A Spreadsheet-Based User Interface for Managing Plural Relationships in Structured Data

Eirik Bakke, David R. Karger, and Robert C. Miller
MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar Street, Cambridge, MA, USA
{ebakke,karger,rcm}@mit.edu

ABSTRACT
A key feature of relational database applications is managing *plural* relationships—one-to-many and many-to-many—between entities. However, since it is often infeasible to adopt or develop a new database application for any given schema at hand, information workers instead turn to spreadsheets, which lend themselves poorly to schemas requiring multiple related entity sets. In this paper, we propose to reduce the cost-usability gap between spreadsheets and tailor-made relational database applications 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. We present Related Worksheets, a spreadsheet-like prototype application, and evaluate it with a screencast-based user study on 36 Mechanical Turk workers. First-time users of our software were able to solve lookup-type query tasks with the same or higher accuracy as subjects using Microsoft Excel, in one case 40% faster on average.

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation: User Interfaces—*Graphical User Interfaces (GUI)*

General Terms
Design

INTRODUCTION
Structured data is everywhere, and information workers need to manage it. On the storage side, the problem has largely been solved through the advent of relational databases; an off-the-shelf relational database management system will serve as a back-end for any database application. Yet, a back-end alone provides no user interface, and the database community has called for more work to be done on the front-end side of the application stack [1].

Certain kinds of database applications may be useful to a large number of people and organizations. For these, high-quality free or commercial GUI implementations are likely to be available. Examples include Personal Information Managers (PIMs), Customer Relationship Management (CRM) software, and business accounting software. Other applications are far more domain-specific, and have to be developed from scratch, possibly by an organization's IT department or an external consulting firm. Real-world examples range from cultural venue and event management systems[1] to software used by Norwegian immigration authorities to manage applications for political asylum[2].

When the effort required to either adopt or develop a new *tailor-made* relational database application is too high, information workers instead turn to a general and more familiar tool: the spreadsheet. One survey [7] 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. In contrast, less than half use "tabulation and summary measures such as averages/totals"—one of the design goals of the original VisiCalc spreadsheet—or more advanced features. Furthermore, spreadsheet users "shun enterprise solutions" [8] and "do not appear inclined to use other software packages for their tasks, even if these packages might be more suitable" [2]. Presumably these ostensibly suitable software packages include specialized database applications; after word processors, database software is the second kind of software most commonly encountered by spreadsheet users, with 60% of spreadsheet users reporting some use of them [7].

Once it is established that spreadsheets are being used for database tasks, we should ask what functionality is lost when we compare them to corresponding tailor-made relational database applications. In this paper we focus on the ability to maintain *plural* relationships—one-to-many and many-to-many—between different kinds of entities in a dataset, a key feature of any relational database application [3]. The world of tailor-made database applications has seen some fairly standard user interface paradigms for viewing and editing such relationships, see Figure 1. The user retrieves records using a *search form*, views results in a *table view*, and edits or views the details of individual entities in a *record view*. We consider two aspects of these user interfaces to be crucial to their relationship management capabilities. First, they can provide the user with joined views of related ta-

CHI 2011, May 7–12, 2011, Vancouver, BC, Canada.

[1] E.g. `http://us.artifax.net`
[2] Unique Flyktning by Visma AS. `http://www.visma.com`

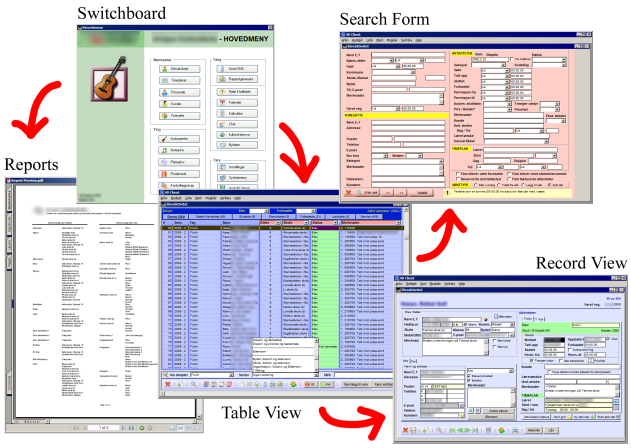

Figure 1. The stereotypical user interface of a database application made with FileMaker, Microsoft Access, or similar software development tools. (Screenshots from a Norwegian software system for public music schools.)

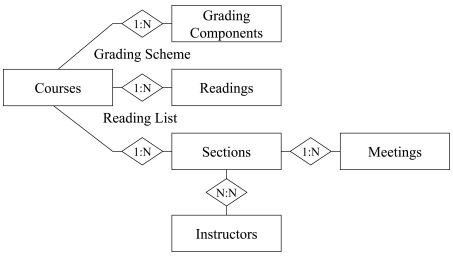

Figure 2. Simplified schema of an academic course management system (Entity-Relationship diagram).

bles in the database. Second, tailor-made interfaces are not restricted to simple tabular views, but can expose relationships in a hierarchical fashion. For instance, a record view for a single entity can itself contain tab rows or miniature table views—*subforms* or *portals*—to represent multiple, independent sets of entities related to the main entity through plural relationships. Note that in tailor-made applications, these views are hard-coded for the particular schema in question. Hard-coded views may sometimes offer significant usability benefits in terms of learning and efficiency, since they make many decisions on the user's behalf and may be fine-tuned for specific tasks. However, they are not an option in general-purpose interfaces.

As a running example, we consider the schema of an academic course management system, see Figure 2. Here, each Course entity has multiple Sections (lectures, precepts, labs, etc.), each Section has multiple Instructors, and so on. A tailor-made interface might expose many different views of the underlying data, for instance, "a list of sections by instructors, each section showing its associated course," or "a list of courses, each course showing its reading list, grading scheme, and sections, each section showing its meetings and assigned instructors." Both of these views are hierarchical, since they call for lists contained within lists. The first view could just as easily be displayed as a flat table, like that returned by a simple join query in SQL. This is not the case for the second view. Since the second view involves parallel joins on multiple independent relationships, i.e. Grading Scheme and Reading List, such joins would produce an extremely large number of rows of redundant data. Whenever we wish to represent an entity together with related entities from two or more plural relationships, hierarchical views become a necessity.

Hierarchical views can often be represented in spreadsheets using clever formatting tricks, e.g. indentation, skipped cells, or comma-separated lists. However, this strategy has a number of problems. First, such views must be manually created and, if more than one view of the same data is desired, kept in sync with the original data. Second, the strategy does not generalize well, with the sheet structure quickly growing unwieldy as relationships fan out and cascade. For instance, while a comma-separated list in a cell might be easy enough to maintain, a list of lists is not. Third, when the data representation deviates from the standard tabular format, also known as first normal form (1NF), key spreadsheet features such as sorting, inserting, charting, filtering, or even simple navigation between individual units of data become hard or impossible to apply correctly.

Because traditional spreadsheet UIs provide no easy way to create joined views of related tables, it becomes impractical to follow good practices of schema normalization, which call for tables of redundantly represented data to be decomposed into multiple smaller ones. This in turn exposes all the usual problems associated with managing improperly normalized data, i.e. insertion, update, and deletion anomalies [5].

While spreadsheets may be inadequate for managing multiple related tables of data, they are great for managing individual ones. Spreadsheets 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. Tailor-made database UIs seldom reach this level of sophistication, due to the limited resources available to develop an application for only one schema. When it comes to general editing tasks on tabular data, spreadsheet systems have an advantage even over most tailor-made applications.

In this paper, we propose to reduce the cost-usability gap between spreadsheets and tailor-made relational database applications 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. Implementing this idea, we present Related Worksheets, a spreadsheet-like desktop application which provides relationship editing facilities similar to those often developed with tools like FileMaker or Microsoft Access, but which work from a single generic user interface rather than requiring a new one to be developed for every conceivable database schema. Unlike in a regular spreadsheet, the user can model complex database schemas

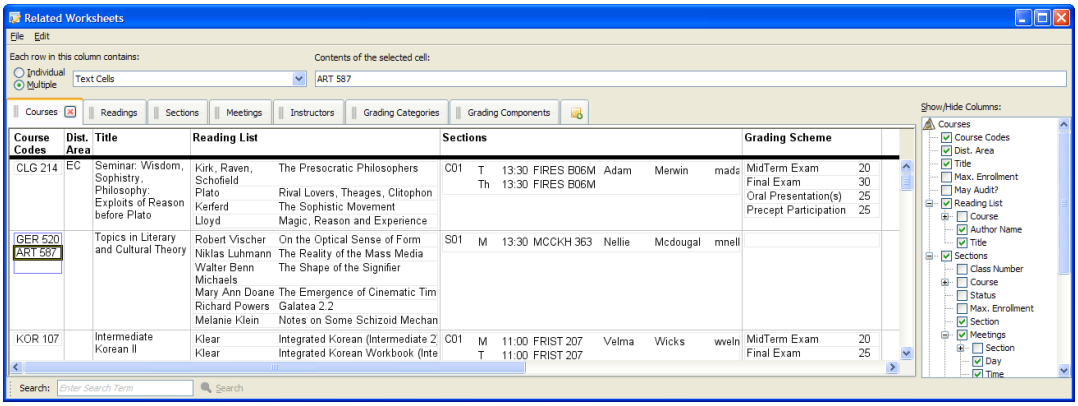

Figure 3. A particular hierarchical view of a dataset with the schema shown in Figure 2, as presented by the Related Worksheets system.

in their proper normalized form without losing the ability to view and edit related data in one place, ensuring long-term manageability of the data. Figure 3 shows one of the hierarchical views mentioned earlier, as handled by the Related Worksheets system.

We evaluate our system with a screencast-based user study on 36 Mechanical Turk workers. First-time users of our software were consistently able to solve lookup-type query tasks with the same or higher accuracy as subjects using Microsoft Excel, in one case 40% faster on average.

RELATED WORK

Visual tools for development of tailor-made database applications first emerged in the 1980s, and have since become a standard genre of office productivity software on par with spreadsheets, word processors, and presentation software. These tools provide visual form designers, wizards, and such to help speed up the development process. Among the first such tools were 4th Dimension[3] (1984) and FileMaker[4] (1985); Microsoft Access[5] appeared later (1992). Some of the newer applications in this category, such as Intuit Quickbase[6], AppForge [11], and App2You [6], have brought usability improvements on the application developer's side, in particular with respect to how well the development environment manages to hide technical details such as join queries, foreign key relationships, and the underlying relational schema. For instance, the application design processes of both AppForge and App2You are form-driven rather than schema driven, that is, the systems use the forms designed by the developer to derive the underlying relational schema rather than requiring them to be defined explicitly upfront. AppForge also has a flexible visual interface for defining updatable hierarchical views. However, these systems are application builders rather than general data manipulation tools: a new application must be built for every schema, and the actual applications produced are just as schema-dependent as any other tailor-made database application. Furthermore, neither project attempts to create a spreadsheet-like environment; these are purely form- and widget-based user interfaces.

There is an increasing tendency among commercial spreadsheet applications to encourage users to organize data in simple, flat tables like those found in a relational database. Microsoft Excel[7], Google Docs[8], and Apple iWork Numbers[9] all encourage the creation of explicit table areas in order to make table-related features such as sorting, filtering, summation, and form generation possible or more reliable.

For relational databases, Query-by-Example [12] is the standard example of a schema-independent visual query language. It inspired many of the standard search interfaces that are found in tailor-made database applications today, although these simpler interfaces typically omit features crucial to the expressiveness of the original language. QBE always retrieves flat tables, and cannot produce hierarchical views.

The concept of *pivot tables*, pioneered in Lotus Improv (1993), exists today both in spreadsheets like Microsoft Excel as well as Online Analytical Processing (OLAP) tools such as Tableau [9] (formerly Polaris) and Oracle Business Intelligence Discoverer[10]. Pivot tables are interactive user interfaces for producing cross-tabulated views of data, that is, views showing data points grouped and aggregated in categories along two axes. While interfaces like Tableau may hide adjacent duplicate data values or organize axis headings hierarchically, both the input and output of a pivot table interface is a flat table of values. Thus, these systems do not provide hierarchical views.

PrediCalc [4] approaches the problem of maintaining consistency between multiple tables in a spreadsheet when the items in the tables are related by plural relationships, including the case where there are temporary or partial inconsis-

[3] http://www.4d.com
[4] http://www.filemaker.com
[5] http://office.microsoft.com/access
[6] http://quickbase.intuit.com
[7] http://office.microsoft.com/excel
[8] http://docs.google.com
[9] http://www.apple.com/iwork/numbers
[10] http://www.oracle.com/technetwork/developer-tools/discoverer

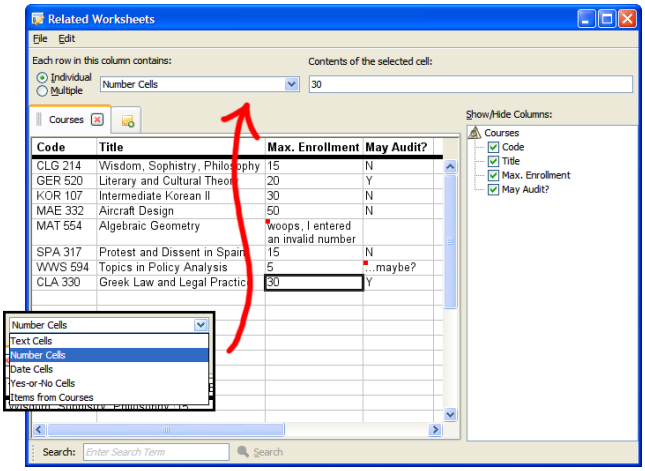

Figure 4. Related Worksheets after creating a new worksheet, entering some simple tabular data, and setting non-text preferred data types for two of the columns. Some of the values entered do not conform to their columns' data types, and have been marked with a red warning box by the software.

tencies in the constraints. It does not, however, extend the structure of the spreadsheet itself beyond two-dimensional cell grids, and thus does not support hierarchical views.

In IceSheets [10], cells in a worksheet can recursively contain entire new worksheets, allowing data to be addressed and organized in a hierarchical fashion. However, the content of the hierarchy is static, and must be built manually by the user. Moreover, the system has no concept of joins or relationships. This makes the system unsuitable for the kind of database tasks we are interested in.

THE RELATED WORKSHEETS SYSTEM

Related Worksheets is a desktop application which, in its initial state, appears largely like a regular spreadsheet. The user can create, open, and save documents known as *workbooks*, and each workbook contains a number of infinitely sized *worksheets* that can be created and selected through a row of tabs. Empty worksheets start out looking like those of Excel or any other spreadsheet application, and can be navigated and edited using standard mouse and keyboard commands—click cells to select, navigate cells with arrow keys, type to edit, drag to resize columns, etc. Since our system's novel features depend on the creation of multiple worksheets, and since worksheets are referred to by name in various parts of the user interface, we force the user to create and name every worksheet explicitly. At startup, we show a blank welcome tab with an instruction on how to create a new worksheet.

Like Google Docs and Apple iWork Numbers, we intend the user to populate each worksheet with a single table of items under a designated row of column headers. Thus, the first row of each worksheet is automatically formatted as a heading, with a bolded font and a thick border underneath, and pinned to remain visible at the top of the worksheet view during vertical scrolling of the rest of the worksheet. Columns can be renamed at any time by editing the contents of their respective header row cells, and can be deleted or moved left or right.

In our system, each column in a worksheet has an explicit preferred data type associated with it. The user may change this type at any time using a drop-down menu in the toolbar area. If the value of a cell does not conform to its column's selected data type—e.g. text in a Number column or an invalid date in a Date column—the cell is rendered with a red warning box similar to those used by PrediCalc and Excel to indicate other kinds of inconsistencies. See Figure 4. The user is always free to keep the default Text type, which accepts any primitive value.

The system's weak form of type checking serves to allow temporary inconsistencies. For instance, it is possible to paste data from Excel without first having to set the appropriate column types, and the data type of existing columns can be changed without special conversion semantics. The type checking functionality applies not only to primitive data types, but also to reference types and column cardinalities, which we discuss next. In the header row, cells always behave as they would in a Text column.

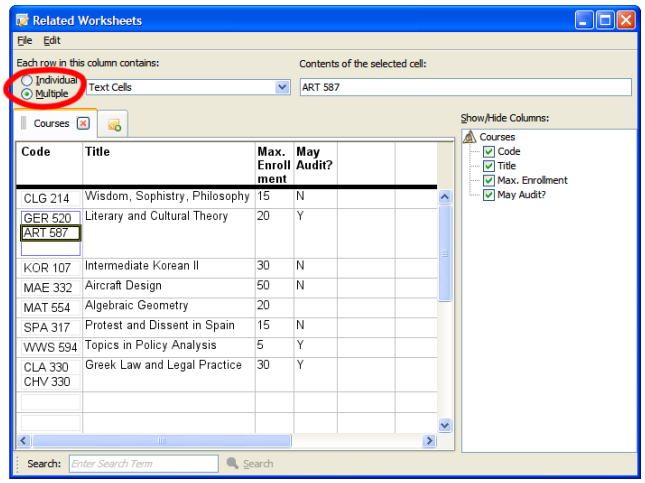

Figure 5. Related Worksheets after the Code column has been set to allow multiple inner cells and some additional course codes have been entered. The outer cell containing the currently selected inner cell is rendered with a blue border, and with one extra blank inner cell as an affordance for adding additional entries.

In addition to a preferred data type, each column also has a preferred cardinality, selected by a pair of radio buttons next to the data type drop-down. Columns preferring *individual* cells per row behave just like columns in a traditional spreadsheet; this is the default. The *multiple* cardinality, however, indicates that the column accepts multiple cells per row. This situation is illustrated in Figure 5. It now becomes necessary to distinguish between *outer* and *inner* cells; in general, outer cells are defined by the intersection of a row and a column, while inner cells are contained within outer cells. Data values are contained in inner cells, and the cursor moves left/right/up/down between the inner cells that are visible on the screen at any time. Outer cells are always rendered to show the contents of each of their non-blank inner

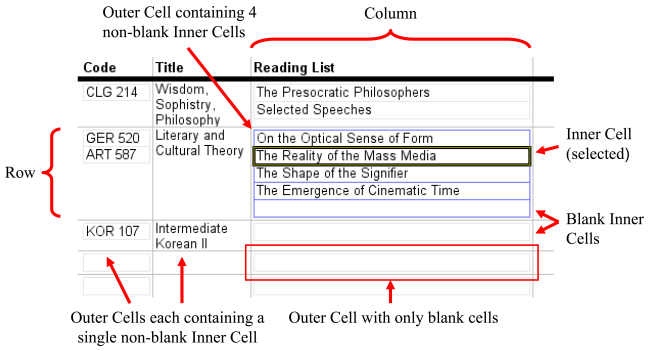

Figure 6. Terminology for describing worksheets in the Related Worksheets system.

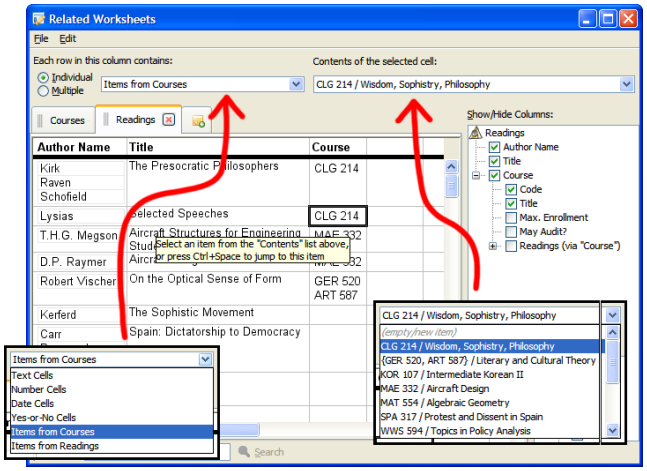

Figure 7. Related Worksheets after adding a new worksheet with data. The Course column has been set to contain references to the Courses worksheet by selecting Items from Courses as the column's preferred data type. Reference values for cells in this column can now be selected from a drop-down list (right) that shows one entry for each row in the Courses table.

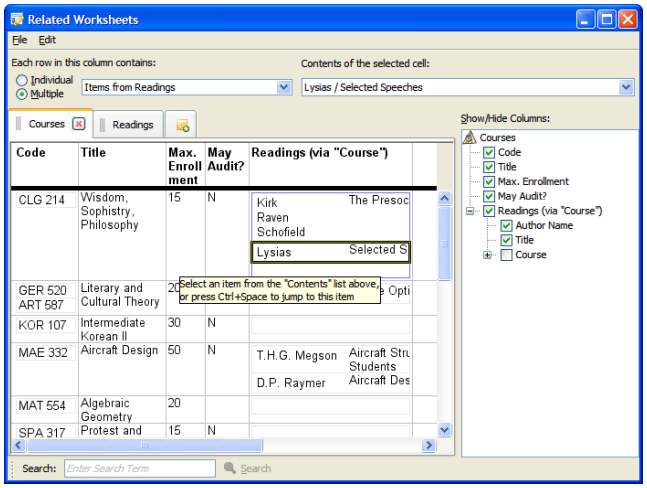

Figure 8. Related Worksheets showing the reverse reference column that was automatically created in the Courses table when the Course column of the Readings table was set to refer to the former. By pressing Ctrl+Space, the user can teleport between the cursor location selected here versus that of Figure 7.

cells, and always with at least one inner cell visible, even if blank. In columns of multiple cardinality, a light gray border is added around inner cells as a visual indicator of the special behavior of these cells. When the cursor enters such a cell, the border is highlighted in blue, and an extra blank inner cell is rendered within the same outer cell as an affordance to enter additional values. See Figure 6. Thus, to add values to a cell of multiple cardinality, the user first moves the cursor to an existing value in the list, then to the new blank cell that appears at the end. Additional blank cells will appear as the user fills the previous ones with values, or disappear again as the user deletes the last value in the list.

Besides the primitive column data types, our system provides reference types. For each worksheet in the open workbook, including the currently selected one, a reference type Items from <worksheet to be referenced> is available in the column data type drop-down, below the standard primitive types. Each inner cell in a column of such a type is allowed to contain, instead of a primitive value, a reference to a row in the target worksheet specified by the column type. When the cursor enters an inner cell in a reference column, the user can select a reference value from another drop-down list. This drop-down shows one entry per non-blank row in the referenced worksheet, plus an option to reset the cell to its default empty state. Note that unlike a relational database, our system does not have a concept of primary or foreign keys. Instead, rows are assigned an internal identifier on first use, and subsequently referenced using this identifier. This is completely transparent to the user.

As an example, suppose the user is modeling an academic course management system like that of Figure 2. In two separate worksheets, the user may have created tables of Courses and Readings, respectively, and now wishes to establish a relationship between the two in order to maintain a reading list for each course. The user can do this by creating a column Course of type Items from Courses to the Readings worksheet. This column will then contain, for each reading, the course containing the reading in its reading list, and the user can select the appropriate course from the reference selection drop-down. See Figure 7. Alternatively, the user can create a column of type Items from Readings in the Courses worksheet—we will explain this symmetry later. Note that *creating* a column simply means making use of a new empty one, like in a regular spreadsheet.

When an inner cell in one worksheet contains a reference to a row in another, a representation of the referenced row is itself rendered within the referencing cell. This representation is a recursive selection of columns from the referenced row, and can be configured through the Show/Hide Columns tree of checkboxes, similar to the Schema Navigation Menu found in AppForge. For instance, in Figure 3, each row in the Courses worksheet contains a list of references in the inner cells of the Sections column to the rows of the Sections worksheet. Each section reference is then rendered just like the corresponding row would be in the Sections table, except

with some otherwise visually distracting borders removed, and with some of its columns hidden according to the user's selection in the Show/Hide Columns tree. Since each section also has a list of references to Meetings as well as to Instructors, this process is applied recursively until primitive values are rendered in the base case. Effectively, the user is able to perform arbitrary nested joins on the relationships between the worksheets, starting from any given worksheet as a base table. Each worksheet has its own recursive Show/Hide Columns configuration. For instance, a course can be shown with its full title when viewed under the Courses worksheet, while only showing its six-letter course code when viewed through several levels of references from the Instructors worksheet.

The entries in the reference selection drop-down are string representations of each potential target row, also built from the fields selected to be visible in the Show/Hide Columns tree, including nested ones, or whatever portion of the string there is horizontal space enough to display. Though not implemented in our prototype system, users should be able to pick items from this drop-down list by typing the first letters of this string representation, using a combo box instead of a list box.

Relationships are bidirectional. When the user creates a reference column in one worksheet in order to maintain a relationship with another worksheet, a corresponding reference column is automatically created in the referenced worksheet, referring back to the original worksheet. For instance, if each Reading is assigned to a Course, then each Course has a set of Readings. When the type Items from Courses is selected for the Course column of the Readings worksheet, this automatically creates a corresponding column of type Items from Readings in the Courses worksheet. See Figure 8. The column name Readings (via "Course") is only an initial default and, as for any other column, can be changed at any time. Like any other column, a reverse reference column can be hidden from the view using the Show/Hide Columns tree. Deleting a reverse reference column deletes the original referencing column as well. Note that the system itself makes no distinction between what we have been calling *forward* and *reverse* reference columns; we are merely using these terms to distinguish between the columns explicitly created by the user versus those automatically made available in the other direction.

By setting the cardinalities of a related pair of forward and reverse reference columns, the user can choose between one-to-one, one-to-many, many-to-one, and many-to-many relationships. By default, an automatically created reverse reference column is set to have the multiple cardinality, resulting in a one-to-many relationship if the forward reference column was of the individual cardinality, or a many-to-many relationship if the forward reference column was of the multiple cardinality. Our system models many-to-many relationships directly, without an intermediate join table, as long as no attibutes are required on the relationship itself. This is an important usability feature, as the concept of join tables is abstract and may make little sense to a spreadsheet user that does not have experience designing relational database schemas.

For the purposes of selecting columns to display in the Show/Hide Columns tree, forward and reverse references can be followed indefinitely. For instance, it is possible to show, for each instructor, the courses taught by that instructor (via the Sections worksheet), then for those courses, what other instructors teach the course, then for those instructors, the complete set of courses taught by those instructors, and so on. Infinite recursion is avoided by expanding the Show/Hide Columns tree lazily and hiding, by default, columns that have already been instantiated as an ancestor in the tree. Like any other column, these can be shown or hidden again by checking or unchecking the appropriate checkbox in the tree.

While references are rendered recursively and the user can easily see across many cascading and parallel relationships simultaneously, the cell cursor moves across the top level only. This was a design decision made to keep the interface simple, in particular to keep cursor behavior similar to that of a regular spreadsheet interface. As a consequence, in order to edit a particular value, whether it be a primitive or a reference, the user must first select the worksheet that contains that value directly, that is, the worksheet that contains the value in one of its inner cells. For instance, in Figure 7, it is not possible to edit the course code or title of a course without first switching to the Courses worksheet tab. This is analogous to restrictions often found in FileMaker-style database applications. However, generalizing a standard idiom often found in the record views of such applications, we introduce a *teleport* feature to simplify navigation across related entities in different worksheets.

The teleport feature allows the user to quickly jump between multiple related worksheets via the references that exist between them. With the cursor located on a cell containing a reference to a row in another worksheet, the user can press Ctrl+Space to switch to the related worksheet, with the cursor placed on the inner cell containing the corresponding reverse reference in the referenced row. After teleporting, the user is free to move the cursor around in the newly selected worksheet, for instance to edit values that were previously too deep in the hierarchy to select with the cursor, or even to teleport another degree away from the original worksheet. To get back to the original point, the user can move the cursor back to the original reverse reference that was selected after the teleport, and teleport again. Consequently, teleporting twice without changing the cursor location will always lead back to the original cell. We believe a regular set of back/forward buttons, not implemented in our prototype, would complement the teleport feature well.

As an example of the teleport feature, consider the situation in Figure 7. A user who wishes to edit the title of the course CLG 214 can press Ctrl+Space to get to the state shown in Figure 8. The user can now move the cursor to and edit the cell containing the Wisdom, Sophistry, Philosophy primitive value. When done, the user can move the cursor back to the

cell containing the Lysias / Selected Speeches reference and teleport to get back to the original location in Figure 7.

We provide a simple search feature accessible from a search bar similar to that found in the Mozilla Firefox[11] web browser. By entering a text string and clicking the Search button, the cursor will move to the next cell whose string representation contains the search term. The string representation is the same as that used by the reference selection drop-down item, minus separator characters; it contains the primitive values of all the columns that have been recursively selected for display in the Show/Hide Columns tree.

In the process of setting up a sample database for our user study, we implemented two special data normalization commands that are likely to be of general utility. The Detect References command replaces each value in a column with a reference to a row in a target worksheet that contains the value in question in its first visible column (as selected in the Show/Hide Columns tree), if a unique such row exists. This is typically used to convert a column of primitive values to a column of reference values, for instance after having imported two flat tables related through a foreign key relationship from a relational database. The other command, Replace References, works oppositely; it replaces each reference value in a column with the value of the first visible column in the referenced row. This command has uses beyond undoing the work of a previously invoked Detect References command; in particular, it can be used as a step to convert the join tables commonly found in relational database schemas into directly supported many-to-many relationships.

USER STUDY

To evaluate our system, we conducted an online user study using Amazon Mechanical Turk[12] and the User Study Console, a Java-based tool we developed to allow test subjects to stream timestamped recordings of their computer screens to our server with a minimum of effort. The User Study Console would also automatically download and open sample spreadsheets we had prepared as well as allow the user to submit changed versions of these in exchange for a confirmation code to be pasted into the Mechanical Turk HIT (Human Intelligence Task, a survey form). Depending on the stage and parameters of the experiment, sample spreadsheets would be opened in either Microsoft Excel, assumed to be already installed on the user's computer, or in our own Related Worksheets application, distributed as part of the same Java Web Start executable.

Our study was a between-subjects design in two stages. The first stage was a recruiting task offered to the general population of Mechanical Turk workers, intended to let us select a group of workers who (1) were willing and able to successfully run the User Study Console on their machines, (2) happened to have Microsoft Excel or a compatible spreadsheet application installed, (3) had demonstrated good faith in solving an otherwise unrelated spreadsheet task, and (4) had provided us with answers to some background and demographic questions. We offered $0.25 for workers to launch the User Study Console, make a chart based on sample data we provided in an Excel spreadsheet, and fill in a survey. The survey included questions about gender, age, occupation, previous spreadsheet experience, and previous experience with Microsoft Access, FileMaker, or other database software, if any.

[11] http://www.mozilla.com/firefox
[12] http://www.mturk.com

Courses(Course, Distribution Area, Title, Max. Enrollment, May Audit)
Readings(Course, Author Name, Title)
Sections(Class Number, Course, Status, Max. Enrollment, Section)
Meetings(Section Class Number, Day, Time, Place)
Instructors(First Name, Last Name, Email)
Grading Components(Course, Grading Category, Percentage)
Instructors-Sections(Instructor Name, Section Class Number)
Cross-Listings(Crosslisted Course Code, Primary Course Code)

Figure 9. The normalized schema as it appeared in the Excel version of the tasks. Primary keys that participate in foreign key relationships are underlined, foreign keys are double-underlined.

In the second stage of our experiment, Mechanical Turk workers with complete and technically unproblematic submissions for the recruiting task were divided randomly into two groups of equal size, and invited by e-mail to do a longer follow-up task, worth $3.00. Workers in the control group would be instructed, like before, to launch the User Study Console and solve tasks using a Microsoft Excel spreadsheet that we provided. Workers in the treatment group would, on the other hand, be instructed to use the Related Worksheets application. The tasks and instructions in the two main forms were identical except in the description of the tool used to solve the tasks. No actual instructions were provided on how to perform the tasks, neither in the Excel nor the Related Worksheets versions of the experiment.

The actual database tasks modeled in our user study were based on the course management system example introduced earlier. The data was a subset of an actual course database, containing 37 courses and related entities.

In the Excel version of the tasks, the database was represented as a set of normalized relations, each stored as a table in its own worksheet. See the Introduction for a discussion of why this particular mapping is a natural one for our investigation. We used natural rather than surrogate primary keys, e.g. course codes in the form "KOR 107," to identify the various entities involved. The full normalized schema of the database is shown in Figure 9. For the Related Worksheets version of the task, we imported the same normalized database tables into our system as individual worksheets, converted foreign key columns into reference columns using the Detect References and Replace References operations mentioned earlier, and made default selections in the Show/Hide Columns tree corresponding to what we believed would be seen in a real course management system. In particular, the Courses worksheet was shown with all fields and subfields fully expanded, excluding reverse reference columns (each course showing its reading list, grading scheme, sections, meetings, instructors, etc.). Each of the other worksheets were shown with all of their immediate

#	*Description*
1	Find a course that is taught by Harry Morrill.
2	In the course "MUS 105: Music Theory Through Performance and Composition," what percentage of the final grade is derived from the Midterm Exam?
3	Find a course that satisfies the "LA" (Literature and the Arts) distribution area, with a lecture (meetings belonging to a section denoted "L01") that starts after noon (after 12.00).
4	What is the e-mail address of the instructor who teaches "KOR 107: Intermediate Korean II"?
5	Who teaches the precept section of "HIS 383: The United States Since 1920" that meets on Wednesdays at 10am?

Table 1. Tasks given in the user study.

fields visible, and some selection of important identifying subfields for each related entity (e.g. course code, distribution area, and course title whenever a course was referenced). Effectively, it would not be necessary for a user to show additional columns or subcolumns to solve the tasks given, though depending on window size and screen resolution, it might have been necessary to scroll horizontally or hide some columns to see the desired information.

The exact tasks given are listed in Table 1. The actual instructions given to users also specified the exact form expected of the answers and gave an example of a correctly specified answer. These tasks are lookup (read-only) tasks of the kind we would imagine to be frequently performed on a real-world course management database. We originally included a number of update tasks as well; however, these uncovered a number of discoverability bugs that would have to be fixed before realistic update performance could be measured.

After the expiration of the main task, we reviewed workers' submissions and associated screencasts, log files, and uploaded spreadsheet files. We graded each of the tasks on a pass/fail basis, and manually estimated the amount of time spent on each task using video frame timestamps and observations of the subjects' actions.

RESULTS

The recruiting stage of the study yielded 52 subjects, screened from submissions by 64 distinct Mechanical Turk workers. The subjects were divided randomly into two groups of 26 subjects each, and invited to participate in the main part of the study. Of these, 18 recruits from the Excel group and 18 recruits from the Related Worksheets group completed the main task. An additional 4 recruits from the Excel group and 4 recruits from the Related Worksheets group started doing a few of the tasks, but soon closed the User Study Console and never submitted a response. The remaining recruits never opened the User Study Console during the second part of the study.

Of the 36 subjects participating in the main part of the study, 25 were male (69%). Subjects were between 19 and 48 years of age, with first, second, and third quartiles at 26, 30, and 35 years, respectively. 16 subjects (44%) described having some prior experience using "database software such as FileMaker or Microsoft Access." Most users had significant previous spreadsheet experience; Table 2 shows the levels of typical spreadsheet use as reported by subjects themselves. The subjects' self-reported occupations are categorized in Table 3.

	"How often do you typically use spreadsheet software?"	*N*
1	Never	1
2	Once a year or less	1
3	A couple of times per year	1
4	A couple of times per month	12
5	A couple of times per week	5
6	Almost daily	10
7	Multiple times per day	6
	Total	36

Table 2. Number of subjects reporting particular levels of previous spreadsheet experience. The numbers to the left are used to calculate averages in Table 3.

Occupation	*N*	*Average spreadsheet experience*
Student, business/finance	2	6.0
Student, mechanical engineering	1	4.0
Student, unknown	1	2.0
Teacher	2	4.0
Business administration	5	5.6
Business process outsourcing (BPO)	3	6.0
Homemaker	5	3.2
Programmer	8	5.8
Other IT	3	5.0
Other technical	5	5.6
Other non-technical	1	4.0
Total	36	5.1

Table 3. Number of subjects and average levels of previous spreadsheet experience by occupations reported. Averages are based on the 7-step scale in Table 2.

Of the subjects who submitted a response to the main task, the number who passed each task is shown in Table 4. The average times taken to complete each task, for subjects who solved each task in question correctly, are shown in Figure 10. Due to intermittent network problems, 5 screencasts from the Excel group and 2 screencasts from the Related Worksheets group were incompletely streamed and could not be used for timing measurements.

Using uploaded screencasts and event logs, we summarized the features used by subjects during the study. This data is

		Task #				
	N	*1*	*2*	*3*	*4*	*5*
Excel	18	18	18	14	17	12
RS	18	18	18	14	17	15

Table 4. Number of subjects who solved each task correctly.

	Excel			*Related Worksheets*		
Feature	*n*	*N*	*n/N*	*n*	*N*	*n/N*
Search	13	17	76%	9	17	53%
Mark Rows	3	15	20%	2	17	12%
Filter	11	15	73%			
Sort	1	15	7%			
Teleport				8	18	44%
Show/Hide Columns				3	17	18%

Table 5. Number of users observed to make use of specific features or techniques during the user study.

shown in Table 5. Some features are found in both Excel and Related Worksheets, others are specific to one system. Except for the teleport feature, the numbers are based on manual review of the screencast videos; the number of data points is slightly less than the total number of users due to some incomplete screencasts. The teleport command was invoked 1, 1, 2, 5, 5, 7, 12, and 13 times by each respective user. Of the Related Worksheets subjects making use of the search feature, 3 discovered it only after having progressed about halfway through the study. Of the Excel subjects making use of the filter feature, 3 did not use the search feature at all. "Mark Rows" refers to subjects visibly working to keep track of rows in a worksheet; this was done in various ways by different users. Two Excel users colored marked rows yellow with the highlighting tool, one of these also at one point wrote down the primary keys of marked rows in an external text editor. A third Excel user marked rows by typing an "x" in the first empty column available next to the table. Two Related Worksheets users marked rows as well; these both used an external editor. It is possible that some subjects may have taken notes on paper as well.

DISCUSSION

For each submission we watched the screencasts and measured the time taken to solve each task. We then, for each task, applied Welch's unpaired two-tailed t-test to the times taken to complete the tasks by those subjects who got them right only. Only Task 4 showed a statistically significant ($p<0.05$) improvement in the Related Worksheets case; it was 41% faster than Excel. Task 5 showed a 31% improvement, but was not statistically significant ($p=0.09$). In all other cases the results were insignificant.

The different tasks can be analyzed in terms of the joins involved and tables traversed in the normalized database schema used in the Excel version of the database in each case. It is useful for this purpose to refer back to the Entity-Relationship diagram that was shown in Figure 2.

For Task 2, only the primary key is needed from the Courses table, and the answer can be found by looking at the Grading Components table. This makes the task simple in both Excel and Related Worksheets. We believe the fact that subjects spent a relatively large amount of time on Task 1 compared to, for instance, Task 4 to be due to the learning time needed to get acquainted with the database and, in the case of the Related Worksheets group, the software itself. For Task 4, we saw a large performance gain for subjects using our system versus those using Excel. We can attribute this to the manual join that must be done between the Sections and Instructors-Sections tables followed by the lookup in the Instructors table in the Excel case. In the Related Spreadsheets case, solving the task was a simple matter of finding the course in the Courses worksheet, which showed all related entities in one place. Task 3 is harder than Task 4, since there are both multiple meetings per section as well as multiple sections per course; in Task 4, only a single lookup was required in the Instructors table. To solve Task 3 in Excel typically requires the user to keep track of numbers of several sections to look for in the Meetings table while browsing the Sections table. In Related Worksheets, one can simply browse for the appropriate course, though since we do not provide any filter-like capabilities, the user must manually look through the entire list of courses. Task 5 involved the Sections, Meetings, and Instructors-Sections tables. Since there is often only a single instructor per section, the task is simpler than Task 3. Like in Task 4, we see indications of a performance advantage among users using the Related Worksheets application over those using Excel. While not statistically significant, it seems that the advantage is large but slightly smaller than that of Task 4. This effect could potentially be due to Excel users getting more familiar with the manual process of joining tables.

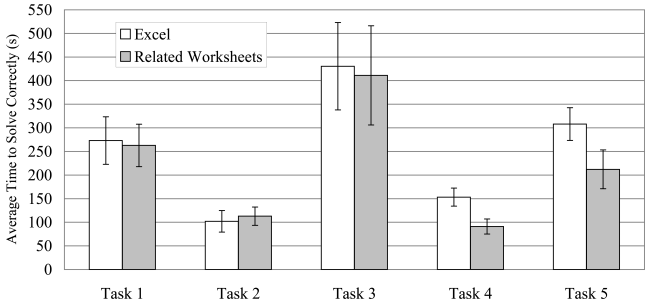

Figure 10. Timing measurements for the user study. The error bars show the standard error of the mean.

The most frequently used feature in both the Excel and the Related Worksheets version of the tasks was search, available in both systems. Typically, users of both systems would use a combination of searching and manual browsing—scrolling back and forth, switching between worksheets, and scanning the data visually—to complete the tasks. This was anticipated: the chief advantage of the Related Spreadsheets system is, in this context, the relevancy of the information seen on the screen at any time. Using a Firefox-style search interface may have put us at a disadvantage due to limited discoverability; only 53% of Related Worksheets users tried searching, vs. 76% of Excel users.

Among the Excel users, we were surprised to find that just about as many subjects made use of the filter feature as made use of the search feature. The filter feature allows the user to restrict rows shown in a table to those containing specific values selected from a drop-down menu above each column. Subjects used the filter feature both to search for specific rows in each table and to mark multiple rows of potential interest for later review. Since the drop-down list provided by the filter feature is always sorted, users could quickly locate a specific item in an otherwise unsorted table, without using the search feature. As mentioned, three users even made extensive use of the filter function without making use of the search feature at all. Given these observations, we consider a filter feature to be an important potential addition to future versions of our interface.

Users sometimes found it necessary to temporarily mark rows of interest in order to solve tasks involving more than a single join between tables; see our earlier observations. Two Related Worksheets users wrote down primary keys in an external editor, despite this not being necessary for completing the tasks in our system. We assume these users must have believed it would be faster to approach the problem in the same way as they would in a standard spreadsheet rather than spending time learning how the hierarchical views worked.

In the Related Worksheets version of the tasks, it was not necessary to modify the Show/Hide Columns configuration, since we had already provided reasonable defaults. Most users thus rightly ignored the feature. A few subjects used the Show/Hide Columns configuration, along with column resizing, to hide unnecessary data from view. The teleport feature was tried by 8 of the 18 Related Worksheets subjects; at least 5 of these subjects seemed to make productive use of it throughout the progression of the study, indicating significant potential value in this feature.

FUTURE WORK

In future work, we plan to incorporate a more expressive visual query language that allows a larger class of hierarchical views to be built by the user, including the ability to apply sorting, filtering, formulas, and aggregation functions. Furthermore, the system should be able to render these hierarchical views using a selection of different layout configurations beyond the simple tabular one, for instance to emulate the record views that are ubiquitous in FileMaker-style applications. Finally, the system should provide a more flexible interface for storing and organizing views themselves. Future evaluation should include a comparison with tailor-made UIs, and also investigate other classes of tasks such as schema creation and more complex querying.

CONCLUSION

We have presented Related Worksheets, a spreadsheet-like application that lets the user establish relationships between rows in related worksheets as well as view and navigate the hierarchical cell structure that arises as a result. These features make the system well-suited for database tasks. A user study on 36 subjects found that first-time users of our system were able to solve lookup-style query tasks with the same or higher accuracy as subjects in a control group using Excel, in one case 40% faster on average.

ACKNOWLEDGEMENTS

We thank Paul Grogan and Phumpong Watanaprakornkul for their help designing and implementing the original Related Worksheets prototype.

REFERENCES

1. Abiteboul, S., Agrawal, R., Bernstein, P., Carey, M., Ceri, S., Croft, B., DeWitt, D., Franklin, M., Molina, H. G., Gawlick, D., Gray, J., Haas, L., Halevy, A., Hellerstein, J., Ioannidis, Y., Kersten, M., Pazzani, M., Lesk, M., Maier, D., Naughton, J., Schek, H., Sellis, T., Silberschatz, A., Stonebraker, M., Snodgrass, R., Ullman, J., Weikum, G., Widom, J., and Zdonik, S. The Lowell database research self-assessment. *Commun. ACM 48*, 5 (2005), 111–118.
2. Chan, Y. E., and Storey, V. C. The use of spreadsheets in organizations: determinants and consequences. *Information & Management 31*, 3 (1996), 119–134.
3. Chen, P. P.-S. The Entity-Relationship model—toward a unified view of data. *ACM Trans. Database Syst. 1*, 1 (1976), 9–36.
4. Kassoff, M., and Genesereth, M. R. PrediCalc: a logical spreadsheet management system. *The Knowledge Engineering Review 22*, 03 (2007), 281–295.
5. Kent, W. A simple guide to five normal forms in relational database theory. *Commun. ACM 26* (February 1983), 120–125.
6. Kowalzcykowski, K., Deutsch, A., Ong, K. W., Papakonstantinou, Y., Zhao, K. K., and Petropoulos, M. Do-It-Yourself database-driven web applications. In *CIDR* (2009).
7. Pemberton, J. D., and Robson, A. J. Spreadsheets in business. *Industrial Management & Data Systems 200*, 8 (2000), 379–388.
8. Raden, N. Shedding light on shadow IT: Is Excel running your business? Tech. rep., Hired Brains, Inc., January 2005.
9. Stolte, C., Tang, D., and Hanrahan, P. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics 8*, 1 (2002), 52–65.
10. Whitmer, B. C. Improving spreadsheets for complex problems. Master's thesis, Brigham Young University, August 2008.
11. Yang, F., Gupta, N., Botev, C., Churchill, E. F., Levchenko, G., and Shanmugasundaram, J. WYSIWYG development of data driven web applications. *Proc. VLDB Endow. 1*, 1 (2008), 163–175.
12. Zloof, M. M. Query-by-Example: A data base language. *IBM Syst. J. 16*, 4 (1977), 324–343.