SRI International

TEAM USER'S GUIDE

Technical Note 343

November 1984

By: Lorna Shinkle
Computer Scientist
Artificial Intelligence Center
Computer Science and Technology Division

APPROVED FOR PUBLIC RELEASE:
DISTRIBUTION UNLIMITED

This research was supported by the Defense Advanced Research Projects Agency under Contract N00039-83-C-0109. The views and conclusions expressed in this document are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advance Research Projects Agency.

SRI International®

333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486

Table of Contents

List of Figures

1. INTRODUCTION

TEAM (Transportable English Data Access Medium) is a transportable natural-language (NL) interface to a database. It is a tool of considerable power that enables the user to retrieve data and elicit answers to queries by asking questions and giving commands in English instead of a formal query language. Moreover, TEAM is not limited to any particular database, but can be adapted to demonstrate natural-language retrieval in a broad variety of application domains. The prototype TEAM software described herein was developed by the Artificial Intelligence Center of SRI International to demonstrate the system's capabilities and adaptive potential.

This user's guide is designed to assist new TEAM users to learn about the concepts and tasks involved in retrieving data and in preparing a demonstration for a new application area. An effort has been made to illustrate some of the problems TEAM must solve in translating an English question into a database query. However, the necessarily limited scope of this guide cannot include a discussion of all the natural-language-processing issues addressed by the system; our emphasis is on a practical, rather than theoretical, understanding of the concepts. Similarly, while this guide cannot cover every detail of creating a new demonstration for TEAM, it does provide a thorough introduction to the procedure to be followed and explains how to use the on-line "help" provided by the system.

Other references on the subject of TEAM include two papers: "Transportability and Generality in a Natural-Language Interface System," by Paul Martin, Douglas Appelt, and Fernando Pereira, and "TEAM: An Experiment in the Design of Transportable NL Interfaces," by Douglas Appelt, Barbara Grosz, Paul Martin, and Fernando Pereira. In addition, a videotape has been prepared that gives an introduction to TEAM and a demonstration.

This introductory manual is designed to be read in conjunction with actual use of the system. While a casual perusal of this document may acquaint the reader with some of TEAM's features, using the examples and suggestions as a practical guide to actually experimenting with the system will prove a much more effective method of learning how to use TEAM and becoming familiar with both its capabilities and its limitations. Much of this guide consists of comments, descriptions, and other background information, but the user is frequently instructed to perform some action as a learning experience. In the examples shown in the text, the portions printed in boldface are typed or selected by the user; these portions may or may not appear in boldface on the screen when TEAM is used. In the examples illustrated by figures, however, the type faces do correspond exactly to the screen display.

2. BACKGROUND

TEAM is a system that allows its users to retrieve information from a database by stating requests in English (which is considered a "natural" language because it is normally used by some human beings to communicate with other human beings) rather than by encoding them in one of the many formal query languages that have been specifically designed for data retrieval. This chapter gives some background on TEAM, its goals and their implications, the two classes of TEAM users, and the major components of the system. Our intention is to provide the user with a perspective that will be helpful in reading the following chapters and in understanding TEAM's capabilities and limitations. For a comprehensive and more technical discussion, see the papers cited in the preceding chapter.

To appreciate the achievements embodied in TEAM, one must first know two fundamental facts about the system:

- TEAM is a research prototype, not a commercial product
- TEAM is a retrieval interface, not a database management system.

Each of these points is amplified below.

TEAM is the concrete result of basic research in natural-language processing; as such, it can be used as a tool for demonstration and teaching. Because the research effort included concern for the needs of users who are not linguists, TEAM has some elegant user-oriented features. However, the problems that must be resolved for true natural-language processing (as opposed to keyword processing) are very broad and difficult ones, and TEAM was developed and modified over time as a vehicle for exploration of the attendant issues. While the TEAM project has expedited significant advances in this area of research, TEAM is not a finished product that can be utilized in a production environment. Consequently, users must be prepared to encounter occasional difficulties in using the system.

Nor should TEAM users expect to find all the features of a database management system, such as easy data entry, protection of data integrity during concurrent access, ability to embed queries in a programming language, or efficient processing of large quantities of data. The only data management feature that TEAM is concerned with is retrieval (although a basic data-entry capability is provided), because the problems involved in retrieval through a natural-language interface were sufficiently challenging to warrant a significant research project. Advances were made in our understanding of the linkage between linguistic concepts and data concepts, as well as in natural-language processing. In the future, this progress may provide the basis for creating effective interface components of database management systems.

TEAM also embodies other consequences of the two project goals: *natural-language retrieval* and *transportability*. Some implications of these goals are touched upon below.

Because TEAM is designed to accept questions in English and translate them into precise queries on a database, extra processing (beyond that required for a formal query language) must be performed by the system in at least two major areas:

- TEAM must resolve the linguistic issues inherent in translating an English sentence, rather than require the user to indicate all data relationships and operators in precise, unambiguous terms.

- TEAM has features that attempt to shield the user from having to know all the details of the database structure (such as the abbreviated or coded names of data fields, or whether more than one file is involved in a query).

Partly because of this extra processing load, the criteria for using a natural-language interface such as TEAM in a worthwhile manner must be based on the following considerations:

- The questions that are typically asked of the system must be "properly" expressed in a natural language such as English. First, typical queries must be "easy to say," rather than easy to program. Second, some questions should take advantage of the power of a natural language, such as the ability (1) to express complex operations or relationships in relatively few words and (2) to combine or modify requests in numerous subtle ways.

- The fact that the user already knows English, and so does not have to spend time learning a different query language, is an obvious advantage. An interface like TEAM is especially suited to the casual user who is willing to make the system do a great deal more work to spare himself the time and/or frustration involved in learning a language tailored to data retrieval. On the other hand, TEAM would not be appropriate for running standard queries that are composed once but executed many times.

The feature of transportability, or TEAM's potential for being moved to a new application domain by a domain expert who is not a linguist, was a major factor in determining the design of the system.

- From the processing standpoint, transportability requires that the internal mechanisms used to resolve linguistic issues be completely domain-independent. This means that no coding shortcuts could be taken to solve a particular problem for a single domain.

- From the user's perspective, transportability implies concern with the way in which information about a new domain is "acquired" by TEAM. This includes care regarding the user interface in general -- apparent in such features as prompts, command menus, and on-line "help" facilities, as well as concern with the way linguistic information is elicited from someone who is not familiar with linguistic theory and jargon -- by using graphics, questions containing examples, and other such techniques.

These points about the implications of natural-language retrieval and transportability have been presented here as an essential preface to the actual process of learning to use TEAM. It is hoped that they will be reinforced by the examples and discussion in the following text.

It will also be helpful for you to know that there are two classes of TEAM users (the end user and the domain expert) and to understand something about the system's different

components. The diagram in Figure 2-1 shows the end user interacting with TEAM by entering a natural-language query and then receiving an answer. The domain or database expert (drawn wearing a wizard's hat) engages in a more extensive interaction, since he must supply all the information required by TEAM before it can respond to questions in a new domain. (In this user's guide, the term "domain" is used in two different but essentially connected ways: in a broad sense, to denote an application area or field, or in a narrow sense, to denote a particular application-related collection of data and linguistic information that has been supplied to TEAM so that end users can query it.) The interaction between the domain expert and the system is called *acquisition*, since it is the process by which TEAM acquires domain-specific information.

In Figure 2-1, the large square surrounds the parts of TEAM that are involved in translating a query from English into a query language. The three major components are represented by rectangles. The NL processor and the schema translator are within the square because they are involved in translating a query, while the acquisition component is outside the square because it does not participate in query translation. Within the query translation square are several ovals, which represent categories of information used by TEAM. Note that the grammar rules are independent of the acquisition process, while the other information is at least partially dependent on any specific domain.

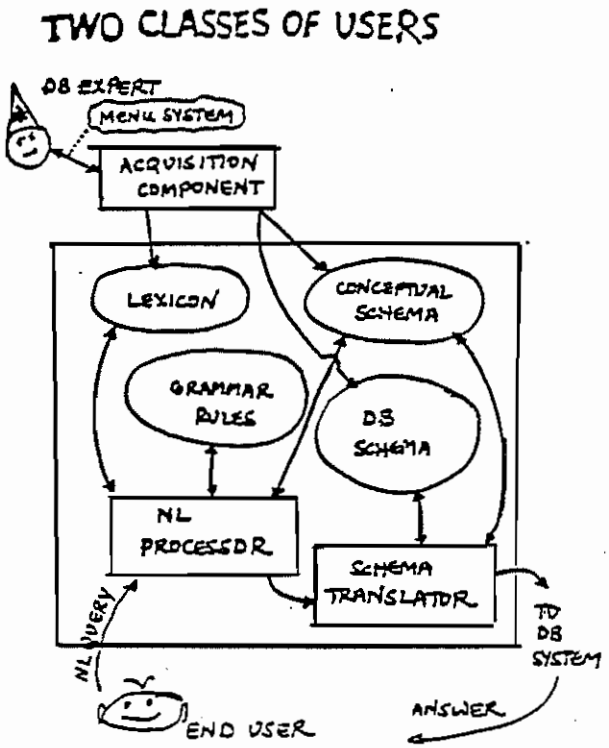


Figure 2-1: Two Classes of Users and the Components of TEAM

The actual processing of the query, once it has been translated into a formal query language, is accomplished by a separate software package called SODA. (The words "TO DB SYSTEM" in the diagram reflect the action of sending the query to SODA). SODA was developed during previous work at the Artificial Intelligence Center to provide a standard

relational interface to a variety of database systems. The demonstration version of TEAM uses an "in-memory" version of SODA (which keeps tables of data in the Lisp Machine memory) as the formal query system to which the natural-language queries are translated.

The NL processor, schema translator, and SODA are diagrammed again on the right side of Figure 2-2. The left side of the latter shows the stages a query goes through as it is processed, starting with the input sentence and ending with the answer. It is not necessary to completely understand these processing stages before one can use TEAM, but a little familiarity with the terminology would be helpful.

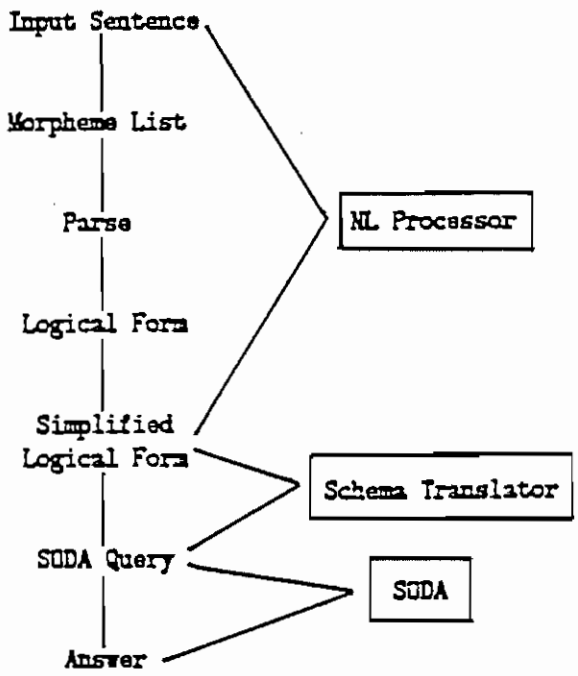


Figure 2-2: Processing Stages

The NL processor first tries to parse the sentence to determine its basic grammatical structure. In the process, it looks up the English words in its lexicon (or vocabulary list) for the domain and transforms them into "morphemes," which are the smallest units of meaning in linguistic analysis. (For example, the NL processor will transform the word "are" into the pair of morphemes "PRES2 BE," meaning the present plural of the verb "to be.") If there is more than one morphological analysis of a word, the morpheme list will be shown a corresponding number of times. (For example, a query that uses the word "more" will have at least two morpheme lists, one with the morpheme "ER MUCH" and one with the morpheme "ER MANY.")

If TEAM is successful in parsing the sentence, it produces one or more parses (or parse "trees"), and ranks them according to likelihood. The NL processor then applies "semantic," "pragmatic," and "scoping" routines to the top-ranked parse to arrive at a "logical form," which represents TEAM's understanding of the sentence's meaning in terms of the concepts of the application domain. If the top-ranked parse fails during any of the further processing, TEAM then tries to transform the next best parse into a logical form.

The logical form is simplified before it is turned into a formal query; this simplification removes nuances of language that are too complex for the current version of the schema translator. It is then the task of the schema translator to transform the simplified logical form into a SODA query. The schema translator is the only part of TEAM that is written in PROLOG, rather than LISP. In the demonstration version of the system, SODA processes the query using the in-memory database, and the answer is returned to the user.

The foregoing brief explanation of TEAM's components and processing stages concludes this background presentation. As mentioned earlier, the references given in the preceding chapter contain more information about TEAM from the theoretical perspective. The following text introduces various features of the system, with Chapter 3 emphasizing the end user and Chapters 4 and 5 focusing on the domain expert.

3. USING TEAM TO ACCESS A DATABASE

This chapter is oriented primarily toward the TEAM end user, who will ask questions in English regarding the contents of a database. The discussion and examples are intended to provide him with an introduction to two aspects of the system: *mechanics* and *concepts*. Thus, this chapter includes not only instructions and hints on using TEAM, but also explanations and examples to help the user perceive some of the underlying linguistic issues and problems.

This chapter also serves as an introduction for the domain expert, since the process of adapting TEAM to a new domain requires a solid understanding of the system's general power and limitations, as well as of the specific ways in which TEAM may be employed in the new domain. Our subsequent discussions on acquisition and demonstration design will build upon the concepts highlighted in this chapter.

As mentioned in the introduction to this user's guide, one cannot learn to use TEAM simply by reading. The descriptions and examples provided here are designed to guide the user through the process of becoming acquainted with TEAM, but it will be essential for him to actually apply the examples in practice -- to experiment with the features of both TEAM and the Lisp Machine environment in which it runs.

3.1 Getting Started

Since TEAM currently runs on a Lisp Machine, the TEAM user must understand some features of the Lisp environment. Those users who are not familiar with the Lisp Machine features should consult either a local expert, the *Lisp Machine Summary* from Symbolics, or a similar document for the appropriate machine. (A few comments are provided here, but they are intended as guidelines, not as a full introduction to a Lisp Machine.) In using TEAM, it will be important to do or know about the following:

- Find out how to get access to a machine, start it, log in with a user name (your own or one you have permission to use), and log out. Instructions for powering up the machine, booting it, and logging in and out are included in the Lisp Machine Summary, but there may also be local practices (such as leaving the machine on even when it is not in use, but reducing screen brightness).

- Learn how to load and start TEAM on the machine you are using. Since this procedure can vary, it is not documented here; check the instructions that came with TEAM or consult with someone who knows about running TEAM at your site.

- The Lisp Machine Summary explains the layout of the screen, including the *status line* at the bottom. A code on the status line indicates whether the machine is running or waiting for the user to type something.

- The Lisp Machine Summary also describes the *mouse*, which is a rolling device used to move a cursor on the screen; it has three buttons on top that can be clicked to issue commands. The effect of clicking one of the buttons will depend on where the mouse cursor is on the screen at the time. A prompt showing what mouse commands are

available at any given moment is located in the *mouse documentation line* or *prompt window*, which is a one-line window in inverse video at the bottom of the screen. The mouse cursor may take on different forms, including that of a small arrow or a circle with a cross in it; occasionally the cursor will turn into an hour glass, indicating that some especially time-consuming task is in progress.

A part of the screen (for example, a word in a menu of commands) may be *mouse-sensitive*, as indicated by a box (hollow or filled) that appears around that part of the screen when the cursor is touching it. The mouse commands that appear in the prompt window at that time apply to the mouse-sensitive object highlighted on the screen. By convention, one click of the leftmost mouse button is used to select a mouse-sensitive object or command from the screen when there is no explicit indication as to which button must be used. (Also by convention, in many circumstances pressing the rightmost button causes a menu of options to appear.)

The mouse can also be used to *scroll* the contents of some windows up or down. The acquisition window, which contains still smaller windows that display lists of words, will be introduced later. If a list is particularly long, only a portion of it will be visible in the window at any one time. However, bumping the mouse cursor against the left side of the window will turn it into a double-headed arrow. The mouse buttons can then be used as indicated in the mouse documentation line to move the contents of the window up and down.

- A few keyboard conventions are important. The following *modifier* keys are designed to be held down while depressing other keys (in the same way that the SHIFT key is used): CTRL (pronounced control), META, SUPER, and HYPER. The combination of one or more modifier keys and another key is usually represented by a dash or dashes; thus CTRL-X means hold the CTRL key down while pressing the "x" key, and CTRL-META-ABORT means hold both the CTRL and META keys down while pressing the ABORT key. The modifier keys are also often indicated in other texts by one-letter abbreviations; thus, c-x is the same as CTRL-X.

  In contrast to the modifier keys, those with white lettering -- such as HELP, ABORT, RESUME, SELECT, and FUNCTION -- are meant to be pressed in *sequence* with other keys, in the same way that the alphabetic keys are normally used. The four examples below illustrate the use of these keys. As mentioned earlier, boldface indicates sequences that are to be typed by the user.

  - **SELECT :** (press and release the SELECT key, then press and release the colon key) will select the main TEAM window, bringing it into view on the screen.

  - **SELECT HELP** will print a list of the keys that can be used with the SELECT key and a description of what each combination does.

  - **FUNCTION C** will change the state of the screen from black-on-white to white-on-black or vice versa. Repeating this key sequence will "toggle" between the two states.

- **FUNCTION HELP** will print a list of the keys that can be used with the FUNCTION key and a description of what each combination does.

The RETURN key is used to signal the end of a TEAM input line. Whenever TEAM is waiting for input from the user, a flashing cursor is displayed on the input line; it will disappear as soon as the RETURN key is typed. The RUBOUT key can be used to backspace over a typing error or the CLEAR INPUT key can be used to clear the input line entirely. Some simple commands for ZMACS, the Zetalisp editor, can also be used to edit the input line, but the cursor must be returned to the end of the input line. (An introduction to ZMACS is presented in the Lisp Machine Summary.)

- The Lisp Machine Summary section on *path names* describes the components of a directory or file name on the Lisp Machine. TEAM requires a path name when an already established domain is loaded by a domain expert or an end user, as well as when a domain is saved by a domain expert.

- The Lisp Machine Summary includes other helpful information, such as how to recover from errors and frozen states. Anyone unfamiliar with Lisp Machines is encouraged to read it through before using TEAM.

Once TEAM has been loaded and started, the main TEAM window will appear on the screen. Figure 3-1 shows this window (with the result of the HELP command displayed). If it does not appear, type **SELECT :** to bring it into view. Note that the prompt window contains the words *Type an English query or select a command from the menu*, as shown in Figure 3-1. The prompt *TEAM>* appears on the screen to indicate that TEAM is ready to accept a question; the cursor is in the form of a circle with a cross inside it. The main TEAM menu extends across the top of the screen (in inverse video) and contains the following commands:

ACQUIRE
SAVE
LOAD
PROFILE
RESTORE
EDIT
DDBG
HELP
QUIT

Each command is mouse-sensitive, so touching it with the cursor will cause a box to appear around it and a sentence to appear in the prompt window at the bottom of the screen.

Positioning the cursor over the **HELP** command and clicking the leftmost button once will cause TEAM to print a message on the screen explaining what each command does, as shown in Figure 3-1.

The ACQUIRE, RESTORE, and EDIT commands are for use by the domain expert rather than the TEAM end user, so they will be covered in Chapter 4. The DDBG command calls the Diamond Debugger, which is used to examine and edit parse trees (TEAM's internal

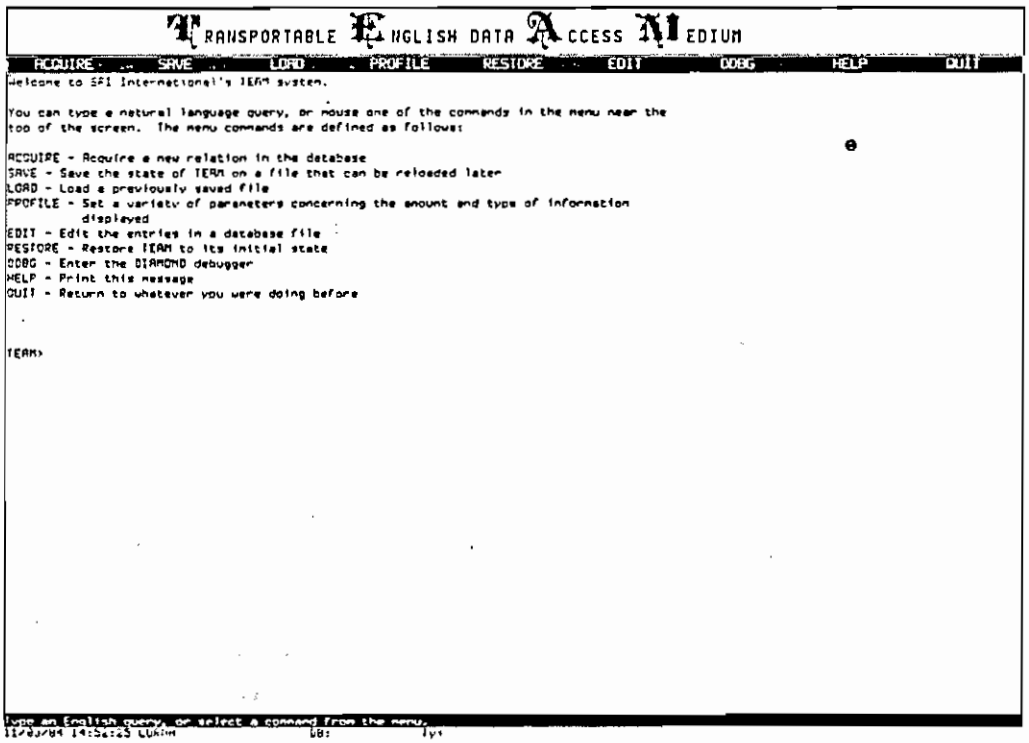


Figure 3-1: Effect of the HELP Command

representation of the parsed query); it is intended for use in debugging and changing TEAM itself, and is not documented in this user's guide.

The PROFILE command allows the user to specify what he wants TEAM to print on the screen as it is processing a question; this command will be explained in Section 3.3. The QUIT command simply leaves the TEAM window and returns to the preceding window.

SAVE and LOAD are opposite commands: SAVE preserves a *TEAM state* as a collection of files; LOAD, on the other hand, retrieves those files and returns TEAM to the state it was in when they were saved. The SAVE command will not be used until Chapter 4, since the domain expert employs this command after he has given TEAM the information it must have to answer questions related to a particular database. The LOAD command, however, may be used by both the domain expert and the end user, since it prepares the system either for further work on a domain or for questioning.

The RESTORE command is used to return TEAM to a pristine or "empty" state, that is, to the state it was in when it was first started. This may be necessary when changing states or domains. Normally the LOAD command, as it loads a new domain, will "write over" the information that was specifically related to a previous domain. However, it is possible for TEAM to get into a disrupted state (for example, when a domain expert tries to save an inconsistent domain); in that case, the RESTORE command must be given before a new domain can be loaded.

Now load the sample domain called *pkcont* that was supplied with TEAM. (This domain is concerned with mountain peaks and countries.) To load the TEAM state, select the **LOAD** command from the main menu (by positioning the cursor over the command until the box appears and then clicking the leftmost button once). TEAM will respond by asking for the path name of the file to be loaded. Answer by typing the path name **>teamdemo>pkcont.acq**, followed by the **RETURN** key, as shown:

What file do you want to load? **>teamdemo>pkcont.acq**

If this path name does not work, check the instructions that came with the system or ask a local TEAM user what path name to use. (No machine name is included in the path name in this example, since it is assumed that the "teamdemo" directory is on your machine; if it is located on a different machine, include the identifier for that machine at the front of the path name.)

The file specified for the LOAD command should always be of type ".acq", indicating that it is an *acquisition file* produced by TEAM via the SAVE command. In fact, the ".acq" at the end of the path name is not necessary, since the system will supply it. TEAM will also load all of the database files (that is, the files with type ".db") that are needed for the state being loaded; the acquisition file tells TEAM which files are needed. In this example, *peak.db* and *worldc.db* will be loaded.

After TEAM has successfully loaded the files, it will momentarily flash a window on the screen that contains the acquisition information -- then display the main window with a clear screen and the TEAM> prompt. It is now ready to answer questions about the sample domain.

3.2 Simple Questions

To use TEAM effectively for retrieving facts from a database, one must have some idea of what that database is about. This is true despite the fact that a natural-language interface is most suitable for those who want to get information without knowing much about the database structure and without any knowledge of a formal query language. TEAM does several things to make it easier for the novice user to ask questions, but it is obviously impossible for TEAM to convert a question about cities (for example) into a database query if the database doesn't contain any information in that category.

Ideally, a database management system with a natural-language interface would also have a "help" facility that would allow users to find out which databases contained information of potential interest to them. However, because TEAM was designed to address the specific problems of the retrieval interface, rather than all those entailed in an integrated system, the user must know what kinds of information a database contains before he tries to query it.

In the case of the sample domain that was loaded, there are two types of things that one can get information about: peaks and countries. Each peak has a name, a country, a height, and an indication as to whether or not it is volcanic. Each country has a name, a continent, a capital, a population, and an area. Figures 3-2 and 3-3 summarize this information in tabular form to make it easier to refer to. They also show some sample values from the database.

Name	Country	Height	Volcanic?
Fuji	Japan	12026	Y
Klyuchevskaya	USSR	15250	Y
Annapurna	Nepal	26504	N
Kanchenjunga	India	28208	N
K2	Pakistan	28250	N
Everest	Nepal	29028	N
Matterhorn	Switzerland	14690	N
Kilimanjaro	Tanzania	19340	Y
Mont Blanc	France	15771	N

Figure 3-2: Sample Information About Peaks

Name	Continent	Capital	Population	Area
China	Asia	Beijing	968230000	3691502
India	Asia	New Delhi	638390000	1229737
USSR	Asia	Moscow	258930000	8647250
USA	North America	Washington	219484000	1256000
Britain	Europe	London	55820000	94209
Nepal	Asia	Kathmandu	13420000	54362
Japan	Asia	Tokyo	114900000	143574
Switzerland	Europe	Bern	6340000	15941
Tanzania	Africa	Dar-Es-Salaam	16070000	363708
France	Europe	Paris	53280000	211000

Figure 3-3: Sample Information About Countries

To introduce the basic mechanics of querying TEAM, this section starts with a few questions that are very simple. Try typing the following sentence after the prompt in the main TEAM window:

TEAM> **what are the countries**

Note that there is no question mark at the end of the example and no capital letter at the beginning -- TEAM ignores capitalization and does not require any punctuation except commas. (Names of countries, capitals, and mountains are capitalized in later examples for the sake of clarity, but this is not necessary.) Remember that the RUBOUT and CLEAR INPUT keys can be used to correct typing mistakes, and be sure to terminate the input line by typing RETURN.

Figure 3-4 shows some of the text that TEAM may generate on the screen as it is processing the query. What you see on your screen may differ, depending on the way your user profile is set; this will be explained in the next section. Note that, before supplying the the answer to your query, TEAM tells you to *Type any character to proceed.* This message gives the user an opportunity to look at the processing output before it is overwritten by the system's reply. In this example, the answer will be the list of countries in the world; this list will fill several screenfuls.

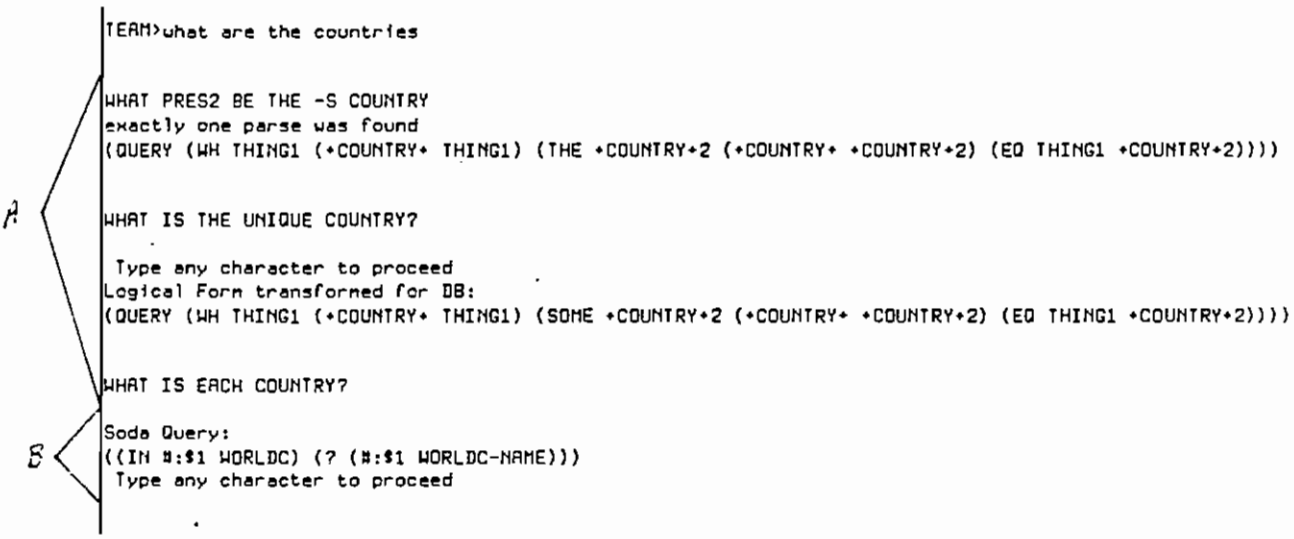


Figure 3-4: Sample Output During Query Processing

The next two sections will discuss the mechanics of setting the user profile and handling various kinds of errors. The following sentences are simple ones to try while experimenting with the profile options and the procedure for correcting spelling errors.

TEAM>**show the peaks**

TEAM>**What is the height of each mountain in Nepal**

TEAM>**show the population of Asia's countries**

While experimenting, try using the following commands (when you see the TEAM> prompt) to repeat a query you have given before:

- CTRL-C will bring back the last question typed
- After CTRL-C has been used once, META-C can be used repeatedly to cycle through the previous questions.

Once the desired query appears, type the END key (or the RUBOUT key, followed by the RETURN key) to signal TEAM to process it. The retrieved query may also be edited by using the RUBOUT key or certain simple ZMACS commands.

3.3 The User Profile

The user profile, illustrated in Figure 3-5, is simply a list of control features that specify what output TEAM should produce as a query is processed. When TEAM is started, it reads the current profile specifications from a file, but the user can change them at any time by using the PROFILE command from the main menu.

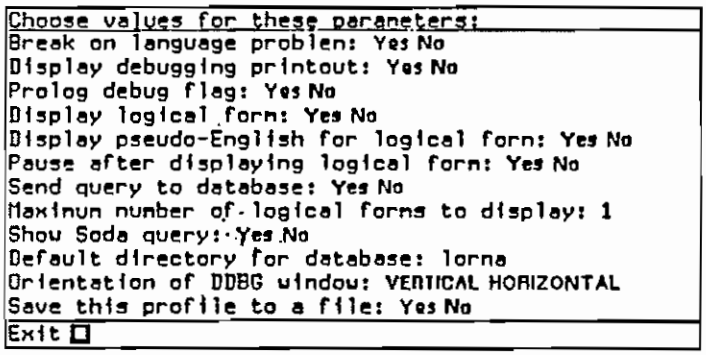


Figure 3-5: The User Profile

To understand the profile options, it is helpful to know what parts of the processing output correspond to the processing stages that were diagrammed in Figure 2-2. The figure shows that the NL processor transforms the query from English into a *logical form*, which is then passed to the schema translator to be converted into a SODA query. Finally the query is passed to a version of SODA, which returns an answer from the database.

Refer back to Figure 3-4 in the previous section, which shows an example of the processing output. The NL processor produces the part of the output labeled "A" in that figure while it turns the English sentence into a logical form. The first line, *WHAT PRES2 BE THE -S COUNTRY*, shows the morphemes that were retrieved by looking up the words in the lexicon. A sequence of morphemes is displayed on the screen as the NL processor tries to parse the sentence. Although there is only one list in this example, multiple lists of morpbemes for a sentence are possible. The second line in the diagram indicates that only a single parse was found. Usually more than one parse is found and all are ranked by the NL processor. Only the number of parses, however, is reported to the user.

The next output line gives the logical form that was obtained after semantic, pragmatic, and scope processing of the top-ranked parse. TEAM then restates the logical form in unambiguous but sometimes baroque English -- in this case, *WHAT IS THE UNIQUE COUNTRY* -- to help the user decide whether the system is interpreting his query the way he intended. The statement *Type any character to proceed* allows him to pause and look at this part of the output before going on. The first logical form is the "true" one from the standpoint of natural-language interpretation. However, the NL processor must simplify it for the purpose of database access. The second logical form and its restatement in pseudo-English are also shown in Figure 3-4, preceded by the line *Logical Form transformed for DB.* In this case, the second logical form hardly differs from the first.

The SODA query, labeled "B" in Figure 3-4, is produced and then printed on the screen by the schema translator. Once again, the statement *Type any character to proceed* causes a pause before the processing output is overwritten by the answer from the database. (Sometimes, when the processing output requires more space than one screenful provides, TEAM prints the message **MORE** at the bottom of the screen. Respond by typing any character; TEAM will continue its output at the top, writing over what is already there.)

Figure 3-5 shows four profile features that can be set to control some of the output discussed above:

Display logical form: **Yes** No
Display pseudo-English for logical form: **Yes** No
Pause after displaying logical form: **Yes** No
Show Soda query: **Yes** No

Note that the current setting for each feature is displayed in boldface, while the alternative is displayed in regular type. (This is typical of situations in which the user must chose among a limited number of answers to a question.) The options that refer to the logical form actually apply to both versions of the logical form.

Experiment with these profile options, combining them in different ways to see which combination(s) you like. To change the profile, click the **PROFILE** command in the main TEAM window. After a few seconds, the profile menu will appear on the screen. To change the answer to any of the profile questions, position the mouse over the answer you want and click the leftmost button; the selected answer will become bold, while the alternative will return to the regular typeface. Once all the changes have been made, position the mouse over the bold square nearest the **Exit** command and click the leftmost button. Then try asking TEAM a question to see just how the output has been altered. The profile changes will remain in effect until you log out of the machine or you change the profile again. To change the profile settings assumed by TEAM when it is started, use the PROFILE command to call up the profile menu. Set the features the way you want them to be and then change the answer to the option

Save this profile to a file: **Yes** No

to Yes, as shown here. Exiting from the profile menu will then cause the current profile settings to be saved in the "team.init" file in your login directory; TEAM reads this initialization file upon starting and sets the profile accordingly.

There is one profile feature that affects acquisition rather than query processing. TEAM uses the directory name specified here as the default directory for the ".db" files:

Default directory for database: **lorna**

The domain expert has the option of changing the directory for a particular database file when it is acquired. He can also change this profile option before beginning an acquisition so that the new directory will be assumed for all the database files. This question is unlike those discussed previously in that there are no options from which to choose. Just position the cursor over the current answer and click the leftmost button; then type the new answer and a RETURN.

The other profile features are designed for use by people who change and debug the system. You should normally leave the answers to these questions set as they are in Figure 3-5. However, you might at some time want to change the replies to these two questions:

Send query to database: **Yes** No
Maximum number of logical forms to display: 1

A negative answer to the first question will cause TEAM to translate queries but not send them to be processed by SODA. As a rule, this is useful only for debugging a new domain or TEAM itself. TEAM uses the answer to the second question when more than one logical form is produced for a given query; if, for example, the response to this question is 2, the top two logical forms will be displayed (but only the top-ranked one will continue to be processed). Setting the answer to -1 will cause TEAM to display all the logical forms. But be careful -- there can sometimes be a large number of them.

3.4 Error States

Although TEAM has proved that a natural-language interface is a viable concept, there are times when errors occur or when TEAM is simply unable to process a query. Translation failures and errors are possible for several reasons, including the following:

- As mentioned earlier, TEAM cannot process queries that refer to things it does not have any information about.

- More generally, TEAM cannot process a sentence containing a word that the system does not recognize.

- Unlike some "natural language" interfaces that are really keyword-driven, TEAM cannot parse a sentence if parts of it are omitted -- or if the English used is inherently incorrect or excessively casual.

- TEAM is a prototype designed to explore a difficult research topic and considerable effort has gone into development of its user interface. Nevertheless, there are still ways in which the system can be disrupted.

Usually, if TEAM cannot complete the processing of a sentence, it will issue an error message such as *Sorry ... unable to parse the sentence* or *semantics failed* and then return to the TEAM> prompt. However, here are five suggestions to assist you in a few other situations that can arise:

1. If TEAM finds a word it does not recognize, it gives a special kind of error message, illustrated in Figure 3-6. The figure shows that the user is given the choice of typing either

   - **SUPER-A** or **RESUME** and supplying a new word, as in the example, or
   - **SUPER-B** or **ABORT** to cancel the query and return to the TEAM> prompt.

   If the error was caused by a misspelled word, it is very convenient to be able to resume the query without retyping the whole sentence. If the error was not caused by a misspelling, try to think of a suitable word that TEAM has recognized before, or abort the query and rephrase it without the apparently unknown word.

```
TEAM>shw the peaks


>>Error: The word SHW is not in the lexicon.

MORPH:
   Arg 0 (WORD-initialization): SHW
s-A, (RESUME):   Specify a different word to substitute in the sentence.
s-B, (ABORT):    Restart process TEAM
+(RESUME) Specify a different word to substitute in the sentence.
Enter new word to substitute for SHW: show

SHOW THE -S PEAK
exactly one parse was found
(IMPERATIVE (THE *PEAK*1 (*PEAK* *PEAK*1) (*SHOW *YOU* *SPEAKER* *PEAK*1)))

(*SHOW *YOU* *SPEAKER* THE  PEAK).

 Type any character to proceed
Logical Form transformed for DB:
(QUERY (WH *PEAK*1 (*PEAK* *PEAK*1) T))


WHAT IS EACH PEAK?

Soda Query:
((IN #:$1 PEAK) (? (#:$1 PEAK-NAME)))
 Type any character to proceed
```

Figure 3-6: Unknown-Word Error

2. If TEAM stops on an error break, displaying a message such as *Help Mr. Wizard!* and waiting for user input (that is, *Tyi* appears in the status line at the bottom of the screen), type **RESUME** to continue processing the sentence and get back to the TEAM> prompt. Then check the user profile and change the answer to the question

   Break on language problem: Yes **No**

   to No, as shown here. This option should always be set to No, since it was designed to be used by the system builders.

3. If any other kind of error occurs during processing and there is a consequent wait for user input, (that is, *Tyi* appears in the status line), type **ABORT** to abandon the sentence and get back to the TEAM> prompt. If necessary, type repeated ABORTs until the prompt appears. If the system returns to the Lisp Listener window, use the SELECT : command to get back to TEAM.

4. If any kind of problem arises when TEAM is running (that is, when *Run* appears in the status line), type **CTRL-META-ABORT** to halt TEAM. When the system is not waiting for input, the ABORT key alone will not trigger any response. CTRL-META-ABORT would be useful, for example, if the status line showed that TEAM was running -- yet nothing had appeared on the screen for some time; the system might be caught in an error loop. CTRL-META-ABORT could also be used to signal TEAM to abandon a query with an excessive number of parses. (TEAM attempts to get a logical form from the top-ranked parse first; if that one fails, however, it will continue to try each parse in turn until it finds one that holds up or it exhausts the sequence of possible parses.)

5. If a problem arises, but none of these situations fits (that is, the status line shows something other than *Tyi* or *Run*), try consulting the Lisp Machine Summary section on error handling or find a local user who can help. This anomalous condition is probably not being caused by TEAM.

The next two sections introduce some complexities into the sample sentences. As you experiment with similarly complex queries, there will be times when TEAM cannot process a question, or when the answer or logical form seems incorrect. When this happens, try to restate the question in a different way. Simplifying the sentence will help. For example, if you have stated it in the past tense, try putting it into the present tense or, if it refers to several data items, try leaving one of them out. However, remember that using "pidgin" English will not help, since TEAM will not be able to parse the sentence if it is not grammatically correct. (The system will process some commonly used patterns of speech even though they are not strictly correct. This relaxation of the rules does not extend, however, to slang, careless constructions, or "telegraphic speech.")

3.5 More Complex Questions

In the foregoing discussion, the point was made that TEAM must recognize the words in a query. It will recognize and know how to process certain lexemes that are built into its vocabulary, such as *the*, *in*, or the verb *to be*; these words are so basic to English that they are likely to be used in questions for any database domain. Once a specific domain has been loaded, TEAM will also recognize and know how to process other words -- those that TEAM learned about or *acquired* when the domain expert established that particular domain. This ability to acquire new words for a new domain is a direct result of the design goal of making TEAM a transportable system.

In the simple queries suggested in Section 3.2, the words used were those in TEAM's basic vocabulary, those that referred to the objects in the database (peaks and countries) and the data stored about them (name, area, height, and so forth), and the names "Asia" and "Nepal," which

TEAM learned from the actual data in the database. However, the domain expert can supply adjectives related to such numeric data items as *high* (related to height), *large* and *small* (related to area), and *populous* (related to population). The end user can then ask questions that compare data items with one another by using the comparative and superlative forms of these adjectives (higher, highest, less populous, least populous, and so forth). In addition, the domain expert can define verbs that interrelate parts of the database, such as *cover* (as in "a country covers an area"). The end user can then ask questions incorporating these verbs.

Since the examples given here were acquired by TEAM for the pkcont domain, you can ask the following questions:

TEAM> **which countries are more populous than Argentina**

TEAM> **what is the largest country in Asia**

TEAM> **is the smallest country the least populous**

TEAM> **what is the area covered by each country with a population more than 1000000**

TEAM> **how high is the highest peak**

Although acquisition will be discussed in the next chapter, it is worth mentioning here that one goal of the domain expert is to define enough words so that the end user will not get exasperated by asking a lot of queries that TEAM cannot process. On the other hand, the words should be ones that are likely to be used, since it serves no purpose for TEAM to acquire a large domain vocabulary if many of the words will never actually appear in queries. It is probably a good idea to supply the end user with sample questions or a list of words that are defined in the domain, just as it is necessary to supply some information about what the database contains. (It might also be advisable for the end user to learn how to look at the acquisition window. One can bring up the acquisition window and, without changing anything, look at the words that are defined; the ABORT key will then return control to the main TEAM window without any unnecessary processing being performed.)

One further consequence of TEAM's transportable design is the fact that the linguistic information acquired must be gathered in a manner that is natural to a domain expert who is not a linguist. This will become more obvious when examples of the acquisition process are discussed, but it is instructive for the end user to realize that words vary in their usage and that TEAM had to elicit from the domain expert all the information it utilizes to interpret correctly the meaning of the end user's sentences. The next section highlights a few of the natural-language-processing problems that TEAM attempts to solve, and the accompanying examples illustrate some of the different ways words that are ostensibly the same can be used in English.

3.6 A Few Linguistic Issues

A number of problems arise in natural-language processing that have to do with interpreting implicit or imprecise information in a sentence. In a formal query language, the user is required to state all relationships precisely, identify all items explicitly, and use each command or portion of a command in exactly the way it is specified. In English, however, as in other natural languages, people constantly rely on the listener's subconscious awareness of the various ways in which words may be used, as well as on his ability to perceive implicit interconceptual relationships and supply "missing links" from the context of the conversation or from his knowledge of the world. Understanding one's native language may seem to be a relatively simple task -- but it seems so only because it is easy to forget how much one has learned since birth that contributes to this ability.

This user's guide certainly cannot examine all of the issues involved in natural-language processing, even from the end user's point of view. However, a few of the problems that must be taken into account can and should be touched upon here. Three broad topics are mentioned: knowledge about how words may be used, connections among the various types of things in the database, and sources of ambiguity. One or more examples, drawn from the sample domain about mountains and countries, are given to illustrate each topic or subtopic.

Each of us possesses a great deal of knowledge as to how different kinds of words can be employed in English; this knowledge now appears to us to be "built in," but in fact we learned it over the course of many years. TEAM also has built-in information about some different categories of words and how they can be applied to diverse types of data in any given database. It is this built-in knowledge that enables TEAM to acquire words and characteristics about the pkcont domain, and then to handle the following situations.

- As a simple example, take the fact that in English it is reasonable to compare numeric data like populations by using the comparative and superlative forms of adjectives. On the other hand, it is not reasonable to compare names of continents in any way except by equivalency. This contrast can be seen in the following sentences:

  TEAM> **is the area of Japan greater than 100000 square miles**

  TEAM> **is Asia the continent of Japan**

- Figure 3-2 showed that the data indicating whether a peak is a volcano are represented by simply a "Y" for yes or an "N" for no. This is what is called a *feature* in TEAM; there are several ways one can talk about whether a peak has this feature or not. All of the following sentences will be translated into database queries that include a test for "Y":

  TEAM> **is Everest a volcano**

  TEAM> **which volcanic peaks are higher than 15000 feet**

  TEAM> **does K2 have volcanism**

TEAM> **can the highest peak in North America erupt**

- One very common habit in English is to use the name of something when one really means the thing itself, or vice versa. For example, the sentence

  TEAM> **show the countries**

  uses the word "countries" as shorthand for "the names of the countries." However, the sentence

  TEAM> **what is the area of Kenya**

  uses the name "Kenya" to represent the country known by that name. Normally people make this substitution of name and thing unconsciously but a system like TEAM must distinguish between the two and yet accept sentences that use one in place of the other.

Often sentences in English relate one type of object or concept to another. TEAM can interpret questions of this kind correctly if the connection is explicit in the question and/or can be derived by the system from information that was supplied by the domain expert. Perhaps the simplest connection is between an object such as a peak and a property of that object such as its height. When the user asks

TEAM> **what is the height of Everest**

he is making such a connection. The following sentences all depend on somewhat more complex connections between data about peaks and data about countries:

TEAM> **what peaks are contained in each country in Asia**

TEAM> **does the least populous country in Europe contain any volcanos**

TEAM> **show Asia's highest volcano**

TEAM> **what volcanos in North America are higher than every volcano in Asia**

The last broad topic considered is ambiguity. This presents difficult problems for TEAM, since human listeners so often resolve ambiguity by context, shared associations, or other methods that allow them to "know what you mean." TEAM tries to resolve three major types of ambiguity: syntactic, semantic, and scoping.

Syntactic ambiguity arises when there is more than one way to parse a sentence. A classic example of this category is *I saw the man in the park with a telescope.* In one context, this

could mean that park in the statement was the one with a telescope in it. In a different context, however, it might mean that the speaker saw the man through a telescope. In a third context, it could be that the man referred to in the statement had the telescope. The following example in the sample domain shows two queries that differ by only one word, yet are parsed very differently:

TEAM> **is the highest peak in Europe**

TEAM> **is the highest peak in Europe volcanic**

A common type of semantic ambiguity involves different meanings for the same word. TEAM can handle a limited amount of this type of ambiguity, but will become confused if the same word has too many meanings. In the sample domain, the word *Kenya* is both the name of a peak and the name of a country. Here are two sentences using the different meanings:

TEAM> **how high is Kenya**

TEAM> **how populous is Kenya**

TEAM invokes a number of rules to resolve ambiguity regarding the scope of such quantifiers as *a*, *any*, *some*, *each*, and *every*. Consider the following queries:

TEAM> **show the height of each peak in Asia**

TEAM> **show the height of every peak in Asia**

The use of *each* in the first query leads TEAM to give *peak* a wider scope than *height*, so the interpretation becomes "for each peak in Asia, show the height." In translating the second query, TEAM must decide whether *every* should trigger the same interpretation as *each*, or whether it should be treated like *all*; in the latter case, *height* is given the widest scope and the query is taken to mean "show a height such that it is the height for all the peaks in Asia." Because of our knowledge of the world, we know that a single height for all the peaks is a nonsensical interpretation, but TEAM must try to resolve the ambiguity by other rules, using only the context of the sentence. As it happens, TEAM makes the wrong choice for this query. (Since it does not find a height equal to the heights of all the Asian peaks, it returns the answer *There are none.*) As another example of the ambiguity of *every*, suppose there were a domain dealing with people who had climbed mountains. The query "who has climbed every peak higher than 15000 feet" might be asking for a list of the climbers of each peak in that category, but it is more likely to be a request for a single person who has climbed all such peaks.

3.7 A Concluding Comment

Before leaving the end user's view of TEAM, it is appropriate to emphasize that the natural-language-retrieval approach exemplified by TEAM is particularly suitable for some types of queries, but particularly unsuitable for others.

TEAM allows the user to state questions in a language he already knows, placing upon the system the burden of interpreting these questions correctly with respect to the defined domain. As long as the questions the user wants to ask can be stated relatively easily in English, the advantages of having such a capability are obvious and potentially significant. From the examples shown in the two previous sections it should be apparent that many queries, both simple and complex, can be phrased naturally in English by the user. However, some queries are not suitable for a natural-language interface. For example, certain queries are most easily expressed in terms of formulas or logic, using phrases such as *let x = ...* or *if y implies z then ....* The ability of a system to translate English sentences into precise formalisms is not applicable to such queries, since they can be more easily defined in precise terms from the start. Other queries are relatively easy to express by means of special-purpose operators available on certain database management systems (such as *sum all*, *scan*, or *order by ...*), especially if the user is accustomed to thinking in terms of such operators. Sometimes it is difficult to compose sentences that express these queries in a way that allows TEAM to translate them into SODA queries. Admittedly, this is a constraint on the types of interrogatives and imperatives that can be handled -- but it is not a limitation on natural-language interfaces in general, since the origin of the problem is typically in the restricted set of SODA operators rather than in the system's natural-language-processing capabilities.

4. ACQUISITION

We now turn from the end user to the domain expert and begin the discussion of *acquisition*, the process by which TEAM acquires information about the structure of the database and the meaning of words in a particular domain.

Making TEAM work in a new domain is not a cut-and-dried task that can be accomplished by following a prescribed formula. It requires not only an understanding of the fundamentals of acquisition, but also considerable effort in designing and testing the new domain's structural and linguistic characteristics. To make it easier to learn about TEAM, the basic concepts and mechanics of acquisition are introduced in this chapter, using a sample domain that has already been designed. In the following chapter, we shall consider various issues in domain design and offer suggestions regarding methods that may prove helpful.

The sample domain used here, which involves mountain peaks and countries, is the one the reader is already familiar with from the preceding chapter. The first major section in this chapter (Section 4.1) explains how to browse through the acquired information for this domain, and acquaints the user with various aspects of acquisition. Section 4.2 then outlines the mechanics of building the sample domain from a pristine TEAM state. This dual view allows the user to become comfortable with the essential ideas underlying acquisition and with the sample domain before trying to start building a domain himself.

This user's guide does not have sufficient scope to include complete directions for building the sample domain. It does provide a general outline and hints on how to use TEAM to acquire the domain. But it also relies on the user's ability to learn from the example of a completed domain and to make use of the self-help facilities available. TEAM offers extensive on-line "help" information through menus, the prompt window, and expanded versions of the questions asked by the system. In addition, much information about the domain is included in Appendix A for convenient reference during the building process. Other sources of explanation and examples on the subject of acquisition are the papers and videotape referred to in Chapter 1.

Throughout this chapter, the mouse will be used to select mouse-sensitive options or objects on the screen, to give commands, and to scroll windows. If you are not familiar with these actions, please refer back to the pertinent remarks in Section 3.1 and to the Lisp Machine Summary.

4.1 Acquisition Concepts

This section will take you on a "guided tour" through the sample domain, pkcont, which we loaded in Section 3.1. Although the previous chapter made use of the main TEAM window only, the acquisition information is available in other windows without any further loading. The only part of the acquired information that is not reviewed in this section is the actual data. The insertion and editing of data, however, will be discussed in Section 4.2.

4.1.1 The Acquisition Window

To move from the main TEAM window to the primary acquisition window, select **ACQUIRE** from the menu at the top of the screen. Since the pkcont domain was loaded previously, the screen should look like Figure 4-1.

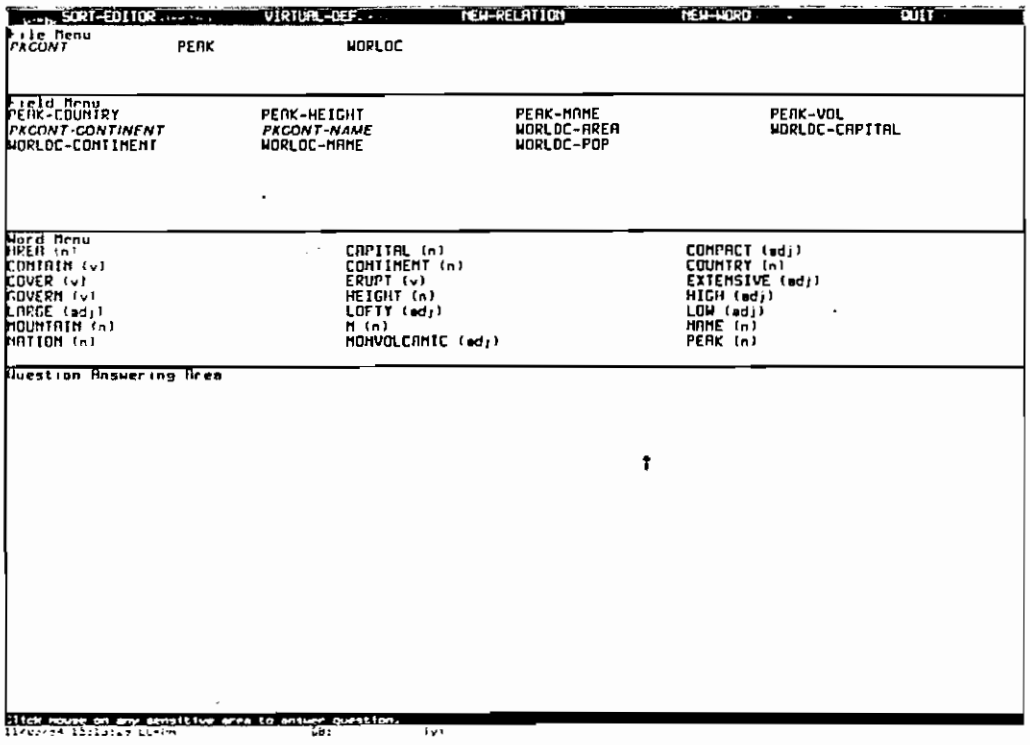


Figure 4-1: The Acquisition Window

Like the main window, the acquisition window has a menu of commands at the top, in inverse video. It contains the following commands:

SORT-EDİTOR
VIRTUAL-DEF
NEW-RELATION
NEW-WORD
QUIT

Position the cursor over each command to see the resultant explanation in the prompt window. All of these commands will be used in Section 4.2, but the SORT-EDITOR, and VIRTUAL-DEF commands will also be executed in the subsections below to display other windows used during acquisition. The QUIT command causes TEAM to update its internal state to be consistent with the acquisition window; this command should be used to leave the acquisition window whenever anything was changed during the acquisition session. If TEAM finds any inconsistency in the domain during the updating process, it issues an error or warning message; otherwise, it returns to the main TEAM window. The ABORT key can be used as a quick way of returning to the

main window if no domain changes have been made; this will hypass both the updating and the consistency checking.

There are three areas on the screen that contain lists of names: the *file menu*, the *field menu*, and the *word menu*. These areas are called menus because each contains a list of objects (or more precisely, names of objects) that can be selected with the mouse. Move the mouse around on the screen; note how the prompt window changes as the cursor moves from the empty portion of the screen to any mouse-sensitive object.

The last area of the screen is the question-answering area. This area is used jointly by TEAM and the domain expert in the process of defining a file, field, or word. Position the mouse over the file object **WORLDC** and click the leftmost button. Questions and answers should appear in the QA area as shown in Figure 4-2.

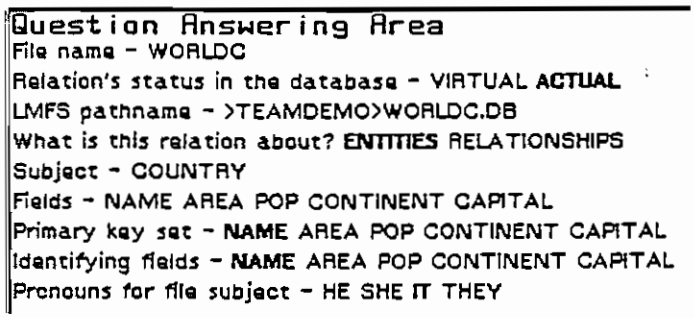


Figure 4-2: The Question-Answering Area for *Worldc*

Select objects from the file, field, and word menus and examine the QA area, keeping in mind the following comments:

- *Both the questions in the QA area and the responses to them are mouse-sensitive.* Position the cursor over a question and click the leftmost button to get an expanded version. Use this on-line help facility extensively while learning about TEAM.

- The QA area for an already defined object shows the full list of questions and answers that were involved in the definition process. However, when a new object is defined, TEAM asks a few questions at a time. Once the user has replied to some of them, TEAM will present further requests that are appropriate to the situation, as clarified by the earlier questions. This is why two objects that are both of type "field" may trigger different questions in the QA area.

- Some questions have a limited number of answers, which appear as choices following the question; the current reply always appears in boldface. Other questions have answers that were typed in by the user (or supplied by TEAM, such as a path name); these appear entirely in regular type.

- Whenever separate words are to be considered part of a multiword phrase (for instance, "capital city") in an answer, the user must enclose them in parentheses in the QA area. Even though TEAM sometimes removes the parentheses when it records such an answer, the system still recognizes the compound. If the latter appears in the word menu, it will have dashes between the words. Such phrases may be used in queries as long as the words are consecutive; they do not require either parentheses or dashes in a query. (The user may also choose to convert a phrase into a single "word" himself by typing it with dashes instead of spaces and omitting the parentheses, as in "square-meter.") A multiword phrase, such as "capital city", is essentially different from multiple answers to a question, as illustrated by the list of adjectives "tall high lofty," which might be supplied for the *peak-height* field. The latter should not be enclosed in parentheses.

Use the mouse to display the QA area for various objects while reading or reviewing the rest of this subsection.

Files

Figures 3-2 and 3-3 presented tables showing the kinds of information contained in the pkcont domain. A *file* or *relation* in TEAM can be thought of as a table of data, similar to the tables in these figures. The term *file* comes from the fact that database management systems often maintain disk or tape files of related information; one or more of these constitute a database. In TEAM, however, *file* is used interchangeably with *relation*. TEAM follows the *relational model* of database organization, and thus assumes that a database is a collection of relations or tables -- each of which contains data about a single kind of object or a single type of connection between objects. The examples below will use the *peak* file, which corresponds to the table in Figure 3-2, and the *worldc* file, which corresponds to Figure 3-3.

Both of these files contain data about things: in one case about mountain peaks, in the other about countries. TEAM uses the relational database term *entities* to refer to things that are the subject of a file, and distinguishes between files about entities (that is, files that have subjects) and files about relationships among the subjects of other files. If you display the QA area for the *worldc* file, you will see that it is specified as a file about entities. Then click the question *What is the file about* and read the explanation regarding the two types of files.

Note also that TEAM asks for the subject, in singular form, and that it may or may not be the same as the file name. TEAM adds the subject (not the file name) to its word menu, and will thereafter be able to recognize that word in a sentence as designating an entry in that file.

There is one other file name in the file menu, *pkcont*. This file name is in italics in the menu because it is a *virtual* file, which is used to express a connection between other files, in contrast to the two *actual* files, which correspond to database files that actually contain data. Subsection 4.1.3 looks at the virtual-file definition.

Fields

When defining a new file or relation, the domain expert must tell TEAM what *fields* it contains. The QA area for *worldc* shows that each column in the table about countries becomes a field in the *worldc* file. The column or columns used to distinguish one entry in the table from another are designated as *key fields*. TEAM puts each field name into its word menu, and inserts the combination of file name and field name (separated by a dash) into the field menu.

However, TEAM must know more about the characteristics of each field before it can properly interpret sentences using the field name. These additional facts are part of the implicit information (the kind that human beings already know how to use) that must be acquired by the system through its dialogue with the domain expert. Thus, even though the fields are partially defined in the QA area for the file, the fields have their own questions that must be answered.

TEAM knows about three basic types of fields that are used in different ways in English: symbolic, arithmetic, and feature. A brief definition of each follows:

- *Symbolic* fields are the simplest type, in the sense that they are just names or symbols. A value in a symbolic field can be retrieved or checked against another value to see if the two are equivalent, but it cannot be used in any other way. *Worldc-name*, *peak-name*, and *peak-country* are all symbolic fields (as are some others in the sample domain). Most symbolic fields have character data in them, but a numeric field would be symbolic if it could only be retrieved or checked against another value for equality (for instance, a social security number).

- *Arithmetic* fields contain numbers that can be compared using the comparative and superlative forms of adjectives (such as "higher" and "highest"). TEAM supports three subtypes within the arithmetic type, each of which has different linguistic properties. *Peak-height* is an arithmetic field that is a *measure*, and *worldc-pop* is a *count*. The third subtype is *date*. Click the question about type in the QA area for one of these fields to see an explanation as to the difference between them. Note also that the domain expert specifies the adjectives that apply to each field. Each adjective becomes a word recognized by TEAM, and its comparative and superlative forms appear in its QA area. If the comparative and/or superlative forms are irregular (such as "better" and "best"), the domain expert can correct TEAM's assumptions in the QA area.

- *Feature* fields contain only an indication as to whether the file entry does or does not have some arbitrary property. *Peak-vol* is a feature field because it contains a "Y" if the peak is volcanic, an "N" if it is not. TEAM supports several linguistic constructs that refer to such a field: adjectives modifying the subject of the file (volcanic), abstract nouns representing the property the subject has or lacks (volcanism), concrete nouns representing subsets of the subjects that have or lack the property (volcano), and verbs applied to the subjects that have or lack the property (erupt). The QA area for *peak-vol* illustrates the way that TEAM acquires the information about the possible nouns and adjectives for a feature field.

For a better understanding of these three types, look at the QA area for each of the fields in the field menu. (The fields in italics are virtual fields from the virtual file *pkcont*.) Use the mouse to

get expanded versions of the questions asked for each type, and think about how the end user would talk about the various fields.

Words

Most of the words TEAM adds to its vocabulary are acquired in the process of defining the files and fields. However, it is also possible for the domain expert to define two other categories of words: nouns and adjectives that are *synonyms* of words TEAM has already learned, and *verbs* that relate file subjects and/or fields. All of these words must be related to things that the system already knows about, so that an end user's sentence can be translated into a database query. One common use of the synonym capability is to define English words that are synonyms for abbreviated or coded field names. In the sample domain, for example, "population" is defined as a noun that is synonymous with the pseudo-word "pop."

The verbs *contain* and *cover* are both defined in the sample domain. You will note that the sentence relating the verb to the database structure must be in the present tense and active voice. Thus, even though the end user may say *what area is covered by Japan* , the domain expert must give the definition as *a country covers an area*. This and other points concerning the definition of a verb are discussed in the chapter on domain design. Note also that TEAM ascertains the linguistic properties of the verb by generating sentences that use the verb in various ways and asking the domain user whether he perceives them to be correct English sentences.

By defining files, fields, and words, the domain expert tells TEAM about the structure of the database and what words are used to refer to it. However, it is also very important for the system to know how the concepts represented in the database structure relate to one another. Much of this information is kept in the sort hierarchy, which is the topic of the next subsection.

4.1.2 The Sort Hierarchy

In the context of TEAM, the term "sort" is a noun (derived from the technical term "a many-sorted logic") that means a class or kind of thing. As an example of this usage, think of the sentence *This sort of padded chair is the most comfortable*. The phrase "this sort of padded chair" refers to a subcategory of the category of objects called chairs. It may overlap with the subcategory of chairs that have rolling casters, but it is disjoint from the subcategory of chairs that are not padded. In the process of learning language, human beings learn a great deal about the categories of things in the world and how they are related to one another. TEAM must absorb some of this information to interpret correctly sentences in a particular domain.

As part of the acquisition process, TEAM and the domain expert build a structure called the sort hierarchy, which diagrams the manner in which the sorts of things referred to by file subjects and fields are related to one another. To look at the sort hierarchy for the sample domain, click the **SORT-EDITOR** command in the menu at the top of the acquisition window. This will bring up the sort-editor window and display the top of the sort hierarchy, as shown in Figure 4-3.

When TEAM is first started, it has a generic sort-hierarchy relating categories that have particular meaning for the system because of linguistic constructions it recognizes. For example,

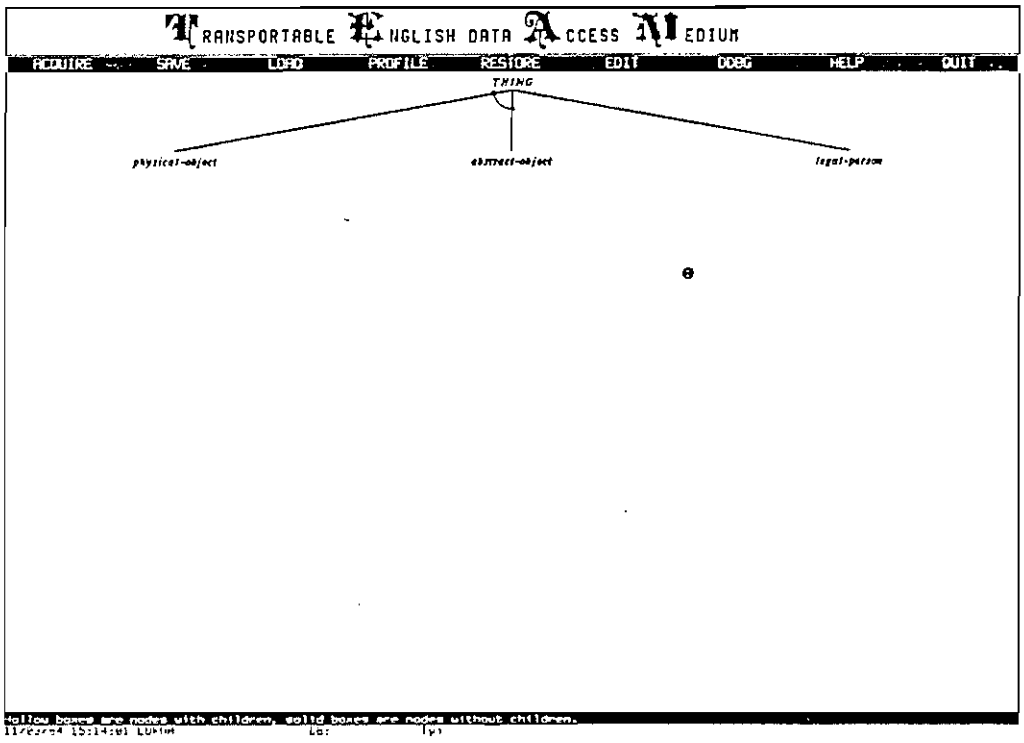


Figure 4-3: The Sort-Editor Window

if a question begins with the word "where," TEAM will try to produce a logical form that gives a subcategory of "location" as its answer. In contrast, if a question begins with "who," TEAM will try to answer with a subcategory of "person" or "legal-person." As files and fields are defined by the domain expert, the system adds the names of the arithmetic and feature fields to the sort hierarchy, underneath the appropriate nodes. Symbolic fields and the subject of each file must be inserted into the sort hierarchy by the domain expert, since TEAM does not know where they belong.

The top entry or "node" in the hierarchy is the most general category *thing*. *Thing* has three subcategories or "children": *physical-object*, *abstract-object*, and *legal-person*. The arc between the line to *physical-object* and the line to *abstract-object* indicates that these two categories are disjoint. Because the arc does not include the line to *legal-person*, the latter may overlap with either or both of the former categories. (A dot is placed on each line that is affected by a disjoining arc; a line that happens to touch a disjoining arc but is not affected by it will not have a dot.) Some of the characteristics of the various nodes will be discussed in Chapter 5, where the focus is on designing a new domain.

The mouse can be used to expand the sort hierarchy, showing where each of the subjects and field names for the sample domain has been placed. The nodes in the hierarchy are mouse-sensitive, and the type of box that appears around a node is significant: a hollow box indicates

that the node has children while a solid box means that the node has none. The children of a node can be displayed by clicking the leftmost mouse button.

Some nodes will have two or more labels, which means that the sorts represented by these labels are really identical. This is called an *equivalence* in TEAM terminology, and the two (or more) labels are called *equivalent* or *alias sorts*. In translating a query into a logical form, TEAM will always identify an equivalence by the top label in the list. The most common form of equivalence is between the subject of one file and a field in another file that acts as a "logical pointer" or reference to the subject of the first file. An example of this is the equivalence between *country* and *peak-country*, shown in Figure 4-4. (Note that the equivalence is **not** between *peak-country* and *country-name*, which are quite different sorts.)

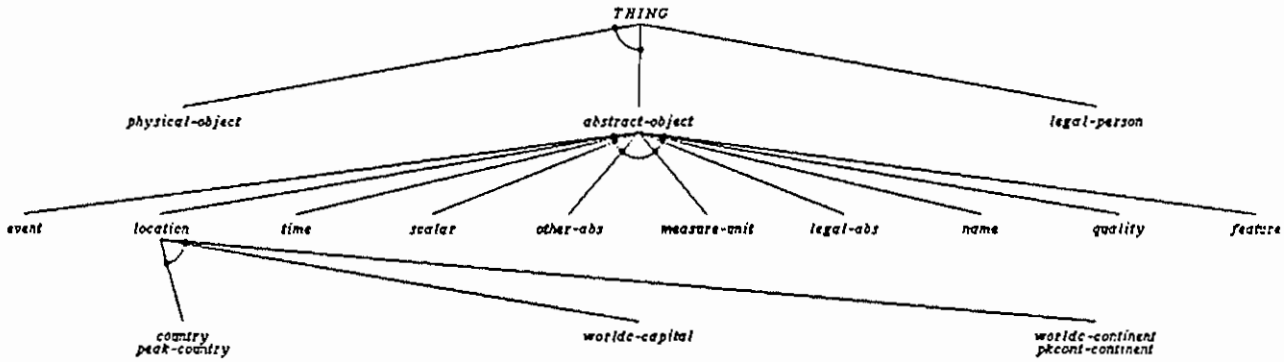


Figure 4-4: An Example of Equivalent Sorts

Sometimes a category or sort has properties that make it appear to fall beneath two nodes in the hierarchy. If (1) the two nodes are not disjoint and have no disjoint ancestors and (2) the resultant connections do not create a cycle (that is, no node is a parent of any of its own ancestors), then the category in question may be a child of both nodes. In the sort diagram, a category that is a child of two parents will appear with a solid line to one parent and a dotted line to the other. The solid line indicates the path that was completely displayed with the mouse, while the dotted line indicates the alternative path; there may be "hidden" nodes along the alternative path. The *person* sort (shown in Figure 4-5) is an example of a category in the sample domain that has two parents. The *country* sort exemplifies a category whose properties suggest that it could be placed underneath two disjoint nodes: *inanimate* and *location*. In such a case, the domain expert must make a choice based on the questions that an end user is considered likely to ask.

To leave the sort-editor window and return to the primary acquisition window, click the **QUIT** command at the top of the screen. Since nothing has been changed, the ABORT key could also be used to abandon this window and return to the main TEAM window.

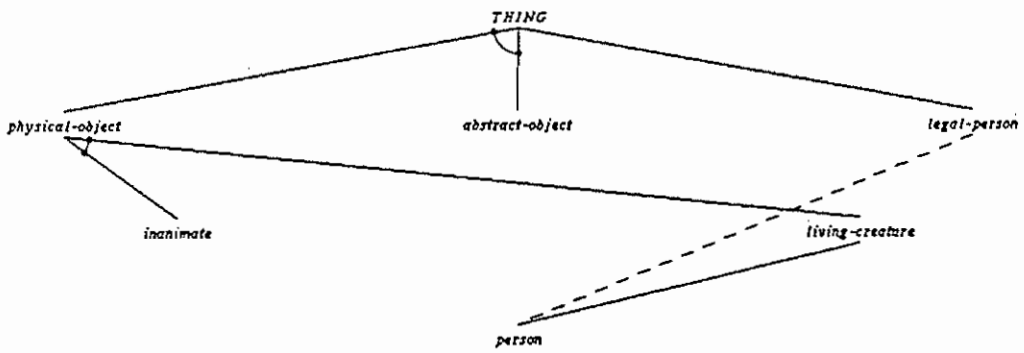


Figure 4-5: An Example of a Sort With Two Ancestors

4.1.3 The Virtual-Definition Window

A virtual relation (or virtual file) defines a logical connection or correspondence between two actual files. The domain expert defines a virtual relation so that TEAM can interpret questions from the end user that depend implicitly on that correspondence.

For example, the field *peak-country* not only contains the name of the country each peak is in, but also acts as a "logical pointer" or reference to the country file. Consider these two questions, both of which use this logical connection:

TEAM> **what peaks are contained in each country in Asia**

TEAM> **show Asia's highest volcano**

TEAM can translate the first of these questions without having a virtual relation defined, because the word *country* appears explicitly in the question and *peak-country* is equivalent to the subject *country* in the sort hierarchy (as explained in the previous section). However, the second question does not refer to *country*. To allow TEAM to recognize that an implicit connection is involved, the domain expert must define the virtual relation *pkcont*. The virtual relation can be thought of as a way of telling TEAM that certain actual properties of a country (such as its continent) should be considered virtual properties of the peaks in that country. (Of course, others should not; for instance, the population of a country should not be considered the population of its peaks as well.)

The virtual relation *pkcont* is defined in the same way that an actual relation is defined, except for the following:

- The file type and field types are specified as "virtual" in the QA area, which causes them to appear in italics in the menus.

- No path name is specified for the database file, since no data will be stored.

- The virtual field *pkcont-name* is defined similarly to *peak-name*, the key field of the peaks file, and the *pkcont-continent* field is defined similarly to *worldc-continent*, a nonkey field of the countries file.

- Once the file and fields have been defined, the various parts of the logical connection between the virtual file and the actual files are specified schematically in the virtual-definition window, as explained below.

Information needed by the language-understanding part of TEAM is supplied by the definition of the virtual file and fields. The schematic definition, however, supplies information needed to process the database query. To see the schematic definition, click the **VIRTUAL-DEF** command at the top of the acquisition window. This will bring up the virtual-definition window. Then position the mouse over the ***PKCONT*** file name and click the leftmost button. This will make the schematic definition of the virtual file appear in the window, as shown in Figure 4-6.

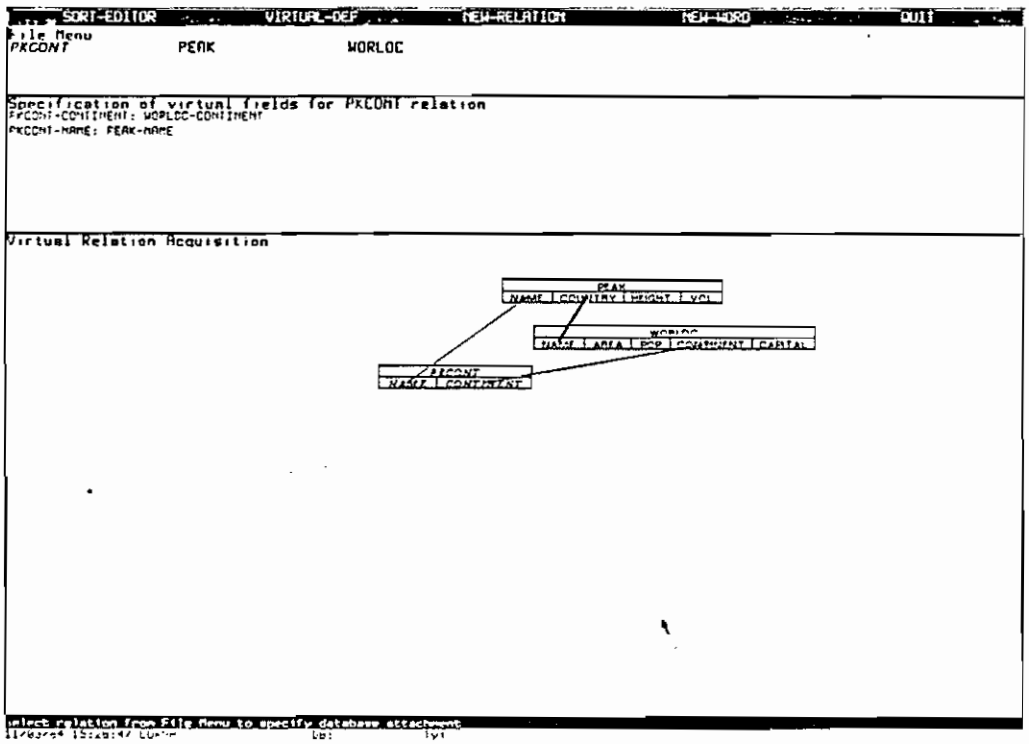


Figure 4-6: Schematic Definition of *Pkcont*

Note that the virtual fields are linked by thin lines to the actual fields that they were designed to resemble. Called "links," these lines specify that the virtual field is to be considered logically "the same as" the actual field. In contrast, the connection between the two actual files is shown by a thicker line, with the reference field in the peaks file at one end and the key field in

the countries file at the other end. This is the connection TEAM must use to process a query that depends on the implicit link between the two files. To compute the "contents" of the virtual file, the system will match values from the reference field of one file to values from the key field of the other file; this connection is therefore identified by the database term "join."

To leave the virtual-definition window and return to the primary acquisition window, click the **QUIT** command at the top of the screen. Since nothing has been changed, the ABORT key could also be used to abandon this window and return to the main TEAM window.

4.2 The Mechanics of Acquisition

The best way to learn the mechanics of acquisition is to actually use TEAM to acquire an already familiar domain, such as the "pkcont" sample. This section is intended to furnish information and make suggestions that will make the learning process easier. First, a basic approach to acquisition is outlined. Then the essential steps in working with the main acquisition window, the sort hierarchy, data, and virtual relations are described.

In working your way through this section, refer back to the overview in the previous section and make extensive use of the on-line "help" facilities available through the prompt window, the mouse-sensitive questions in the QA area, and the sort editor's HELP command (explained below). For ease of reference, Appendix A contains figures showing much of the acquisition information for the sample domain; you can also reload and study the sample domain when questions arise.

4.2.1 The Basic Strategy

Once a domain has been loaded into TEAM, the system must be returned to a pristine state before a new domain is acquired. This is done by selecting the **RESTORE** command from the menu at the top of the main TEAM window. Wait until TEAM has restored its internal data structures to the initial state; the TEAM> prompt will then appear and the status line will show the "Tyi" indicator. Now select the **ACQUIRE** command and you will see acquisition window appear with no entries in the menus.

The domain expert has considerable control over the order in which the various acquisition tasks are accomplished. By using the mouse to select commands from the acquisition window or objects from the file, field, and word menus, the domain expert tells the system what the current task will be. TEAM's role in the dialogue is primarily one of responding to the user's cues and ensuring that certain constraints will be met. For example, when the domain expert has answered the questions that first appear in the QA area for some object, TEAM causes further questions to appear that are appropriate to the situation as specified up to that point.

However, it may be helpful to you, as a new TEAM user, to have a plan in mind for the task of defining the sample domain or a new domain. A basic strategy is outlined here, but remember that it is meant only as a flexible guide that can be modified to suit circumstances and your personal style. Methods of accomplishing the steps presented here are described in later subsections.

1. Begin by defining an "actual" file (like *worldc*) and its fields. The main acquisition window will have to be used for defining the file, the sort editor for entering the file subject into the sort hierarchy. When this has been done, select each field from the field menu and complete the questions for that field. Finally, use the sort editor to put the symbolic fields into the sort hierarchy.

2. Define any synonyms that are needed for the file subject and field names. Then go back and repeat Steps 1 and 2 for the other actual files. It might be wise to save the acquisition (as described below) after defining each file, so that a recent state can be reloaded if it proves difficult to recover from a mistake made in current work.

3. Define any additional adjectives that are synonyms of existing adjectives. (Since synonymous adjectives can be defined by giving multiple answers in the field's QA area, this may not be necessary.) Check the word menu for adjectives that have irregular comparatives or superlatives, and correct TEAM's assumptions regarding these forms. For example, replace "populouser" with the compound "(more populous)".

4. Define the verbs for the actual files. Save the acquisition at this point.

5. Add the data for each actual file by using the data editor. (An alternative strategy is to add data to each file immediately after it is defined. As far as TEAM is concerned, data can be defined at any time once the file itself has been defined completely.)

6. Add the virtual relations for the domain, one at a time. This will involve using (1) the main acquisition window to define the file and fields, (2) the sort editor to enter the fields in the hierarchy, and (3) the virtual-definition window to specify the links and join. You may also want to define verbs that apply to the virtual relations. Save the acquisition after each virtual relation has been defined.

To save an acquisition, return to the main window using the **QUIT** command. If a file was defined during the acquisition session, a message similar to the one in Figure 4-7 will appear on the screen. Note that the path name used is the one that was specified in the QA area when the file was defined. Answer "Y", which TEAM will expand to "Yes" as shown in the figure. (The RETURN key is **not** needed.) TEAM will create a Lisp Machine file, with file type ".db", containing LISP code that will recreate the database file when the domain state is reloaded. Each time the data editor is used to add data to the domain file, a new version of the LISP code will be saved. (The ".db" file will not actually appear in the file directory until the data editor has been used on the domain file for the first time.)

```
No DB file B:>LORNA>worldc.DB -- Shall I create a new one? (Y or N) Yes.
TEAM>
File name? sample

TEAM>
```

Figure 4-7: Screen Display After Saving an Acquisition

To continue saving the acquisition, select the **SAVE** command from the main TEAM window. This instructs the system to create a file of LISP code that can be used to recreate the domain state. The file type ".acq" is always used to indicate that it is an acquisition file written by TEAM, and the code includes instructions to load the ".db" files that are associated with the acquisition. Since the name of the acquisition file has not been specified previously (unlike the database files), the system will ask for it, as shown in Figure 4-7. Supplying a simple file name, as shown in the figure, will cause the file to be saved under the login directory; a path name can be used instead to specify a different directory.

How often the domain state should be saved is a matter of experience, available disk space, how much work has been done since the last save, and how difficult it would be to correct a mistake in the next task. Some hints have been given above, but it is usually better to err on the side of saving too often. In the interests of maintaining free space on the disk, periodically clean out old acquisition and database files that have been superseded.

The next four subsections describe some of the details of the tasks mentioned in the basic strategy. While experimenting with TEAM to create the sample domain, remember to refer back to Section 4.1, which contains the introduction to the acquisition concepts; Appendix A also contains reference material relating to the sample domain.

4.2.2 Acquiring Files, Fields, and Words

The definition of a domain begins with the acquisition window, which is reached from the main window by selecting the **ACQUIRE** command, as described previously. Files, fields, and words are discussed in turn below. Remember that the mouse can be used to get an explanation of each question that appears in the QA area.

Files

To begin the definition of a file, select the **NEW-RELATION** command from the acquisition menu at the top of the screen. TEAM will then set up the QA area for the file with the question *File name* -, and put a placeholder composed of question marks in the file menu. Position the cursor over the answer space of the file name question and click the leftmost button. A blinking rectangle will appear on the line to indicate that TEAM is waiting for you to type your answer. Type the name of the file and press the RETURN key. (The mouse cursor can be moved out of the way once the blinking rectangle appears.) Note that all input is converted to uppercase as soon as you type RETURN. TEAM will replace the question marks in the file menu with the file name, and further questions will appear in the QA area.

As the questions appear, either accept TEAM's assumption (such as the path name) or give a new answer. Remember that there are two types of questions: those that have a list of possible replies, from which you must select one with the mouse, and those that require an answer to be typed. Remember also that the mouse can be used to get an expanded version of a question.

It is often possible to correct a mistake in one's answer simply by selecting a different alternative or typing a new value. In principle, any response should be reversible. Because of limitations in the implementation, however, problems may arise if you change the answer to a question posed by TEAM for the purpose of determining which of several possible questions it

should subsequently ask. The question of file type (with answers *virtual* and *actual*) belongs to this category. It may also be difficult to change a file name or the fields in a file once TEAM has accepted them, so check such responses carefully before typing RETURN. This is one reason it is wise to save the domain state before acquiring a new file.

Note TEAM's reminder that the file subject must be inserted into the sort hierarchy. (This is described in the next subsection.) Once this has been done and the basic questions in the QA area have been answered, the file name will turn boldface in the file menu. This means that no further actions are needed to define the file. However, actions may still be necessary for completing the definitions of the file fields.

Fields

Field names are defined by their entry in the QA area of the file, but TEAM still needs further information about most fields. Supply it by mouse-selecting each field from the field menu, and then answering the questions presented in the QA area. Arithmetic and feature fields will be inserted into the sort hierarchy automatically, but symbolic fields will have to be entered as described below. (TEAM will print a reminder for each symbolic field.) The field name will turn boldface in the field menu as soon as the field definition (including insertion into the sort hierarchy) is complete.

The field type question (with choices *symbolic*, *arithmetic*, or *feature*) and the arithmetic field subtype question (with choices *dates*, *measures*, or *counts*) are used by TEAM to determine what further questions should be displayed (and where the field should go in the sort hierarchy). Thus, the potential exists for changes in these answers to cause difficulties.

Indicating that you want to edit the lexicon for a symbolic field may cause the word menu to become very large if there is a large number of different values for that field in the database. This lexicon-editing feature may be very useful (for instance, to add synonyms or irregular plurals), but it should be used cautiously.

Do not try to alter the symbols indicated for "unknown" and "not applicable" in the QA area, since the system's current version will not know how to make that change. These symbols are shown in the QA area because the "full" version of SODA recognizes and distinguishes between them. However, the "in-memory" version of SODA used to demonstrate TEAM does not recognize them as having any particular meaning.

Note how TEAM utilizes the sample value for a symbolic field to explain the questions about classifiers in terms that can be understood easily by a domain expert who is not a linguist.

Words

If the adjectives supplied for arithmetic fields have irregular comparatives or superlatives, TEAM's assumptions for these forms will be incorrect. For example, it will assume that "populouser," rather than "more populous," is the comparative form of the adjective "populous." When the irregular form is simply "more" or "most" used with the basic adjective, as in this example, it is not strictly necessary to correct the assumption. TEAM will recognize "more populous" in a sentence as meaning the comparative of populous, regardless of whether it

is shown in the QA area. However, because TEAM uses the comparative and superlative from the QA area to generate the pseudo-English restatement of the query, correcting the "more" and "most" forms will affect what the user sees during the query translation. To correct the irregular comparatives and superlatives, select each adjective from the word menu (to bring up the QA area), click the answer space of the appropriate question with the mouse, and type in the correct word or multiword phrase. (Remember to enclose a phrase in parentheses.)

To define a new word, select the **NEW-WORD** command from the acquisition window and answer the questions as they appear in the QA area. Remember that nouns and adjectives must be synonyms of words already in the domain, and that verbs must relate fields and/or file subjects.

A word can be deleted from the domain by positioning the mouse over the word in the word menu and clicking the middle button. However, words that were generated by TEAM from file subjects and file names should not be deleted; TEAM will print a warning message if you try to do this.

4.2.3 The Sort Editor

The acquisition process also involves use of the sort editor. To move to the sort-editor window, select the **SORT-EDITOR** command from the main acquisition window. As explained in Subsection 4.1.2, the leftmost mouse button can be used to expand various parts of the sort hierarchy on the screen.

The prompt window shows what other mouse commands are available. Two quick clicks of the leftmost button will insert a new node underneath the node the cursor is touching. One or two clicks of the middle button will delete the node the cursor is touching: one click signals TEAM to retain the structure that is underneath the deleted node so that the entire subtree can be inserted somewhere else, whereas two clicks indicate that this substructure should be deleted. One click of the rightmost button brings up the sort-editor menu; the leftmost button can then be used to select a command from this menu. Two clicks of the rightmost button will bring up the Lisp Machine system menu, which contains operations related to windows on the screen and to LISP programs.

Display the sort-editor "help" information by using the rightmost mouse button (while the cursor is positioned over some node) to bring up the sort-editor menu, then selecting the **HELP** command. Several pages of information will be displayed on the screen, including a short description of each command available from the menu or by mousing a node.

To insert a file subject or symbolic field into the sort hierarchy, position the cursor over the node that should be the parent of the new node. Click the leftmost button twice in quick succession. This will bring up a menu containing labels for the file subjects and symbolic fields that have been defined but not yet inserted into the hierarchy. Select the correct label with the leftmost mouse button. TEAM will insert the new label as a child of the current node. A menu entry *ANY* is also provided so that, when appropriate, intermediate nodes can be put into the hierarchy (see Chapter 5); TEAM will display a small box in which to type the new label.

Moving the mouse cursor out of the sort-editor menu "box" will make the menu disappear without any command having been executed. Similarly, moving the cursor out of the menu of nodes that may be inserted under the current node will simply abort the insert command. (This is what normally happens when the cursor is moved out of a menu of mouse-sensitive commands. However, it is not true of a box that is used to specify values and has a specific EXIT command, such as the user profile menu.) Thus, to see what file subjects or fields remain to be inserted, simply give the insert command at any node, examine the choices in the menu, and then abort the command by moving the cursor away from the menu.

To instruct TEAM to create a symbolic field equivalent to a file subject in the sort hierarchy (see Subsection 4.1.2 and Figure 4-4), position the mouse over the file subject and use the rightmost mouse button to bring up the sort-editor menu. Select the **Insert Equiv.** command from the menu, which will bring up the menu of labels that need to be entered. Select the appropriate field name from the list. TEAM will place the new label underneath the main label on the same node. Remember: *once an equivalence has been inserted, it cannot be removed.*

The sort-editor menu also contains the commands that are used to mark children of a given node as disjoint sorts (and to "undisjoin" nodes) as well as to add a second parent for a node (an "ISA" link). Refer back to Subsection 4.1.2 for an introduction to these concepts, and consult the "help" information in the menu and the mouse prompts at the bottom of the screen for guidance in using these commands.

To return to the main acquisition window, select the **QUIT** command from the menu at the top of the screen. (It looks the same as the main TEAM menu, but the QUIT command is the only one that is actually mouse-sensitive from the sort editor.)

4.2.4 Entering and Editing Data

Adding data to the files in the sample domain is another part of the acquisition process. This topic was not covered in Section 4.1 (because the process of viewing the data is no different from the process of changing it), but samples of the data in the pkcont domain were seen in Figures 3-2 and 3-3.

To add data to a database file or relation, go to the main TEAM window and select the **EDIT** command from the menu at the top of the screen. (If going to the window causes the system to print a message about creating a DB file, answer "Y" as shown in Figure 4-7.) The data editor will display a menu showing the names of the domain files, as shown in Figure 4-8. Select one of these files to edit.

Figure 4-8: A Data-Editor Menu of File Names

Once a file has been selected, the data editor presents a menu showing the key fields of each entry in the file as well as the entry *NEW*, as shown in Figure 4-9. To add a new entry, select *NEW* from the menu; to change some value for an entry, select its key from the menu. (Note that moving the mouse cursor out of either of these menus will abort the editing session with no harmful effects.)

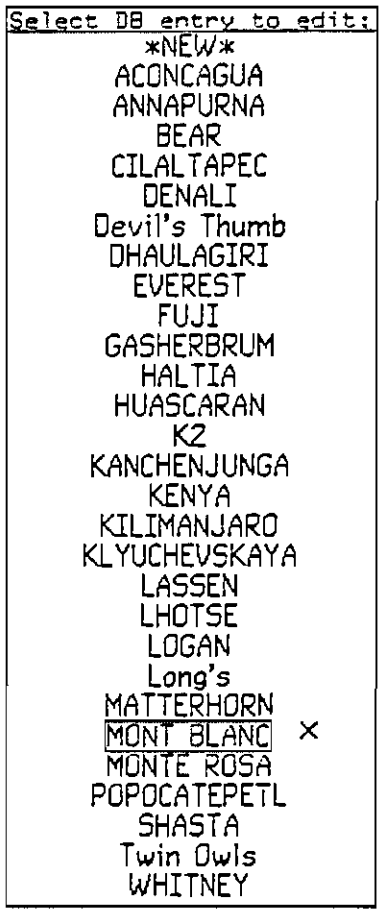


Figure 4-9: A Menu of File Entries

The data editor will display in a box (such as the one in Figure 4-10) the field names and values (actual or null) of the selected entry. Use the mouse to select the value position of a field and then type in a new value, ending the input with a RETURN. If the new value is a multiword phrase, enclose it within parentheses, just as in other parts of the acquisition. TEAM will convert the parentheses to vertical bars (as shown in the example) after capitalizing the whole phrase. When all the fields have been assigned their correct values, position the mouse over the square box adjacent to the **Exit** command and click the leftmost button. The data editor will then return to the menu of entries so that another entry can be edited. (Don't be alarmed if changes you have made to key values do not appear in the entry menu; leaving the menu and returning to it will cause the display to be updated.)

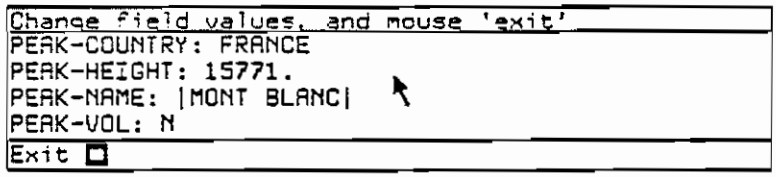


Figure 4-10: Values For A Particular Entry

To finish editing a file, simply remove the cursor from the menu of entries. Another TEAM> prompt will then appear in the main window. To edit entries in a different file, select the EDIT command once again and follow the procedure described above. Using the data editor can be tedious if a significant amount of data must be entered. A faster method is to insert a few entries with the data editor and then to use the ZMACS editor to add further entries to the ".db" file created by TEAM. Some details regarding this procedure are given in Chapter 5.

4.2.5 Defining a Virtual Relation

The remaining task in the acquisition process is to define one or more virtual relations. The acquisition of the already familiar *pkcont* relation will serve to illustrate this task.

As described earlier, the first steps in defining a virtual relation are very similar to those used in defining an actual relation. Give the **NEW-RELATION** command to create a new file, and define the file and fields appropriately. (Refer to Appendix A for details not supplied here.) The file type is *virtual* and no path name is required. The subject is *peak*, to correspond to the subject of the *peak* file. The virtual fields are designed to correspond to fields in the actual relations: the key field, *pkcont-name*, is like the key field *peak-name*, and the other field, *pkcont-continent*, is like the nonkey field *worldc-continent*.

The subject of the *pkcont* file is already in the sort hierarchy, since it is also the subject of the *peak* file. However, the two fields must be inserted as equivalent sorts of the corresponding fields from the actual relations. Thus, *pkcont-name* should be inserted as equivalent to *peak-name*, while *pkcont-continent* should be made equivalent to *worldc-continent* (as in Figure 4-4).

Once the file and fields have been defined, the schematic connections must be specified in the virtual-definition window. Select the **VIRTUAL-DEF** command from the acquisition menu. The virtual-definition window will resemble Figure 4-6, except that there will be no diagram and no textual specification of the virtual fields. Mouse commands will be used to create the diagram, and TEAM will fill in the textual specification of the virtual fields as the schematic "links" are created. (Review description of the "links" and "join" in Subsection 4.1.3.)

First, click the leftmost mouse button to select the *pkcont* file from the file menu. Move the cursor to the large open window at the bottom of the screen, where you will see a hollow box appear and move along with the cursor. This box is a schematic template for the *pkcont* relation. Move it into a convenient position within the window (see Figure 4-6) and click the leftmost

mouse button. TEAM will fix the position of the template and fill in the file and field names. Once the template is fixed in place, position the cursor over the file name portion of the template; look at the prompt window to see that mouse commands can be used to delete the template (leftmost button) and to pick it up with the cursor so that it can be relocated (middle button). Use the mouse to select each of the actual files and position its template in a convenient location in the window.

Now place the cursor over one of the field name portions of a file template, and note that different mouse commands are now available. The leftmost button is used to create (one click) or delete (two clicks) "links" between field names, while the middle button is used to construct or delete a "join" between two fields.

To create a link between the fields *pkcont-name* and *peak-name*, position the cursor over one of the fields (making sure it is on the field, not the file) and click the leftmost button once. Then move the cursor to the other field and click the leftmost button again. (To delete the link, place the cursor over either field and click the leftmost button twice in quick succession.) Use the same procedure to create the link between *pkcont-continent* and *worldc-continent*. Then make the join between *peak-country* and *worldc-name* by following the same steps but using the middle button. This completes the definition of *pkcont*.

You will observe that the virtual-definition join is made between the reference field *peak-country* and the key field *worldc-name*. This is because TEAM will need to "match" values from these two fields to connect the correct peak and country entries. This is different from the connection that was made in the sort hierarchy when the *peak* file was defined. There the field sort *peak-country* was made equivalent to the subject sort *country*, which represents not just the **name** of the country but the **thing itself**. The sort hierarchy is concerned not with matching values from fields, but with the relationships of the essential linguistic concepts in the domain.

This subsection completes the introduction to acquisition. Using the acquisition window (including the QA area), the sort editor, the data editor, and the virtual-definition window, the domain expert tells TEAM what the system must know to enable it to answer questions in a new domain. In the next chapter, we shall continue our discussion of the methodology and problems of adapting TEAM to a new domain by addressing some concerns that arise in the course of domain design. It is recommended that users familiarize themselves thoroughly with the concepts and mechanics presented in this chapter before moving on to the next one.

5. DOMAIN DESIGN

As mentioned earlier, the process of moving TEAM to a new domain includes not only interacting with the system to supply the acquisition information, but also deciding what such information should comprise. This involves designing, testing, and revising the database structure and linguistic support. Our objective in this chapter is to supplement the preceding introduction to the concepts and mechanics of acquisition by providing guidelines and details that will be helpful in the design process.

In designing a new demonstration, it is important both to pick a domain of interest to both the designer and the prospective end users, and to consider whether the characteristics of the intended application are indeed suitable for TEAM. The domain expert may design a demonstration either by adapting an existing database "schema" or organization, or by creating a new database schema from his own knowledge of the application. In either case, the domain expert should have enough familiarity with database concepts and with the domain data to be able, when necessary, to design and modify the logical organization of the data. A few comments about designing the database structure are given later, but some fundamental knowledge is assumed -- particularly regarding the relational model of database organization. Many books and articles on database concepts and the relational model are available; one such reference is An Introduction to Database Systems by C. J. Date, published by Addison-Wesley in 1977.

Adapting an existing database schema for TEAM will require more or less work, depending on how similar its original organization is to that required by TEAM. For example, a schema that is already in a relational format will require less adaptation than one organized as records and sets. We suggest that the new user begin by building a fairly small domain with straightforward data relationships that fit easily into the relational model. It is also important to remember the injunctions expressed in Chapter 2 in regard to the nature of the TEAM prototype.

5.1 Judgment and Experimentation

No formula or fixed procedure can ensure a well-designed demonstration domain in every case. As in most design processes, including database design methodologies, a considerable amount of judgment and experimentation is necessary. However, as remarked earlier, this chapter does contain guidelines that may assist the domain expert in his approach to the design process and in his thinking about the judgments involved.

Since TEAM's purpose is to process database queries that are expressed in English, an essential basis for making design judgments should usually be consideration of the types of English queries that end users are likely to ask (or, in other words, how users will "talk about" the database). An example of one such design decision was mentioned in the preceding chapter, with respect to the placement of file subjects and symbolic fields in the sort hierarchy. Because decisions about file structure, word usage, and virtual relations may also have linguistic implications, they should not be made without taking typical queries into account. For this reason, an important part of designing a new application is to gather one or more sets of English sentences that are appropriate for TEAM, use the application data in various ways, and are typical expressions of data retrieval requests in the application domain. The domain expert can use these sentences to assess the effects of various choices and to test versions of the domain as

they are acquired. One or more subsets of the test sentences can be selected for use in demonstrating the system.

Often design judgments cannot be arrived at without some experimentation, particularly if the domain expert is not a natural-language interface expert and has little experience with TEAM. As a rule, the best way to understand the linguistic implications of a choice among design alternatives is to actually acquire different versions (test domains) of part of the application and then run queries against them. Thus, the design process becomes an iteration through the stages of planning, acquiring, testing, and revising.

Because of this iterative process, and the practical difficulties that can be encountered in trying to change some of TEAM's acquisition information, it is a good idea to plan to experiment with small segments of the demonstration. This can be done in at least two ways: (1) by identifying and building individual parts of the demonstration that require testing to answer particular design questions, and (2) by saving the acquisition state at strategic points so that various experimental domains can be built on a common base. If there is ever any doubt as to whether a problem might have been caused by trying to change an answer previously given to TEAM, be sure to check this out by reacquiring the domain from a pristine TEAM state (that is, the state that the system is in when started or after the RESTORE command has been used). It is also a wise precaution to acquire the entire domain from a pristine state once all the experimentation has been completed.

5.2 Some Characteristics of an Effective Demonstration

As the question of what types of queries are suitable for a natural-language interface such as TEAM has already been addressed, we shall not discuss it any further here. Nevertheless, the demonstration designer is urged to choose an application domain that will lend itself to appropriate English queries, and to create a set of sentences for the domain that will utilize TEAM's capabilities. In addition, the selected domain should be of interest to those for whom the eventual demonstration is intended.

Some issues relative to the database structure will be brought up in the next section, but the designer should begin by considering both the types and interrelationship of data that might be stored in the database. The results of such an analysis will help determine which files and fields the database should contain. Experience has shown that the most interesting demonstrations involve just a few files, with a small number of fields in each. There should be at least two files, with some relationship between them (as in the pkcont sample domain), so that the system's ability to interrelate data from different files can be exploited. Similarly, the database should include several of the diverse field types offered by TEAM: symbolic, feature, count, measure, and date. However, the use of a large number of files, or of many similar fields, generally entails additional acquisition effort without making the demonstration commensurately more interesting.

After sketching out an initial plan for the database files and fields, the domain expert may proceed to plan the linguistic support for the database; included therein are answers to the questions in the QA area for each file and field, the placement of the file subjects and symbolic fields in the sort hierarchy, and the actual words that will be used. Here too, the most impressive

demonstrations will be those that exhibit a good portion of TEAM's capabilities: a field that is a classifier of the file subject, enough adjectives for interesting comparisons, synonyms, verbs relating fields to the file subject or to one another, an equivalence in the sort hierarchy (possibly arising from a field that acts as a reference or "pointer" from one file to another), and so forth. In planning these details, the designer may realize that certain modifications of the database structure would make the domain more interesting.

One of the powerful features of TEAM is that it enables the domain expert to define virtual relations, so that the end user's queries are supported by an implicit connection between certain database files. However, to design a domain that allows this capability to be demonstrated properly, the domain expert must understand when a virtual relation is needed and what its essential function is. More will be said on this subject later when we discuss virtual relations in greater detail.

Finally, an effective demonstration should involve a moderate amount of data. Enough data is needed in each file to ensure that typical queries will elicit positive results. Too much, however, will slow up the processing of a SODA query, besides being tedious for the domain expert to enter. Furthermore, once the meaningful patterns that appear in the application have been represented in the sample data, large numbers of entries that simply repeat these patterns will probably detract from the effectiveness of the demonstration. This is one of the ways in which TEAM, as a research prototype of an English-language interface, differs from a commercial database management system.

5.3 The Database Structure

The database schema for a TEAM domain must be relational, that is, the data are organized in terms of relations or tables like those in the pkcont sample domain we examined earlier (see Figures 3-2 and 3-3). Furthermore, the schema must be in "third normal form" (or fourth normal form) so as to avoid anomalies in retrieving the data. The concepts and methodology of normalization will not be presented here formally, but they are described in many publications dealing with the relational model. This section does contain some suggestions on the procedure of constructing a TEAM database and some comments about how the resulting structure is related to the way that people think and talk about the data. These suggestions and comments touch upon some of the problems of normalization, as well as on some general issues in database design.

As mentioned in the previous section, the domain expert should begin the task of designing the files and fields of a TEAM database by deciding what object categories should be included (such as mountain peaks and countries), and what properties, features, or attributes (such as height and population) should be recorded for each category. The term "entity type" is often used in database design, and will be employed here, to refer to a category of objects about which data is to be stored. Various terms (notably "attribute") are used to refer to the kinds of data kept about an entity type, but the term "property" will be used most often in this discussion. In the TEAM database schema, each entity type chosen by the domain expert will become the subject of a file. The file will be about "entities" (see Section 4.1.1), and each entry in the file will represent a particular entity (such as a mountain or a country). The chosen properties of the entity type will become fields in the file. In selecting the properties to be included, the domain

expert should consider what the TEAM field type (such as *symbolic* or *count*) of each possible property would be, and choose one or more properties of each type for the demonstration database.

It is very important for the domain expert to avoid combining data about more than one subject or entity type in a single file (which is a violation of third normal form). For example, if all of the data about a country were included in the *peak* file (along with the *peak-country* reference), TEAM would interpret the *capital*, *population*, and *area* fields as properties of a peak, since *peak* is the subject of the file. Since this does not correspond to the way users think and talk about these concepts, it would cause confused interpretations of the user's queries.

The domain expert should also consider the relationships between entity types. The *peak-country* field in the sample domain illustrates one common kind of relationship. The country name in the *peak-country* field can be considered, in a sense, a property of the peak. However, since it also acts as a pointer to the country that is related to a given peak, it represents a relationship between the two entity types. In contrast, the city name in *worldc-capital* is simply a property of the country because there is no *city* entity type (or file) in the pkcont domain. The critical factor in the decision to make *country* an entity type (with its own file) and *capital* simply a property (with just a field) was that the designer wanted to record properties of countries, but not properties of cities. Once this decision had been made, it became clear that *country* had to be the subject of a file and that *peak-country* would act as a reference to the appropriate *country* entity. Making a country and its properties part of the *peak* entity would have violated the normal form constraints, as mentioned earlier. Making *city* the subject of its own file and having the capital of a country refer to it would not have violated any constraints, but there would have been no fields in the *city* file other than the city's name. Thus, it was simpler to just treat the capital of a country as an ordinary property rather than as a reference field.

The kind of relationship among entity types that is described above is special in that there is a one-to-many correspondence between the entities of one type and those of another type. For example, there is a one-to-many relationship between countries and peaks, since there is exactly one country for every peak (at least in the demonstration, if not in the world), although there may be many peaks in a given country. Relationships that possess this one-to-many characteristic can be represented in the database schema by a reference field that is included in the file for the "many" side of the relationship and that contains the value from the key field for the "one" side. (If there are two or more key fields, there must be an equal number of reference fields.) The reference field must be equivalent to the **subject** of the referenced (or one-side) file in the sort hierarchy. A common mistake is to make the reference field of one file equivalent to the key field of the other file instead of the latter's subject. Relationships that are characterized by a one-to-one correspondence can also be represented by a reference field, which can be placed in either file.

It is possible to have a many-to-many relationship between two different entity types or between an entity type and itself, such as a *border* relationship between two countries. It is also possible to have a relationship between three or more entity types (not necessarily distinct), such as a *supply* relationship between suppliers, parts, and customers. Furthermore, relationships sometimes have properties of their own, that do not correspond to properties of any of the entity

types involved. For example, an *assembly* relationship between parts and assembled objects might have a *quantity* property. In all of these cases, normal-form constraints prevent the domain expert from using the reference-field strategy (unless, as described later, a new entity type is introduced). Instead, these relationships can be represented as files that are "about relationships." (Remember that this is the alternative to "about entities" in the QA area used to specify a file.) In the relationship file, each entity type is represented by a field like its key, and this field becomes a child of the entity file's subject in the sort hierarchy. Thus, the *border* file would have two fields, *c1* and *c2* perhaps, which would both be children of *country* in the sort hierarchy; the data in these fields would be country names. The *assembly* file would have three fields: two for the entity types (one field a child of *part* and the other a child of *assembled-object*), and a third for the *quantity* property.

A file that is "about relationships" has no subject as far as TEAM is concerned, since it does not hold data about a type of thing. Questions that refer to properties of things, such as "what is the height of each peak," are appropriate to a file about entities (that is, a file with a subject) but not to a file about relationships. However, verbs can be defined that relate the fields of a relationship file, so that questions like "what suppliers supply each customer" or "what customers buy widgets from Universal Widget Co." can be asked.

There is a way to represent the above relationships with reference fields, but it involves introducing a new entity type in each instance. The "supply" case is a good example. As noted above, the designer could choose to represent the connection between a supplier, a part, and a customer as a relationship file. On the other hand, he could decide to call the connection a *supply agreement* and make it the subject of an entity file. The *supply agreement* file should have a key field, such as an *sa-number*, and three fields that act as references to *supplier*, *part*, and *customer* entities. Technically, the designer's choice is to model the connection as a single relationship among three entity types or as three one-to-many relationships between the three entity types, on one hand, and a fourth entity type, on the other. Either way of organizing the data is acceptable, so the designer should base his choice on whether being able to "talk about" supply agreements is useful and proper for the application. For instance, the question "What is the supplier of each supply agreement?" would be appropriate only if "supply agreement" were the subject of an entity file, but "Which supliers supply widgets?" would work in either case (as long as the verb "supply" is defined).

Another matter of judgment relates to data items that have different values but very similar roles, such as quarterly-earnings figures for a year, or several possible "descriptors" of a geographical feature (with values such as *flat*, *circular*, or *temporary*). In designing the database structure, the question arises as to whether these data items are really separate "properties" of a file subject, each of which plays a different role, or are simply multiple values that all play the same role. If their roles are indeed dissimilar and will be distinguished by different words in the English queries that are given to TEAM, such data items should then be made each a separate field in the entity file. Quarterly earnings might be treated this way, for example, if the domain expert decided to distinguish first-quarter earnings from second-quarter earnings, and so forth. However, this would preclude references to quarterly earnings in general, since the underlying SODA software is not sophisticated enough to merge data from different fields into a single field.

If the data items do not perform different functions and it would be unnatural to refer to them with different English words, they should not be represented as separate fields. This would violate a normalization constraint having to do with "repeating groups," as well as making English questions awkward or ineffective. Instead, a relationship file should be designed to represent a many-to-many relationship between the entity type and the data in question. This might be the best way to deal with feature descriptors, for example. In the relationship file, each feature would be paired with each of its descriptors. Assuming that a feature was identified by its number and a descriptor by its name, sample entries might be: (24, circular), (24, temporary), (32, gable), (46, without-superstructure). One or more verbs would be defined so that questions such as "What descriptors describe each feature?" could be asked. There would also be an entity file with subject "descriptor" if the database contained any properties of a descriptor other than its key.

Other situations can arise in which the domain expert must make a design judgment, either on the basis of formal normalization requirements or by trial and error when TEAM does not seem to be interpreting queries correctly. Often a problem that is discovered when a design is tested can be traced to a normal-form violation. For example, situations in which the same information is represented in more than one way are not always obvious when the domain expert begins a design, but are very likely to cause problems in "talking about" the database. As mentioned before, these conflicts must be resolved by considering typical queries for the application.

5.4 Defining Words

As files and fields are acquired, TEAM also adds words to the lexicon for the domain. The name of each field and the subject of each entity file become words. In addition, TEAM learns words from the answers to questions about files and fields, such as the adjectives acquired for measure fields and the nouns, adjectives, and verbs acquired for feature fields. The words that appear in symbolic fields in the database (such as the names of countries, capitals, continents, and peaks) are also acquired, although they do not appear in the word menu unless the option to edit the lexicon is selected for the appropriate field (see Section 4.2.2).

The "type" of each noun is specified during the acquisition. A noun may have one of four types: **common** (or **count**), which includes things that can be counted, like *country*, *ship*, or *supplier*; **mass**, which applies to things that are measured in bulk, such as *wheat* or *oil*; **proper**, which indicates proper names like *Nepal* or *Universal Widget Co.*; and **abstract**, which includes nouns that can be used without determiners, like *truth* in "the pursuit of truth" or *type* in "what patients have blood of type AB?" If a noun is not clearly a mass, proper, or abstract noun, *common* is the best choice. TEAM makes an initial selection for the type of each noun, which generally does not need to be corrected by the domain expert. If a noun's type must be changed, however, this is done by mousing the entry in the word menu and answering the appropriate question in the QA area. The only exception is nouns that do not appear in the word menu because they were acquired from symbolic data and the lexicon-editing option was not selected; the type of such a noun is specified in the QA area that defines the symbolic field.

In addition to learning the basic vocabulary related to files and fields, the system can acquire verbs and synonyms (which may include nouns, adjectives, and verbs). The synonym

capability is especially useful for defining multiword phrases. Although a file subject or field name may be a compound phrase that is typed with hyphens instead of spaces (since spaces are not permitted in file subjects, file names, or field names), it is often convenient to use a single word or abbreviation and then define one or more compound phrases as synonyms. For example, *supply agreement* might be defined as a synonym for the abbreviated file subject *sa*.

The ability to define verbs is a powerful feature, which broadens substantially the range of sentences that can be processed. In addition to selecting verbs for the domain, the designer must consider how the linguistic properties of each will be conveyed to the system. The present-tense (third person singular), past-tense, and past-participle forms of the new verbs are supplied as answers to questions in the QA area. The designer also furnishes a sentence that uses the verb in association with elements of the database structure; this sentence must use the verb in the present tense and active voice. (TEAM does not currently have a way to define a verb that can be used only in the passive voice.)

The defining sentence must also use the verb in its most general sense -- that is, with as many elements of the database structure as are applicable. From this most-general usage, TEAM will determine whether the verb is intransitive, transitive, or ditransitive, and whether particles or prepositional phrases may be used. For example, the sentence "a component fails" indicates that the verb *fail* is intransitive, because it has a subject (component) but no object. For this input to be accepted, *component* would have to be (or be synonymous with) a file subject, a field name, or a common noun for a feature field; all the nouns used in the following examples conform to this requirement.

The sentence "a country covers an area" indicates that *cover* is a transitive verb, with both a subject (country) and an object (area). The sentence "a supplier supplies a part to a customer" tells the system that *supply* is a ditransitive verb, which can be used with a subject (supplier), a direct object (part) and an indirect object (customer). TEAM can also translate sentences employing the dative form of a ditransitive verb, such as "Which suppliers supply Universal Widget Co. bolts?" You will note that "a supplier supplies a part" would have used the verb *supply* correctly, but it would not have been the most general usage of the verb from the domain standpoint, and would not have informed TEAM that *customer* could be used in a sentence with *supply*. Similarly, the sentence "a component fails on a failure date" shows not only that the verb is intransitive, but also that a prepositional phrase can be used to relate a component to a failure date. Putting the prepositional phrase into the defining sentence does not compel the user to include it in a sentence that uses the verb, but omission of the phrase from the defining sentence will prevent its use in queries. Some verbs require a verbal suffix, such as "up" in "a building material makes up a structure." The system will recognize a verbal suffix in the defining sentence and take it into account in translating queries.

If there are two or more unrelated nouns that can be used with a verb, the verb should be defined twice. For example, *make* might be defined once as "an employee makes a part" and once as "a machine makes a part." TEAM will try to decide which of a verb's possible meanings is being used in an input sentence by examining the context in which the verb is used. If there are two or more related nouns that combine with a verb in a particular way, then the defining sentence should use the noun that appears closest to the top in the sort hierarchy. For example, suppose that *part* and *widget* are two file subjects, and that *widget* is a child of *part* in the sort

hierarchy. If the defining sentence for *supply* used *widget*, the verb could not be applied to parts. However, defining the verb by using *part* makes it possible to apply it to widgets as well.

As a second example, suppose that *country* and *continent* are file subjects, both are children of *location* in the sort hierarchy, and each has an *area* property. By specifying "a location covers an area," the domain expert can define the verb *cover* to apply to either a country or a continent, as well as to the corresponding area. But the verb must be acquired **after** the files and fields have been fully defined, including the insertion of the sorts in the hierarchy. In general, one should always define verbs after the sorts to which they apply have been inserted in the hierarchy. This is especially important in the case of equivalent sorts, so as to prevent TEAM from marking a verb as potentially ambiguous when, in fact, equivalence excludes that possibility.

Once TEAM has interpreted the sentence in a verb definition, it will display other sentences containing the verb and ask the designer to signal whether they are correct, thereby eliciting other facts about how the verb may be used. When the domain expert quits the acquisition, TEAM will print a warning message in the QA area of the screen if it cannot find a database match for some part of a verb definition. The user is instructed that pressing any of the regular keys (such as the space bar) will cause TEAM to continue exiting from the acquisition. However, the system will not be able to process sentences containing that problematic verb. When such an error appears, the ABORT key can be used to stop exiting and return to the acquisition mode for correction and completion of the verb definition.

As shown above in the case of verbs, it is possible to introduce ambiguity into the domain by having the same word defined in more than one way. The database is another source of ambiguous words; "kenya", for example, could be both the name of a mountain and the name of a country. Using the same field name in two different files or listing the same adjective as being applicable to two different fields will also produce ambiguous words. (The field names will be differentiated in the field menu because of the file-name prefix, but the word formed from the field names will have two different database attachments.) When TEAM encounters an ambiguous word in a sentence, it must try to resolve the situation by choosing one of the word's possible meanings. Since the system's success in choosing the correct meaning depends on many factors related to the context in which the word is used, ambiguity will be handled more successfully in some cases than in others. It is wise to experiment with particular instances of ambiguity in a domain -- then, before building them into the finished domain, to decide which ones are handled most appropriately.

5.5 The Sort Hierarchy

As noted previously, the system inserts the sorts for arithmetic and feature fields into the sort hierarchy, but the domain designer must do this himself for symbolic fields and file subjects. Just where these should be placed in the hierarchy depends on the way that the subjects and fields relate to the core concepts in the hierarchy and to one another.

The most basic distinction in the generic hierarchy is among the three first-level nodes, *physical-object*, *abstract-object*, and *legal-person*. In the TEAM context, a physical object may not be an abstract object and vice versa; these sorts are disjoint. A "legal person," however, may

also be either a physical or an abstract object. (It should be remembered, in this connection, that the domain expert can give a node more than one parent, as long as no cycles are formed and no disjoint ancestors are implied.) The designer seldom has to put new nodes of the hierarchy directly beneath these first-level nodes, since each of them has several subcategories. Nevertheless, it is important to understand something about the characteristics of the objects in each of these classes, since these characteristics apply to the subcategories as well.

- A physical object has properties such as weight, height, or color. It also has a location, which is theoretically changeable. In practice, of course, this may not be possible, as something like a building or a mountain is not readily moved.

- An abstract object does not usually have weight, color, or other such properties, nor does it have a location in the same sense that a physical object does. In fact, we do not usually think of an abstract noun as representing an "object" at all, even though "abstract object" is standard terminology for this concept. The sort *number*, for example, represents an abstract object, and it is clear that a number does not have physical properties or a location. In contrast, the sort *meeting* represents an abstract object which could have a location, but not in the sense that one could pick it up or give it to someone else. *Country* also has a location in a similar sense and, like a physical object, it has an area. However, we talk about a country as a location itself (for such things as cities and mountains), and locations are usually abstract objects. The designer of the pkcont domain had to make a choice between making a country a physical object or making it an abstract location; in this case, he decided in favor of the latter.

- A "legal person" or "agent," in the TEAM sense, is someone or something that can be the direct reply to a question beginning with "who." As examples, consider the three subcategories: *person*, *corporation*, and *government*. *Person* also comes under the physical-object category, while *corporation* and *government* also belong to the abstract-object category. The sort *country* could be considered a "legal person" (as a child of *government*) in a "politics" database, since we refer to countries as agents in such sentences as "Who sold wheat to Russia?" In a geography database, however, a country is not likely to act as an agent.

The subcategories of *physical-object* and *legal-person* are easily comprehensible, but the subcategories of *abstract-object* may be made clearer with a brief explanation.

- A *location* can be the direct reply to a question beginning with "Where is ...?" This is why *continent*, *capital*, and *country* are children of *location* in the pkcont domain. A location is unrelated to time.

- An *event* has both a time and a location, though the latter does not exist in the sense that the event could be picked up and given to someone. *Meeting* is an example of a sort that would fall beneath *event* in the hierarchy. An event can also be used as a "shorthand" reference to a time or a location. For example, the answer to "Where is he?" might be "At the party."

- *Scalar* is a sort used to handle several concepts related to numbers. TEAM places the names of date, count, and measure fields underneath the appropriate subcategories of *scalar* during acquisition.

- *Other-abs* is provided as a node underneath which the designer can place nouns that do not fall into any of the generic subcategories of *abstract-object*. The only reason this node is furnished, instead of having the designer insert nodes directly under *abstract-object*, is that the screen would become too crowded if more than a few direct subcategories were added.

- A *measure-unit* is an abstract noun like *foot* or *square-meter*. Many of the standard units of measure are in the generic hierarchy, but a new one is inserted if the domain designer specifies a new type of unit for a measure field.

- *Legal-abs*, an abbreviation of "legal abstraction," should be a parent of legal entities that are not physical objects, such as *government*, *corporation*, or *partnership*; typically these sorts also have *legal-person* as a parent.

- *Name* is a very important sort, since placing a field as a child of *name* tells TEAM that the field may be used interchangeably with the thing of which it is the name (normally a file subject). An identifying number, such as a social-security number, should be considered a name. In fact, any symbolic field that could be used as a single-field key for the subject of the file should probably be inserted here.

- *Quality* is used as the parent for abstract nouns, such as *volcanism*, that are associated with a feature field.

- *Feature* becomes the parent of feature fields, such as *peak-vol*. The two categories of objects -- those that possess the feature and those that do not -- become disjoint children of the file subject, along with any count nouns, such as *volcano*, defined for the feature field.

Normally, the domain expert will add new sorts to the hierarchy as children of the "lowest" or most specific nodes, but this is not always the case. For example, the noun *employee* might be added as a subcategory of *person* in spite of the fact that the subcategories *man*, *woman*, and *child* already exist. The designer would probably also make *employee* disjoint from *child*. In general, it is good strategy to disjoin any nodes you can, since this may prevent TEAM from making "bad guesses" when interpreting references to the domain concepts in the input sentence.

The designer must consider not only where to place new sorts in the generic hierarchy, but how the new sorts will be related to one another. We have already seen that the relative position of certain sorts can be important when verbs are defined. In fact, the domain designer may need to introduce an extra node into the sort hierarchy as the parent of two related nodes that can both be used with a verb. In the example discussed in the previous section, the *location* node already existed -- but sometimes a new node must be created. We have also seen that equivalent sorts result from the use of reference-field relationships. Whether to make two sorts equivalent is a particularly significant decision, not only because of its linguistic consequences, but also because an equivalence cannot be "undone." Before making such a decision, it is particularly important to try the options out by creating small acquisitions that contain parts of the domain or by building the experimental levels of the domain "on top" of a saved (and therefore recoverable) state.

5.6 Virtual Relations

A virtual relation is required in a domain when sentences from end users may depend on a reference-field relationship between two files without mention of either the file subject or the reference field. As noted earlier, the question "What peaks are contained in each country in Asia?" can be interpreted in the pkcont domain without a virtual relation, because *country* is mentioned and *peak-country* acts as a reference to *country*. In contrast, the sentences "What is the continent of each peak?" and "Show Asia's highest volcano?" cannot be translated without the virtual relation *pkcont*.

In essence, the virtual relation tells TEAM to recognize *continent* as a virtual property of a peak. Because of this, the interface can interpret the phrases "the continent of a peak" and "Asia's highest volcano." Furthermore, because of the schematic definition of the "links" and "join" involved in the virtual relation, the system can find the continent of a peak by matching the appropriate value in the *peak-country* field to a value in *worldc-name* and then retrieving the *worldc-continent* value. Because the *continent* field in the virtual relation is specified in the QA area as being a classifier of the subject of the virtual relation (which is *peak*), TEAM can also recognize that the phrase "Asia's peaks" means those peaks that have continent "Asia," and use the same retrieval strategy to find them.

The domain designer should examine each reference field in a new domain and decide whether end users are likely to depend implicitly on the relationship between the two files. If one or more fields of one file may be considered virtual properties of the subject of the other file, a virtual relationship should be created to capture this fact. (If there are no fields that would be considered virtual properties, there is no reasonable way for a sentence to depend implicitly on the relationship, or vice versa.) The subject of the virtual file should be the same as the file subject that will gain the new properties. The fields of the virtual relation should correspond to (1) the key field(s) of the "subject" file, and (2) the nonkey field(s) of the other file that should be considered virtual properties of the subject of the first file. (Remember that key fields identify the entities in subject files uniquely -- this is why the key(s) is (are) included for the first file. No key is needed for the other file, since the subject of that file does not participate in the virtual relation.) For instance, in the *pkcont* virtual relation, the subject is *peak*, the *pkcont-name* field corresponds to the key of the *peak* file, and *pkcont-continent* is the same as *worldc-continent*.

The virtual fields may have the same names as the actual fields or they may have different ones, but each one must be made equivalent to its corresponding actual field in the sort hierarchy. The questions in the QA area should be answered as they were for the actual fields, with the exception of the field type question (for which "virtual" should be selected) and the two questions about whether the field is a classifier of the file subject (for symbolic fields). As illustrated in the example below, these two questions should be answered by considering whether the virtual properties can be classifiers of the subject of the virtual file. Thus, the replies may be different from simply stating whether or not the actual properties can be classifiers of the actual file's subject.

Let us suppose there is a file, *gf*, with subject *geo-feature*, that has key *gf-id*, a reference field *gf-class*, and some other fields (see Figure 5-1). The field *gf-class* contains values from the *class-number* field of a file that also has a descriptive field *class-name*. Since end users may use

phrases like "the name of each geo-feature," a virtual relation such as *gfname* should be defined, with subject *geo-feature* and two fields, perhaps called -- for example -- *gfname-id* and *gfname-name*. *Gfname-id* must be equivalent to *gf-id* in the sort hierarchy, and *gfname-name* must be equivalent to *class-name*. The figure shows the schematic definition of the links from *gfname-id* to *gf-id* and from *gfname-name* to *class-name*; it also shows the join from *gf-class* to *class-number*. It should be noted, however, that, as a reference field, *gf-class* must be equivalent to the **subject** of the *class* file in the sort hierarchy, not to *class-number*. The sort hierarchy represents the linguistic reality, whereas the links and join represent the database reality.

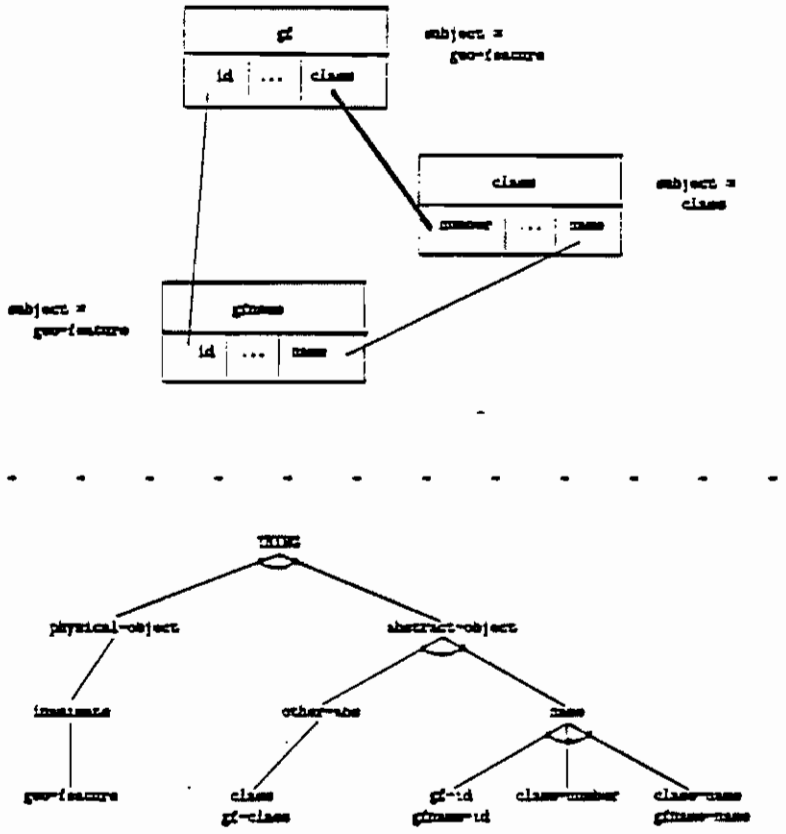


Figure 5-1: Diagram and Partial Sort Hierarchy for a Sample Virtual Relation

As for the classifier questions, observe that *class-name* is unlikely to be considered a classifier of *class*: the question "How many water tower classes are there?" doesn't make much sense in this domain, since "water tower" is by definition a single class. Still, "How many water tower geo-features are there?" is a sensible question, which indicates that *gfname-name* should be considered a classifier of the subject of the virtual file. Similar reasoning holds for the implicit-classifier question, which uses the test sentence "How many water towers are there?" Note that these test sentences are the ones that would be shown in the extended explanation of the two classifier questions if the domain designer gave "water tower" as the sample value for the *class-name* and *gfname-name* fields.

For those readers who are familiar with the "view" concept in database management, it should be noted that a virtual relation is like a view in that it provides a shorthand for a join. A virtual definition does not, however, have the "template" aspects of a view (although some control over what is displayed can be exercised by marking convenient identifying fields). In keeping with the fact that TEAM is an interface, not a database management system, virtual relations are used to increase the range of sentences that can be processed by the system, whereas views may be used for several other reasons, including data protection and preservation of old data interfaces.

5.7 Editing Data Files and Testing a Set of Queries

Adding more than a few entries to a database file can be a tedious process. One way to make this task easier is to use the TEAM data editor to put the first few entries into the file, and then add the remaining entries by editing the LISP ".db" file directly with ZMACS or the appropriate editor for the Lisp Machine on which TEAM is running. (Instructions for using ZMACS are included in the Symbolics documentation.)

Using the EDIT command to add the first few entries will ensure that the ".db" file will have the proper code structure. Figure 5-2 shows a sample ".db" file, with the field names and database entries moved to separate lines for better visibility. Note that the field names are in alphabetical order and that the data values in each entry are listed correspondingly. It is a good idea to begin editing a ".db" file by formatting it as shown in the figure. First insert a "return" character before each part of the code that should be on a separate line. Then position the cursor at the left parenthesis before "CREATE.TABLE" and give the ZMACS command CTRL-META-Q to indent the field names appropriately. Move the cursor to the left parenthesis before the first "PUTPROP" and give the command again to format the database entries.

```
;;;;-*- MODE:Lisp; PACKAGE:D8; BASE:10.; -*-
(CREATE.TABLE? (QUOTE FID) (QUOTE (
                                   (FID-NAME)
                                   (FID-NUMBER))))
(PUTPROP (QUOTE FID) (QUOTE (
                             (DRYLAND 175)
                             (|CONIFEROUS WOODLAND| 201)
                             (|COMMERCIAL BUILDING| 39)
                             (HOUSE 41)
                             )) (QUOTE TABLEDATA))
(PUTPROP (QUOTE (FID-NAME)) (QUOTE NIL) (QUOTE SPEC))
(PUTPROP (QUOTE (FID-NUMBER)) (QUOTE NIL) (QUOTE SPEC))
```

Figure 5-2: A Database File, Formatted for Editing

New entries may be inserted anywhere in the list of existing entries. In the figure, for example, a new entry such as "(APARTMENT/HOTEL 40)" could be inserted before the entry "(DRYLAND 175)", after the entry "(HOUSE 41)", or between any two entries. For the sake of clarity, each new entry should be put on a new line. In this example, it would also make sense to reorder the entries in numeric sequence, since the numeric field is the key; this would make it easier to keep track of new entries. Note that the order of the values within each entry should not be changed, but the order of the entries themselves may be. Compound phrases must be

enclosed within two vertical bars, as shown in the example. (These bars indicate to LISP that the characters between them should be treated as one literal atom.)

Once the file has been edited and saved, the new version must be reloaded before TEAM will recognize the changes. Loading the domain will automatically cause the most recent version of each database file to be loaded, as well as the acquisition file. A more efficient way of loading an altered data file is to type **(load 'data-file-pathname)** to the Lisp Listener window (found by using **SELECT L**), where "data-file-pathname" is replaced by a path name such as ">teamdemo>peak.db".

Another convenience is the ability to run through TEAM a group of test sentences that have been put in a LISP file, directing the results to another file. The easiest way to begin a test file is to use ZMACS to copy it from an existing one and edit the contents, as this ensures that the file will have the proper mode and package; for example, ">teamdemo>userguide.questions" is a test file that can be copied. As shown in Figure 5-3, each sentence in a test file is enclosed in parentheses (to make it a LISP list) and "nil" is used to signal the end of the sentences to be processed (which may or may not be the end of the file). In a query, a punctuation mark other than a period or question mark must be preceded by a slash ("/"). A comment can be included in the file by preceding it with a semicolon.

```
; -*- MODE: LISP; PACKAGE:DIAMOND; BASE:10. -*-

(what are the countries)
(show the peaks)
(what is the height of each mountain in nepal)
(show the population of asia/'s countries)            ;note that / must be placed before '
(which countries are more populous than argentina)
(what is the largest country in asia)
(is the smallest country the least populous)
(what is the area covered by each country with a population more than 1000000)
(how high is the highest peak)
(is the area of japan greater than 100000 square miles)
(is asia the continent of japan)
(is everest a volcano)
(which volcanic peaks are higher than 15000 feet)
(does k2 have volcanism)
(can the highest peak in north america erupt)
(show the countries)
(what is the area of kenya)
(what is the height of everest)
(what peaks are contained in each country in asia)
(does the least populous contry in europe contain any volcanos)
(show asia/'s highest volcano)
(what volcanos in north america are higher than every volcano in asia)
(is the highest peak in europe)
(is the highest peak in europe volcanic)
(how high is kenya)
(how populous is kenya)
(show the height of each peak in asia)
(show the height of every peak in asia)
(show the height of all the peaks in asia)
(what area is covered by japan)
(What is the continent of every peak/?)
nil
```

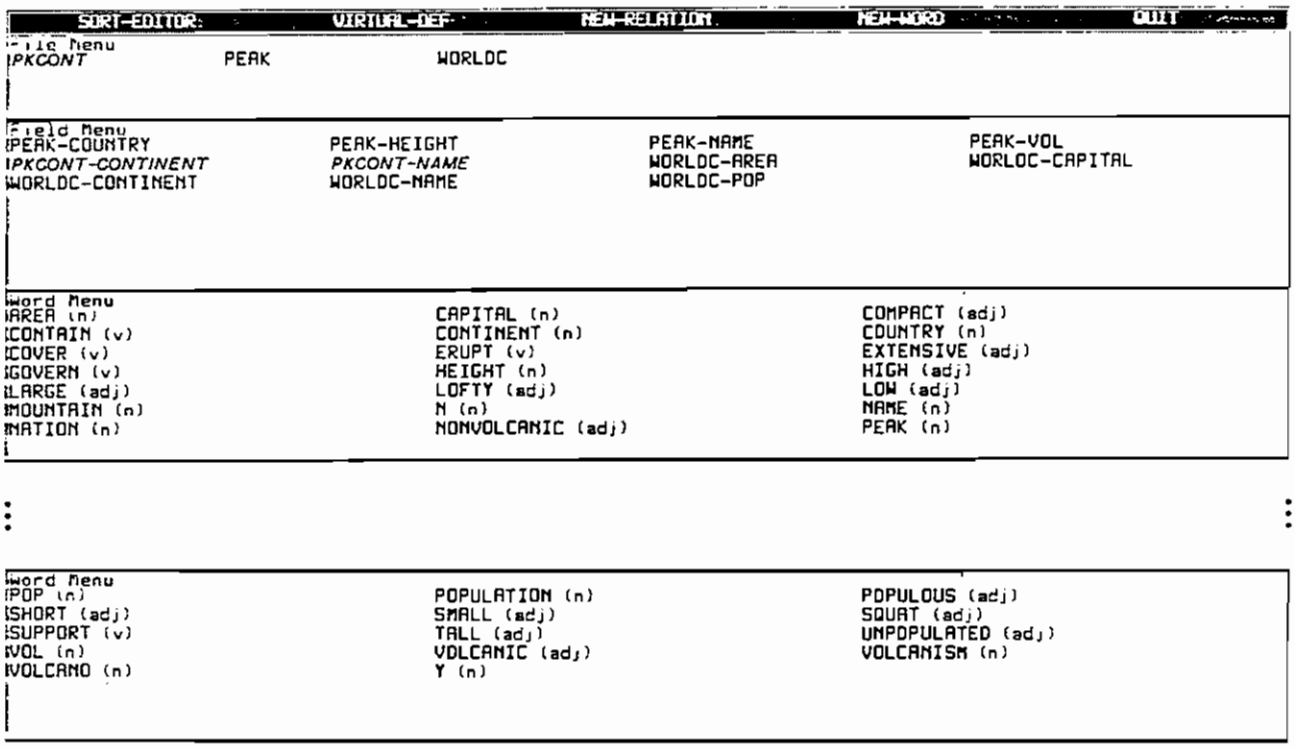
Figure 5-3: A File of Test Questions

Before running the test sentences through TEAM, make sure that the correct domain is loaded and that the profile is set so as to produce the output you want to see in the resulting file. Load the "tester" by typing **(load '>team>tester)** in the Lisp Listener window. Then run the tester by typing **(test.team 'infile 'outfile)**, where "infile" is the path name of the file of sentences and "outfile" is the path name to be used for the output file. If an error occurs while the tester is running, the error message and options for continuing will appear in the Lisp Listener window. The output to the file will be similar to the output that normally appears in the TEAM window, except that the actual answer from the database will be printed as a LISP form rather than as a table.

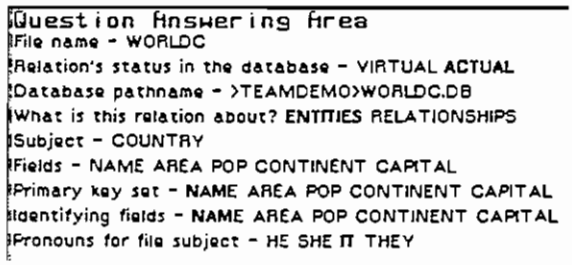
Appendix A
Information From the Pkcont Domain

Appendix A
Information From the Pkcont Domain

Menus



Files and Fields



Question Answering Area
Field WORLDC-AREA is part of an ACTUAL relation.
Type of field - SYMBOLIC ARITHMETIC FEATURE
Value type - DATES MEASURES COUNTS
Are the units implicit? YES NO
Enter implicit unit - SQUARE MILE
Measure type of this unit - TIME WEIGHT SPEED VOLUME LINEAR AREA WORTH TEMPERATURE OTHER
Abbreviation for this unit? - SQ MI
Conversion formula from SQUARE-METERS to (SQUARE MILES) - (/ X 2589998.)
Conversion formula from (SQUARE MILES) to SQUARE-METERS - (* X 2589998.)
Positive adjectives - LARGE EXTENSIVE
Negative adjectives - SMALL COMPACT

Question Answering Area
Field WORLDC-CAPITAL is part of an ACTUAL relation.
Type of field - SYMBOLIC ARITHMETIC FEATURE
Edit lexicon for words in this field? YES NO
'Unknown' convention for this field - *
'Not applicable' convention for this field - **
Are field values units of measure? YES NO
Database Nouns subcategory - PROPER COUNT MASS UNIT
Typical value - PARIS
Will values of this field be used as classifiers of the file subject? YES NO
Will the values in this field be used alone as implicit classifiers? YES NO

Question Answering Area
Field WORLDC-CONTINENT is part of an ACTUAL relation.
Type of field - SYMBOLIC ARITHMETIC FEATURE
Edit lexicon for words in this field? YES NO
'Unknown' convention for this field - *
'Not applicable' convention for this field - **
Are field values units of measure? YES NO
Database Nouns subcategory - PROPER COUNT MASS UNIT
Typical value - ASIA
Will values of this field be used as classifiers of the file subject? YES NO
Will the values in this field be used alone as implicit classifiers? YES NO

Question Answering Area
Field WORLDC-NAME is part of an ACTUAL relation.
Type of field - SYMBOLIC ARITHMETIC FEATURE
Edit lexicon for words in this field? YES NO

Question Answering Area
Field WORLDC-POP is part of an ACTUAL relation.
Type of field - SYMBOLIC ARITHMETIC FEATURE
Value type - DATES MEASURES COUNTS
Type of object counted - PERSON
Positive adjectives - POPULOUS
Negative adjectives - UNPOPULATED

Question Answering Area
File name - PEAK
Relation's status in the database - VIRTUAL **ACTUAL**
Database pathname - >TEAMDEMO>PEAK.DB
What is this relation about? **ENTITIES** RELATIONSHIPS
Subject - PEAK
Fields - NAME COUNTRY HEIGHT VOL
Primary key set - **NAME** COUNTRY HEIGHT VOL
Identifying fields - **NAME** COUNTRY HEIGHT VOL
Pronouns for file subject - HE SHE IT THEY

Question Answering Area
Field PEAK-COUNTRY is part of an ACTUAL relation.
Type of field - **SYMBOLIC** ARITHMETIC FEATURE
Edit lexicon for words in this field? YES **NO**
'Unknown' convention for this field - "
'Not applicable' convention for this field - ""
Are field values units of measure? YES **NO**
Database Nouns subcategory - **PROPER** COUNT MASS UNIT
Typical value - NEPAL
Will values of this field be used as classifiers of the file subject? **YES** NO
Will the values in this field be used alone as implicit classifiers? YES **NO**

Question Answering Area
Field PEAK-HEIGHT is part of an ACTUAL relation.
Type of field - SYMBOLIC **ARITHMETIC** FEATURE
Value type - DATES **MEASURES** COUNTS
Are the units implicit? **YES** NO
Enter implicit unit - FOOT
Measure type of this unit - TIME WEIGHT SPEED VOLUME **LINEAR** AREA WORTH TEMPERATURE OTHER
Abbreviation for this unit? - FT
Conversion formula from METERS to FEET - (/ X 0.3048)
Conversion formula from FEET to METERS - (* X 0.3048)
Positive adjectives - TALL HIGH LOFTY
Negative adjectives - SHORT LOW SQUAT

Question Answering Area
Field PEAK-NAME is part of an ACTUAL relation.
Type of field - SYMBOLIC ARITHMETIC FEATURE
Edit lexicon for words in this field? YES **NO**

```
Question Answering Area
File name - PKCONT
Relation's status in the database - VIRTUAL ACTUAL
What is this relation about? ENTITIES RELATIONSHIPS
Subject - PEAK
Fields - NAME CONTINENT
Primary key set - NAME CONTINENT
Identifying fields - NAME CONTINENT
Pronouns for file subject - HE SHE IT THEY
```

```
Question Answering Area
Field PKCONT-CONTINENT is part of a VIRTUAL relation.
Type of field - SYMBOLIC ARITHMETIC FEATURE
Edit lexicon for words in this field? YES NO
'Unknown' convention for this field - *
'Not applicable' convention for this field - **
Are field values units of measure? YES NO
Database Nouns subcategory - PROPER COUNT MASS UNIT
Typical value - ASIA
Will values of this field be used as classifiers of the file subject? YES NO
Will the values in this field be used alone as implicit classifiers? YES NO
```

```
Question Answering Area
Field PKCONT-NAME is part of a VIRTUAL relation.
Type of field - SYMBOLIC ARITHMETIC FEATURE
Edit lexicon for words in this field? YES NO
```

Sample Words

```
Question Answering Area
Enter word - COUNTRY
Syntactic category - ADJECTIVE NOUN VERB
Plural - COUNTRIES
Noun subcategory - PROPER COMMON ABSTRACT MASS
```

```
Question Answering Area
Enter word - POPULATION
Synonym - POP
Syntactic category - ADJECTIVE NOUN VERB
Plural - POPULATIONS
Noun subcategory - PROPER COMMON ABSTRACT MASS
```

```
Question Answering Area
Enter word - SHORT
Syntactic category - ADJECTIVE NOUN VERB
Comparative - SHORTER
Superlative - SHORTEST
```

```
Question Answering Area
Enter word - POPULOUS
Syntactic category - ADJECTIVE NOUN VERB
Comparative - MORE POPULOUS
Superlative - MOST POPULOUS
```

```
Question Answering Area
Enter word - CONTAIN
Syntactic category - ADJECTIVE NOUN VERB
Third person singular present tense (he she it) - CONTAINS
Past tense - CONTAINED
Past participle - CONTAINED
Sentence - A COUNTRY CONTAINS A PEAK
'A PEAK CONTAINS.' <=> 'Something CONTAINS a PEAK.' YES NO
'A COUNTRY CONTAINS.' <=> 'A COUNTRY CONTAINS something.' YES NO
'A PEAK is CONTAINED.' <=> 'Something CONTAINS a PEAK.' YES NO
```

```
Question Answering Area
Enter word - COVER
Syntactic category - ADJECTIVE NOUN VERB
Third person singular present tense (he she it) - COVERS
Past tense - COVERED
Past participle - COVERED
Sentence - A COUNTRY COVERS AN AREA
'AN AREA COVERS.' <=> 'Something COVERS an AREA.' YES NO
'A COUNTRY COVERS.' <=> 'A COUNTRY COVERS something.' YES NO
'AN AREA is COVERED.' <=> 'Something COVERS an AREA.' YES NO
```

```
Question Answering Area
Enter word - ERUPT
Syntactic category - ADJECTIVE NOUN VERB
Third person singular present tense (he she it) - ERUPTS
Past tense - ERUPTED
Past participle - ERUPTED
Sentence - A VOLCANO ERUPTS
```

```
Question Answering Area
Enter word - GOVERN
Syntactic category - ADJECTIVE NOUN VERB
Third person singular present tense (he she it) - GOVERNS
Past tense - GOVERNED
Past participle - GOVERNED
Sentence - A CAPITAL GOVERNS A COUNTRY
'A COUNTRY GOVERNS.' <=> 'Something GOVERNS a COUNTRY.' YES NO
'A CAPITAL GOVERNS.' <=> 'A CAPITAL GOVERNS something.' YES NO
'A COUNTRY is GOVERNED.' <=> 'Something GOVERNS a COUNTRY.' YES NO
```

```
Question Answering Area
Enter word - SUPPORT
Syntactic category - ADJECTIVE NOUN VERB
Third person singular present tense (he she it) - SUPPORTS
Past tense - SUPPORTED
Past participle - SUPPORTED
Sentence - A COUNTRY SUPPORTS A PEAK
'A PEAK SUPPORTS.' <=> 'Something SUPPORTS a PEAK.' YES NO
'A COUNTRY SUPPORTS.' <=> 'A COUNTRY SUPPORTS something.' YES NO
'A PEAK is SUPPORTED.' <=> 'Something SUPPORTS a PEAK.' YES NO
```

Portions of the Sort Hierarchy