Entity-Relationship Database User Interfaces

T. R. Rogers, R. G. G. Cattell

Information Management Group
Sun Microsystems, Incorporated
Mountain View, California

Abstract

We report on experience with database user interfaces that are entity-relationship oriented, rather than relation-oriented, and provide a new level of ease-of-use for information management. Our goal is to allow technical workers with little or no knowledge of database systems, query languages, or relational terminology to use databases to solve simple information management problems. Our tools also provide new capabilities for expert users, such as database browsing using a mouse and database display using bitmap graphics.

Keywords: Entity-Relationship, Ease-of-Use

1. Introduction

Relation-oriented user interfaces allow the definition and manipulation of tables. The default interfaces deal with a single table at a time; the relationship between the data in different tables is in the mind of the user instead of being supported by further semantics and primitives in the database system or its user interface.

By contrast, an entity-relationship user interface supports the definition and manipulation of entities, which may be instantiated as many records from many tables, and relationships, which are instantiated as one or many records from one table. Our model supports entities and relationships similar to those defined in the entity-relationship data model [Chen 76]. An *entity* table is defined as a table with a key. A key consists of a set of fields guaranteed to be unique over the set of records in the table. An entity is a record in an entity table. By constrast, a *relationship* table does not have a key. A *reference* is field of a table (either an entity or relationship table) whose values are keys of another (entity) table. That is, a reference logically "points" to entities; a reference is sometimes called a *foreign key*, and their implementation *referential integrity*. References are the glue that put together many records that compose one logical entity.

Entity-relationship database user interfaces provide easy-to-use database access for novice users as well as advanced capabilities for experienced users. The interfaces take advantage of the bit-mapped display and mouse capabilities available on advanced workstations. Instead of requiring a high-level query language or program to access data, our user interfaces allow a database to be accessed by using the mouse alone. Users can

---

Sun Workstation® is a registered trademark of Sun Microsystems, Inc.
UNIX™ is a trademark of AT&T Bell Laboratories.
UNIFY® is a registered trademark of Unify Corporation.
SQL™ is a trademark of International Business Machines Corporation.

"browse" from one entity in the database to another. Edits can be made to an entity spanning many tables. Normal relation-oriented query capabilities such as SQL are also available; the entity-relationship interface complements these relation-oriented interface capabilities.

In this paper, we first discuss the underlying database system that is required to support entity-relationship interfaces. We then describe the architecture and operation of two new tools that were made possible by the underlying entity-relationship platform.

2. Database System

We feel that a database system must provide *at least* the following four classes of features in order to support entity-relationship capabilities in a production engineering environment:

- Data model independent capabilities
- Relational capabilites
- Entity and relationship definition primitives
- Entity and relationship manipulation primitives and performance

We discuss each in the following sections.

First, an entity-relationship database system must support data model independent capabilities such as concurrency control, report generation, forms-based data entry and data modification, large amounts of persistent data, and transactions for recovery.

Next, the system has to support all relational capabilities since these provide the necessary simplicity and power for many types of database activity. We do not want to lose or trade such important capabilites for entity-relationship features. The system must support a high-level query language such as SQL [Cham 76], interactively and programmatically.

Entities can be defined using normal relational database DDL semantics without providing any special new semantics. A possible implementation for these new semantics would be to provide several new tables, including one for the definition of key fields and another to contain reference field information. However, each database client then has to build a set of functions to provide access to and modification of these extra tables. In addition, these add-ons can be cumbersome and do not meet our interactive performance needs.

Thus, we require the DBMS be extended to support entity definition primitives. The primitives can be provided in a simple yet practical manner by extending the relational model in two ways. First we provide a way to specify a key for a table. In this manner we can identify unique entities in our database. Second, we provide a reference primitive that allows linking a field in one table to the key field in another table. This is similar to other approaches [Lorie 83] [Astrahan 76]. An entity consists of a record from an entity table plus records from all of the other tables in the database that reference the entity record.

The last feature class for entity-relationship operations involves capabilities for entity

manipulation with acceptable interactive response time. We would like to emphasize that *it is not enough for the system to support programmatic access using an embedded query language*. Many database systems have fairly impressive complex query response times. The system needs a programmatic interface that can provide very high performance for operations on entities, either by parsing and optimizing SQL at compile-time, or by providing lower-level (typically access method) scan capability. Many relational systems support embedded query languages, but almost no relational systems support the performance we require for entity-relationship user interfaces [Rubenstein 87] [Learmont 87].

Our performance requirements have led us to use an access method different from the familiar database system methods such as B-trees, hashing, ISAM, or sequential access: we use links to implement reference fields. Logical connections between records in different tables are implemented by real physical pointers in the older hierarchical and network systems, or by unique record id as in the RSS component of System R [Astrahan 80]. In essence, links perform the function of a pre-join. Approaches to pre-joining records have been along the lines of logically pre-joining by building pre-join indexes or physically pre-joining by doing the join on update and physically clustering the related records. The latter method really is not a solution to the performance problem since the number of ways in which records may be joined is very large and unnacceptable burdens would be placed upon the system in maintaining pre-joined tables. Furthermore, clustering of records can only be done according to one criteria, limiting the usefulness of this approach. B-trees could be used to implement pre-join indexes, thereby retaining the advantages of a data independent approach to entity manipulation. However, a large number of indexes would be required for all of the different join combinations, and the amount of storage used would be greater than that required for a links-based approach.

We used SunUNIFY [Sun 86c] as the underlying base for the database system because it satisfied all four requirements for supporting entity-relationship user interfaces. Fast interactive response for simple entity manipulation is possible due to the implementation of a low-level programmatic interface which includes hash indexes on keyed fields in addition to scans on b-trees and pointer chains. Furthermore, the database system supports the definition of combined fields, where several fields can be grouped together and referred to by one name. This capability enhances the power of the referencing scheme.

We now discuss the architecture and operation of two new entity-relationship user interfaces made possible by the SunUNIFY platform. The first tool is schemadesign, used to define the database schema. The other tool is databrowse, used for editing and browsing data in the database.

3. Schemadesign

Schemadesign is a window-based tool with which users can graphically create and display the database schema by using an entity-relationship diagram. Schemadesign's graphical representation is much easier to understand than the linear listing of tables in conventional relational systems. It is most commonly used by database administrators to define databases; however, it is also a very useful tool for viewing the schema for an existing database. The tool contains a message window for messages and a command subwindow for typing in information or clicking commands with the mouse. The editing subwindow is used for mouse and keyboard entry operations.

In Figure 1, two types of boxes are shown in the editor subwindow: entity (square) boxes and oblong (relationship) boxes. Entity boxes are used to represent tables with keys, and relationship boxes denote tables that have no key. Typically, relationship tables reference at least two entities, although this is not a requirement. Relationship tables can be transformed into entity tables by defining a key for them.

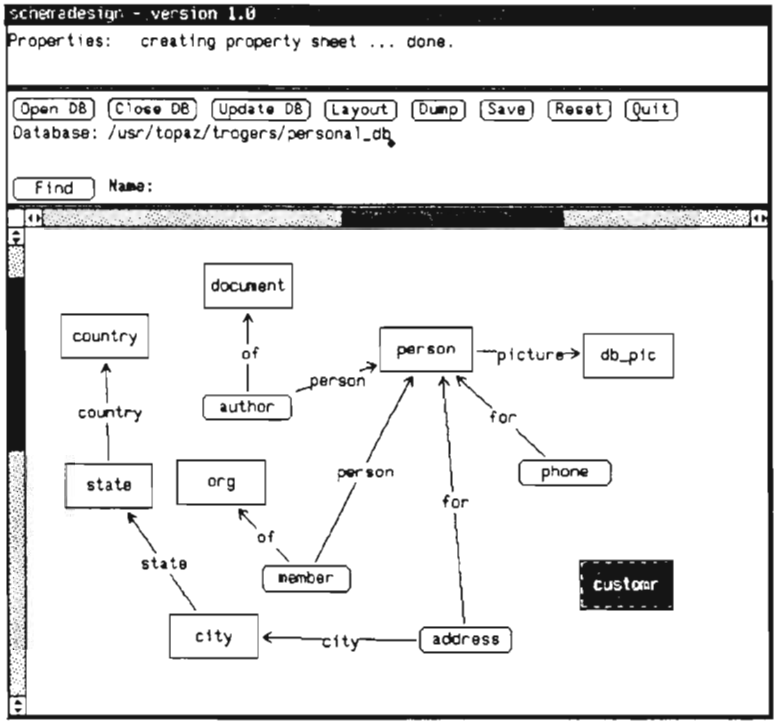

Figure 1: schemadesign with customr table selected

Constraint-checking through the use of a built-in referential integrity [Codd 81] feature provides a level of data integrity checking not found in most database management systems. The constraint checking is provided by the database system itself; it is not provided as an add-on by schemadesign. The arrows drawn between the boxes represent graphically what would be referred to as foreign keys in the relational model, although most relational database management systems do not implement them. The ability to perform integrity checks when records are inserted, deleted, or modified is recognized as a key entity-relationship enhancement to a database system [Schwarz 86]. In fact, such extensions are being considered for for inclusion in the ANSI SQL2 standard [ANSI 87]. As an example, the referential integrity facility would prevent the deletion of an organization referenced by its members. Likewise, it would not be possible to directly add a person as a member of an organization if the organization did not exist in the database.

There are three main steps to using schemadesign:

Identify 'entities'

The user identifies entities (objects) in the real world that he wants to model in the database system. Examples are people, cities, and documents.

Make entity tables for them

An entity table is created for each real-world object. Creating a table using schemadesign is a simple process of menu/mouse selection and table naming. A "property sheet" provides field name entry and field type selection and definition, as shown in Figure 2. Data types for fields are defined either by selecting the type with the mouse by clicking on the circle made from arrows icon or by selecting the type from a pull-down menu. For example, in Figure 2 we illustrate using a menu to select the string data type for the logon_name field.

Identify features

A feature can be thought of as more information about an entity. Examples would be the age of a person or a list of the organizations to which the person belongs. There are a few simple rules to follow when identifying features of entities. Following these guidelines encourages normalization ([Date 85] [Codd 72]) of the data without any knowledge of normalization theory on the part of the user. The user makes either fields or relationship tables for the features depending on whether they are 1-to-1 or many-to-1 with the corresponding entity, respectively. For example, since a person has only one date of birth, date of birth is said to be "1-to-1" with the person entity. Therefore, we model this relationship in our schema by making the date of birth feature a field in the entity table for person. Similarly, since an employee can have many phones (such as one at work, one in the car, and one at home), i.e. phone numbers share a "many-to-1" relationship with a person, we capture this information in the schema with the relationship table phone. It has a field in it that references the people entity. In this manner, we can add as many phones for the same person as we like. Schemadesign lets the user create many-to-1 relationships simply by selecting the reference table with the mouse, selecting connect from a menu, and drawing a line to the referenced (entity) table. The required reference field in the referencing table is automatically generated. It is also very simple to model many-to-many relationships by composing them out of 1-to-many connections: drawing two or more connections from a reference table to entity tables.

Schemadesign helps the user understand the structure of the schema by not allowing the creation of a database from an incorrectly defined schema. Although it is possible to automatically generate reference fields, the tables do not become fully defined until the entity tables have key fields defined for them. In Figure 1, we notice that the customr table is surrounded by a dotted box. This indicates that the customr table is not completely defined. In this particular case, it is not defined because no key field has been specified.

In addition to schema definition, schemadesign allows easy schema modification. Users delete or rename tables and fields by selecting commands from a pop-up menu, and they can customize the graphical layout of lines and icons. Scrollbars can be used to move about in large schemas, and a find facility is available to locate tables with particular names. B-tree and password properties can also be changed using schemadesign. After modification, schemadesign automatically reconfigures the existing database as necessary to fit the new design.

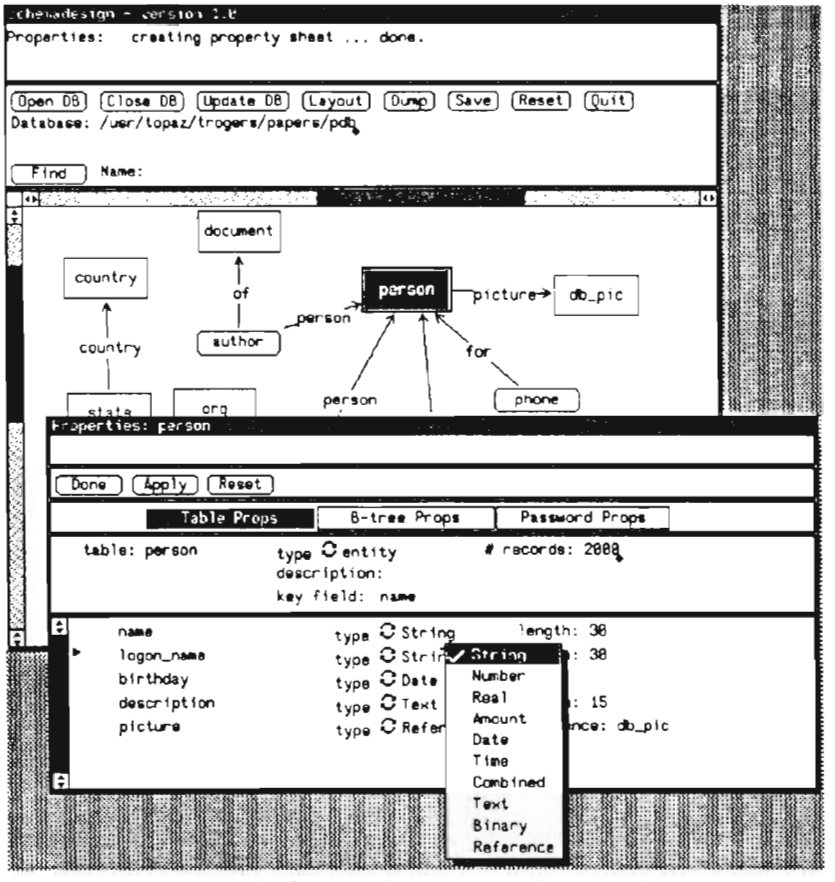

Figure 2: schemadesign with property sheet and menu

We have found that a graphical display is a useful and powerful tool for both the neophyte and expert users. We are implementing further extensions for the tool such as a "verbose mode" to allow seeing all fields for a table at once, and a group move facility to allowing moving a number of tables and references as one unit.

4. Databrowse

Databrowse is a window-based program that allows viewing and editing of logical entities instead of the single records found in conventional relational databases. It is built upon the standard SunView window system [Sun 86d] and the ERIC SunUNIFY programmatic interface [Sun 86b]. It can also be used to browse and edit information in the schema, but not change the schema itself.

Databrowse has a number of subwindows. From top to bottom they are a message subwindow for messages; a command subwindow for selecting entities and relationships, changing between databases anywhere in the computer network, and other operations; an editor-browser subwindow for viewing and editing data; a text subwindow for displaying and editing text fields; and a picture display subwindow with a picture editing subwindow. The text subwindow is only present if the database contains text fields, and the picture subwindows are only present if the database contains picture fields.

Data can be displayed in the familiar tabular format as shown in Figure 3. The person table was selected simply by using the mouse and a menu which are also displayed in the figure. Table names and entity names appear in **bold** face. Each field name (label) has a

colon appended to it. Most data appears as ASCII text, but picture fields appear as camera icons. The user can scroll back and forth through the table using the scrollbars. Not shown is a property sheet that allows the user to tailor data display characteristics.

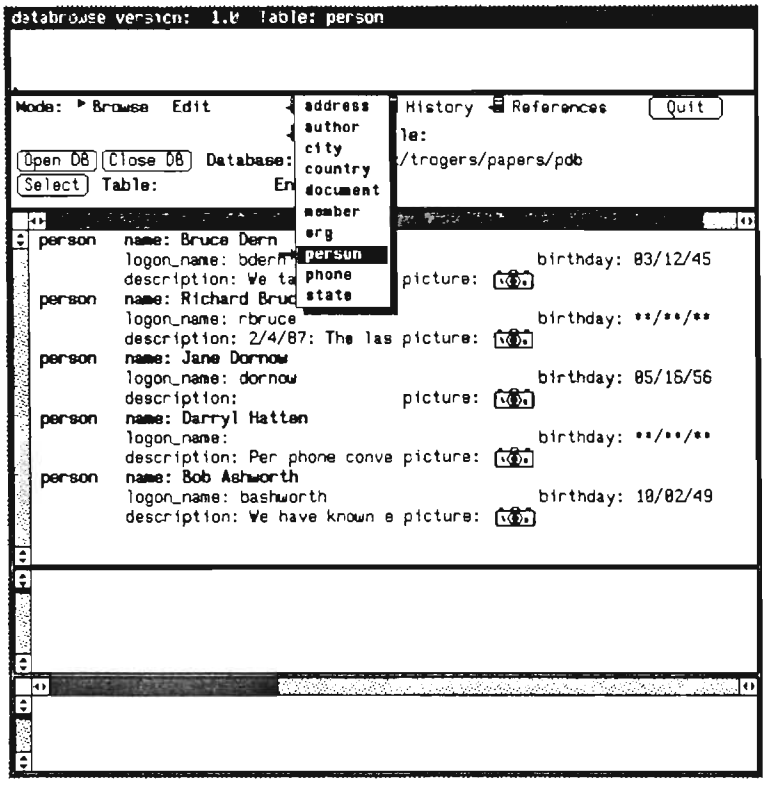

Figure 3: databrowse in table mode displaying person records

In addition to looking at single tables, databrowse lets the user view entities. The data in an entity consists of information from a number of records in potentially many different tables in the database. The user can click on any bold face data, with the exception of table names, to display an entity. For instance, if the user clicks on either "Bob Ashworth" or the field label "name:" for Bob Ashworth in Figure 3, the editor-browser subwindow is erased and the new entity information is displayed as shown in Figure 4. The entity record itself is above the dotted line, and the reference records are listed below the dotted line. The entity name which is a field in each referencing record is not shown, to reduce the display of redundant data. There are several ways to scroll to the other reference records for this person that are not shown in the figure.

Databrowse can be used to store pictures either in the database itself, or pictures can be stored in files, with simply the file name in the database. The underlying database system supports the binary data type, but it is only used by databrowse to store pictures. User-written programs can do whatever they want with the binary fields, however, including using them for storing binary program files, images, or other types of unformatted data. In Figure 3 the person's picture is displayed by clicking the camera icon or the "picture:" field label. The selected camera remains in reverse video.

Note that the instantiated entity is *not* one record, and it is *not* possible to get the necessary display information using a relational view. It would be difficult to get the same display information using a purely relational system. First, a simple relational view would be cumbersome because the targetlist of the view would have to be needlessly

complicated. Second, a join would not work in any case where there was no matching record in any one of the join tables, unless the outer join syntax were supported [Date 83]. Third, retrieving the information would probably have to be done as a series of two-table queries which would take too long. Using the underlying capabilities of ERIC, the new display of an entity is obtained directly from the database in a fraction of a second, without read-ahead or caching in the program.

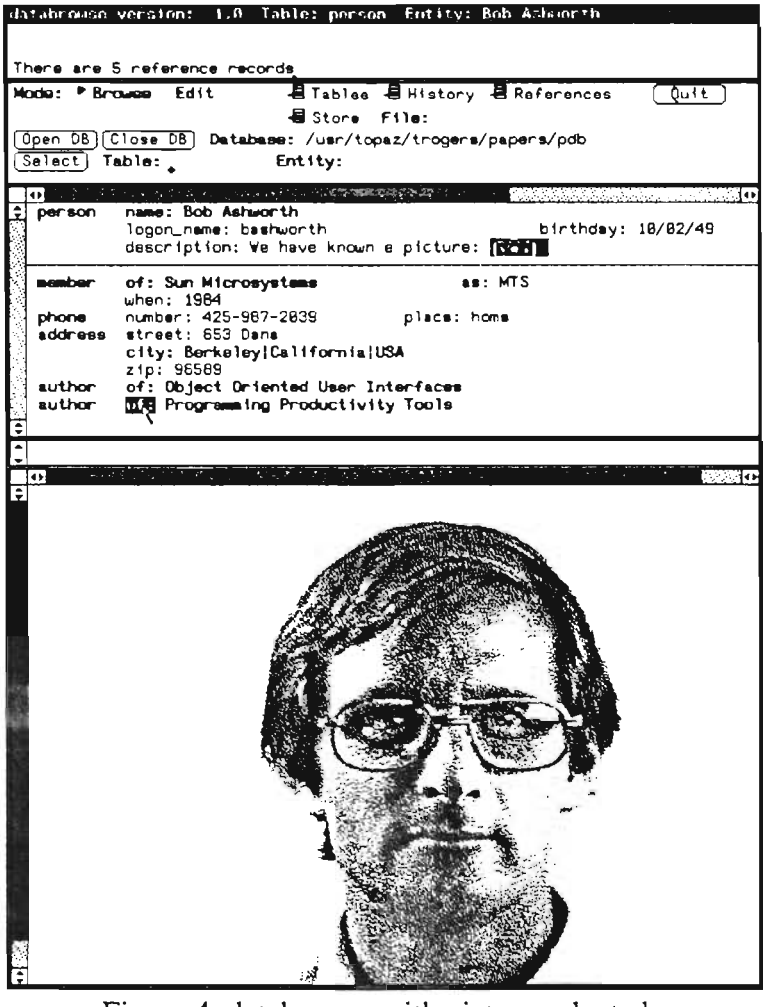

Figure 4: databrowse with picture selected

After viewing the person entity, the user may want to see more information about one of the papers that the person authored. This is done simply by clicking on the name of the paper or the field label for the particular paper. In Figure 4, we show the user selecting the "of:" field label. Figure 5 shows the new entity (the paper) being displayed. There are several new features shown in this figure. Foremost is the fact that the user is now in edit mode, and can edit any data on the screen. In this mode, labels are used for browsing and data, when clicked, can be edited. Delete, "D", and insert, "I", boxes are shown. It is possible to use these to delete reference records and add reference records. These operations, of course, are subject to the referential integrity constraints applied by the underlying database system.

Note that no forms design is necessary for accessing an entity. Databrowse essentially creates a default form. In editing mode, an empty record is shown for each reference

record type that has no actual records which reference the current entity. Included is a reference menu item which shows a menu of referencing from which a specific reference set can be chosen. The default form has most of the desired information about an entity, which is what the user wants most of the time.

Databrowse can also be used for text manipulation. Text fields are edited by the user in the text subwindow which has the full power of the normal Sun View text editor. It is possible to scan in entire pages of data using scanners, or an ASCII text file can be created outside of databrowse with the user's favorite text editor. The file can then be read into the text subwindow for editing. When the editing is completed, the text can be saved to the database or a file. In Figure 5 we show the contents of the description field for the document. We could just as easily have stored the entire document in the database. As with picture fields, the selected field is highlighted using reverse video.

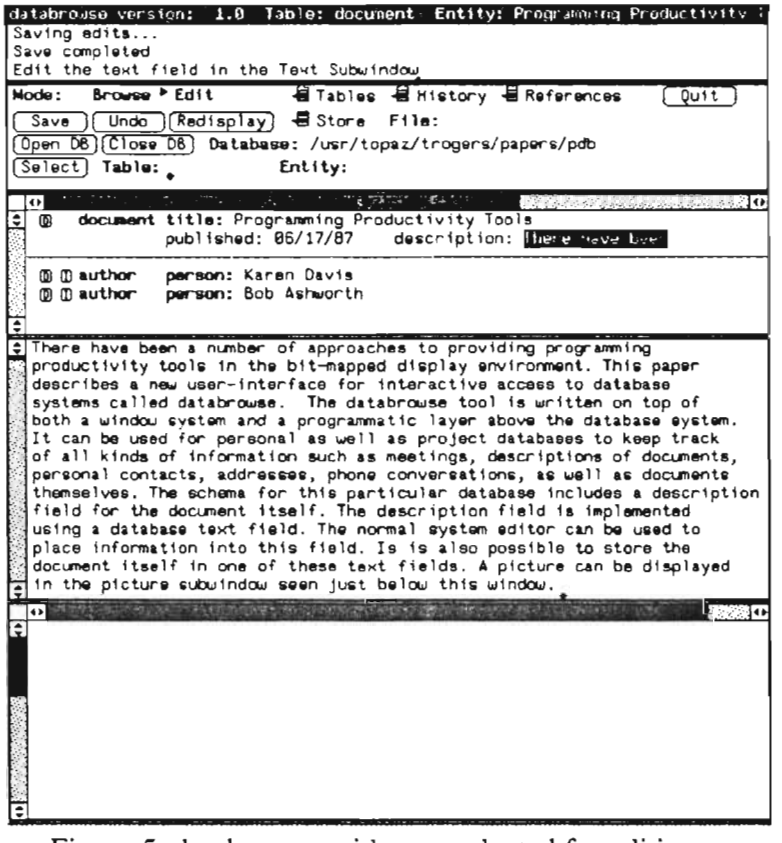

Figure 5: databrowse with text selected for editing

5. Related Work

The idea of an entity-relationship diagram editor such as schemadesign is not new -- almost anyone who read the original [Chen 76] probably thought of it. Surprisingly, there have been few actual commercial implementations until recently [Chen 87] [CCA 87] [ETH 87]. There have also been some research prototypes in this direction [Chan 80] [Gold 85].

We have presented experience with the implementation and use of such a tool, and

demonstrate that such a tool can easily be built on a relational system. In our experience, our users have always tended to draw entity-relationship diagrams on paper; now this task has been automated.

The databrowse tool is unique in both the research and development worlds, to our knowledge, except for the author's previous work [Cattell 83]. The two features that make databrowse unique are its entity-centric orientation, possible only through custom form design in other systems, and its browsing capability to move between database entities. The closest comparable work is probably [Motro 86], who reports on navigating entity-relationship connections by typing natural-language-like commands.

Other work on so-called database "browsers" has actually consisted of *scrolling* through records in a single table [Stonebraker 82] or of typing queries that allow a "fuzzier" specification of data [Motro 87]. Larson [Lars 86] also reports on some early similar approaches for browsing.

There is some work analogous to databrowse in the user's mode of interaction. Recently, [Delisle 86] reports work on such a system for hypertext. However, hypertext lacks the underlying database structure provided by the relational system, and thus (1) the same database of entities and relationships cannot be queried through other means, and (2) there is less uniformity of structure, so users are more likely to get confused [Mantei 82].

Unlike databrowse, some systems allow query specification and thus "browsing" from a graphical schema representation [Wong 82] [King 84] [Fogg 84] [Gold 85] [Lars 86] or from combinations of menus and type-in areas [Grap 87]. This is generally a multi-step process of moving between specification and query results or incremental query construction using the schema whereas databrowse allows the user to move directly from entity to entity based on selection of actual data values.

6. Future Work

Schemadesign and databrowse are available as a commercial product [Sun 86b]. We are looking at a number of extensions to them, to allow user-defined data types and associated procedures, control of databrowse's record display to allow custom applications, and convenience features for schemadesign. Some minor extensions can be made to databrowse to allow queries, and a more sophisticated query tool utilizing graphics is being constructed. In addition, we have built a tighter coupling between databrowse and the schema in which the portion of the schema corresponding to the current entity is graphically displayed.

We are also porting databrowse, schemadesign, and the collection of tools we build to other commercial database products. This is not a trivial task, since (1) we require extensions to the underlying relational model to incorporate new semantics, and (2) we require high interactive performance often not available at the level of SQL. However, we believe the task is feasible with most DBMSs, including some non-relational ones.

7. Summary

In summary, we have built entity-relationship user interfaces to a DBMS oriented towards ease-of-use.

Schemadesign focuses on a difficult problem in DBMS ease-of-use: designing a database. It provides a simple several-step process not requiring knowledge of relational design and normalization theory.

Databrowse provides a default form for database entities spanning many tables. In our experience this is the database "form" that is required for most applications, and thus provides a user interface with no further action on the part of the user after schema design. Furthermore, the browsing provides an easy-to-use alternative to a more complex query language for casual users. We use databrowse in our every-day work.

Both databrowse and schemadesign are also interesting to "experts". Databrowse browsing provides a fast way to get to simple pieces of data, and schemadesign diagrams are an efficient way to view database designs. Together, they implement zooming, panning, and rudimentary filtering functionality in Larson's browsing taxonomy [Lars 86].

A number of issues arose in the construction of these tools on relational databases. Very fast response time is required to give databrowse the "feel" of an interactive tool, as it executes the equivalent of many SQL queries upon most mouse selections by the user. Extensions to the data schema and functionality of the underlying database system are required to implement the semantics of schemadesign and databrowse on an underlying DBMS; we achieved this through a programmer's interface layered on top of the DBMS.

8. Acknowledgements

Mike Freedman and Bob Marti implemented schemadesign. Tom Rogers implemented databrowse. Tim Learmont, Mike Freedman, Bob Marti, Richard Berger, and Liz Chambers worked on the underlying ERIC implementation.

References

[ANSI 87]
ANSI SQL2 base document. ANSI X3H2-87-144, July 1987.

[Astrahan 76]
Astrahan, et. al: *System R: A Relational Approach to Database Management.* Transactions on Database Systems, 1, 2, ACM, June 1976.

[Astrahan 80]
Astrahan, et. al: *A History and Evaluation of System R.* IBM Technical Report RJ2843, June 12, 1980.

[CCA 87]
CCA: *DB Designer - Product literature.* Cambridge, MA.

[Cattell 83]
Cattell, R. G. G.: *Design and Implementation of a Relationship-Entity-Datum Data Model.* Xerox PARC Technical Report CSL-83-4, April 1983, see Section 7.

[Cham 76]
Chamberlin, Astrahan, et. al.: *SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control.* IBM Journal of Research and Development. November, 1976.

[Chan 80]
Chan, E. and Lochovsky, F.: *A Graphical Database Design Aid Using the Entity-Relationship Model.* In Entity-Relationship Approach to Systems Analysis and Design, North-Holland, 1980, pp 295-310.

[Chen 76]
Chen, P.: *An Entity-Relationship Model -- Towards a Unififed View of Data.* ACM Transactions on Database Systems, 1, 1. January 1976.

[Chen 87]
Chen & Associates, Inc.: *ER designer - Product literature.* Baton Rouge, LA.

[Codd 72]
Codd, E.: *Further Normalization of the Data Base Relational Model.* Data Base Systems, Courant Computer Science Symposia Series, Vol. 6. Prentice-Hall. 1972.

[Codd 81]
Codd, E.: *Referential Integrity.* Proceedings of the Seventh International Conference on Very Large Data Bases, Cannes, France, September, 1981.

[Date 81]
Date, C.J.: *Referential Integrity.* Proc. 7th Int. Conf. on Very Large Databases, 1981, pp. 2-12.

[Date 83]
Date, C.J.: *The Outer Join.* IBM Technical Report TR 03.181. January 1982.

[Date 85]
Date, C.: An Introduction to Database Systems. Volume 1. Fourth Edition. Addison-Wesley. September 1985.

[Delisle 86]
Delisle, N. and Schwartz, M.: *Neptune: a Hypertext System for CAD Applications.* ACM SIGMOD Proceedings, 1986.

[ETH 87]
Institut fur Informatik, ETH: *Gambit - Product literature*, Zurich, Switzerland.

[Fogg 84]
Fogg, D.: *Lessons from a "Living In a Database" Graphical Query Interface.* Proceedings ACM SIGMOD 84, pp 100-106.

[Gold 85]
Goldman, K.J., Goldman, S.A., Kanellakis, P.C., Zdonik, S.B.,: *ISIS: Interface for a Semantic Information System.* Proceedings ACM SIGMOD 85, pp. 328-342.

[Grap 87]
Soretas-Graphael: *G-BASE - Product literature.* Waltham, MA.

[King 84]
King, R.: *Sembase: a Semantic DBMS.* Proceedings 1st International Workshop on Expert Database Systems, Kiawah Island, South Carolina, October 24-27, 1984, pp.

151-171.

[Lars 86]
Larson, J.A.: *A Visual Approach to Browsing in a Database Environment.* IEEE Computer, June 86, pp. 62-71.

[Learmont 87]
Learmont, T., and Cattell, R. G. G.: *Object-Oriented Database Systems.* To appear in Springer's *Topics in Information Systems* Series.

[Lorie 83]
Lorie, R., Plouffe, W.: *Complex Objects and Their Use in Design Transactions.* Database Week: Engineering Design Applications Proceedings. San Jose, May, 1983.

[Mantei 82] Mantei, M.: *Disorientation Behavior in Person-Computer Interaction.* PhD dissertation, University of Southern California, 1982.

[Motro 86]
Motro, A.: *BAROQUE: A Browser for Relational Databases.* ACM TOOIS 4, 2, April 1986, pp 164-181.

[Motro 87]
Motro, A.: *VAGUE: A User Interface to Relational Databases that Permits Vague Queries.* CS Dept, University of Southern California.

[Rubenstein 87]
Rubenstein, W. R., Cattell, R. G. G.: *Benchmarking Simple Database Operations.* Proceedings ACM SIGMOD 1987, pp 387-394.

[Schwarz 86]
Schwarz, P, et. al.: *Extensibility in the Starburst Database System.* Proceedings of the 1986 International Workshop on Object-Oriented Database Systems, September 23-26 1986, pp. 85-92.

[Stonebraker 82]
Stonebraker, M. and Kalash, J.: *TIMBER: A Sophisticated Relation Browser.* Proceedings of the Eighth International Conference on Very Large Databases, September 1982, pp 1-10.

[Sun 86a]
Sun Microsystems, Inc.: *SunUNIFY Programmer's Manual.* Sun Microsystems, Mountain View, California, 1986.

[Sun 86b]
Sun Microsystems, Inc.: *SunSimplify Programmer's Manual.* Sun Microsystems, Mountain View, California, 1986.

[Sun 86c]
Sun Microsystems, Inc.: *SunUNIFY Reference Manual.* Sun Microsystems, Mountain View, California, 1986.

[Sun 86d]
Sun Microsystems, Inc.: *Sun View Programmer's Guide.* Sun Microsystems, Mountain View, California, 1986.

[Tuori 86]
Tuori, Martin: *A Framework for Browsing in the Relational Data Model.* PhD thesis, University of Toronto CSRG, 1986.

[Wong 82]
Wong, H.K.T., Kuo, I.: *GUIDE: Graphical User Interface for Database Exploration.* Proceedings of the Eighth International Conference on Very Large Databases, September 1982, pp 22-32.