INSIDE:
Methods and Materials, Data Base Design, Data Base Management Systems, Results, Discussion of Results

22-01-40

A Comparison of Three User Interfaces to Relational Microcomputer Data Bases

Carl Medsker
Margaret Christensen
Il-Yeol Song

PAYOFF IDEA. Different styles of user interfaces can dramatically affect data base capabilities. In an environment comprising many different data bases, the goal is to select one data base management system (DBMS) that provides the best selection of design tools, minimizes development times, and enforces relational rules. This article presents a case study performed at the Hospital of the University of Pennsylvania, in which a test data base was developed for implementation with three DBMSs, each with a distinctly different user and programmer interface.

INTRODUCTION

This article examines the effects of different human computer interface styles on data base functional capability. Graphical user interfaces (GUI) are spreading across computing platforms. In the rush to iconic, bit-mapped graphic displays, however, underlying capabilities must not be ignored.

From a human-factor perspective, two interfaces are of importance in data base design: the end-user interface to the application and the programmer interface to the data base development system. The user interface affects data integrity, productivity, product satisfaction, and system acceptance. The programmer interface affects the ability to deliver the requisite application in a timely manner. Interface styles (e.g., command line, command menu,

Auerbach Publications
Data Base Management

or GUIs) affect development times, interface options, and amount of program code.

The effect of each interface style on functional capability, design options, and production efficiency was evaluated. Oracle Corp.'s HyperCard for the Apple Computer, Inc., Macintosh; Oracle for DOS; and Borland International, Inc.'s Paradox for DOS were chosen because each is based on the relational model and each represents a distinctly different approach to interfacing with the relational model.

METHODS AND MATERIALS

Data base applications for this study were developed within the Department of Anesthesia at the Hospital of the University of Pennsylvania. Because this department is divided into clinical, research, and teaching functions, each with its own information needs, it is not surprising that it is a heterogeneous computing environment. A Digital Equipment Corp. miniVAX; Sun Microsystems, Inc., workstations; Apple IIs, Macintosh SEs, and Macintosh IIs; and IBM Corp. PCs and 80386s coexist. Applications are even more diverse, including real-time data collection, statistics, graphics, business spreadsheets, data bases, word processing, and online searching.

Research studies are largely pharmaceutical based and require the collection of data concerning a wide range of physiological parameters. The sheer number and diversity of research data base needs require systems that reduce development times. Staff computer experience varies from the indifferent, reluctant user to the expert assembly-level programmer. User interfaces and data base management systems must, therefore, be appropriate for these different groups. It is within this environment that a test data base was designed as the basis for evaluation.

DATA BASE DESIGN

The test data base consisted of hospital patients and six common physiological measurements collectively referred to as pH/blood gas measurements. An entity-relationship (ER) diagram, derived relational schemas, and data dictionary were the common specifications used with each data base system.

The test data base consists of patients who receive pH/blood gas analyses. The results determine the course of pulmonary and cardiac therapies. One pH/blood gas set consists of:

- F_iO_2. The fraction of inspired oxygen provided at sampling time.
- pH. The pH value of the blood.
- PCO_2. The partial pressure of carbon dioxide in the blood.
- PO_2. The partial pressure of oxygen in the blood.
- BE. The base excess of the blood.
- HCO_3. The bicarbonate concentration in the blood.

The ER diagram (see Exhibit 1) represents the fact that a patient has many pH/blood gas tests, and each test belongs to exactly one patient. Each occurrence of a Patient is identified by the hospital identification number (HUP_ID_#). The data and time of testing are included with the relationship

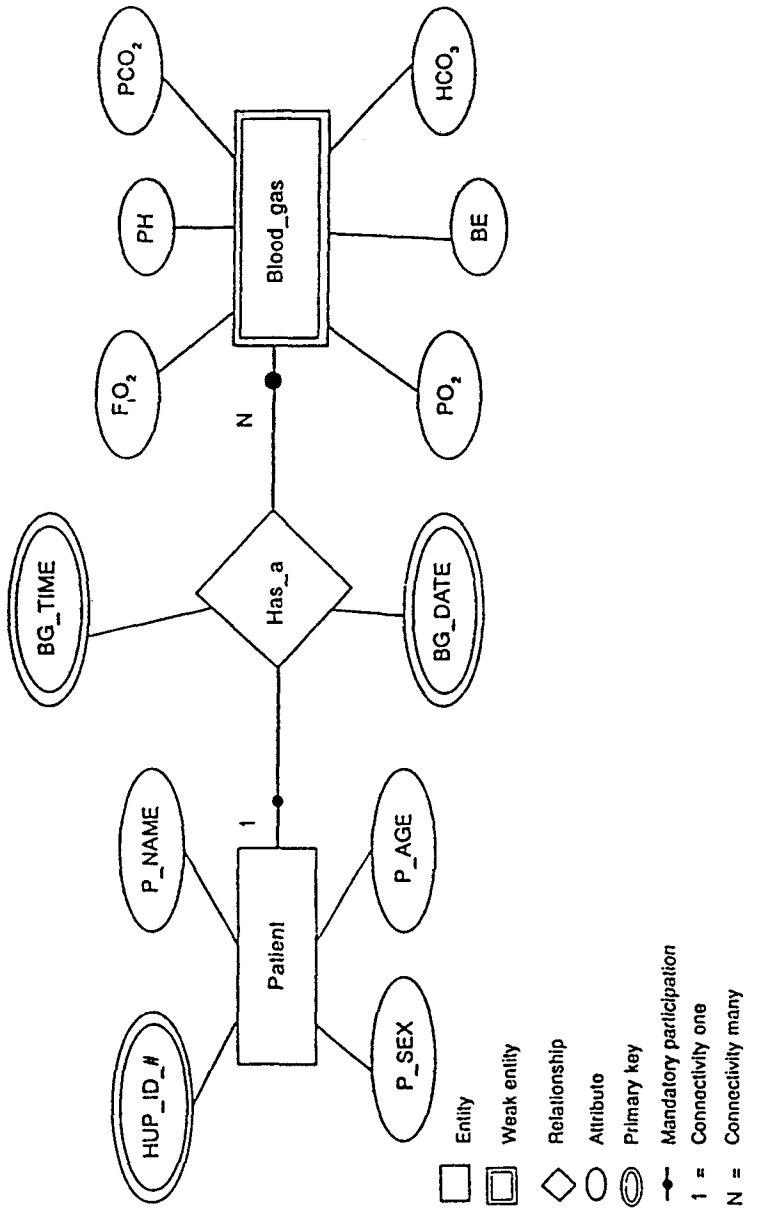


Exhibit 1. ***Entity-Relationship Diagram***

Exhibit 2. *Attribute Constraints of pH/Blood Gas Data Base*

	Attribute	Type	Template	Minimum	Maximum	Default
1	HUP_ID_#	Char 9	Digits	0	999999999	
2	P_NAME	Char 20	Uppercase			
3	P_SEX	Char 1	M or F			
4	P_AGE	Num	##[#]	0	150	
5	BG_DATE	Date	mm/dd/yy	01/01/90	today	today
6	BG_TIME	Char 5	##:##	00:00	23:59	
7	F_iO_2	Num	[#].##	0.21	1.00	0.21
8	PH	Num	#.##[#]	6.00	8.00	
9	PCO_2	Num	##[#].[#]	0.0	200.0	
10	PO_2	Num	##[#].[#]	0.0	770.0	
11	BE	Num	#[#].[#]	-50.0	+50.0	0
12	HCO_3	Num	#[#].[#]	0.0	50.0	23.0

HAS_ a because, logically, they are attributes of an event at the time the blood specimen was drawn. Alternatively, analysis date and time could be included with the relationship Blood_gas, but this example describes only when the blood sample was taken from the patient, not when it was analyzed. Each occurrence of Has_a is identified by the combination of the date (BG_DATE) and the time (BG_TIME) of sampling plus the patient hospital number (HUP_ID_#), using BG as an abbreviation for blood gas. A set of blood gas results has no unique identifier and therefore is a weak entity. The ER diagram was mapped to two relations, combining the weak entity Blood_gas with the relationship Has_a: PATIENT (HUP ID #,P_NAME, P_SEX,P_AGE) HAS_A_BLOOD_GAS (HUP ID #,BG DATE, BG TIME, F_iO_2,PH,PO_2,PCO_2,BE,HCO_3). The data base systems were evaluated on the ability to directly accommodate attribute constraints (see Exhibit 2) by relational tables.

DATA BASE MANAGEMENT SYSTEMS

Three data base systems were evaluated: HyperCard Version 1.0 for the Macintosh, Oracle for DOS Version 2.0, and Paradox for DOS Version 3.5. Oracle/HyperCard is a hybrid consisting of an Oracle relational back end and a Macintosh HyperCard front end. Most administrative chores, including table creation and modification, are handled by HyperCard. A window for SQL is provided for direct, command-line interaction with tables.

Oracle for DOS consists of a relational back end with a command-line interface for administration and table creation and modification. SQL Forms was used for creation of the user interface. Query-by-example and a menu-bar command structure characterize Paradox. The built-in program generator creates applications based on 15 or fewer tables and was used for this evaluation. In general, the program generator is useful for small applications, prototyping, and creating menu bars. Applications using more than 15 tables must be programmed in the Paradox application language.

Exhibit 3. *Shneiderman's Guide for Form Fill-In Screens*

Meaningful Title

Avoid computer terms and jargon and prefer words related to the users' activities.

Comprehensible Instructions

Be brief but clear in phrasing instructions and be consistent across the applications.

Logical Grouping and Sequencing of Fields

Sequence fields to match common usages.

Visually Appealing Layout

Align and space fields to avoid clutter. When possible, match screens to paper forms.

Familiar Field Names

Use common terms for field names.

Consistent Terminology and Abbreviations

Work from a prepared list of standard terms and abbreviations.

Error Correction for Characters and Fields

Ensure that the user can correct parts of fields or entire fields.

Visual Templates for Common Fields

Indicate the number of characters per fields, provide templates, and mark option fields.

Help Facilities

Provide meaningful help messages with example corrections.

Hardware Configurations

Macintosh software ran on a Macintosh SE with 2M bytes of RAM and a 20M-byte hard drive. DOS software ran on a Northgate 20-MHz 80386 with 4M bytes of RAM and a 75M-byte hard drive.

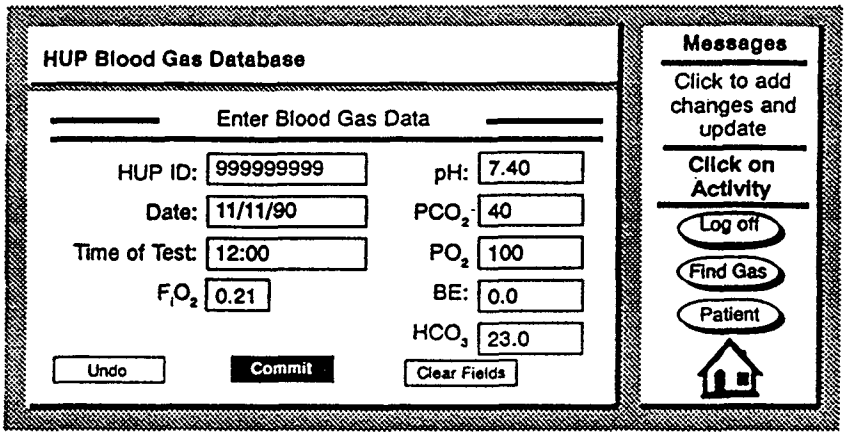
Evaluation Criteria

The user interface was evaluated against Shneiderman's guidelines for form fill-in data entry screens (see Exhibit 3). The screens shown in Exhibit 4 were rated against six of these nine guidelines, or criteria. The criteria meaningful title, familiar labels, and consistent terms refer to screen titles, field or attribute names, and abbreviations or units, respectively. Because term selection is dictated by programming policy and is independent of any particular data base management system, these criteria were excluded from the evaluation.

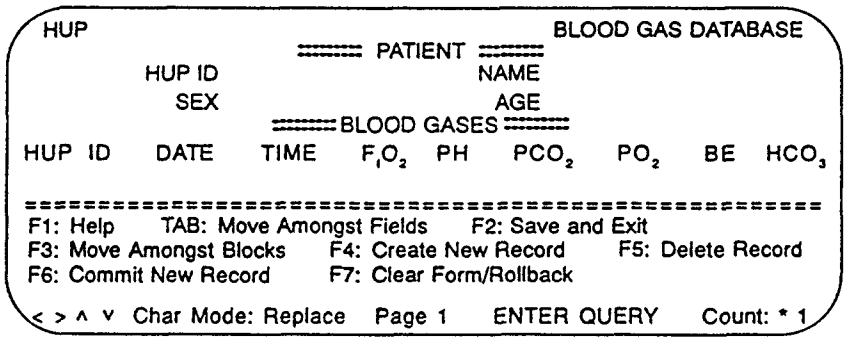
Data base functional capability was defined as adherence to Codd's rules of relational design, as listed in Exhibit 5. To facilitate analysis, his earlier set of 12 base rules and one foundation rule was chosen over the recent revision that includes 20 fundamental laws and 330 rules.

The programmer interfaces were compared in regard to development times, lines of code written, and richness of design tools. Two additional appli-

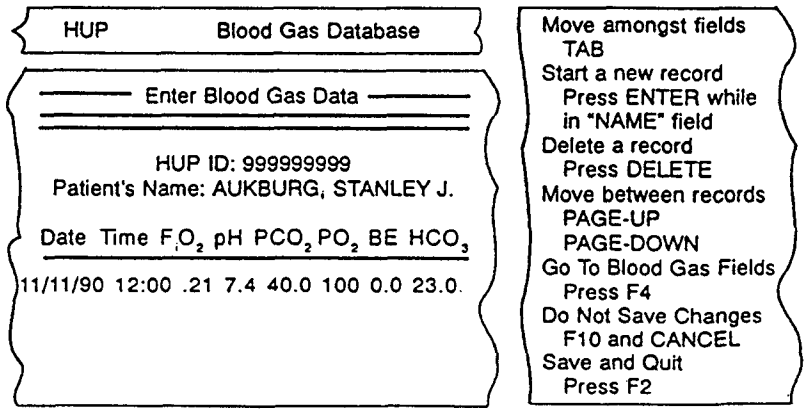
Exhibit 4. *Application Screens*



a. The HyperCard Data Entry Screen



b. The Oracle for DOS Data Entry Screen



c. The Paradox Menu Bar and Data Entry Screen

Exhibit 5. ***Codd's Relational Rules***

Rule 0:	Foundation Rule: A relational DBMS must manage all data base activities through relational operators.
Rule 1:	All information in a relational data base, including metainformation about tables (i.e., names, attributes, constraints) is contained solely as values in tables.
Rule 2:	Every individual item of information must be accessible by table name, primary-key value, and attribute name.
Rule 3:	Regardless of data type, missing or unknown data (nulls) must be represented and functions for manipulation supported. It must also be possible to disallow null values for primary keys.
Rule 4:	All descriptions of the data base are stored in relational tables just as regular data and must be subject to relational queries and manipulations.
Rule 5:	A relational data base must provide at least one language with a complete syntax for data definition, view definition, data manipulation; integrity constraints, access authorization, and transaction boundaries.
Rule 6:	The data base system must be able to determine whether updates, deletions, and insertions are possible through any given data base view.
Rule 7:	Base relations and derived relations are treated as single operands with regard to retrievals, insertions, updates, and deletions.
Rule 8:	Tables and applications are unaffected by changes in the physical computing environment.
Rule 9:	Tables may be changed without affecting applications, assuming the changes protect fundamental information.
Rule 10:	No part of a primary key or a foreign key may be null and these facts are stored in a catalog table.
Rule 11:	Data may distributed over multiple sites without affecting applications.
Rule 12:	Integrity constraints may not be undermined by addition of a record-at-a-time language.

cations were written as preliminary practice with each data base and discarded; this served as a means of reviewing each system's design tools and ensured that the programmer was equally adept with each system before creating the final applications. The work was completed with production times logged on two consecutive days.

RESULTS

This section presents data base application interface features followed by an evaluation of data base functional capability. Finally, programmer productivity is summarized.

User Interface Evaluation

A variety of data base application features were evaluated, including instructions, field grouping and sequencing, layout, error correction capabilities, visual templates, and help facilities. The evaluations of the three data base systems with Shneiderman's guidelines, as conducted by the

Exhibit 6. *Evaluation of Shneiderman's Guidelines*

Number	Guideline	HyperCard	Oracle for DOS	Paradox for DOS
1	Comprehensible instructions	Excellent	Poor	Good
2	Logical grouping	Poor	Excellent	Excellent
3	Visual appeal	Excellent	Poor	Good
4	Error corrections	Excellent	Good	Good
5	Field templates	Poor	Good	Excellent
6	Help facilities	Excellent	Poor	Good

programmer, are summarized in Exhibit 6. Each individual feature is discussed in the following section.

Comprehensible Instructions. Although mainly a choise of terminology and consistent use of meaningful abbreviations, the presentation of instructions varied. HyperCard provided the most flexible, creative options (e.g., pop-up fields or the ability to disable or hide irrelevant options), thereby reducing the amount of information that must be processed and the choices that must be considered. All information does not have to be presented on the screen at once, as with Paradox and Oracle for DOS. Context-sensitive instructions were easily provided by pop-up fields. Cognitive load was lowered by presenting subsets of instructions as needed.

Menu bars were easily generated within Paradox, resulting in an on-screen display of menu explanations before the forms interface was run. Once in a form, however, application-specific instructions were limited to the data entry screen. Context-sensitive command help is linked to a function key and is always available. Oracle for DOS allowed only the instructions that fit on one data entry screen.

Logical Grouping and Sequencing of Fields. Fields were grouped as desired in all three tools, but the smaller Macintosh screen did not display as many fields. In particular, it was difficult to create a one-to-many form with HyperCard. An example is an order form on which a person is associated with any number of line item orders. In the test data base, a patient may have many pH/blood gas tests. One-to-many forms were quickly and easily created with Oracle for DOS and Paradox, but Paradox's screen design tools allowed a more creative presentation and easier manipulation of fields than SQL Forms. In Paradox, the designer can easily mark and move areas of the form or add color and other video attributes (e.g., inverse video or blinking characters), all from the menu bar.

Visually Appealing Layout. HyperCard was superior in screen appearances (see, for example, Exhibit 4). The ability to include graphics and buttons from existing applications was a significant design aid. Paradox lacks graphic capabilities but has extensive screen painting tools. Oracle for DOS, using SQL Forms, has a more limited set of creative screen painting tools. Oracle for DOS was limited to lines for demarking areas of the screen and

Exhibit 6. *Evaluation of Shneiderman's Guidelines*

Number	Guideline	HyperCard	Oracle for DOS	Paradox for DOS
1	Comprehensible instructions	Excellent	Poor	Good
2	Logical grouping	Poor	Excellent	Excellent
3	Visual appeal	Excellent	Poor	Good
4	Error corrections	Excellent	Good	Good
5	Field templates	Poor	Good	Excellent
6	Help facilities	Excellent	Poor	Good

programmer, are summarized in Exhibit 6. Each individual feature is discussed in the following section.

Comprehensible Instructions. Although mainly a choise of terminology and consistent use of meaningful abbreviations, the presentation of instructions varied. HyperCard provided the most flexible, creative options (e.g., pop-up fields or the ability to disable or hide irrelevant options), thereby reducing the amount of information that must be processed and the choices that must be considered. All information does not have to be presented on the screen at once, as with Paradox and Oracle for DOS. Context-sensitive instructions were easily provided by pop-up fields. Cognitive load was lowered by presenting subsets of instructions as needed.

Menu bars were easily generated within Paradox, resulting in an on-screen display of menu explanations before the forms interface was run. Once in a form, however, application-specific instructions were limited to the data entry screen. Context-sensitive command help is linked to a function key and is always available. Oracle for DOS allowed only the instructions that fit on one data entry screen.

Logical Grouping and Sequencing of Fields. Fields were grouped as desired in all three tools, but the smaller Macintosh screen did not display as many fields. In particular, it was difficult to create a one-to-many form with HyperCard. An example is an order form on which a person is associated with any number of line item orders. In the test data base, a patient may have many pH/blood gas tests. One-to-many forms were quickly and easily created with Oracle for DOS and Paradox, but Paradox's screen design tools allowed a more creative presentation and easier manipulation of fields than SQL Forms. In Paradox, the designer can easily mark and move areas of the form or add color and other video attributes (e.g., inverse video or blinking characters), all from the menu bar.

Visually Appealing Layout. HyperCard was superior in screen appearances (see, for example, Exhibit 4). The ability to include graphics and buttons from existing applications was a significant design aid. Paradox lacks graphic capabilities but has extensive screen painting tools. Oracle for DOS, using SQL Forms, has a more limited set of creative screen painting tools. Oracle for DOS was limited to lines for demarking areas of the screen and

single- or double-line boxes. Paradox allowed more options (e.g., different colors, inverse, blinking characters, and Lotus-like menu bars).

Error Correction for Characters and Fields. Error correction in HyperCard was aided by use of the mouse to place the cursor within a field. Paradox and Oracle for DOS required the user to backspace and delete characters to the correction point, which increased keystrokes.

Visual Templates for Common Fields. The extensive field formatting capabilities of Paradox allowed embedding of regular expressions and specific values in field definitions—for example:

- "*!". This converted all characters to uppercase as they were typed.
- "##:##". This allowed only two numbers on either side of a colon, which was automatically provided.
- "###.##.####". This was a template for a Social Security number that automatically filled in the "-".
- "[YES,NO"]. This restricted input to these two values.

Field templates are partially availablein HyperCard and Oracle for DOS. Uppercase conversion and date formatting were available by menu selection in the DOS version. Uppercase conversion in HyperCard required a loop structure to sequentially convert each character. The Social Security template and restricted, discrete input values were unavailable options in either version of Oracle.

Help Facilities. Extensive help facilities were added to HyperCard in a variety of formats. Buttons linked help messages to screens and pop-up fields, allowing context-sensitive support. Brief guidelines appeared in a fixed field as the mouse pointer entered each button. In Oracle for DOS, help text was limited to what fit on a screen and lacked the interactive, context-sensitive support provided in HyperCard.

Oracle, on both platforms, displays numeric error codes (of which there are more than 1,500) that are meaningless without the error manual (166 pages long). Subroutines could be written to trap and interpret these errors, but that represents an excessive amount of programming. Paradox error codes are text messages, some of which are obvious (e.g., This field must be filled in), whereas others require some knowledge of data base terminology (e.g., Key violation error) but are further explained in the user guide. These may also be trapped by subroutines that provide more meaningful messages. The F1 function key provides context-sensitive help with Paradox commands from within any application.

Relational Functional Capability

Codd's rules call for all table definitions and constraints to be stored in tables for manipulation with data base system statements. Although supported by some data bases, many constraints and relational features are embedded in the application code instead. For this article, the two alternatives were referred to as direct and indirect. Exhibit 7 lists key relational features indicating direct or indirect support by the three chosen systems.

Exhibit 7. *Support for Relational Features*

Relational Feature	HyperCard	Oracle for DOS	Paradox for DOS
Primary key designation	Indirect	Direct	Direct
Entity integrity	Indirect	Direct	Direct
Referential integrity	Indirect	Indirect	Indirect
Null values	Indirect	Direct	Direct
Set-at-a-time processing	Indirect	Direct	Direct

Exhibit 8. *Evaluation of Ability to Ensure Data Integrity*

Integrity Constraint	Hypercard	Oracle for DOS	Paradox for DOS
Data typing	Indirect support	Direct support	Direct support
Field templates	Indirect support	Indirect support	Direct support
Minimum and maximum limits	Indirect support	Indirect support	Direct support
Default values	Indirect support	Indirect support	Direct support
Table look-up	Indirect support	Direct support	Direct support
Automatic fill-in	Indirect support	Indirect support	Direct support

Exhibit 9. *Programmer Productivity*

	HyperCard	Oracle for DOS	Paradox for DOS
Development Time	8.0 hours	—*	2.5 hours
Lines of Typed Code	121	316**	19

Notes:
*The Oracle DOS application was not completed.
**Based on lines of code generated by Paradox program generator.

With Hypercard, new data may be committed a row at a time, or rows may be buffered in memory and committed as a set. The first option slows performance but allows immediate checking of key integrity, referential integrity, and null values. The second option improves response times but makes it difficult to verify integrity because HyperCard handles data a row at a time, whereas relational systems process data a set at a time. Oracle for DOS specifies the features partly through the SQL table creation command and partly through SQL Forms. Paradox provided direct specification of these relational features, excluding referential integrity, by the menu bar interface for storage in tables. On the basis of the constraints listed in Exhibit 2, Exhibit 8 summarizes the ability of each system to ensure data integrity.

Programmer Productivity

Programmer productivity (see Exhibit 9) was evaluated on the basis of development times and lines of typed code required to implement the design in Exhibit 1. The programmer had received graduate school training in programming and data base design and had created clinical and research data base applications for the host institution of this study. Times are within

Exhibit 10. *SQL Updates Within Each Data Base System*

SQL:

```
Insert into HAS_A_BG values ('999999999',11/11/90,'12:00',0.21,7.40,40,100,0.0,23.0);
```

HyperTalk:

```
Put ""& card field HUP_ID_#& ""& "," into A
Put ""& card field BG_DATE & ""& "," into B
Put ""& card field BG_TIME &""& "," into C
Put card field FiO2 & "," into D
Put card field PH & "," into E
Put card field PCO2 & "," into F
Put card field PO2 &"," into G
Put card field BE & "," into H
Put card field HCO3 & "," into I
Put "insert into HAS_A_BG values ("into SQLSTR
Put A & B & C & D & E & F & G & H & I after SQLSTR
Execsql SQLSTR
```

Oracle for DOS:

```
Insert into HAS_A_BG values ('999999999',11/11/90,'12:00',0.21,7.40,100,0.0,23.0);
```

Paradox for DOS (by on-screen table fill-in):

HUP_ID_#	BG_DATE	BG_TIME	F_iO_2	PH	PCO_2	PO_2	BE	HCO_3
999999999	11/11/90	12:00	0.21	7.40	40	100	0.0	23.0

15 minutes of the actual time, and include tables, screens, and program creation.

Finally, embedding SQL queries in HyperTalk required collecting values from HyperCard fields, concatenating these values and the appropriate SQL terms, and then executing the query. Certainly, the user could directly enter SQL commands in the HyperCard SQL window, but the purpose of HyperCard as an application interface is to shield the user from command line entry by attaching the code to buttons. This requires additional programming codes to collect user input and construct SQL statements. Embedding SQL in other programming languages (Pascal or C) is more straightforward in that the SQL query is contiguous, not requiring concatenation of terms in the host language. Exhibit 10 shows SQL updates within each data base system.

The paradox menu bar sequence to directly append a new tuple is as follows:

<M>odify<E>dit<table-name><ENTER><END><DOWN>
<999999999><ENTER><11/11/90><ENTER>
<12.00><ENTER><0.21><ENTER>

<7.40><ENTER><40><ENTER><100><ENTER>
<0.0><ENTER><23.0><[F2]>

The <> brackets delimit user keystrokes. This sequence selects modify from the menu bar, selects edit from the submenu, inputs the requested table name, moves to the end of the table, and enters a new tuple.

The HyperTalk version is verbose and convoluted and illustrates the coding required to recover the relational capabilities lost with the addition of the HyperCard interface. HyperCard does, however, require fewer user keystrokes than Paradox to accomplish the same input.

DISCUSSION OF RESULTS

Rapid turnout of data base applications with effective interfaces and relational capabilities is needed to meet burgeoning user demands and tap distributed computing resources. Which system supports the fastest design of relationally true data bases with effective user interfaces? Is there necessarily a trade-off?

The human-computer interface is critical in the way it addresses certain clearly defined needs. It can help—or hinder—the developer, serve as a catalyst for increased user productivity and satisfaction, and provide or guarantee data base capability. Cursor movement, error correction, helpful instructions, cognitive load, data accuracy and consistency, and data entry speed are all affected by the interface. Other considerations include development time, file sizes, response times, reusability of code, and ease of maintenance.

The ability to specify constraints on possible attribute values is a significant feature of a data base management system. For instance, forcing uppercase characters guarantees consistent ASCII sorts or control over medical terminology. In addition to ensuring data integrity, value constraints are also an aspect of the user interface because the user is aided in keying in data while minimizing mistakes and editing.

In a data base environment, the programmer interface is defined by the available programming tools and the access to these tools in designing user interfaces and applications. Furthermore, adherence to the relational model is partially a function of the user interface because no current data base system fully implements Codd's relational rules. So choice of a data base should be based in part on the ability to enforce or subvert the relational model using the interface.

HyperCard adds an object-oriented graphical user interface to a relational/SQL foundation. The system excels at generating attractive screen displays, reusing code by cutting and pasting buttons and scripts, and managing presentation of instructions. Text management is better accomplished in HyperCard than in the other data bases because a card may contain unstructured, searchable text fields. It is also excellent for prototyping because the screens can be created before the underlying tables or code.

Both Oracle for DOS and Paradox require creation of tables before screens. However, HyperCard suffers in implementing the relational model,

handling large volumes of data, and processing time-sensitive transactions. Smaller research, teaching, and office applications, however, are appropriate for HyperCard, particularly when there is infrequent use by novices who require additional prompting and instructions.

Oracle for DOS presents a character-based structured query language (SQL) interface. Applications are commonly generated by creating text files of commands that are then executed at the command prompt. This allows correction of mistakes and application modification by editing the text files. Rerunning the text file is quicker than retyping each command and the text files document the application. Oracle for DOS provides more sophisticated, powerful form design tools and computer-aided software engineering (CASE) tools, but at considerably higher cost.

Borland's Paradox, although character based, uses a hierarchical menu bar structure and a query-by-example (QBE) interface for the presentation of tables. Paradox is appropriate for a production environment in which time is critical, large volumes of data are involved, data integrity is crucial, and relational functional capability is more important than screen appearances. All three systems provide table look-ups, but Paradox also provides automatic fill-in. This feature is particularly useful for dealing with medical terminology, in which misspellings are easy and multiple terms for the same concept exist. The Paradox program generator and screen design tools give it a clear advantage in development times for applications within the 15-table limit per application. Paradox offers the most direct control of data entry and integrity, whereas HyperCard and Oracle for DOS require more coding—in some cases extensively—to provide an equivalent level of control.

From a purely aesthetic point of view, HyperCard is a pleasure to work with, particularly with its graphic capabilities. Reusability of parts and scripts reduces coding and frees time for creative screen design. The danger lies in forever tinkering to get every screen feature just right to the neglect of data integrity issues.

CONCLUSION

Is Oracle with a Macintosh HyperCard front end just another pretty face or is it a serious data base engine? Are capabilities present behind the showy icons and buttons? The choice of a data base system for microcomputers based on this analysis of interface capabilities, relational adherence, and developer productivity, is between HyperCard and Paradox. Oracle resides on many platforms, from microcomputers to workstations and mainframes, and should be considered in meeting distributed processing needs. In addition, triggers are a powerful feature of SQL Forms; they are used, for example, to restrict data base access, collect and store audit data, enforce referential integrity, perform calculations, and embed SQL statements.

The addition of CASE and prototyping tools makes Oracle for DOS a productive, rich development environment. The goal in this study, however, was to examine the capabilities of systems of approximately equal cost that were readily available for user-development projects and that represented different approaches to interfacing with the relational model. That is, for

a given investment, which system provides the richest collection of design tools, minimizes development times, and enforces relational rules?

On the basis of Shneiderman's guidelines, HyperCard and Paradox provide better form fill-in interfaces than Oracle for DOS. HyperCard is the interface of choice, though, if visual display, flexible field editing, and a user help system are the major consideration. Oracle for DOS and Paradox are relationally truer and provide one-to-many and many-to-many relationships, essential features in a production environment. Codd's Rule 12 for relational data bases explicitly disallows subversion of integrity and functional capability by lower-level languages, which is precisely the result of HyperCard forcing access to tables by HyperTalk.

Perhaps the recently released update to HyperCard resolves the interface—functional capability problem. However, the essential point remains and is applicable to selecting a system on any platform. Just as a book should not be judged by its cover or a car bought by its color, an application should not be judged by its icons alone. Functional capability should be carefully evaluated when selecting a Macintosh data base or one of the increasing number of DOS Windows data bases.

Carl Medsker is director of the critical care laboratory of the Hospital of the University of Pennsylvania and a doctoral student in the College of Information Studies at Drexel University, Philadelphia. His research interests are in the application of expert system technology and emergent computation techniques to problems in health care and medical informatics. He is a member of the ACM, IEEE, ASIS, American Medical Informatics Association, and Computer Professionals for Social Responsibility.

Margaret Christensen is an assistant professor at the College of Information Studies at Drexel University. Her research interests include user interfaces and applied artificial intelligence.

Il-Yeol Song is currently an assistant professor at the College of Information Studies at Drexel University. His research interests include data base design methodologies, knowledge-based systems, and methodologies and tools for intelligent information systems. He received his PhD in computer science from Louisiana State University in 1988. He currently serves as a member of the program committee of the First International Conference on Information and Knowledge Management, and is a member of the International Conference on Developing and Managing Intelligent Systems.

Recommended Reading

Codd, E.F. *The Relational Model For Database Management*, Version 2. Reading MA: Addison-Wesley, 1990.

——. "Is Your DBMS Really Relational?" *Computer World*, 1985.

Cronin, D. *Mastering Oracle*. Carmel IN: Hayden Books, 1988.

Krumm, R. *Understanding and Using Paradox 3.5*. New York: Brady Books, 1990.

Parsaye, K.; Chignell, M.; Khoshafian, S.; and Wong, H. *Intelligent Databases*. New York: John Wiley & Sons, 1989.

Shafer D. *Using Oracle With HyperCard*. Carmel IN: Hayden Books, 1990.

Shneiderman B. *Designing the User Interface*. Reading MA: Addison-Wesley, 1987.