

 **TANDEM** COMPUTERS

An Approach to End-User Application Design

Jim Gray

Technical Report 81.1
March 1981
PN87600

An Approach to End-User Application Design

Jim Gray

March 1981

Tandem Technical Report 81.1

Tandem TR 81.1

An Approach to End-User Application Design

Jim Gray
Tandem Computers Incorporated
19333 Vallco Parkway, Cupertino CA. 95014

March 1981

ABSTRACT: Soon every desk will have a computer on it. Software to do mundane things such as payroll, mail, and text processing exists and as a by-product produces vast quantities of on-line information. Many users want to manipulate this data, often in unanticipated ways. These unexpected uses cannot justify substantial programming costs. This paper argues that the relational data model and operators combined with a screen-oriented forms design and display system answers many of the needs of such users. In such a system, all data are represented in terms of records and fields. The user defines the screens (forms) he wants to see, and then specifies the mapping between fields of these screens and fields of the data base records in terms of predicates and relational operators.

Copyright © 1981 D. Reidel Publishing Company. Originally appeared as a chapter of "Data Base Management and Applications", Andrew Whinston Ed., D. Reidel Publishing, 1981. Republished by Tandem Computers Incorporated with the kind permission of D. Reidel Publishing Company.

TABLE OF CONTENTS

INTRODUCTION: The Economics of Application Programming	1
A STANDARD DATA MODEL: The Case for Relations	2
DATA MANIPULATION: Database Editors	5
FORMS: Displaying Records in Comprehensible Ways	7
AN END-USER APPROACH TO APPLICATION DESIGN	11
SUMMARY	15
ACKNOWLEDGMENTS	15
REFERENCES	16

INTRODUCTION: The Economics of Application Programming

Application programmers cost about fifty dollars per hour these days. On average, they produce about one line of code per hour (this includes time to design, code, test, document, sell and maintain). When computers were expensive, this was not a big problem. But as the price of computers goes down, the application programming problem becomes a barrier to the use of computers.

Every eight years the price of processing and memory drops by about a factor of ten. The price of transducers (printers, displays, disk arms, and tape drives) is not declining and so there continue to be some economies of scale in printing and archival storage. However the declining cost of computer hardware has made computing equipment attractive to almost all business operations.

Standard packages for payroll, accounting and billing, inventory control, text processing and many other areas have been available for many years. These packages have evolved to extremely versatile systems which can be installed with almost no application programming effort. Typically, a customer fills out a questionnaire describing his accounting practices and tax computations. These answers are used to "customize" an accounting package tailored to that particular customer's needs.

Inexpensive computer hardware along with inexpensive standard application packages result in all the operational data of the business being captured in machine readable form. These packages do the "standard" things (print the payroll, generate inventory reports,...). But these packages do not do non-standard things: if a user wants the accounting program to give him an analysis of shipping costs by order size and this analysis is not a feature of the accounting package then the user must hire an application programmer to do the analysis. More likely the user has a clerk do the report by hand.

There is yet-another source of interesting on-line data. The wire services, airlines, stock markets, phone companies, and information services (e.g. Viewdata) each "publish" information in machine readable form. Unfortunately, each of these services has its own protocols and data representation. These days one needs a different type of terminal for each service. As a consequence, there are lots of good and interesting data out there which you cannot get to from an "ordinary" computer terminal.

If these trends continue, one can assume that:

1. Computer hardware is free and everybody has it.
2. Application programming is prohibitively expensive.
3. Standard application packages capture all the operational data of a business but only present it in standard ways.
4. Interesting data are available from outside the organization in vast quantities, but the information has diverse structures and formats.

The problem is that end-users will want to examine this data without involving application programmers. The proposed solution is to offer an easy-to-use data analysis package which will allow users to examine and manipulate the operational data captured and stored by standard packages and information supplied by information vendors. This proposal requires two components:

1. An easy-to-use data analysis package.
2. A common data format which will be presented to the data analysis package by the standard packages and information suppliers.

A STANDARD DATA MODEL: The Case for Relations

The previous section argues that representing the operational data of a computer system in a standard form is essential to end-user analysis and display of the information. This section argues for the relational data model as that standard.

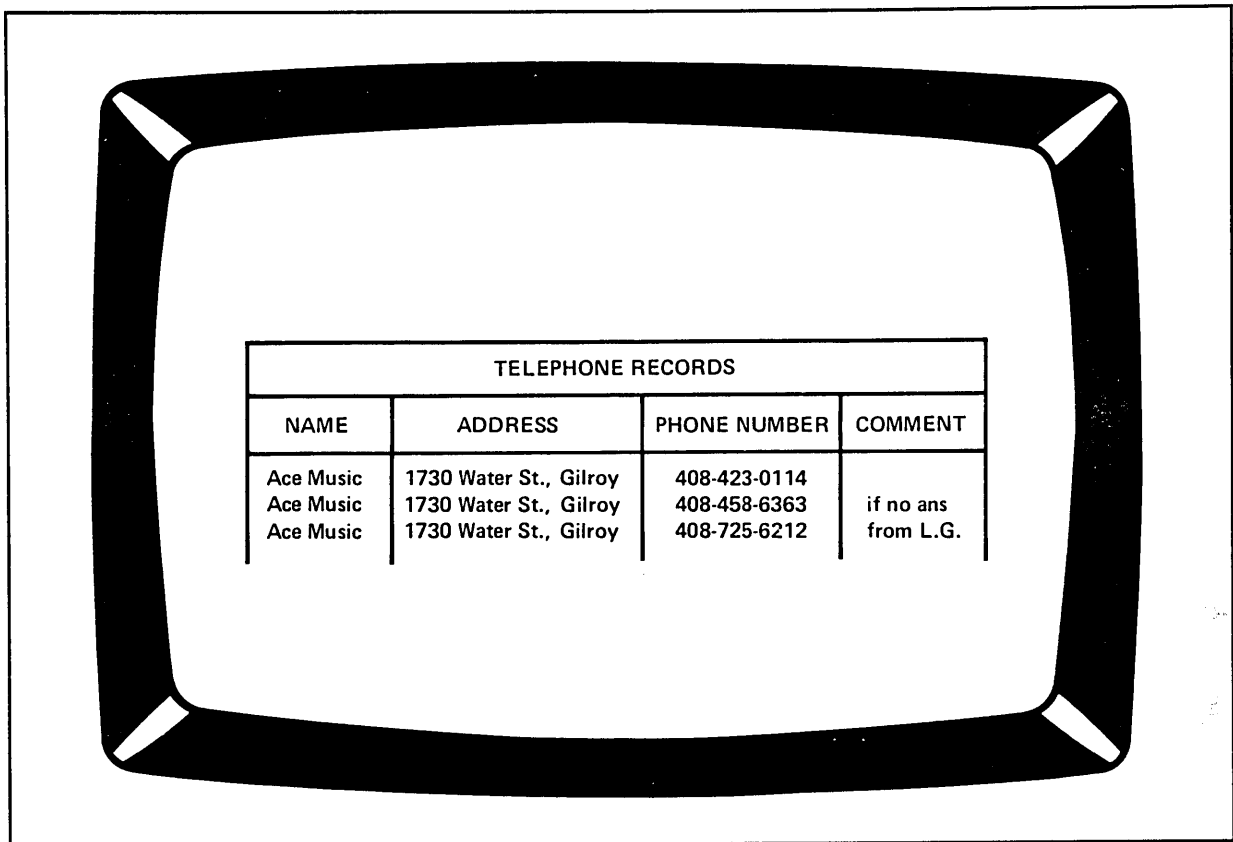
The simplest data model is the null data model. Data is represented as an unstructured sequence of characters. The problem with the null data model is that it is very difficult to write general programs to manipulate completely unstructured information. The program needs to be told what to look for in the data.

Therefore, virtually all data models support the notion of RECORD. A record is supposed to represent a fact about the world. A data base is composed of various records (facts). A data manipulation language provides operators to add, delete, alter and query the records in various ways. In such a system, each instance of a record has a type which tells how to interpret that instance. For example, an employee record and a department record have different structures and are distinguished by their record types.

Some data models treat all records as unstructured sequences of characters. This is fine for applications in which records do not have much structure: For example a text processing system may treat each line of a document as a single record. But, if records have structure, an unstructured record model suffers from the same flaws as the null data model: One cannot write general purpose programs which manipulate records unless there is some general description of the "meaning" of the record.

So again, almost all data models interpret a record as a collection of FIELDS. Each record type is declared to have a sequence of fields. An instance record of that type will assign a value to each of those fields.

A sample record type (TELEPHONE RECORDS), its fields and some record instances are displayed below:



TELEPHONE RECORDS			
NAME	ADDRESS	PHONE NUMBER	COMMENT
Ace Music	1730 Water St., Gilroy	408-423-0114	
Ace Music	1730 Water St., Gilroy	408-458-6363	if no ans
Ace Music	1730 Water St., Gilroy	408-725-6212	from L.G.

Example 1. A record type and three instances of it.

This example demonstrates some of the problems of putting "real" data into a rigid format. The phone book which looks so simple is really quite complex. Ace Music Company has three phone numbers, one with the notation "If No Answer Call" and another with the notation "From Los Gatos Telephones Call". Each of these additional entries has another phone book record for the Ace Music Company and the notations appear in the (usually null) comment fields.

On examination, the simple phone book is best described as a tabular system with occasional excursions into "unstructured" text. It is probably representative of the type of data one will get from pre-existing information sources. "Real" data is not uniform and some creativity may be required in mapping it to a uniform data model.

In addition to reducing the difficulty of writing general purpose programs to process data, the imposition of the record and field structure makes it possible for users to understand the data structure.†

Beyond this simple model of records and fields, data models proliferate in two directions:

1. They add semantics and constraints to fields.
2. They add relationships among records.

For our purposes it suffices that one can write a program which will take any of these “fancier” data models and map them into a representation in which there are records and fields and where field values are strings of characters. Tools which translate data from network and hierarchical form to a tabular form are available [4]. Although these tools are now hard to use (that is the user is expected to be a programmer), they could be made easy to use. The translation backward (from simple to fancy) is not easy.

Not only must a standard data model have a standard format for records, but it must also have a standard format for record definitions. So the standard data model must describe how a file description looks. The description defines the number of fields in the record and for each field its name, attributes and display format. This information allows the other programs to interpret the records of a tabular file.

The major virtue of a tabular representation is that we can all understand it. We can print it or put it on a screen and we can construct a language which manipulates such records in an understandable way—a way understandable to clerks who think in terms of fields and records in tables on the screen or on a piece of paper.

A simple data model is essential to a simple language to manipulate data. A simple data model is also essential to a user model of the data. The described tabular model has the minimal complexity to allow the representation of facts in a computer data base.

†This is a tabular or flat file model rather than relational model:

1. There are no domains—only character strings.
2. There is no notion of key—records are ordered in the table and have an existence independent of their values.

This is a simplified version of the SQL data model [8].

DATA MANIPULATION: Data base Editors

Given this model of files consisting of records consisting of fields, it is a straight-forward task to display the data base to a user. Such a program has many of the attributes of a text editor and so is called a data base editor.

One imagines that all records of a particular type are arranged in a table with one row per record (see Example 1). The terminal becomes a window with which one can examine this table. If there are many records, or if the records are very wide then most of the records will not be visible at one time. Operators are provided to move the window around on this table (up, down, left and right).

Users may enter new data and alter stored data by typing on the screen. Like text editors, data base editors support insert, delete and update operations.

Beyond this point, the analogy with text editors breaks down. Data base editors support a data manipulation language which allows the display of logical subsets of the data. This data manipulation language is generally a syntactic sugaring of the relational calculus with some additions (e.g., statistical analysis, pattern match, phonetic search, ...). A typical list of such relational operations is:

PROJECT:	eliminate some columns from the answer table.
SELECT:	eliminate rows which do not satisfy a predicate.
JOIN:	take the relational composition of two tables.
UNION:	concatenate two (similar) tables to make a new one.
INTERSECTION:	consider only record values which are in both tables.
SORT:	reorder the answer based on some criterion.

With these operators, one can easily examine large quantities of data looking for anomalies or trends. Such data base editors are generally available [3,5]. Experience with them has been quite favorable: untrained users are able to learn them quickly. There are several cases in which users saved enough money in the first month to pay for the whole system for several years. In other cases, the application programming backlog was cut from months to weeks because the application programmers were made so productive.

These editors are used in two modes:

1. Data is extracted from the operational system and operated on by analysts and planners.
2. New applications are done entirely on the data base editor because the application programming cost is so much lower.

In the first mode, users only read the data. They may want to use the data base editor to modify operational data but I do not think this should be allowed. The operational data is exposed as tables, but it may have complex internal structure. The data base editor cannot know or enforce this structure. For example, allowing update of operational data via a data base editor would allow the user to credit one account and not debit another. While this is very convenient, it should not be allowed. The operational data must only be manipulated using the standard packages which are auditable and which enforce the standard operating procedures of the company.

Given this prohibition on updating operational data, one might think that the data base editor should not support update at all. But that is not so. Users will have their own private data bases which they should be allowed to alter in any way they like. In particular, they will have to do updates in order to enter the data into the computer. So we must depend on the authorization system to disclose sensitive operational data only to users authorized to see it and not to allow any users to update operational data via the data base editor.

Database editors are usually packaged with a second component which does report generation. Report generators format the data into page-size units which are structured as title, body summary. In addition, the first and last pages of the report summarize the body of the report.

Report generation is symptomatic of a problem with data base editors. People do not want to see tables of numbers; they would like to see charts, graphs and maps which pictorially show statistics and trends. For example, if a user wants to find out about invoice number "34245789", he does not want to see it as a single row of a table preceded by the rows for invoices "34245781",..., "34245788". Rather he would like to see the invoice displayed as one or more screens laid out to look just like the paper invoice. Reports are a small step in that direction: they display tabular data in non-tabular formats. The next section examines the problem of data display in greater detail.

To summarize the arguments so far:

1. If data is represented in or translated to tabular form, a simple data base editor can be used by non-programmers to manipulate data.
2. These editors allow users to analyze data extracted from other sources (i.e. operational data bases) and to install simple new applications of their own design.
3. The display of all data as tables is inconvenient for some applications. Some provision must be made for the display of tabular data in a form more appropriate to the application.

FORMS: Displaying Records in Comprehensible Ways

We choose a simple tabular model because everyone can understand it and because it lends itself to aggregate operators such as sort, select and join. It also admits a simple data display, one line per record. But humans are willing to deal with much more complex data displays. In particular, when displaying data on a printed page or screen, people prefer that the page have some structure. For many tasks a page filled with a table of numbers is rather hard to grasp.

As mentioned in the last section, report generation systems have been aggregating and structuring files and records into page-oriented listings for many years. The structuring typically aggregates the data by some attributes and then prints summary statistics by attribute (subtotal, total, ...).

Increasingly, report generation is being replaced by "interactive reports". The structure of such systems has the flavor of report generation except that the system puts a "blank" report sheet on the screen and the user types in the attributes of interest. The system then generates the pages of the report corresponding to those attributes [1,2,6,7].

To give a specific example, consider the telephone book. In its "batch-oriented" report form the familiar phone book is several thousand of pages and is rather cumbersome to search. An ordinary data base editor would represent the phone book as a table. The user could scroll through the table by using predicates on the various fields. For anyone who can type, this is faster and easier than looking in a phone book but the display leaves a lot to be desired. Nearby entries are displayed along with the requested entry which occupies only a single line of the much larger display. A telephone operator who is always looking up numbers would prefer a more structured display.

A nice display for this application would allow the user to enter last and first name prefixes. The system would then do a phonetic search on those names and display the matching entries (and only those entries) in a structured way. For example, the four records for Ace Music might be displayed as:

```
Telephone Book Search and Edit

Name: _
Address:
Telephone:
HELP          FIND          PANIC
SCROLL-ENTRY  SCROLL-NUMBERS
INSERT        REPLACE        DELETE
```

Example 2: A very simple display of a phone book lookup.

Programs to define such screens are evolving quite rapidly and have wide acceptance [6.9]. Their basic model is that a SCREEN or FORM consists of a set of WINDOWS. Each window consists of sub-windows and of FIELDS. The display above has several windows: the outer window contains the NAME and ADDRESS fields and the TELEPHONE inner window. The inner window contains up to four repetitions of a telephone entry window. The telephone entry window has two fields: a phone number field and a comment field. The user may scroll the TELEPHONE window. If there are more than four entries for the given name and address. Scrolling the outer window causes the inner window to change consistent with the outer window.

Forms are used as follows: the user fills in parts of the form. This constrains the window and causes the system to fill in the remaining parts of the form. The constraints can be literal strings (e.g. "ACE") or can be predicates in the data base editor language (e.g. >1000).

This process of filling out a form is interactive. In an invoice application the user fills in the supplier name and the system fills in the address. The user fills in the item name and quantity and the system fills in item number, the unit price, and total price. The interaction is repeated until the invoice is complete. The system then totals the invoice and enters it into the database. Order entry is a "standard" and "operational" package and is used here only as an example.

Once retrieved, a form may be deleted from the data base, altered to replace its source in the data base, or a new form may be entered in the data base. The user indicates the desired operation by selecting the appropriate screen function.

The template for a form can be defined interactively. In fact the definition process is a forms-oriented application.

1. Each window is described in turn by pointing to its corners and giving its horizontal and vertical repetition factors. In addition a mapping from the window to a particular set of records is specified in the language of the data base editor.
2. A field is like a window but has different attributes. Some of the attributes that may be associated with a field are:
 - a. Literal value which will be displayed when the field is output. Headings, field names, and default values are done in this way.
 - b. Display attributes such as protected (screen user can't change it), color, brightness, etc.
 - c. Constraints such as mandatory (must be entered), alpha (must be alphabetic) and allowed value ranges (e.g. between 1 and 100). In fact any predicate of the data base editor can be used to constrain the possible values of a field.
 - d. For fields which are either input or output fields, a mapping from that field to the data base record of the containing window is specified.

The mapping of windows and fields to data base records is the most difficult part of making such systems workable. Each window is considered to be a record from some file. The screen designer specifies the file when specifying the window. Windows nested within windows may depend on the values of "outer" windows in defining their values. Fields within a window consume or produce values for the fields of the records of the window.

In the example of the phone book, suppose the two name and address screen fields of the window are called field.2 and field.4, then the outer window is related to the "phone.book" record type and is defined by:

```
SELECT UNIQUE name , address
FROM phone.book
WHERE name SOUNDS LIKE field.2
AND address SOUNDS LIKE field.4;
```

Example 3. SQL to lookup name and address given a screen.

The SOUNDS LIKE verb allows phonetic match. If this selection matches multiple record instances, each distinct instance of the outer window will produce a window instance. The terminal user can scroll through each of these screen instances.

The TELEPHONE window can now be described in terms of the values of fields in the outer window by:

```
SELECT number,comment
FROM phone.book
WHERE name= field.2
AND address = field.4;
```

Example 4. Look-up of phone numbers for given name and address.

At present, systems for defining screens and specifying the mapping between screens and the data base are just evolving and so are somewhat limited and un-polished. I expect that these systems will evolve to be quite useable. The experience with Query By Example is promising: users can master simple aspects of a simple language, especially when the alternative is doing the job manually.

I am glossing over several aspects of screen management systems. But one aspect is essential to the discussion that follows. Screen management systems let one screen refer to another. The simplest example is a menu screen. When the user selects a particular menu item, the screen manager displays the corresponding screen. This approach has two virtues. First it prompts the user and leads him through the application. Second it groups a set of operations together into a transaction. Certain operations require that several screens be entered before the operation is complete—for example hiring an employee may require filling in several forms.

A forms-oriented application appears to be a collection of screens. The top screen is a menu of functions. When the user enters a particular function he will be presented with forms to fill out and with further menu selections. The user may do data entry by filling out forms or may do data retrieval by partially qualifying a form and then asking the system to search for the records satisfying those constraints. The constraints can be literal values or QBE-like predicates.

To summarize:

1. The relational data model and relational operators allow a simple data manipulation language.
2. A data base editor allows for the analysis of data by naive users in unanticipated ways.
3. The meaningful display of data for screen-oriented users requires some progress in screen definition and screen management packages but forms-oriented definition facilities seem quite promising.

AN END-USER APPROACH TO APPLICATION DESIGN

Here is a prescription for an end-user to design an application:

Find someone who has already done the application. If possible use his system. Remember how expensive it is to write programs.

If you cannot find a pre-existing version of your application and if it is simple enough to fit the forms-oriented mold then proceed as follows:

1. Describe the application in terms of the screens you would like to see. Do a scenario of screens for each task (data entry, data retrieval, report generation,...).
2. For each screen decide what records must be maintained in the data base in order to support the screen.
3. Use the data base editor to create this data or to extract it from pre-existing data bases.
4. Use the forms design package to describe each screen and the relationships among screens.

I believe that clerical personnel will be able to understand this design process with very little training. Screens are a tangible concept which they can easily grasp. Data base editors have been quite successful precisely because they have the simple conceptual model of editing a table.

For clerical applications, steps 1 and 2 should be relatively straight-forward, especially as forms-oriented applications proliferate and most people gain first-hand experience with them. The typical approach to step 2 is to have a record per screen. This horrifies data base designers because a typical screen displays many facts (e.g. repeating groups) and so is not in normal form. But the relational model discourages repeating groups, and users discover (or are taught) that it is a good idea to keep one fact in one record (not two facts in one record or one fact in two records).

At present, the requisite pieces are not all there: application packages usually do not externalize their data as tables, data base editors are not available on some computing systems, and form definition and management systems are rather difficult to use. But I believe that the approach I have sketched is a viable way to allow users to tailor their own systems.

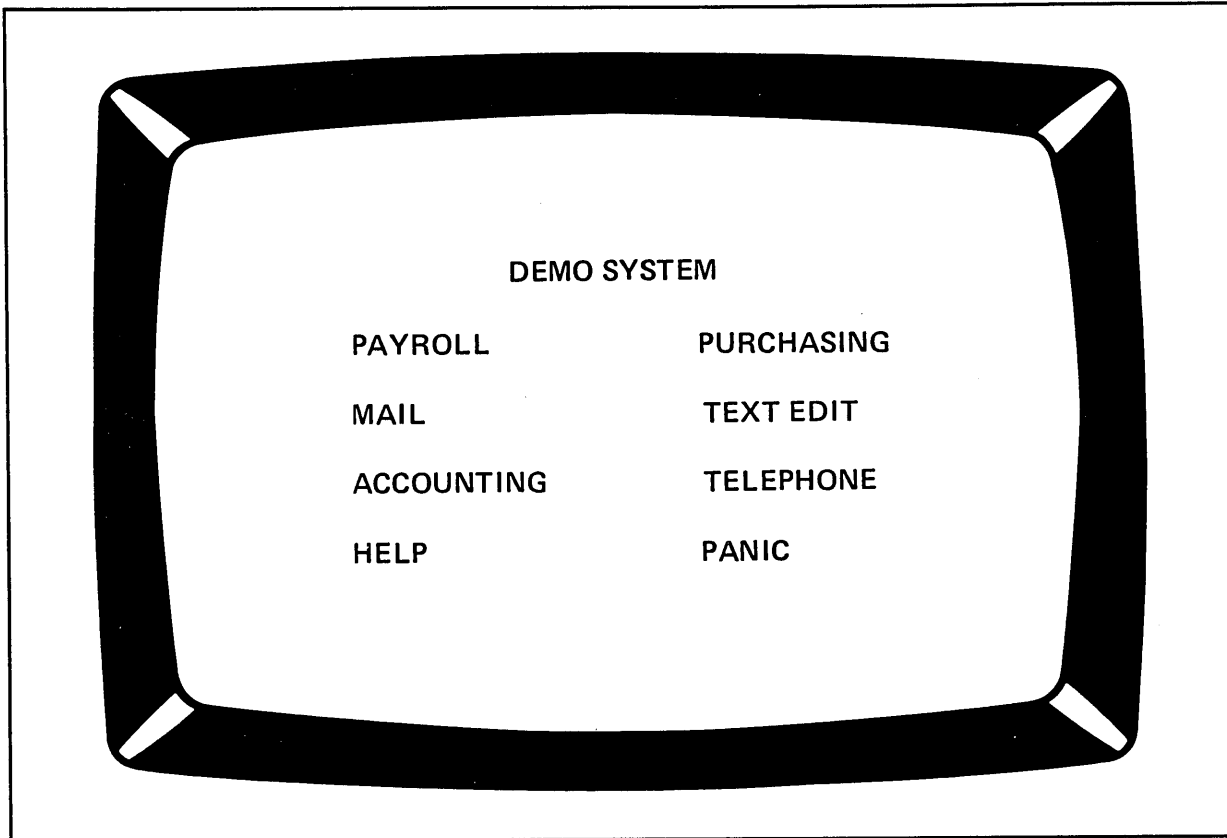
If standard application packages are written on a general-purpose data management system then it is not difficult to extract operational data from the data management system and display it in tabular form. (IBM's Query By Example supports such an extraction mechanism from IMS operational data management system. Similarly Tandem's Enform has full access to the Enscribe data base, subject only to authorization constraints).

Data base editors are increasingly available and certainly have a bright future.

The forms design systems and screen management systems I have seen are not quite ready for the bank loan officer or stock analyst. They have all the needed function but demand too much knowledge of the screen designer. The best systems I have seen still need to be chauffeur driven by an application programmer. It is clear that these systems will evolve and be integrated with data base editors in the future.

To give an impression of how the screen design scenario works, consider the design of the phone-book application. Clearly this application would already have been done by someone else, but it is chosen as a simple illustration of the design approach.

The starting screen is the menu of possible applications. It already exists. We add to it a new item for telephone books:



Example 5. The top level screen of an application.

This screen, like all others, has a title DEMO SYSTEM and a set of functions. All screens have the HELP and PANIC functions located at the bottom of the screen. Help displays a description of the screen in natural language (French). Panic returns the screen to the top level menu (this menu) and cancels any transactions that were in progress. The other functions specify pre-existing applications.

TELEPHONE has been added to the list of functions. A description of the telephone function would also be added to the help file describing the screen. If that function is selected (via light pen, mouse or function key) then the following screen is displayed:



Example 6. A telephone book look-up screen designed by a user.

The designer writes a natural language description of this screen. Each field at the bottom represents a function. HELP triggers the display of the screen description. FIND causes the search for the name in the phone book and displays all matching entries. INSERT puts the data entered on the screen into the file. REPLACE is used after find, the user first finds a name then enters the new data on the screen and then picks the replace function to cause the data base to be modified. DELETE, like replace, is used after FIND to delete items. The SCROLL functions move on to the next set of matching entries. Recall that we defined this window to be up to four vertical repetitions of phone number, comment pairs matching the name and address. If the entry has more than four phone numbers then SCROLL.NUMBERS will move on to the next batch of four. SCROLL.ENTRY moves on to the next names if several names match a find. We could allow a vertical repetition of the phone entries so that the output of a FIND would display several matching items at a time on the screen.

After specifying these screens and their use, the designer would then design the data base records to support them. In this case, this is a simple matter: one entry per phone number. The user might be tempted to have one record per screen but that would be a mistake; both because a name address pair can have a very large number of phone numbers and because it would violate the one fact in one place design rule.

The remaining steps would involve defining the phone book record type to the data base editor and the mapping between the telephone screen and the telephone record. The definition of the records is done as in Example 1 (for a QBE-like system). The mapping between screens and data base records is done by a forms-oriented program that asks for the mapping of each field in the screen. The user would provide answers like the SQL statements of Examples 3 and 4.

SUMMARY

In summary, it is proposed that a data base editor allows clerical personnel to maintain and manipulate their personal data bases and to examine operational data bases. A forms-oriented screen manager allows users to define pleasing displays of data.

How does this proposal differ from the automatic programming bubble of the seventies? First, the problem domain has been restricted to the manipulation of data base records in a clerical environment. Second, the proposal gives the user a very simple model of the data (records and fields) and a simple model of data manipulation (aggregate operators like sort and select). So it differs in that it attacks a very limited application area and attacks it by simulating the things people already do manually. Experience with text processing systems suggests that this approach makes a system accessible to non-programmers.

Instances of such systems are emerging from the experimental stage. These new systems will change the prevailing attitude that end-users cannot implement simple application programs. As a side effect, these systems will dramatically improve the productivity of applications programmers defining interactive screen-oriented data base applications.

ACKNOWLEDGMENTS

The ideas for this paper stem from discussions with Peter DeJong (IBM Yorktown) about Query By Example (QBE) and the much more ambitious System for Business Automation (SBA) which he is now constructing. Several of the ideas relating to screen-oriented application design stem from Gary Hass and Frank Nargie (IBM Santa Teresa) who used this approach in making System R available to end-users. Discussions with and critical comments from Allen Berglund, Keith Dick, Jonathan Perry, Robert Shaw, Bob Welles, Kim Worsencroft, and Joan Zimmerman helped clarify my ideas and improved the presentation of this paper.

REFERENCES

- [1] *Tandem 16 Pathway Screen Cobol Reference Manual*, Part No. 82146, Sect. 7, Tandem Computers, 19333 Vallco Parkway, Cupertino CA 95014, 1980.
- [2] *Display Management System/Virtual Storage, General Information Manual*, IBM Form No. GH20-18863, IBM Data Products Division, White Plains NY 1978.
- [3] *Tandem 16 Enform Reference Manual*, Part No. 82034, Tandem Computers, 19333 Vallco Parkway, Cupertino CA 95014, 1980.
- [4] Shu, N.C., B.C. Housel, R.W. Taylor, S.P. Ghosh, V.Y. Lum, *EXPRESS: A Data EXtraction, Processing, and REStructuring System*, ACM TODS Vol. 2, No. 2, June 1977.
- [5] *Query By Example Program Description/Operations Manual*, IBM Form No. SH20-2077, IBM Data Products Division, White Plains NY 1978.
- [6] *QUBE, Query/Update by Example*, Intel Form No. PR60M008, Intel Commercial Systems Division, Box 9968, Austin TX 78766, 1980.
- [7] Rowe, L.A., K. A. Shoens, *Screen Definition Facilities for Interactive Database Applications*, Computer Science Division, U.C. Berkeley, Berkeley CA 94720, Draft September 1980.
- [8] Chamberlin, D.D., et. al., *Sequel 2: a Unified Approach to Data Definition, Manipulation and Control*, IBM J. R&D, Vol. 20, No. 6, Nov. 1976.
- [9] *Using HP View 3000, An Introduction to Forms Design for Non-programmers*, Part No. 32209-90004, Hewlett Packard Corporation, 19447 Pruneridge Ave., Cupertino CA 95014, 1979.

Distributed by
 **TANDEM** COMPUTERS
Corporate Information Center
19333 Vallco Parkway MS3-07
Cupertino, CA 95014-2599

