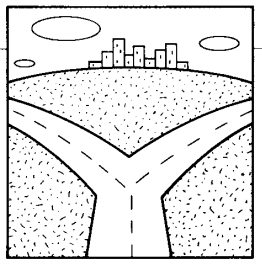


A Full-Screen Facility for Defining Relational and Entity-Relationship Database Schemas

Gary H. Sockut and **Ashok Malhotra**, *IBM Thomas J. Watson Research Center*

While you can use linear languages to define database schemas, it is not very convenient to do so. The main goal of this interactive system is ease of use.

A data model is an abstraction that can be supported by a database system. It specifies the types of structures a database can contain and the operations with which a user accesses the database.

One of the first steps in building a database is to define a schema, a specification of the logical structure of data in a particular database. For example, if a data model allows tables, a schema might specify an Employee table. After the database system has processed the schema and created the Employee table, a user can insert a data row to represent a particular employee.

A linear language (text strings) is not convenient for defining schemas. The user must remember the language's features, such as its syntax and the set of available data types, and must specify most information explicitly, without using defaults. A linear language's syntax can be verbose. Definitions of interrelated parts of a schema might be split into different statements in the language.

Our goal was to develop an easy-to-use, interactive facility for defining schemas for a mainframe database system. We wanted it to be much more convenient than a linear language. Our facility, called Dbdefs, has several features that make it easy to use. It provides similar interfaces for defining schemas for two data models, relational[1] and entity-relationship,[2] whose structures we summarize later in the article.

Features. Dbdefs integrates features that reduce the amount of time, typing, and memorization required to define a schema. The features include:

- Full-screen displays for creating, displaying, and changing information.
- Menu-driven selection of alternatives.

Sockut is now with IBM's Santa Teresa Laboratory.

0740-7459/88/1100/0068/$01.00 © 1988 IEEE

• Default values.

• The ability to customize defaults and some other aspects of the facility's behavior, with either a user profile or on-line actions.

• Function keys.

• Feedback that either indicates successful completion of an action or explains the error.

• Protection from harmful actions you request inadvertently.

• Prompting and on-line help.

Dbdefs is flexible: It lets you choose how to write definitions, where to write them, and how to document them:

• It provides interfaces for defining databases in two data models, relational and entity-relationship.

• It lets you write and delete definitions in a database (create and delete tables and indexes).

• It lets you read and write definitions in files, even if you do not use a database in a particular session with the facility. This lets you start to define a schema today, save the draft definitions overnight, and resume defining the schema tomorrow. Also, you can export files to and import files from an application program, with only minor editing (if needed). In this manner, Dbdefs can help define schemas for several database systems.

• It lets you record comments, which can later help another database user who is not sure which tables and columns contain data that pertains to his task.

It lets you choose the temporal order of composing or changing parts of the definitions:

• Testing proposed definitions for correctness takes place late enough to permit flexibility but early enough to prevent an attempt by the facility to write erroneous definitions to a database.

• You can restructure a draft definition of a schema that does not already exist in a database.

Dbdefs is consistent, which makes it easier to learn and use:

• The relational and entity-relationship interfaces are very similar.

• For relational definition, the one interface integrates the logical-level information that generates four types of statements in the SQL database language (Create Table, Create Index, Label, and Comment).

• The screens give you similar sets of functions and similar invocations of the functions when appropriate.

Our prototype, Dbdefs, integrates features that reduce the amount of time, typing, and memorization required to define a schema.

• Creating a new definition and displaying (and optionally changing) an existing definition use the same screen format.

Overview

Dbdefs is a prototype we developed at IBM Thomas J. Watson Research Center. We have implemented all the features this article describes, except some extensions described at the end of the article. Dbdefs runs on the CMS operating system, which is part of the VM system for IBM 370 computers.

Erlang[3] is our entity-relationship system. Erlang is a successor to the EAS-E system.[4] Both systems provide an integrated programming language and database system. We wrote Dbdefs in the Erlang programming language.

Erlang is a front end to IBM's Structured Query Language/Data System, a relational database system. For both the relational and entity-relationship interfaces, Dbdefs generates tables and indexes for a SQL/DS database. Characteristics of SQL/DS determined our facility's version of the relational data model and influenced our version of the entity-relationship model.

The EAS-E system was developed from 1977 to 1985 with IBM 3270-class terminals. Erlang shares much of the code of EAS-E, operates in the same environment, and supports the same terminals. Dbdefs uses colors extensively on the 3279 terminal, but it also supports other 3270-class terminals. A 3270-class terminal includes keys that move the cursor among the fields you can type in.

Organization. You invoke Dbdefs from CMS by typing "dbdefs." Figure 1 shows the two branches of Dbdefs. Figure 1a shows the screens and transitions for the relational interface; Figure 1b shows the screens and transitions for the entity-relationship interface.

With one exception, the screens obey a stack discipline—you eventually return to the previous screen or to CMS. The exception is that from either main menu you return to CMS, not to the previous screen. All the screens in Figure 1 have on-line help, which you access and exit by pressing function keys.

This article neither illustrates every screen, nor shows how we use color to reinforce the identification of parts of a screen. Features that are common to both interfaces are described in the overview section or in the relational section.

Function keys. We tried to assign function keys in each screen similarly. Sometimes the number of available function keys constrained us. We often use the Enter key as a function key. Key 1 always invokes on-line help. Key 3 always cancels the current screen's partially performed actions (if any) and returns from that screen *without* performing any new ac-

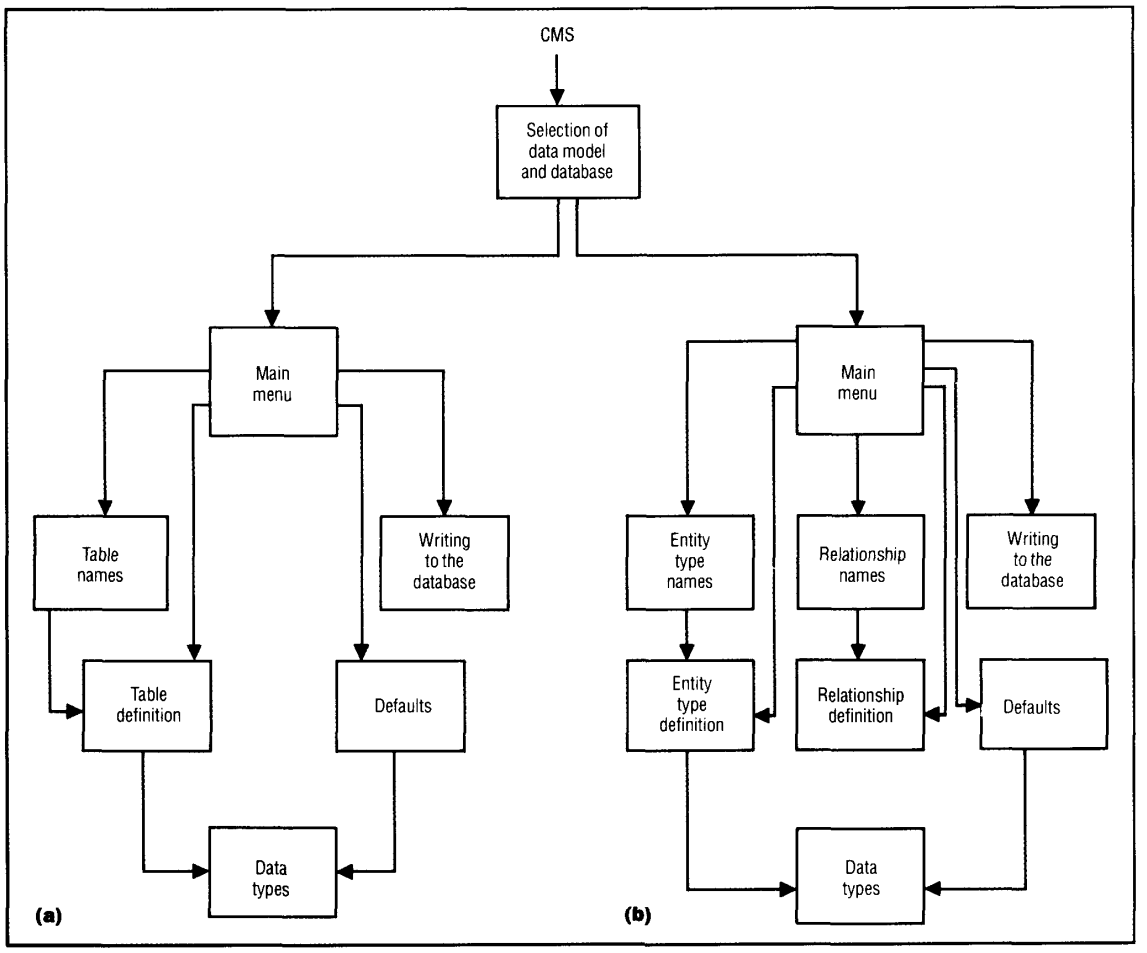


Figure 1. Screens and transitions between screens for the **(a)** relational interface; **(b)** entity-relationship interface.

tions. In several screens, the Enter key returns from that screen *after* performing the appropriate new actions. In all screens with scrolling, key 8 scrolls to the next group of data, and key 7 scrolls to the previous group of data. Many programs on the CMS operating system use this same assignment of keys 1, 3, 7, and 8.

Screen format. Each screen has an identifying header. The bottom of each screen has space for a message. For example, after you delete a table named Employee, the message says "deleted table Employee."

Or suppose you try to define a table with a column whose data type is Char (characters in SQL/DS) and whose size is 300 (which exceeds the SQL/DS maximum for Char). In this case, Dbdefs displays a message that says "size of char must be in range <1, 254>" and moves the cursor to the "300," automatically scrolling to it if necessary.

When Dbdefs finds an error, it explains the error and lets you correct it and perform other actions. Dbdefs does not look for additional errors until you tell Dbdefs to proceed.

Certain actions (such as deleting a table definition) might cause harm if you request them inadvertently, so Dbdefs asks you to confirm a request for such an action, and it performs or cancels the action according to your reply. Again, in each screen, key 3 cancels the screen's partially performed actions (if any) and returns from that screen.

User profile. A Dbdefs user can (but need not) create a profile, a file that customizes the behavior of Dbdefs for that user. A profile can specify

- the data model,
- the name of a database (or a blank to indicate no database),
- defaults for data types and sizes, and
- defaults for the names of files for reading and writing definitions.

An example profile statement is "let default length of char = 25." When you invoke Dbdefs, Dbdefs processes your user profile if one exists. Dbdefs omits the processing if you invoke Dbdefs by typing "dbdefs noprof." Use of a profile can, of course, reduce the amount of typing during a Dbdefs session.

Selection of data model and database. This initial screen lists the two available options: Define tables (which selects the relational data model) and define entity and relationship types (which selects the entity-relationship model).

This screen also includes a field for the database name after each option. You select a data model and a SQL/DS database by typing the name of the database after the name of the option and then pressing the Enter key to proceed to

the main menu.

You select a data model without using a database by leaving the database name blank (with the cursor at that model's name field) before pressing Enter. Dbdefs can read and write files of definitions even if no database is specified.

If the user profile specifies both the option and database name (which might be blank), Dbdefs proceeds to the main menu without first showing the screen for selection of data model and database. When a user expressed a desire to use Dbdefs to define entity-relationship schemas for a system other than Erlang, we included the ability to omit the use of a database and the ability for a profile to specify the option and database name.

Relational interface

In the relational data model, a schema specifies tables to describe the data structure. Each table has columns. For example, an Employee table might contain columns for Name, Salary, and Project. Each column has a data type and (if necessary) a size. For example, a Name column might contain 40 characters.

A table's rows contain values (for example, a Salary of $40,000) that conform to the table's specification in the schema. A schema might specify that data rows of a particular table cannot contain a null value in a particular column. Also, a schema might specify that no two data rows of a table can equal each other in all of a particular set of columns.

Main menu. Figure 2 shows the main menu for table definition. This screen appears when the user has specified the relational data model and a database named Central, either as part of a profile or from the data model and database selection screen.

To invoke a function, you press a function key or Enter. To read or write a file of definitions, you first type the desired file name and file type following the menu entry if a different name and type appear on the screen.

From this screen you can also go to the screens where you can define new tables, display the names of existing table definitions, write table definitions to a database, and display and optionally change the data-type defaults.

Testing the definitions for correctness here means just checking for distinctness of table names. You can request this test explicitly; Dbdefs will perform it automatically when you request writing to the database.

We perform this test in the main menu after you have defined all of the tables, instead of performing it during table definition. This makes it easier for you to change tentative names of tables while constructing a draft definition of the database. For example, while the names of two tables are being exchanged, the tables might temporarily have the same name.

Dbdefs can read and write definitions in ordinary text files; it just stores definitions in data structures in main storage until you write them to a file or to a database. In Figure 2, the user has typed "payroll erlang" as the file name and file type for both reading and writing. The default pairs of name and type are "input erlang" for reading and "output erlang" for writing; the user's profile could have specified different defaults.

The syntax in the files is a sequence of the SQL statements Create Table, Create Index, Label, and Comment. For writing, Dbdefs indents to enhance readability.

```
          ************************************************************
          *          DBDEFS -- Main Menu for Table Definition        *
          ************************************************************

Fill in file name and type (if read or write file), and press a PFKey or ENTER:


PF4:    Define New Table

PF7:    Display Table Names

ENTER:  Test Definitions                    PF2:  Display and Change Defaults


PF6:    Read Definitions from               PF10: Write Definitions to Database
        File: PAYROLL  ERLANG
                                            PF11: Delete Definitions from Database
PF9:    Write Definitions to
        File: PAYROLL  ERLANG

                                  (Using database CENTRAL  on machine CENTRAL )

Other PFKeys: 1=Help  3=Return
```

Figure 2. Screen for main menu for table definition. "PF" means program function .

```
                    ************************************
                    *     DBDEFS -- Table Definition     *
                    ************************************

TABLE NAME: EMPLOYEE              LABEL:
  COLUMN NAME         DATA TYPE       SIZE  NOT-NULL?  LABEL            UNIQUE?
  |                   |               |     | ,--------'                     |
: NAME                CHAR            40    X                              X
: SALARY              INTEGER               X Annual salary in dollars
:
:
:
:
:
:
:
:
PFKeys:  1 = Help  2 = Data types  3 = Return without defining
 4 = Default data type & size  5 = Show comments on table & columns
 7 = Scroll to previous group of columns  8 = Scroll to next group of columns
 9 = Do prefix command (", D, C, M, F, P)  10 = Insert a column at cursor
 11 = Delete column at cursor  12 = Copy another table's columns at cursor
 ENTER = Go to next table if displaying; define table if defining
```

Figure 3. Screen for table definition.

For reading, indentation is optional. Use of the SQL syntax has these advantages:

1. A person who knows SQL can understand (and even edit) the files with no training. However, use of Dbdefs alone (including its functions for reading and writing files) to define SQL/DS databases does *not* require that you know SQL syntax or examine the files.
2. You can create statements via Dbdefs, perform minor editing, and embed the statements in an application program that uses SQL/DS or another SQL database system.
3. You can extract SQL definition statements from an application program, perform minor editing, and read the statements into Dbdefs.
4. The parts of Dbdefs that write to a file and write to a database share code.

If a user profile or the screen for selection of data model and database named a database, Dbdefs can write the definitions to and delete them from the database.

Table definition. Figure 3 shows the screen for table definition. From this screen, you can define a new table or display (and optionally change) an existing definition. In this example, the user has defined a simple table named Employee. It has two columns: Name (40 characters) and Salary (an integer). Each row on this screen represents a column in the table. Key 5 switches between hiding and showing comments; Figure 3 hides them.

To create or change a definition, you can type the table's name, label, and comment and each column's name, data type, size, label, comment, and Not-Null and Unique signals (which are any nonblank character).

The table name, column names, and data types are required — they cannot be left blank. Some but not all data types have a size. For such a data type, leaving a blank field for size tells Dbdefs to use the data type's default. The table label, table comment, column labels, column comments, Not-Null signals, and Unique signals are optional – they may be left blank.

Below the header for column fields (Column Name, Data Type, Size, Not-Null?, Label, and Unique?), the screen uses a vertical line (or a bent indicator for Label) to point to the beginning of each column field. For example, "Char" is a data type, and "Annual salary in dollars" is a column label.

SQL/DS forbids null values in data rows for a Not-Null column. Both columns in Figure 3 specify an "X" to signal Not-Null. Any nonblank character in the Not-Null field indicates forbiddance of null values; a blank field indicates allowance of null values. We copied the term Not Null from the syntax of the SQL database language, adding a hyphen.

Labels are optional. SQL/DS stores them in its catalogs, which are sets of information that describe the database. An application program that uses SQL/DS can read the catalogs and display labels (if any) when presenting data to the user. Labels can make the presentation easier to understand. The label of Salary in Figure 3 is "Annual salary in dollars."

If you specify one or more columns as Unique, SQL/DS ensures that no two data rows can equal each other in all the Unique columns: Any two rows must differ from each other in at least one Unique column. In Figure 3, the Name column specifies an "X" to signal uniqueness; no

two employees can have the same name. Any nonblank character in the Unique field indicates that the column is one of the Unique columns. A blank field indicates that the column is not one of the Unique columns. Zero or more columns can be Unique. We copied the term Unique from the SQL syntax. A more realistic specification of an Employee table might make Name nonunique and might add a Unique column for an employee's identification number, but we wanted to keep this example very simple.

When Dbdefs shows comments, it lets you examine and create or change the comments for the table and for each column. SQL/DS stores comments, which are optional, in its catalogs. A SQL/DS comment can contain 254 characters; a screen that shows comments can display only one column at a time, so Dbdefs hides comments when you first request the screen for table definition.

To specify a column's data type and size without typing them, you can place the cursor anywhere on a column line and press one of two function keys:

- Key 4 writes a default data type and its size on the screen. Using a profile or the screen for defaults can change the default data type and its size.
- Key 2 displays a menu of the available data types and sizes and lets you select a data type and a size, which Dbdefs then writes on the screen for table definition.

To insert or delete columns, you can select a column (by placing the cursor anywhere on that column line) and then press one of three function keys:

- Key 10 pushes down the selected column and all successive columns and inserts a blank column at the cursor.
- Key 11 deletes the selected column and pulls up all successive columns.
- Key 12 copies another table's columns (including column names) at the selected position and the successive positions (after pushing down any columns). Dbdefs prompts you to choose between entering the other table's name and canceling the copy. The copy function, also suggested by prospective users, makes it easier to
- define two or more tables with identical or similar columns (permanent and temporary employees, for example),
- move columns from one table to another, and
- combine two tables or split one table in two.

Dbdefs also provides *prefix* commands to duplicate, delete, copy, and reorder columns. You type a command over the colon that precedes a column's name and (if necessary) type a position over the colon that precedes another column's name. Key 9 invokes the command. The available commands are:

" — Duplicate the selected column after that column.

D — Delete the selected column. This is redundant with the deletion function key.

C — Copy the selected column to the indicated position. The new column need not follow the column copied.

M — Move the selected column to the indicated position.

Dbdefs can read and write definitions in ordinary text files; it just stores definitions in data structures in main storage until you write them to a file or database.

For copying and moving, the available positions (with respect to another existing or blank column) are:

F — Following this column.

P — Preceding this column.

Duplicating and copying make it easier to define several columns with similar characteristics. Dbdefs initially assigns a blank column name to the newly created column, because a table's columns must have distinct names. For example, typing a "C" over the colon before "Name" and a "P" over the colon before "Salary" would push down the Salary column and create a new column between Name and Salary with a blank column name, a 40-character data type and size, a blank label, and Not-Null and Unique signals.

Keys 7 and 8 scroll through groups of columns. For any screen with scrolling, if more groups of nonblank data follow, Dbdefs tells you that more data follows.

Pressing Enter returns to the previous screen after defining a new table or changing (if the user specified changes) an existing definition. Dbdefs automatically tests for correctness (such as distinctness of column names and validity of data types) within the table definition. Deferring this test to the end of table definition makes it easier for you to change tentative names of columns while constructing a draft definition of a table. Key 3 returns without creating or changing the definition.

From the screen for table names, you can request the display (and optional change) of the definitions of all the tables successively. If Dbdefs reached the screen for table definition via such a request, leaving this screen by pressing Enter changes this table's definition (if you specified changes) and also causes the display of the next table (if any). But leaving this screen by pressing key 3 ends the succession of displays of (and possible changes to) tables. Thus, for any method of reaching this screen, Enter tells Dbdefs to proceed with the actions that you have started, and key 3 tells Dbdefs to abort those actions.

Data types. Figure 4 shows the screen for data types. You view the menu of the names and sizes of the data types in this release of SQL/DS. For each data type that has a size, the text at the right shows the allowed sizes and current default. Thus you need not remember the set of available data types or their allowed sizes.

You can select a data type by pressing Enter after performing any of these actions:

- Typing a size for the data type (near the center of the screen) if the data type has a size.
- Typing any nonblank character at the right of the data type's name (below the arrow in the prompt that begins "Mark a data type").
- Leaving the cursor in the data type's size area (if any) or at the right of the data type's name.

If the data type has a size, you can type a size or can select the current default size by not typing a size. In this example, the user has prepared to select Char (by typing an "X" at the right of Char below the arrow) and has typed a length of 40 characters. In

this example, the "X" is not necessary, because the "40" suffices to select Char. Pressing Enter returns with selection of a data type and size. Pressing key 3 returns without selecting and does not write a new data type and size on the screen to which you return.

Table names. The screen for table names shows the names of all definitions. Function keys scroll through groups of names. You can move the cursor to select a name and can then press one function key to delete the selected definition or another key to invoke the screen for table definition to display (and optionally change) the selected definition. Another key (with the cursor anywhere) successively invokes that screen to display (and optionally change) all definitions. Another key invokes that screen to define a new table.

Writing to the database. The data area in a SQL/DS database consists of named portions called *Dbspaces*. Each table in a database resides in a Dbspace. The SQL statement that writes a table definition to a database has an optional parameter for the Dbspace. For later reading and writing of a table's data, a SQL/DS user does not specify (or even need to know) a table's Dbspace. Dbdefs does not invoke the SQL/DS facilities for creating a Dbspace, so at least one available Dbspace must exist before you can write to the database.

Half the screen for writing to the database displays the names of the available Dbspaces. For each table that this Dbdefs session has defined, the other half also displays the table name, an indication of whether the table already exists in the database, and a field for the name of the Dbspace in which the table will or does exist. Function keys scroll through the two halves separately.

For writing to the database, you specify a Dbspace name for each table definition. You can type these names, or you can move the cursor to a Dbspace name in either half of the screen and press one function key to fill in all blank Dbspace name fields with the selected name.

Dbdefs initially fills in a Dbspace name for each table that exists in the database. It also initially fills in a Dbspace name for each table that does not exist in the database if only one available Dbspace exists or if any of this Dbdefs session's tables exist and all are in the same Dbspace. However, you can type over a name that Dbdefs filled in.

In the 1986 ANSI standard for SQL,[5] the Create Table statement has a Unique option. In our release of SQL/DS (for which Dbdefs generates statements), the Create Index statement has the Unique option. When writing to a file or to a database, Dbdefs generates a Create Index statement with the Unique option for each table with Unique columns. It generates no Create Index statement for a table without Unique columns.

When reading from a file, Dbdefs accepts Create Index statements with or without the Unique option, but it ignores any that omit the option. Uniqueness is our only reason for accepting and generating Create Index statements.

One key returns to the main menu after writing to the database all table and index definitions that did not already exist there. Dbdefs does not rewrite definitions that already exist.

```
                ********************************************
                *     DBDEFS -- Select Data Type and Size      *
                ********************************************

Mark a data type | or place cursor next to it.  Fill in size or use default.
                 V
INTEGER:                                      31-bit binary integer.
SMALLINT:                                     15-bit binary integer.

DECIMAL:            Precision:    digits.     1-15 digits; default is 9  .
                    Scale:    dec. places.    0-15 digits; default is 0  .

FLOAT:                                        Double precision (8 bytes).

CHAR:             X Length: 40  characters.   1-254 characters; default is 4   .
VARCHAR:            Max. length:     chars.   1-254 characters; default is 254 .
LONG VARCHAR:                                 Maximum 32767 characters.

GRAPHIC:            Length:     characters.   1-127 characters; default is 4   .
VARGRAPHIC:         Max. length:     chars.   1-127 characters; default is 127 .
LONG VARGRAPHIC:                              Maximum 16383 characters.

PFKeys:  1 = Help  3 = Return without selection  ENTER = Return with selection
```

Figure 4. Screen for data types.

Defaults. This screen displays the defaults for data types. You can type over any of them and can then press Enter to change the current defaults and return to the previous screen. A user profile can also customize the default values.

The screen for defaults includes the default size for each data type that has a size. The *current* default sizes appear during any invocation of the screen for data types, as shown in Figure 4.

The screen for defaults also includes the default data type and its size. To define them, you can press one function key to use the screen for data types. Earlier we described how key 4 in Figure 3 can write the current default data type and its size on the screen for table definition.

During the display of the screen for defaults, the *current* defaults are unchanged until you press Enter, so you can type proposed new defaults here without *yet* affecting the *current* defaults that appear on the right in the screen for data types.

Entity-relationship interface

In the entity-relationship data model, a schema specifies entity types and relationship types to describe the data structure.

Each entity type has attributes. For example, an Employee entity type might contain attributes for Name and Salary. Each attribute has a data type and (if necessary) a size. For example, a Name attribute might contain 40 characters.

In the Erlang version of this data model, each relationship type connects two entity types: a source and a target. For each relationship type, the schema specifies whether the relationship is one-to-one, one-to-many, many-to-one, or many-to-many. For example, a Project_Member relationship type might be a one-to-many relationship whose source is a Project entity type and whose target is an Employee entity type.

Entity occurrences and relationship occurrences contain values (a Salary of $40,000, for example) that conform to the specifications of the entity types and relationship types in the schema. A schema might specify that no two occurrences of a particular entity type can equal each other in all of a particular set of attributes.

Several features of the Erlang version of the entity-relationship model differ from Chen's original entity-relationship model:

- Only entity types, not relationship types, have attributes.
- Each relationship type involves only two entity types (perhaps the same entity type). It must be declared as one-to-one, one-to-many, many-to-one, or many-to-many.
- Each entity type must have a key, which consists of only one attribute. No two entity occurrences can have the same value in

Related work

Many of Dbdefs's characteristics appear in Ben Shneiderman's guidelines on user interfaces.[1]

Although there are many interactive facilities for tasks such as database design (especially on personal computers), there are few published research articles about them. Some other systems related to Dbdefs are:

Office-by-Example. An office-information system, Office-by-Example[2,3] includes a relational database system and full-screen capabilities for creating, deleting, changing, displaying, and listing definitions of tables and other objects. From a list of object names, you can display or delete individual definitions. A user's profile can customize the behavior of OBE. You can define a table two ways:

- You press a function key or type a command to create a table skeleton (simple picture). After typing the table's name, you add or delete columns by pressing function keys and typing the columns' names. Each column's data type is character string. All the columns constitute the set of unique columns.
- You type a command or press a function key to create a definition table (a more detailed picture of a table). After typing the table's name, you add or delete columns by pressing function keys and typing the columns' names. Here you can type each column's data type, format for display, and uniqueness signal. Each table must have at least one unique column. You can default any of the information except for table name and column names. Use of a definition table can also change a definition that you created via a skeleton.

IBM's Xedit. A full-screen text editor, Xedit is popular with users of IBM 3270-class terminals. It includes a field for prefix commands before each line of text. Our prefix commands' syntax is a subset of that of Xedit; thus Xedit users can transfer their knowledge to Dbdefs. Xedit also has function keys for scrolling and a key for inserting a line at the cursor position. It has a command for inserting lines of another file into the file that Xedit is currently editing. A user's profile can customize the behavior of Xedit.

ER-Designer. This full-screen facility for defining entity-relationship schemas (from Chen & Associates) provides graphics-based entry and display of entity types and relationships and text-based entry and display of attributes. Its techniques include menus, scrolling, and mouse-based selection. ER-Designer runs on a personal computer; a related product, SchemaGen, uses the output of ER-Designer to generate schemas for any of several mainframe database systems whose data models are not entity-relationship.

Paradox and dBase III Plus. Both Paradox (from Ansa Software) and dBase III Plus (from Ashton-Tate) include full-screen facilities for defining relational schemas. The facilities' techniques include menus, function keys, and scrolling of columns. Their functions include viewing the set of available data types, copying another table's columns, and other functions.

Database Design and Evaluation Workbench. This workbench[4] for database design supports each of several steps of design.[5] The steps in the workbench include requirements analysis, conceptual design (using an entity-relationship model), logical design, and physical design. The workbench supports a graphics-oriented user interface.

References

1. B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley, Reading, Mass., 1987.
2. K.-Y. Whang et al., "Office-by-Example: An Integrated Office System and Database Manager," *ACM Trans. Office Information Systems*, Oct. 1987, pp. 393-427.
3. M.M. Zloof, "Office-by-Example: A Business Language that Unifies Data and Word Processing and Electronic Mail," *IBM Systems J.*, 1982, pp. 272-304.
4. D. Reiner et al., "A Database Designer's Workbench," in *Entity-Relationship Approach*, S. Spaccapietra, ed., North-Holland, Amsterdam, 1987, pp. 347-360.
5. T.J. Teorey and J.P. Fry, *Design of Database Structures*, Prentice-Hall, Englewood Cliffs, N.J., 1982.

the key attribute. A key attribute in Erlang corresponds to a set of Unique columns in SQL.

Most of our screens and functions for entity-relationship schemas (data types, entity type names, relationship names, writing to the database, and defaults) resemble the corresponding screens and functions for relational schemas, so we will not repeat their descriptions here. The main differences appear in three screens: main menu, entity-type definition, and relationship definition. The relational and entity-relationship functions share most of their code.

Main menu. Figure 5 shows the screen for the main menu for entity-relationship definition. This screen appears when the user has specified the entity-relationship data model and a database named "Central."

The menu's header differs slightly from that of the table-definition menu. The menu's entries include defining and displaying entity types and relationships instead of tables. For brevity, we say "relationship" instead of "relationship type." The syntax for files of definitions is an entity-relationship format, not SQL statements – you may define a schema for an entity-relationship system other than Erlang.

In testing the definitions for correctness here, Dbdefs checks for:

- Distinctness of entity type and relationship names.
- Presence of definitions of the source and target entity types of each relationship.

Again, deferring the test to the main menu makes it easier for you to change names of entity types and relationships while constructing a draft database definition. The deferral also lets you define a relationship before defining its source and target entity types.

Writing to (or deleting from) the database consists of creating (or deleting) SQL/DS tables and indexes that store entity types and relationships. Dbdefs maps an entity type with n attributes into a table with n columns and a unique index on the key column. Dbdefs maps a relationship into a table with two columns (for the keys of the source and target) and a unique index on both columns.

Entity-type definition. In this screen, you type the entity type's name. For each attribute, you can type a name, a data type, a size, and a signal for whether the attribute is the key. Exactly one attribute must have a key signal.

The set of available data types differs slightly from that for tables. The prefix commands and all function keys (except the one for comments) match those for table definition.

Relationship definition. Figure 6 shows the screen for relationship definition. From this screen, you can define a new relationship or display and optionally change an existing definition.

In this example, the user has defined a relationship named Project_Member. Its inverse name is Assignment. It is a one-to-many relationship between the Project and Employee entity types.

To create or change a definition, you can type the relationship name, inverse name (which may be blank to indicate no inverse name), source entity-type name, target entity-type name, and type of relationship.

```
          ************************************************************
          *  DBDEFS -- Main Menu for Entity-Relationship Definition  *
          ************************************************************

Fill in file name and type (if read or write file), and press a PFKey or ENTER:

PF4:   Define New Entity Type            PF5:  Define New Relationship

PF7:   Display Entity Type Names         PF8:  Display Relationship Names

ENTER: Test Definitions                  PF2:  Display and Change Defaults


PF6:   Read Definitions from             PF10: Write Definitions to Database
       File: PAYROLL  ERLANG
                                         PF11: Delete Definitions from Database
PF9:   Write Definitions to
       File: PAYROLL  ERLANG

                                (Using database CENTRAL  on machine CENTRAL )

Other PFKeys: 1=Help  3=Return
```

Figure 5. Screen for main menu for entity-relationship definition.

```
                ****************************************
                *    DBDEFS -- Relationship Definition     *
                ****************************************


NAME OF RELATIONSHIP:   PROJECT_MEMBER

NAME OF INVERSE:        ASSIGNMENT


NAME OF SOURCE ENTITY TYPE:   PROJECT                M or 1:  1

NAME OF TARGET ENTITY TYPE:   EMPLOYEE               M or 1:  M



PFKeys:  1 = Help  3 = Return without defining a relationship
12 = Copy source and target information from another relationship
ENTER = Go to next relationship if displaying; define relationship if defining
```

Figure 6. Screen for relationship definition. "M" means many.

Key 12 copies another relationship's type (one-to-one, for example), its source's name, and its target's name. Dbdefs prompts you to choose between entering the other relationship's name and canceling the copying.

The meanings of the Enter key and key 3 correspond to their meanings in the relational interface's screen for table definiton. If you press Enter, Dbdefs automatically tests for correctness within the relationship definition.

Our small community of users has reacted favorably to the features of our prototype. An inexperienced user of databases remarked that he likes the menus, use of color, simplicity of instructions, ability to cancel inadvertent actions, and straightforward creation of database objects.

A more experienced user of databases likes

- the ease of specifying data types and using defaults,
- the minimal memorization and typing that are required,
- the automatic testing for correctness, and
- the screen layout for table definition.

She also suggested improvements, including

- more descriptive prompting, messages, and help information;
- more flexibility in three screens;
- the ability to extract information from table definitions that already exist in a database;
- the ability to define indexes on nonunique columns; and
- revised function-key assignments.

Some other possible extensions are:

- Provide a screen (to replace the current use of a text editor) for creating, deleting, changing, and displaying a profile.
- Expand the capabilities of profiles (and perhaps add a screen) to let you customize the assignment of function keys to functions.
- Expand the set of prefix commands (for example, to duplicate a column or attribute many times).
- For tables that already exist in a SQL/DS database, provide extraction of Dbdefs definition information from the several system catalogs where SQL/DS stores it. This would let you view the stored information in one place. You can browse the information and discover which items pertain to your tasks.
- When writing to the database, rewrite the definition of a table, entity type, or relationship that exists in the database if you have changed the definition. Such schema restructuring[6,7] can, of course, require a major effort (and perhaps additional information from the user) if data values already exist.
- Show you an entity-relationship view of tables that you have defined or a relational view of entity types and relationships that you have defined. Such a mapping between data models[6] can be more complex than the one that we use now for entity-relationship definition.
- Combine Dbdefs with interfaces to other tools, such as for conceptual design, physical design (including indexes on nonunique columns), creation of Dbspaces, and manipulation of data occurrences.
- Provide a workstation-based facility with graphics and mouse-based interactions.

Our techniques can be applied to other facilities. The key features in Dbdefs are the similar interfaces for two data models and the integration of techniques like menus, defaults, function keys, and user profiles. ❖

Acknowledgments

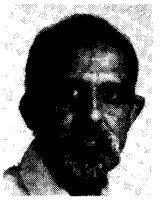
Maria Butrico, Peter Chen, Sham Navathe, Anil Nigam, Kyu-Young Whang, and Stanley Zdonik discussed an earlier draft of the article with us. The referees also suggested several improvements.

References

1. E.F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM*, June 1970, pp. 377-387.
2. P.P. Chen, "The Entity-Relationship Model — Toward a Unified View of Data," *ACM Trans. Database Systems*, March 1976, pp. 9-36.
3. A. Malhotra et al., "An Entity-Relationship Programming Language," Research Report RC 11816, IBM T.J. Watson Research Center, Yorktown Heights, N.Y., April 1986.
4. H.M. Markowitz, A. Malhotra, and D.P. Pazel, "The EAS-E Application Development System: Principles and Language Summary," *Comm. ACM*, Aug. 1984, pp. 785-799.
5. *Database Language SQL*, X3.135-1986, American Nat'l Standards Inst., New York, 1986.
6. G.H. Sockut, "A Framework for Logical-Level Changes Within Database Systems," *Computer*, May 1985, pp. 9-27.
7. S.B. Navathe, "Schema Analysis for Database Restructuring," *ACM Trans. Database Systems*, June 1980, pp. 157-184.

Gary H. Sockut is an advisory programmer at the IBM Santa Teresa Laboratory. His main areas of interest are database management, office systems, and operating systems.

Sockut received a BS in applied mathematics from Brown University, an MS in electrical engineering from the Massachusetts Institute of Technology, and a PhD in applied mathematics from Harvard University. He is a member of the IEEE Computer Society and ACM.



Ashok Malhotra is a research staff member at the IBM T.J. Watson Research Center. His main areas of interest are application-development technology, database management, and object-oriented systems.

Malhotra received a BS and MS in electrical engineering and an MS and PhD in management, all from the Massachusetts Institute of Technology. He is a member of the IEEE Computer Society, ACM, and the Institute for Management Science.

Address questions about this article to Sockut at IBM Santa Teresa Laboratory, P.O. Box 49023, San Jose, CA 95161-9023.