for database tasks despite great pains should be an indicator that such tools are the way to go. If we could make a general-purpose data manipulation tool that covers even, say, 80% of the functionality afforded by typical tailor-made database applications, information workers would never look back.

We should give credit to others who have proposed similar grails before. Quoting Yannis Ioannidis:

"Visual database query languages that can express the full spectrum of complex queries desired and data visualization mechanisms that can capture the essence of large and complex data sets in ways that match users' intuition are somewhat of a Holy Grail in database user interfaces. Unfortunately, almost nobody seems to be looking for it!" [4]

Visual query languages are indeed central to the idea of of making a general-purpose data manipulation tool. A tool that is intended to replace the majority of tailor-made database applications must certainly give the user the power to express SQL-like queries from a graphical interface. In fact, we need more than relational completeness, since many of the views desired in a database front-end are hierarchical rather than tabular; we discuss this below. Now quoting the 2005 The Lowell Database Research Self-Assessment:

"It is a long-standing lament that the database community does too little about user interfaces. (...) A small number of slick visualization systems oriented toward information presentation were proposed during the 1980s, notably QBE and VisiCalc. There have not been comparable advances in the last 15 years, and there is a substantial need for better ideas in this area." [1]

It is interesting to note that one of the two notable visual query languages cited is VisiCalc: the spreadsheet itself. Keep in mind, spreadsheets are indeed one kind of visual query language. We need better ones, so that they can replace our tailor-made database applications.

5. THE STATE OF THE ART

Since the stereotypical user interface design of a tailor-made database application follows rather mechanically from the underlying database schema (see again Figure 1), application builders like FileMaker et al. provide plenty of shortcuts and wizards to create them, as do development frameworks such as Ruby on Rails[4]. Systems like AppForge [9] and App2You [5] take these ideas further than their predecessors, abstracting away much of the low-level form design

[4] `http://guides.rubyonrails.org/getting_started.html#getting-up-and-running-quickly-with-scaffolding`

and data binding work required earlier, making the application development process more WYSIWYG, and hiding the technical details of relational schema design from the user. Still, these systems are "application builders"—rather than aiming to be general data management applications that can be used with any schema, they aim to make it easier for developers to churn out special-purpose ones at a faster pace. The user interfaces produced are just as schema-dependent as before, they lack good features for general-purpose data management, and users have to learn to use them from scratch.

Despite being in the application builder category, we consider AppForge [9] to have made a major step in the right direction, by creating a visual query language that allows the user to retrieve joined hierarchical views of the data in the database. We believe such a language must be a core part of any general-purpose data manipulation tool that intends to replace tailor-made database applications. Query-by-Example was already mentioned as a visual query language [10, 1]; however, it is only able to retrieve flat tables similar to those returned by SQL queries, and is as such not expressive enough for our purposes. A major and relatively recent visual query language is that of Polaris/Tableau [8]; this system provides a very expressive user interface for defining visualizations and aggregations on tabular or multidimensional data, based on the pivot table concept found originally in Lotus Improv (1993). The output, however, remains in flat table form, possibly rendered on screen using a selection of visualizations. Pivot table systems provide cross-tabulated, as opposed to hierarchical, views of the user's data.

6. SOME SUGGESTIONS

What would a universal tool for relational data look like? It would likely be a blend of interface norms from the spreadsheet world along with more powerful data management capabilities from the database world. In particular, we imagine:

- The tool is not a builder for the interface. The tool *is* the data interface. Users should be able to construct the particular views they need as they go, at a small enough cost that these views can be treated as disposable objects.
- The tool should allow users to work with data as a set of views. The underlying relational structure should be available for power-users, but hidden by default.
- The tool can read and write both data and schemas. A flexible, universal interface for existing relational schemas would be an achievement in its own right, but we believe such tools should be able to create and modify schemas as well. Users, however, should not need to be aware of these operations unless they want to be.
- The tool should present data to the user as complex structures where appropriate, even if the data is not displayed as such relationally. One-to-many or many-to-many relationships may, for instance, appear as a bulleted list from the perspective of a particular object of focus.
- The tool should contain a visual query language. That is, it allows the user to construct and compose view queries over the data using a graphical user interface.
- As a necessity, the tool must have features common to spreadsheets (multiple selection, search/replace, etc) and existing database applications (form-style browsing, reports, etc).
- The tool errs on the side of off-the-shelf usability, requiring technical input from users only when necessary to resolve a potential ambiguity. Calendar applications provide a good example of this: when an event in a repeated series is modified, the application provides the user with several options: modify one item, modify the entire series, or modify all future events. Relationships between entities create many such ambiguous situations that should be handled as conversations between the computer and the user rather than by requiring the user to specify *a priori* how she intends to modify information in the future.

We propose that a universal tool may be constructed by relying on hierarchical views as the interface metaphor. Hierarchical views are ubiquitous in the user interfaces found in traditional tailor-made database applications, because they naturally encode the one-to-many and many-to-many relationships that are found in just about every business-oriented database schema. Some examples of hierarchical views include:

- Patient records in a medical database, with doctor visits as an embedded table of each patient.
- Course records in a school database, with required readings and teaching staff as embedded tables.
- Emails in an inbox, decorated with tags, subject, and sender name.
- Books in a bookstore, displayed by category.

Given this, the role of the small-business interface database consultant appears to be writing code to translate a relational programming interface into a series of hierarchical views. If what we want are hierarchical views, then we should *build databases that allow us to pretend the information is hierarchical in the first place.*

Note that many hierarchical views, such as the second example above, do not map naturally to the flat tables that could be returned by a SQL-style query, because they involve multiple unrelated one-to-many relationships being encoded in parallel (e.g. reading lists and teaching assignments). A standard SQL-style join would lead to an exponential number of duplicated rows from each table being included in the result set.

Building database tools which embrace the commonality of hierarchical information browsing would enable the creation of databases that interact with casual users in the same way as they think about their data. At any given time, most users want to look at an entity, or a list of entities, along with properties and possibly related other entities. SQL can always be used in the exceptional cases.

7. RELATED WORKSHEETS

One of our own systems, Related Worksheets [2], attempts to explore the spreadsheets-as-a-database concept by extending the spreadsheet paradigm to let the user establish relationships between rows in related worksheets as well as view and navigate the hierarchical cell structure that arises as a result. A user study on 36 regular Excel users showed

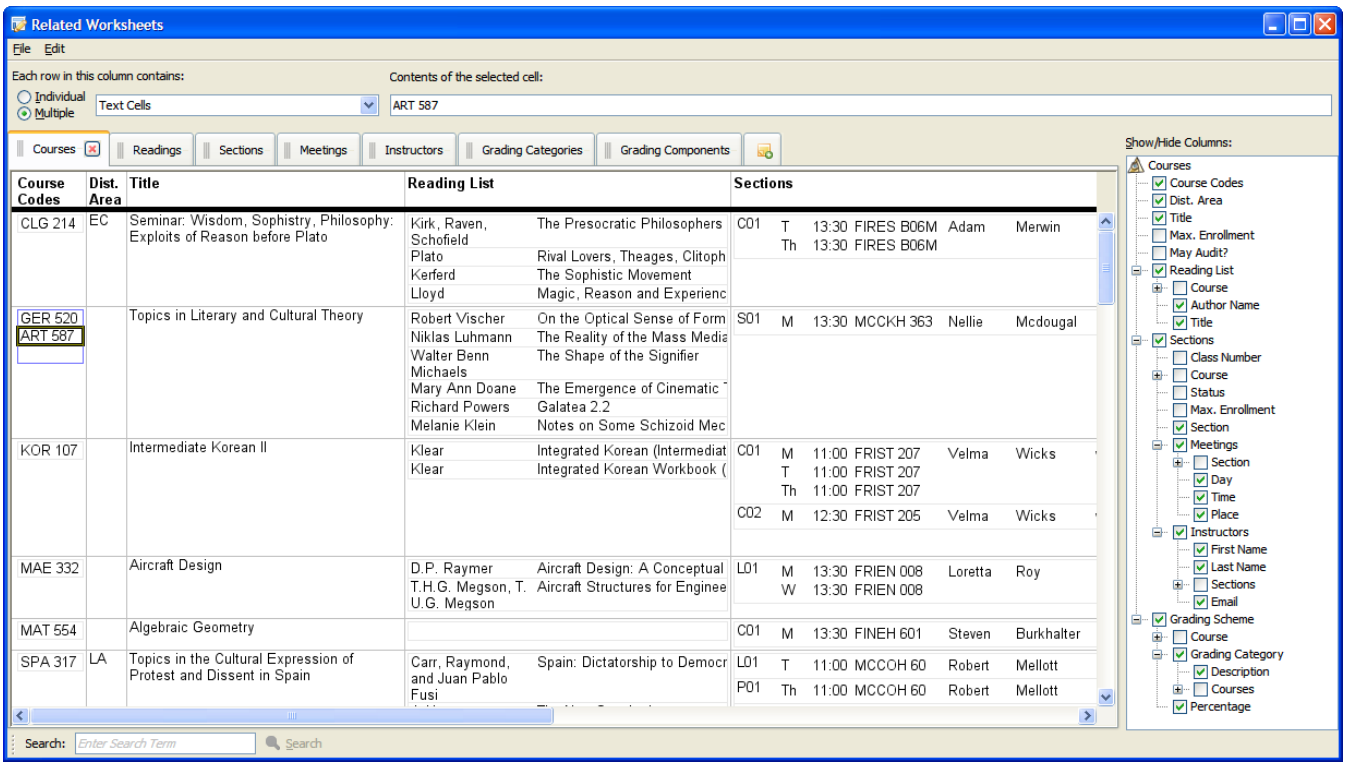

Figure 2: Screenshot of our Related Worksheets application, showing a hierarchical view of entities related through one-to-many and many-to-many relationships.

that first-time users of our system were able to solve lookup-style query tasks with the same or better accuracy than subjects in a control group using Excel, in one case 40% faster on average ($p<0.05$). See Figure 2.

8. CONCLUSION

Since the early years of database management systems, we have kept building a new database application for every schema that came along. These resulting *tailor-made* database applications are expensive to develop, hard to maintain, hard to use, and hardly much more than graphical front-ends to their underlying databases. By looking to the success of spreadsheets as a general-purpose data management tool, and by developing new visual query languages to let end-users access the full power of relational database management systems from a simple and unified interface, we can eliminate the pains of using either spreadsheets or tailor-made applications for database tasks, and get the best of both worlds.

9. REFERENCES

[1] S. Abiteboul, R. Agrawal, P. Bernstein, M. Carey, S. Ceri, B. Croft, D. DeWitt, M. Franklin, H. G. Molina, D. Gawlick, J. Gray, L. Haas, A. Halevy, J. Hellerstein, Y. Ioannidis, M. Kersten, M. Pazzani, M. Lesk, D. Maier, J. Naughton, H. Schek, T. Sellis, A. Silberschatz, M. Stonebraker, R. Snodgrass, J. Ullman, G. Weikum, J. Widom, and S. Zdonik. The Lowell database research self-assessment. *Commun. ACM*, 48(5):111–118, 2005.

[2] E. Bakke, D. R. Karger, and R. C. Miller. A spreadsheet-based user interface for managing plural relationships in structured data. In *CHI*, 2011 (to appear).

[3] Y. E. Chan and V. C. Storey. The use of spreadsheets in organizations: determinants and consequences. *Information & Management*, 31(3):119–134, 1996.

[4] Y. E. Ioannidis. Visual user interfaces for database systems. *ACM Comput. Surv.*, page 137, 1996.

[5] K. Kowalzcykowski, A. Deutsch, K. W. Ong, Y. Papakonstantinou, K. K. Zhao, and M. Petropoulos. Do-It-Yourself database-driven web applications. In *CIDR*, 2009.

[6] J. D. Pemberton and A. J. Robson. Spreadsheets in business. *Industrial Management & Data Systems*, 200(8):379–388, 2000.

[7] N. Raden. Shedding light on shadow IT: Is Excel running your business? Technical report, Hired Brains, Inc., January 2005.

[8] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002.

[9] F. Yang, N. Gupta, C. Botev, E. F. Churchill, G. Levchenko, and J. Shanmugasundaram. WYSIWYG development of data driven web applications. *Proc. VLDB Endow.*, 1(1):163–175, 2008.

[10] M. M. Zloof. Query-by-Example: A data base language. *IBM Syst. J.*, 16(4):324–343, 1977.