deebee: General, Personal and Relational Database Management System for Casual Users

Dept. of CIS - Senior Design 2009-2010

Yan Yanovskiy
ayanov@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

Dr. Susan B. Davidson
susan@cis.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

ABSTRACT

deebee *is a general, personal and relational database management system specifically designed for casual users who have no knowledge of database concepts. Currently, users have heaps of data that they need to manage without any niche applications to help them. Thus, we will create a general data management system based on the relational database model so that the system can be flexible to the user's needs. The three main pillars of the user's experience will be to* ***edit****,* ***view****, and* ***share*** *tables. In the creation stage, the user will quickly and easily set up the schema for their databases and fill it in with data. Then, they will be able to share this data with colleagues, family and friends and customize privacy settings for each invited user. Finally, the system will allow users to access the data using Query By Example, a powerful yet easy way to allow users to query their data.*

Because deebee *is built as a website instead of a desktop application, it is automatically cross-platform. In addition, the guiding philosophy of the project is to create intuitive design and artfully abstract away technical details of database administration.*

1. INTRODUCTION

We are all currently living in what is commonly referred to as the "Information Age." This is a time where data is extremely important, and keeping track of it all is getting harder and harder. Fortunately, E. F. Codd [4] solved the problem of creating powerful yet flexible databases with his relational model in 1970. A relational database consists of relations, which are sets of uniquely identifiable tuples with the same attributes. Later, Peter Chen [20] developed a way to conceptually visualize data in a relational database with the Entity Relationship (ER) model, in which entities (nouns) are connected with relationships (verbs) and attributes (adjectives). Since then, computer programmers have adopted the relational model, and it is now used ubiquitously on the World Wide Web.

There are several theoretical concepts that address how to access data set up in these relations. Two such concepts are relational algebra and relational calculus. SQL (Structured Query Language) was the practical outgrowth of these concepts [18]. Although SQL has become the de facto standard querying language for these relational databases, another way to query, called Query By Example (QBE), was invented by Moshé M. Zloof [23]. QBE allows users to easily build queries (that can be translated into SQL) by simply inserting values into an empty table. For instance, if one wanted to search for all "Person" records with the first name "John" who live in Pennsylvania, one would simply fill in an empty table with the word "John" under "Name" and "Pennsylvania" under state. Then the equivalent SQL query (*e.g.*, SELECT * FROM Person WHERE FirstName = 'John' AND State = 'Pennsylvania') would be created automatically. One should refer to Figure 1 for a User Interface (UI) mockup of QBE. The goal was to appeal to casual users who have little to no experience in computers or mathematics [23].

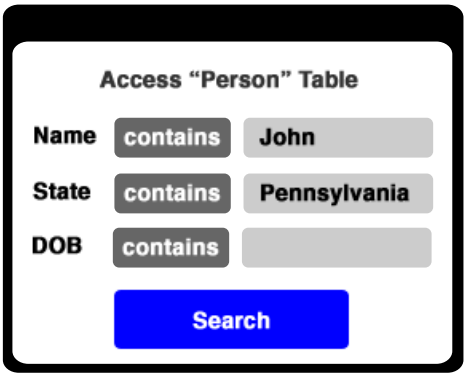

Figure 1: QBE UI Mockup

Users have benefited from immensely popular data management software that are based on the aforementioned concepts, such as Apple's iTunes (to manage music and other media) and iPhoto (to manage photos). iTunes [13], for instance, uses QBE for its "smart playlists." However, there has not been a focus on general, personal database management systems for casual, non-business users. The reason creating a very general system is important is because while software developers often fill niches where data management is needed (*e.g.*, music management), not every niche is profitable nor suitable for a standalone application. For example, where should a piano teacher keep track of his or her lessons (how much did it cost, how long was it, was there noticeable improvement, etc.)? The market for piano lesson data management may not be large enough, so currently, the solution most users flock to as a way to store data is Microsoft Excel [7]. However, while Microsoft Excel offers some database management features, such as Sort and Fil-

ter, its primary purpose is not to be a relational database system. Instead, it is a spreadsheet application. Thus, users lack the tools to build powerful queries, semantically connect their data and share data easily with colleagues, family and friends. In addition, they do not adhere to good database management techniques, such as reducing redundancies where it is not needed [7].

Thus, an application that leverages the powers of relational databases in an easy-to-use interface would give people more power to keep track of their data. This application would allow users to input data with fields of their own choosing. Additionally, users would be able to quickly and easily share their databases with certain groups of people or the world. For example, the coach of a little league baseball team could keep track of all the children's positions and statistics and then share it with parents who could track their children's status and progress.

2. RELATED WORK

2.1 Research

This is not the first project to attempt to bring relational databases to the "casual" user. In fact, the father of the relational model, E.F. Codd himself, wrote a paper entitled "Access to Relational Data Bases for a Casual User" [5]. Codd's project, called "RENDEZVOUS," however, focused on connecting natural language questions with unambiguous queries. *deebee*, on the other hand, will focus on a Graphical User Interface (GUI) as the liaison between casual users' thought processes and SQL queries.

Moreover, there has also been research that studies the effectiveness of various database interface concepts. For instance, Cattell in [3] proposed a user interface based on the ER model as a way to allow casual users to construct databases. One should refer to Figure 2 to see what an ER UI could look like. Another study in [8], which gave a wide set of users database-related tasks to complete in a two-hour time limit, corroborated the effectiveness of this proposal by showing that people who used the spatial database view demonstrated 21% better performance than did those given the nonspatial view. Similarly, in the category of ease of use, subjects in the spatial view group rated ease of use 16% higher than the subjects in the nonspatial view.

In addition, Thomas *et al.* in [21] provided a psychological study of Zloof's QBE idea. Although they acknowledge some weak points (*e.g.*, users had problems with aggregate operations such as COUNT), QBE has characteristics that make it ideal for casual users. For example, since the user is provided with a template, there is no need for the user to generate queries "free style." Moreover, because there is no natural language component, colloquial and formal languages do not clash (*e.g.*, "and" as it used in natural English is not always equivalent to logical "and"). Finally, QBE is easy, yet extendable with experience and need for more complicated queries [21].

2.2 Competitors

The primary competitor to the proposed system would be spreadsheet applications, such as Microsoft Excel [12], and more importantly, Google Docs [10]. Google Docs allows people to create tables (*i.e.*, sheets in a spreadsheet file) and share this data with other users in real time with a detailed revision history. However, as with any spreadsheet application, this option does not focus on being a database and therefore is not optimized for handling data and relationships between data. That is, one would have to already understand database concepts to use spreadsheet applications in a more powerful way. Also, spreadsheets created in Google Docs do not provide for advanced filtering features. For instance, Google Docs does not allow for a QBE search such that searching for "main" in a record that contains the string "123 Main St." would result in a positive hit.

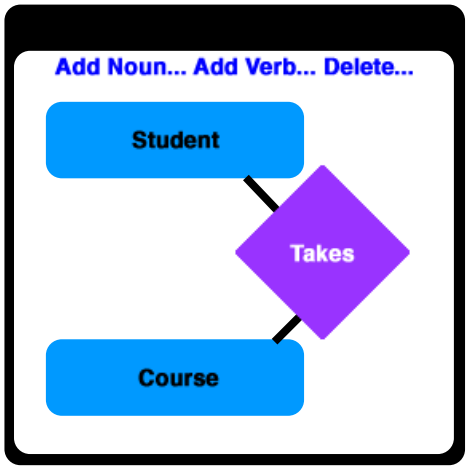

Figure 2: ER UI Mockup

Another Microsoft product that is a competitor is Microsoft Access [1], which is a stripped down version of a complete database management system. Despite being stripped down compared to software like Microsoft SQL Server, it provides the power of a relational database to end users and is already included in many Office suites. However, like some of the other competitors that will be mentioned, it is clearly focused on business clients using Windows. The first clue is the price point: $229 for a single user license. In addition, the interface is clearly designed for a user that has experience in database management and is not designed to be extremely intuitive. Moreover, because it is a desktop application, a user cannot share access with other users as quickly and easily as in Google Docs for example[1] [1].

Similarly, there are many other websites that provide data management systems online, but all are very clearly targeting business clients only. Some examples of these sites include Intuit QuickBase, Caspio Bridge, TrackVia and Dabble DB. The last [9] is an especially interesting database management system, because although it is designed for business clients, it attempts to make database building slightly more intuitive and accessible. For example, it has a data visualization view in which data can be expressed on a map of the United States. On the other hand, Dabble DB users are charged $8 per editing user per month, which means there is a high threshold for newcomers to overcome. Most probably do not understand the benefits of a relational database

[1]It should be noted that it seems Microsoft is attempting to integrate online solutions directly into the application in Office 2010. However, these services are still very clearly focused on business users and remain too unintuitive for casual users. Additionally, Microsoft will even feature web versions of applications such as Excel, but not Access. These web applications will resemble Google Docs.

system and will not adopt it. Plus, sharing write ability of databases with friends and family is infeasible at such rates. Finally, the interface still does not abstract away from enough database concepts to be suitable for casual users [9]. Another example of a business-oriented database system that is slightly more user-friendly is Zoho Creator [6]. It is very powerful and even includes a custom scripting feature. But again, it does not walk people through the creation of a database slowly enough, and it is priced to discourage sharing (even at $45 per month, one can only share editing control over a database among ten users) [6].

Yet another potential competitor is Bento [2], a Mac and iPhone/iPad application by FileMaker. It is a powerful yet easy-to-use personal database, filled with templates, forms and even Query By Example. Its greatest advantage is that it the interface is clearly designed for personal and small business use (*e.g.*, CD collections, art collections, client lists). However, it has several flaws that the proposed system would improve on. First, Bento's only platforms are the Mac and iPhone/iPad, which means both that Windows and Linux users cannot use it, and since it is not on a web platform, sharing is severely limited (Bento 3 has just added the ability to share databases on up to five computers on a Local Area Network). Also, it does not provide extra features, such as data visualizations of locations and time [2].

Thus, there is no application that addresses the problem of allowing casual users to create and share databases quickly and easily. The current solutions are either constrained to a platform or targeted to business and experienced users. The vision of this project, as a result, is to introduce people to the idea of a relational database, but most importantly, abstract away from the technical details. One should not need to understand SQL queries or relational algebra to directly leverage their power. Hence, the goals of this project are to be:

1. Cross-platform
2. General for many different types of data
3. For casual users with no prior experience (*i.e.*, personal)
4. Have an easy-to-use UI
5. Allow for users to dynamically share data with other users

3. SYSTEM MODEL

3.1 Developer Model

deebee consists of five "layers" as seen in Figure 3. The most important layer is the "deebee" layer, which is the backend application written in PHP. It is a liaison between the CodeIgniter web framework (to be further detailed in Section 4) and the web interface that end users will interact with. The underlying server technologies (PHP, MySQL and Apache) lie underneath the CodeIgniter framework, thus making CodeIgniter the liaison between the *deebee* system and the SQL queries for the backend database. This way, the interface layer should be not be affected by significant changes in how the system interacts with the backend database. In the future, an Application Programming Interface (API) will be created that will sit on top of the Main System next

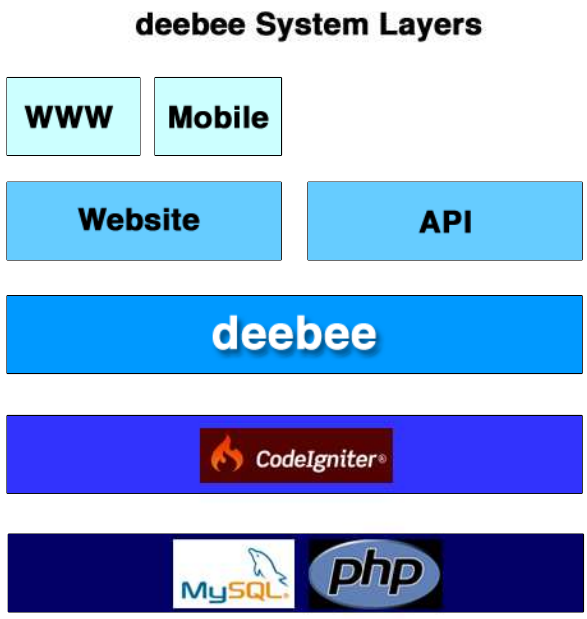

Figure 3: System Layers

to the interface and will provide developers an outlet to the database on the bottom. Similarly, it will be helpful to implement a portion of the website to be optimized for mobile viewing. This would allow users to use *deebee* in mobile settings, expanding the reach of relational databases in average users' lives.

3.2 User Model

Figure 4: Create Share Access Model

In response to the flaws examined in related systems and competitors, *deebee* will focus on a different target audience as well as a different type of interface. The most crucial part of the project is to abstract away important database concepts (*e.g.*, relations, relational algebra, SQL queries) so that casual end users can easily use the system. Therefore, the system will allow the user to perform three easy steps: create, share and access (see Figure 4). First, the user will be able to **create** tables and relationships to store his or her data. The system will allow users to do this by building fields and then inserting records as one is accustomed to in a spreadsheet.

Next, the user will be able to **share** the data with a select group of individuals, who will be able to see the respective table in their "Tables" section under "Tables You Have Access To." Alternatively, the user can set the database to be viewable (and possibly even editable) by the public. Granular privacy settings will allow the user to customize which users have read or write access.

Finally, the user (and the invited users) will be able to **access** the data using Query By Example, much like they would in iTunes. For example, the user can build a query

(called a "smart playlist" in iTunes) by setting that the "Name" field contain "John" and the address contain the string "Main St." Thus, the user will be able to leverage some of the power of SQL queries and relational algebra or calculus without any knowledge of the theory behind these queries. One can see a mockup of a QBE UI in Figure 1 and the implementation in Figure 8. Note that the system will automatically logically AND all the individual field searches. Additionally, wildcards will automatically be appended to both sides of search strings. For instance, a search for "main" in the "Address" field will automatically display records that contain "123 Main St." in the "Address" field.

Figure 5: Tables

Although this user model was originally intended to be explicitly part of the user interface, the sections that are exposed to the user are to "Edit," "View," and "Share" tables. The reason that this user model was not explicitly implemented as part of the UI was because it would be too constricting due to its very linear nature. Thus, users can freely move from one mode to another in the current implementation. Users can access both their own tables and tables which have been shared with them in the "Tables" section (see Figure 5).

In the "Edit" mode (see Figure 6), users are able to create and delete tables as well as fields (*i.e.*, columns). In the "View" mode (see Figure 7), they are able to view, edit, add and delete entries. In addition, the QBE search is located in the "View" mode (see Figure 8). Note that the Uniform Resource Locator (URL) of search queries has been specifically designed to be easy-to-read and use by implementing key-value pairs (field names followed by a forward slash followed by the value). For example, a user's search (*e.g.*, names containing "red" and addresses containing "main") would yield the following URL ending: "tables/view/12/Name/red/ Address/main."

Next, in the "Share" mode (see Figure 9), users can select the aforementioned privacy settings: view and edit permissions. One simply has to select "Block" to remove the user from this privacy list.

4. SYSTEM IMPLEMENTATION

First, the backend of the system runs on a LAMP setup: Linux Apache MySQL PHP. This set of applications was chosen because it is the industry standard for web applications of this kind. In addition, because of its open and prevalent nature, we will be able to continue leveraging third

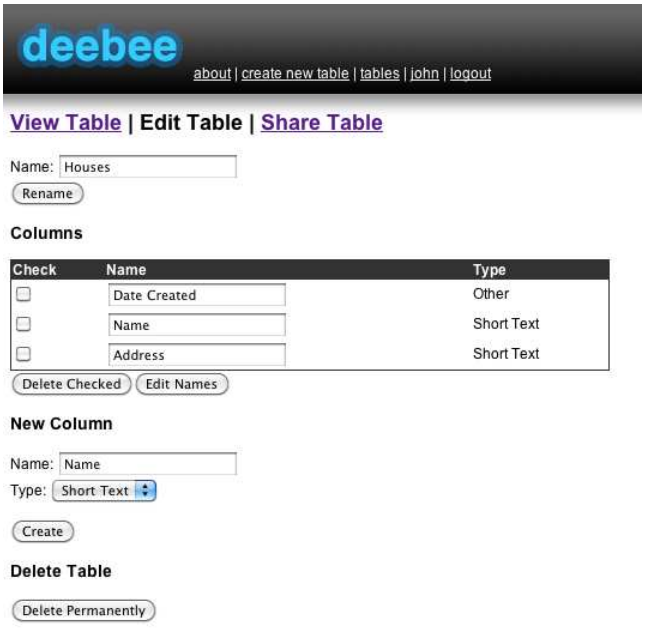

Figure 6: Edit Mode

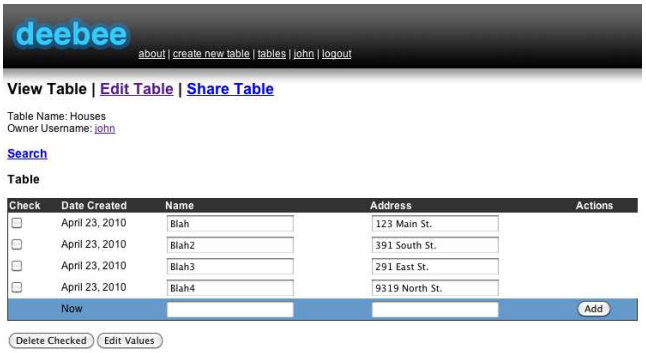

Figure 7: View Mode

party libraries. Additionally, this setup is being version controlled by Subversion.

In order to implement a more robust search engine, we used both inverted indices and Porter's Stemming Algorithm, an algorithm for suffix stripping by M.F. Porter [16]. Inverted indices allow for faster keyword searching because they allow for easier and quicker search calculations (*e.g.*, one can easily extend basic functionality to include relevance and partial matching) [18]. More specifically, this suffix stripping algorithm allows a user to search for "raining" and get back results that contain the word "rained." Fortunately, an openly available implementation of this algorithm in PHP is available from M.F. Porter's official website [17].

In addition, as can be seen in Figure 3, over the the base system lies the CodeIgniter framework [11]. This PHP framework was chosen because it provides a wealth of useful libraries (*e.g.*, form submissions, precautions against SQL injections), uses the Model-View-Controller (MVC) architectural pattern (see Figure 10) and has been shown to be faster and more responsive than other PHP frameworks [19]. The MVC design pattern was implemented because it al-

Figure 8: QBE Search

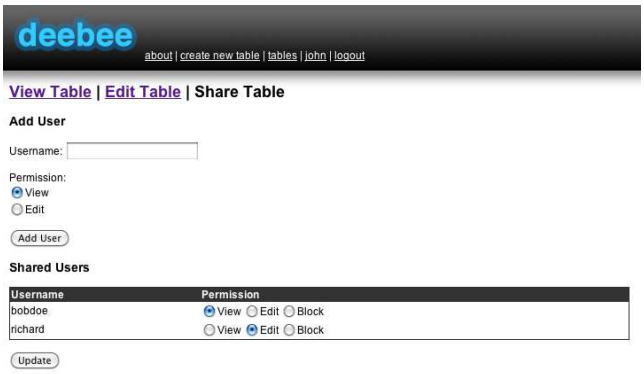

Figure 9: Share

lows for easily scalable modifications to data retrieval (the "model"), the output (the "view") and the interaction between the two (the "controller"). The decoupling of the interface from the logic will also allow easier deployment to targets other than the desktop browser in the future, such as the mobile browser and the API. Additionally, the models themselves become inherently modular, thus aiding in easy modification and expansion.

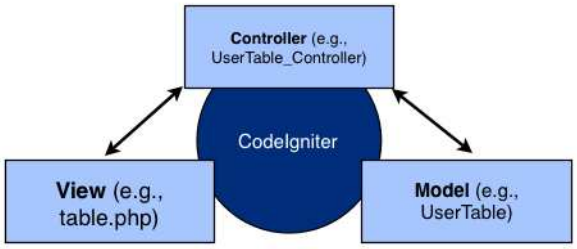

Figure 10: Model-View-Controller

Moreover, part of the interface is based on jQuery, an AJAX framework [14]. AJAX allows the developers of a system to create flexible and dynamic user interfaces (where appropriate) to minimize page refreshes and create a more intuitive interface experience for the user. Moreover, jQuery provides for easy-to-implement effects, such as a search area that slides up and down at the toggle of a button. This creates for a smoother and more natural user experience.

Clearly, an important part of the project is how the *deebee* layer accesses the backend database. The implemented approach is to translate users' tables into MySQL tables in a simple one-to-one mapping. That is, a user's table will be a real table in the SQL database. As users update and delete their tables, commands to the CodeIgniter database library are called, which subsequently call SQL commands such as "ALTER TABLE" and "DROP TABLE" to the backend database. Thus, in following the aforementioned MVC pattern, when a user clicks on "Create New Table" in the view, the controller gathers the inputs (*e.g.*, name) and calls the UserTable model. Although there is a one-to-one mapping of UserTables to SQL tables, because the developers of MySQL note that there is a performance disadvantage to having too many tables in the same database, the tables are split into different databases by alphabetical order of the table's name [15].

5. RESULTS

The most important criteria for the success of the project is clearly how easy and intuitive the interface is. The goal of this project was not to invent a new paradigm of database management; instead, it was to abstract away these important database concepts so that casual users can use them. Essentially, this is an optimization between ease of use and how powerful and flexible the system is.

The possible use cases for this project are numerous. For example, one could use *deebee* to keep track of bills, to-do lists, movies watched, wish lists, athletic and fitness-related progress and more. At the same time, there are many professional use cases as well. For instance, real estate agents and prospective home buyers can use this project to keep track of the houses that they find interesting. Imagine a table with fields like "Address" and "Comments." The real estate agent can give "edit" permission to their clients and then add new records to the table whenever a new house appears on the market. Likewise, the clients can add their thoughts about the house to the "Comments" field.

Other use cases include keeping track of collectable items (both obtained and on a wish list), bills, clients and more. Small business managers could use *deebee*, for example, to keep track of their suppliers. The managers could share these tables with salesmen and a new channel of dynamically-updated communication has been created. And since *deebee* is built on top of a web platform and uses open standards (like JavaScript), people can access the tables from almost any Internet-enabled device (*e.g.*, desktop, laptop, smartphone, tablet).

6. FUTURE WORK

Now that the fundamental technology for this project has been implemented, there are many more features that one could add to this website in the future.

For example, it would be useful to have RSS feeds for each user and database so that people can easily keep track of the data they care about. Next, in order to make the

sign up process easier and more connected with the web (since the "share" part of the project is essentially a "social network"), the project could incorporate Facebook Connect and/or OpenID into the signup process.

There are also a lot of features one could implement related to data visualizations. For instance, one can have a special "Location" field in a table and then embed and populate a Google Map into the page. Similarly, one could use the SIMILE Timeline Widget (JavaScript-based timeline) [22] to populate a dynamic timeline full of the table's records.

Additionally, it would be very helpful to have a mobile version of the site, so that users can access their data anywhere their phone can access the Internet. An additional feature that could be implemented would be to build an API so that developers can tap this wealth of data. As a result, third parties, such as various to-do list interface developers, could use the same source of personal data to populate their software. Just as in the MVC design pattern that was used to build this project, this decoupling of data from interface development would allow for a broad range of creative solutions, all synchronized and personal to you. Clearly, very carefully constructed privacy controls would also have to be created for this API in order to alleviate privacy concerns.

Also, as with any website that requires membership, interoperability will also be important. First, the system could import and export standard files such as Excel spreadsheets (.xls and .xlsx), XML files and Comma Separated Value (CSV) text files. As can be seen in [7], research in how to smartly convert Excel tables into a relational database has already been published. That is, the process will decide how to smartly get rid of unnecessary redundancies often prevalent when a user tries to use a spreadsheet application as a database. This process is based on the theory of data normalization and functional dependencies, concepts from the theoretical study of databases that describes a certain type of relationship between data in a relational database.

Finally, the system needs to be optimized to allow for scalability if the number of users in the system should exponentially grow.

7. CONCLUSION

deebee was designed to give casual users the power and flexibility of relational databases, while abstracting away from all technical considerations. Thus, a casual user, with no experience in computer science or mathematics, could sit down, and using any platform that has access to the web, start creating tables that they can then dynamically share with family, friends and coworkers. By using the concept of QBE, users can build powerful queries easily. And by being built on top of the CodeIgniter PHP framework, which implements the MVC design pattern, this project also ensures to be scalable and expandable in the future.

8. REFERENCES

[1] Microsoft Access. `http://office.microsoft.com/en-us/access/default.aspx`.

[2] FileMaker Bento. `http://www.filemaker.com/bento/`.

[3] R. G. G. Cattell. An entity-based database user interface. In *SIGMOD '80: Proceedings of the 1980 ACM SIGMOD international conference on Management of data*, pages 144–150, New York, NY, USA, 1980. ACM.

[4] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[5] E. F. Codd. Access to relational data bases for a casual user. *SIGART Bull.*, (61):31–32, 1977.

[6] Zoho Creator. `http://creator.zoho.com/`.

[7] Jácome Cunha, João Saraiva, and Joost Visser. From spreadsheets to relational databases and back. In *PEPM '09: Proceedings of the 2009 ACM SIGPLAN workshop on Partial evaluation and program manipulation*, pages 179–188, New York, NY, USA, 2009. ACM.

[8] Steven S. Curl, Lorne Olfman, and John W. Satzinger. An investigation of the roles of individual differences and user interface on database usability. *SIGMIS Database*, 29(1):50–65, 1997.

[9] Dabble DB. `http://dabbledb.com/`.

[10] Google Docs. `http://docs.google.com/`.

[11] EllisLab. Codeigniter. `http://codeigniter.com/`.

[12] Microsoft Excel. `http://office.microsoft.com/en-us/excel/default.aspx`.

[13] Apple iTunes. `http://www.apple.com/itunes/what-is/`.

[14] The jQuery Project. jquery. `http://www.jquery.com/`.

[15] MySQL. Disadvantages of creating many tables in the same database. `http://dev.mysql.com/doc/refman/4.1/en/creating-many-tables.html`.

[16] M. F. Porter. An algorithm for suffix stripping. pages 313–316, 1997.

[17] M.F. Porter. Porter's stemming algorithm. `http://tartarus.org/ martin/PorterStemmer/`.

[18] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill, 3 edition, 2003.

[19] SellersRank.com. Php framework benchmarks. `http://www.sellersrank.com/web-frameworks-benchmarking-results/`.

[20] Peter Pin shan Chen. The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1:9–36, 1976.

[21] John C. Thomas and John D. Gould. A psychological study of query by example. In *AFIPS '75: Proceedings of the May 19-22, 1975, national computer conference and exposition*, pages 439–445, New York, NY, USA, 1975. ACM.

[22] SIMILE Widgets. `http://www.simile-widgets.org/`.

[23] Moshé M. Zloof. Query by example. In *AFIPS '75: Proceedings of the May 19-22, 1975, national computer conference and exposition*, pages 431–438, New York, NY, USA, 1975. ACM.