A CASE STUDY OF AD HOC QUERY LANGUAGES TO DATABASES

by

John E. Bell and Lawrence A. Rowe

Memorandum No. UCB/ERL M90/78

7 September 1990

A CASE STUDY OF AD HOC QUERY LANGUAGES TO DATABASES

by

John E. Bell and Lawrence A. Rowe

Memorandum No. UCB/ERL M90/78

7 September 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A CASE STUDY OF AD HOC QUERY LANGUAGES TO DATABASES

by

John E. Bell and Lawrence A. Rowe

Memorandum No. UCB/ERL M90/78

7 September 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A Case Study of Ad Hoc Query Languages to Databases†

John E. Bell
Lawrence A. Rowe

Computer Science Division–EECS
University of California
Berkeley, CA 94720

ABSTRACT

This paper describes a case study performed to compare three different interface styles for ad hoc query to a database. Subjects with wide ranging computer experience performed queries of varying difficulty using either an artificial, graphical, or natural language interface. All three interfaces were commercial products. The study showed that the artificial language was best for novices and interface experts, the graphical language was best for programmers, and the natural language was best for end users.

1. Introduction

Increasing computer power and decreasing costs are causing radical changes in user interfaces. Graphical and natural language interfaces are now practical alternatives to alphanumeric interfaces. This paper describes a case study that compared three different interface styles for database query: 1) artificial language, 2) graphical language, and 3) natural language.

Other researchers have compared different interfaces to query languages with varying success. Reisner, Boyce, and Chamberlin compared a relational algebra query language (SQUARE) and a relational calculus query language (SQL) and found that the relational calculus was easier for non-programmers to learn [23].

Greenblatt and Waxman [13] and Boyle, Bury and Evey [7] performed experiments comparing SQL and Query-by-Example (QBE). QBE allows its users to fill-in blanks in a two-dimensional table that is similar to the output format of the query [31]. Greenblatt and Waxman concluded that QBE was better but they did not use actual systems. Boyle, Bury and Evey used actual systems and they found that subjects took longer to learn QBE and that SQL was better for some types of queries while QBE was better for others.

Shneiderman [24] and Small and Weldon [25] performed experiments that compared SQL and natural language. Their results were inconclusive because the experiments were limited by experimental design problems (e.g., limited query language functionality) and by the lack of a working natural language interface.

Jarke, et al. [14] performed a field study that also compared SQL and natural language, but again with inconclusive results. SQL appeared better than natural

---

† Research supported in part by a grant from BP America and by the National Science Foundation under grant MIP 87-15557.

language because users achieved a higher success rate on queries. However, there were very few subjects and little control of the experimental conditions.

Other experiments have been performed which studied only one interface (QBE [28] or NL [21], [12], [22], [15], [11]) or were designed to answer different questions (e.g., procedural versus non-procedural [29], different data models [17], [8], or programming using natural language [6]).

This paper describes a case study comparing full function query languages with sixty subjects working on actual interfaces in a controlled environment. Subjects ranged in computer experience from none at all to experienced database users with programming experience. The tasks included simple queries (e.g., single table) to complex queries (e.g., multiple table and aggregate counting). Subject performance was analyzed across interfaces and across task types giving a rich and broad picture of the usability of these interfaces.

A case study was chosen as the research method for this study because of the need to gain an understanding of *how* these different interfaces compare as a prerequisite to finding the result of such a comparison. A case study, like ethnographic research, allows us to rely on a variety of means of observation, to discover what questions should be asked in a comparison of such interfaces, and to recognize the complexity of comparing the use of these interfaces by actual users [30]. The need to emphasize qualitative aspects of research has been recognized as well by other researchers in computer science (e.g., [19] and [26]).

This research stands out from the previous research because: 1) it is the first to include a graphical interface for database query, 2) it is a controlled experiment that used actual products, and 3) it emphasizes the qualitative along with the quantitative aspects to help understand the results we find.

The remainder of this paper describes the study and the results. Section 2 describes the sample database and the three interfaces. Section 3 describes the details of the experiment, including its design, the subjects involved, and the treatment. Section 4 presents the quantitative results and section 5 presents the qualitative results. Section 6 describes a small experts experiment and those results compared to the results presented in section 4. Finally, section 7 contains our conclusions.

2. The Experimental Database and Interfaces

The experimental database included information about students, teachers, classes and activities for a high school. Figure 1 shows an entity-relationship diagram for the database. Entities are represented by boxes, and relationships are represented by diamonds.

The high school database was chosen because it would be familiar to all subjects and it is easy to understand. In addition, it has sufficient entities and relationships to allow a variety of simple and complex questions to be asked.

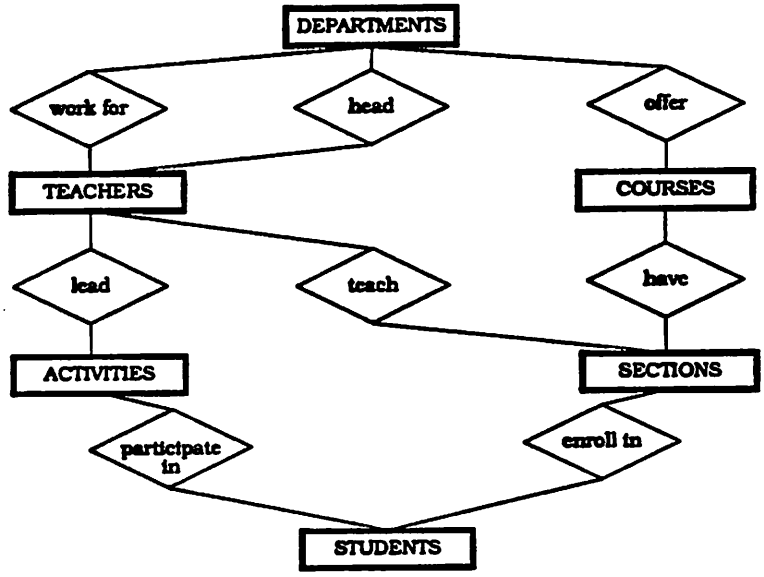

Figure 1: ER diagram of high school database.

A fundamental problem encountered when comparing different interfaces for the same task is to find a fair method to present the task to the experimental subject. For example, a natural language description of the task is inappropriate for this research because one of the interfaces being compared is a natural language interface which could lead to biased results. In addition, a single task presentation method was needed so that differences could be eliminated in how subjects using different interfaces understood the task.

Other researchers have tried other task presentation methods including: problem solving [24], showing the answer which subjects are to get [25], and missing data [21]. We chose a pictorial representation of the tasks because it causes minimal bias toward any of the interfaces, it can represent all of the desired tasks, and it was readily understandable by subjects.

Figure 2 shows an example. The task represented in the figure is to find the first and last names and salary of teacher number 101.[1] The icon represents the entity, in this case a teacher, and the captions represent the attributes. Attributes with values are restrictions and attributes without values specify the information to be retrieved. Other icons used in the pictures are shown in figure 3. Primary icons involve a single type of object (e.g., students), and secondary icons involve a combination of primary icons (e.g., sections involve teachers and courses). Subjects could refer to a sheet showing these icons and their meaning during the experiment.

[1]The actual representation used in the study was slightly different but the difference did not affect understanding by the subjects. For more details, see [Bell, 1990 #53].

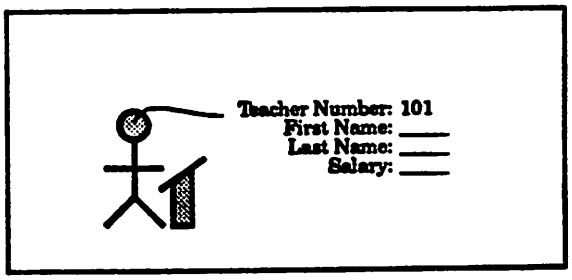

Figure 2: An example of pictorial task presentation.

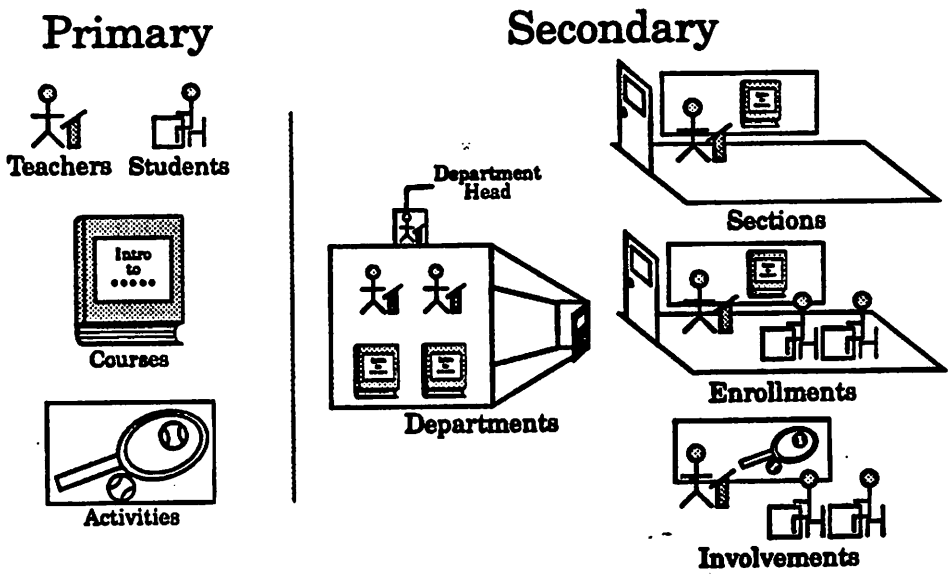

Figure 3: The icons used for pictures in the high school database.

Figure 4 shows a task description for a two table query. The task is to list all courses taught by each teacher. Notice that the join is not specifically stated but is implied because it uses attributes from different entities (i.e., teachers and section). The remainder of this section shows how this query can be entered into the three interfaces.

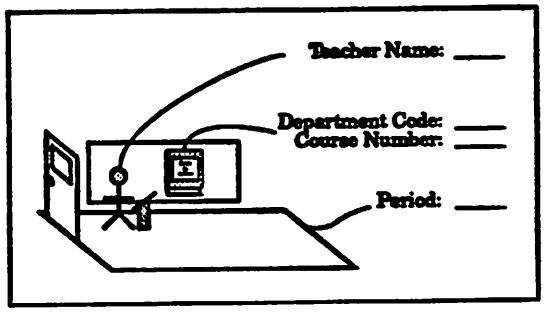

Figure 4: Task description: List all courses taught by each teacher.

2.1. SQL

The first interface was an artificial language (AL) interface called *isql* that used SQL [3]. SQL uses English keywords (e.g., *select*, *from*, and *where*) to make it more readable and easier to remember. SQL is claimed to be user friendly because it is a non-procedural language. An SQL query that solves the sample task is:

```
select teachers.tfname, teachers.tlname, sections.dcode,
   sections.cnumber, sections.speriod
from teachers, sections
where teachers.tnumber = sections.tnumber
```

2.2. Simplify

The second interface was a graphical language (GL) called Simplify developed by Sun Microsystems [2]. Earlier versions of Simplify were developed at Xerox PARC [9]. Simplify is essentially a graphical interface to an SQL processor. A query constructed in Simplify is translated into an SQL query which is then run. Consequently, Simplify and SQL users specify the same commands but they enter the commands in different ways.

Figure 5 shows a Simplify window that contains the query for the sample task. The query is created by the following steps:

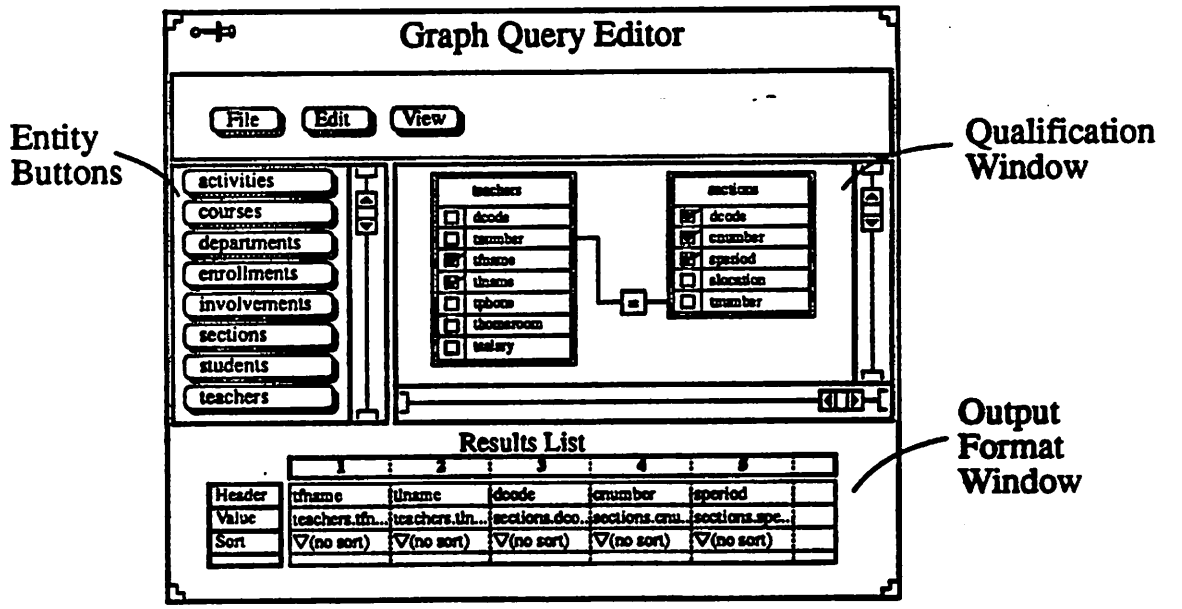

Figure 5: A sample Simplify screen.[2]

[2]The entity boxes in Figure 5 are shown side-by-side to enhance readability.

- Click on entity buttons for the tables in the query (i.e., teachers and sections). The entity boxes in the qualification window are displayed when the entity is selected.
- Click on the attributes in the entity boxes that are to be included in the query. Selected attributes are indicated by check marks (✓) and are automatically entered into the output format window.
- Click on tnumber in teachers, select "Create a join" in a pop-up menu, and click on tnumber in sections. This specifies the join between teachers and sections. A line joining these two attributes is displayed.

The query is then executed by choosing "Execute query" in a pop-up menu. The query result is displayed in another window.

2.3. *DataTalker*

The third interface was a natural language interface (NL), called DataTalker, developed by Natural Language, Inc. [1]. It allows users to specify a query by entering plain English on a keyboard.

One solution to the sample task is the following:

```
Show the courses taught by each teacher
```

It is important to realize that this is only one solution among many that the user can enter to request this information. For example, another way to ask for this information is:

```
List the teachers and their courses
```

One objective of a natural language interface is to allow a variety of questions to access the same data.

All interfaces used the Ingres DBMS to store data. The SQL interface used in the experiment was the *isql* full-screen editor supplied with Ingres 5.0. The Simplify interface used was version 1.0 (beta) released in May 1989. The DataTalker interface used was version 3.0 released in March 1989. The experiment was performed on a Sun 3 computer. The same database was used for all subjects.

3. The Experiment

The experimental design is shown in figure 6. Subjects were assigned to one of four groups (the vertical axis) based on prior experience and were randomly assigned to one of three treatments (the horizontal axis). Each subject worked through two phases: a learning phase and a performance phase. The learning phase was composed of task levels that covered seven types of queries. The query types taught in each level are described in figure 7. The performance phase included queries to test how well the user learned to use the interface to which they were assigned during the experiment.

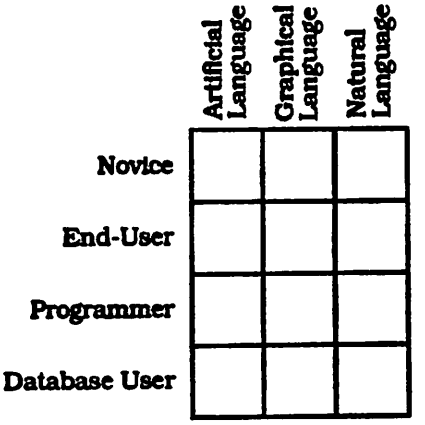

Figure 6: Experimental design.

LEVEL	DESCRIPTION
L1	One table, no restrictions
L2	One table, one restriction
L3	One table, two restrictions
L4	One table, one or two restrictions, sorting
L5	One table, no restrictions, aggregation counting
L6	Two table join
L7	Three table join

Figure 7: Learning phase task level definitions.

Most subjects were drawn from students at the University of California at Berkeley. They were all volunteers who responded to announcements or advertisements soliciting subjects. Each subject spent about two hours working with one of the interfaces. The subjects were classified according to prior computer experience as shown in figure 8.

GROUP	DEFINITION
Novice	less than 10 hours of computer experience ever
End User	experience with applications (e.g., spreadsheet or word processing), but no programming experience
Programmer	programming experience, but no database experience
Database User	knowledgeable in SQL or QUEL

Figure 8: Subject group definitions.

Over 80 people participated in the experiment of which 55 were used for analysis. Some subjects were used for a pilot study and others were rejected because they had

experience with the particular interface to which they were assigned. There were fifteen subjects in each user group except for database users. Since the database users knew SQL, none of them were assigned to the AL condition. Consequently, only ten database users were included in the study.

Subjects were randomly assigned to one of the three treatments: AL, GL or NL. Everything was identical about the three treatments except for the interface used and the content of the help materials provided during the learning phase which is described below.

Each subject was given a brief introduction to the experiment, the high school database, and the pictorial task presentation method. The only information they were given regarding the database is shown in figure 9 which was briefly explained conceptually (e.g., "For every teacher, we know his or her department code, teacher number, name, phone number, ..."). The ER diagram in figure 1 was not made available to the subjects.

The subjects were told they would be getting information out of the computer using a query language. After being given a sample task picture, the subjects were shown how that task could be performed with the interface to which they were assigned.

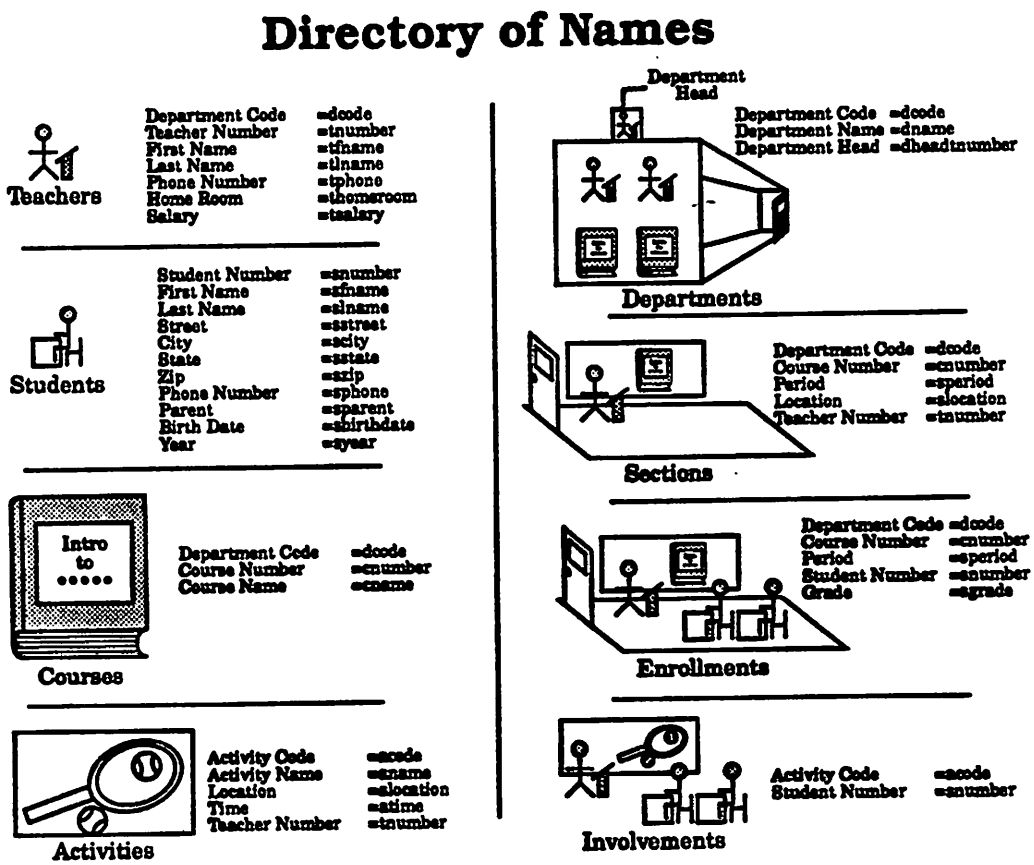

Figure 9: Database attributes description sheet.

3.1. *Learning Phase*

Subjects then worked through the learning phase based on the procedure shown in figure 10. Subjects were given a task beginning at level 1 which they were to perform. If they could not solve the task, help was provided in the form of written advice. Help was broken down into the four levels shown in figure 11. After completing the task, another task of similar difficulty was given. These tasks were given until a task could be solved without help at which point subjects were advanced to the next level. A minimum of two task per level was required even if subjects could complete the first task without help.

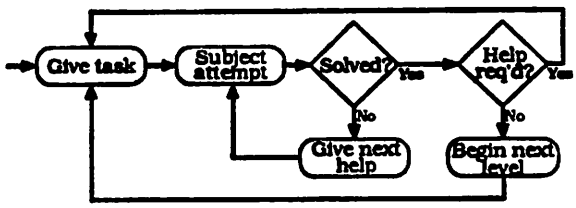

Figure 10: Treatment in Learning Phase

LEVEL	DESCRIPTION
Hint	A two sentence description of the essence of the solution.
Example	A query used to solve a similar task.
Explained Example	The steps followed to generate the example query.
Answer	A query that would solve the current task.

Figure 11: Help level descriptions.

3.2. *Performance Phase*

After completing all levels, subjects moved to the performance phase of the experiment. In the performance phase, subjects were give a series of tasks that represented query types in the learning phase as well as query types not seen before. No help was given during the performance phase. Subjects were allowed to work until they were successful or until they were making no progress toward the solution. The performance phase tasks and the corresponding learning phase levels are shown in figure 12.

The non-existence and self-referential join tasks were included to see if subjects were able to extend their knowledge of the interfaces to situations that had not been explicitly taught, and to see how experts on each of the interfaces would deal with complex tasks.

PERFORM. TASK	DESCRIPTION	LEARNING LEVEL
P1	One table, no restrictions	L1
P2	One table, two restrictions	L2 & L3
P3	One table, aggregate counting	L5
P4	Three table join	L7
P5	Two table join, one restriction, sorting	L4 & L6
P6	Non-existence	none
P7	Self-referential join	none

Figure 12: Performance phase task level definitions.

A non-existence query asks if particular instances do not exist. For example, find all activities that have no student participants. A self-referential join query has a join of a table to itself.

Each subject was videotaped during the experiment. Records were also kept by the experimenter (J. Bell) during each session. These records included the time at each step through the procedure (e.g., when the task was given, when help was given, when the subject moved to next level, etc.) as well as verbal comments made by the subject and difficulties the subject encountered (e.g., forgetting to clear out an old query before creating a new query). In addition, for the AL and NL treatments, a transcript file was recorded.

4. User Performance Results

The quantitative results of the performance phase of the study are presented in this section. Two metrics were examined: success and average time to task completion. Success is defined as completing a given task in a level. This metric assessed how well subjects could perform a variety of tasks after the training they received in the learning phase.

The second metric was the average time required to complete each task. The average is considered only when tasks were successfully completed by at least half of the subjects. This metric assessed how much work was required of subjects to complete tasks in the performance phase.

The performance phase was divided into two groups: trained tasks (P1 thru P5) and untrained tasks (P6 and P7). These groups will be examined below.

4.1. *Trained Tasks*

The first metric examined for the trained tasks was the success rate. The number of tasks completed successfully out of all of the trained tasks is shown by user group and by interface in figure 13. Five tasks (i.e., P1 thru P5) were given to each of the five

	SQL	SIMPLIFY	DATATALKER
NOVICE	0	0	3
END USER	0	0	3
PROGRAMMER	0	0	5
DB USER	•	0	5

Figure 20: Number of subjects who were successful on non-existence task.

The number of subjects who were successful on the self-referential join query is shown in figure 21. Only one subject, a database user using Simplify, was able to perform this task. Hence, Simplify is capable of performing self-referential joins, but most subjects were unable to discover how to do it. Furthermore, no SQL subjects were successful because self-referential joins in SQL are hard to figure out if you've never done one before. No DataTalker subjects were able to perform the self-referential join. This result does not mean that DataTalker cannot solve the task but that no one was able to discover how to do it. We were somewhat surprised by these results since we thought more subjects would get this query. This result suggests that subjects did not really understand the join concept and/or the way joins are specified in the three interfaces. Further study of this issue is needed.

	SQL	SIMPLIFY	DATATALKER
NOVICE	0	0	0
END USER	0	0	0
PROGRAMMER	0	0	0
DB USER	•	1	0

Figure 21: Number of subjects who were successful on self-referential task.

The timing data on the untrained tasks did not reveal any obvious patterns.

5. Qualitative Results

In a case study, qualitative data is a rich source of information which helps the researcher understand more about the object of study. Subject comments both during and after the experiment, common mistakes and usage patterns, and experimenter observations all provided insight concerning what was easy and hard about each of these interfaces for the different types of tasks in this study. This section contains a discussion of the qualitative results as they relate to the following topics: 1) single and multiple restriction tasks, 2) join tasks, 3) counting tasks, 4) unknowingly getting the wrong answer, 5) compounded uncertainty, and 6) the natural language connection.

5.1. *Single and Multiple Restrictions*

For end users, the comparison between single and multiple restrictions highlights some of the differences between SQL, Simplify, and DataTalker. The number of tasks needed to complete the single restriction level (level L2) during the learning phase was the same on all interfaces. The number needed to complete multiple restrictions level (level L3), however, varied widely. Simplify subjects required fewer tasks, while SQL subjects required more tasks, and DataTalker subjects required even more tasks.

In Simplify, multiple restrictions are specified by repeatedly forming single restrictions. A conjunction (i.e., 'and') is the implied connection between these restrictions. Consequently, all that subjects had to do for multiple restrictions was repeatedly apply what they learned for single restrictions. Multiple restriction queries involved almost no learning beyond single restriction queries.

In SQL, multiple restrictions must be connected with the appropriate operator ('and' or 'or'). Few end users were able to figure this out without help, so the use of 'and' for these queries had to be learned. Hence, multiple restriction queries involved some learning beyond single restriction queries. The most common guess by subjects was the use of a comma (','). The use of the comma is very reasonable given that it is used for conjunction in the select phrase.

In DataTalker, multiple restrictions are stated in standard English. Because of that, they can be specified in a variety of ways. For example, the restriction "city='Berkeley' and age>15" on students can be specified by any of the following phrases:

- which students live in Berkeley and are older than 15?
- of students who live in Berkeley, which are older than 15?
- which students live in Berkeley? which of them are older than 15?
- which students who live in Berkeley are older than 15?

Of these various statements subjects had to learn which would work with DataTalker. Furthermore, these statements were very different from, and more complex than, statements with only one restriction. Consequently, DataTalker users had more to learn when beginning to work with multiple restrictions than users of SQL or Simplify.

5.2. *Joins*

Joins between tables form a class of very difficult queries. In order to specify a join, users must understand how a join operator works and they must understand the structure of the tables they are to join.

In SQL, the method for specifying joins is generally easy to understand. All of the tables to be joined are listed in the *from* phrase, and the joins are stated as restrictions

in the *where* phrase in the form of *relation1.attribute = relation2.attribute*. Assuming the user understands the join operation, the hard part is in knowing what to join. SQL provides no help to the user so specifying a query is not easy. Some subjects expressed the desire to write out the join on paper and then translate that into SQL.

Joins are easier to specify in Simplify than in SQL. After the appropriate tables have been selected, lines are drawn between the attributes to be joined. The hard part again is in determining what to join. However, Simplify's graphical display of the join seems to be helpful in thinking about it. Subjects could look at the display and quickly verify their joins. In contrast, subjects using SQL had a harder time re-reading their SQL join to see if it was right.

A problem with Simplify occurred when three table joins were attempted. Simplify did not allow subjects to reposition tables in the window, and the default position was in a single column. The result was that the window had to be scrolled up and down to see all of the tables. The problem was especially bad when both ends of a particular join line could not be seen on the screen at one time. It was not possible in such cases to look at the screen and immediately understand what the command was. Rather, the screen had to be scrolled back and forth in order to see the entire command.

Specifying joins was an area in which DataTalker was clearly superior because users did not have to specify joins at all. All potential joins in the database had to be described to DataTalker in the connection process. When users asked a question that involved attributes in separate tables, DataTalker automatically included the join terms between the tables. Subjects who used DataTalker were unaware that joins ever took place. In fact, some subjects thought that some of the two table join tasks had been given before since they were so much like the earlier tasks in other respects.

This power of DataTalker is similar to the universal relation interface as described by Maier, et al. [18]. A universal relation is a single relation that represents the entire database. Hence, instead of having separate tables for teachers, students, courses and so on, all attributes are in a single table. Users are freed from having to decide which table to include and from having to specify any joins. They just need to specify which attributes they want without reference to tables. Consequently, universal relation interfaces have the potential of giving this power of DataTalker to an artificial or graphical language interface.

Like DataTalker, however, the universal relation model has a problem with ambiguity which needs to be resolved. A system based on the universal relation model must decide which underlying relations to join and what the join terms should be for each query. The queries, however, may be ambiguous. For example, the query to list the teachers and their courses could mean the teachers who teach each course or the teachers who are the head of the department of each course. A universal relation system, like DataTalker, must determine which interpretation the user intended since the query itself did not uniquely identify the intended meaning.

SQL provides a similar though more limited functionality with views. Views are pre-formed queries that can be treated by the user as if they were actual relations. Moreover, views can be defined so as to provide a single relation for every potential join. Users would only need to determine which view to use as opposed to determining the right join terms. For example, one view would be the combination of teachers and the courses they teach and another view would be the combination of teachers and the courses in their department. The user only needs to decide which view is wanted. In Simplify or in SQL without the views defined, users would have to decide which relations to join and which join terms to use. In DataTalker or in a universal relation interface, the system must try to determine which join is intended.

The disadvantage of using views is that the burden is on users to determine which views to use. In this experiment, subjects were sometimes uncertain about which relation to use. If views were defined for all possible joins, there would be more 'relations' to choose from, and the differences between them would be more subtle. Choosing which relation to use, however, is undoubtedly much easier than figuring out the join terms. More research is needed to determine how using views compares with natural language and universal relation interfaces.

5.3. Counting

Counting was a very simple concept for subjects to understand, but it met with varying difficulty in each of the interfaces. Subjects quickly realized what they were to do, but they often had difficulty in doing it. Figure 22 contains the picture for a counting task. This task asks for the number of students involved in each activity.

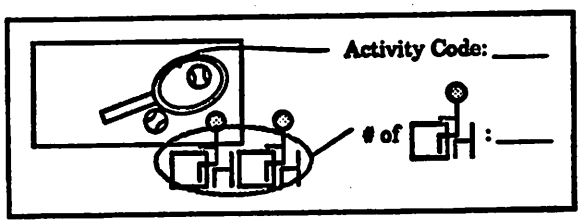

Figure 22: Counting task picture.

Figure 23 contains the SQL query that solves this task. In SQL, counting queries involve two changes to the retrieval tasks. First, the phrase *count(*)* is added to the *select* phrase. The asterisk ("*") can be replaced with an *attribute* that identifies the groups of objects to be counted. For example, *count(acode)* could be used for the example task instead. The asterisk method was taught because it was believed to be easier to learn. Second, a *group by* phrase was added after the *from* phrase. In the *group by* phrase, attributes were entered to identify the groups of records to be treated together. In the example, *group by acode* means that all involvements that have the same activity code should be grouped and counted together.

```
select  acode,  count(*)
from  involvements
group  by  acode
```

Figure 23: SQL query to solve the counting task.

Programmers had an especially hard time learning counting in SQL. They could not see the logic in the use of the *count* and *group by* phrases. Consequently, they refused to believe it had been designed as it appeared. Once they learned how it worked, however, they were able to perform counting tasks relatively quickly. One database user noted that he had to think about the data differently in order to understand counting tasks. For the simple select commands not involving counting, he could think of the data as simple lists. However, in order to do the counting commands, he had to think of the data as relational tables. Given this change in thinking, counting was not a problem for him.

End users did not have this difficulty with counting in SQL. Some end users had the attitude that it did not make much sense, but it worked, so they would go with it. The programmers, on the other hand, wanted it to be more reasonable. The end users were not as concerned about why things were they way they were. If it worked, they were satisfied.

In Simplify, counting is specified by *creating a computation* on an attribute representing the data to be counted. This language construct is equivalent to the *count(attribute)* phrase in SQL. Simplify does not require a direct specification of the *group by* phrase as in SQL. Rather, all attributes that are selected to be displayed are included in the *group by* phrase.

Subjects had little trouble in Simplify once they found the pop-up menu containing the *create computation* menu. There is a sub-menu under *create computation* that allows different types of computation (e.g., max, min, and average). However, since counting was the default, most subjects never used the sub-menu.

In DataTalker, counting tasks can be performed by telling the computer to count the data, or by asking the computer how many records there are. Most subjects initially asked the computer to *show the number of students in each class*. This sentence was interpreted as referring to student numbers, instead of the number of students. Once subjects read the first help given at this level suggesting the *count* and *how many* phrases, however, they had almost no trouble at all.

5.4. *Unknowingly Getting Wrong Answer*

We expected to find that subjects using DataTalker would unknowingly get the wrong answer more often than when using SQL or Simplify. However, this behavior happened with about the same frequency on all systems.

In DataTalker, the most common mistake involved teachers. Subjects would get a list of courses and need to add the names of the teachers of the courses to that list. The simplest approach was to say 'include teachers.' However, in some situations, this command resulted in adding the teachers who were department heads of the department of that course instead of the teachers who taught the course. If subjects had been paying close attention, they might have been suspicious when all courses in each department were taught by just one person, but none of them did. This task was not especially complex, and the problem did not occur because of lack of knowledge by the user, though greater knowledge might have made the users more aware of the possibility.

In SQL and Simplify, unknowingly getting the wrong answer occurred on the join, non-existence, and self-referential join tasks. In each of these cases, subjects managed to get an answer that had the right structure which they assumed was right but was not. For example, in getting a list of students who were enrolled in MTH 110 second period, some subjects got a list that included students of any MTH course, or students of any 110 course such as French or Art, or students in any course offered second period. All of these problems resulted from using incorrect join terms. The result looked like a list of students in MTH 110 second period, but the data were incorrect. For tasks in SQL and Simplify that subjects knew more about, this problem did not occur at all.

Generally speaking, the wrong answer occurred unexpectedly only at the fringes of a subject's knowledge of the database and the interface they were using.

5.5. Compounded Uncertainty

Each part of a query that is uncertain in the user's mind dramatically increases the difficulty of completing the query. All combinations may have to be tried before finding an appropriate way to write each part of the query.

This compounded uncertainty is most apparent in DataTalker. For example, consider the query 'List the departments with code MTH alphabetically.' This task does not work in DataTalker, so a user would have to determine what was wrong and fix it. There are at least three different components to the query: the 'list' component, the math department component, and the sorting component. There could be problems with any of these components or with the interaction between components. To fix the query, subjects often experimented with each individual component until they were successful. Users were fairly confident in 'list' since they used it for many queries, but 'show' and 'include' are possible alternatives. 'Departments with code MTH' could be phrased many ways. For example, perhaps 'department code' should be used instead of just 'code.' Or perhaps 'MTH' should be in quotes or should not be capitalized. The correct phrasing is 'the MTH department.' The sorting phrase 'alphabetically' was used by subjects as they were first learning sorting in DataTalker. Other possible phrasings include 'in sorted order,' 'by department name,' or 'in alphabetical order.' The correct approach is not to sort until after the data has been retrieved. Consequently, three versions of the list component, four versions of the departments component, and four

versions of the sorting component give at least 48 possible phrasings of this query. Partial success on queries may help users narrow the search, but many possibilities remain to be tried, and none of these queries will be successful until sorting is put into a separate query.

This problem highlights the need for a good *connection* with a natural language interface. Any weakness in the connection compounds these problems.

In SQL, compounded uncertainty was less likely because subjects were more confident in the standard parts of the query structure. Once they learned the basic structure of *select, from* and *where*, they rarely called them into question. Instead, they focused primarily on the new aspects of the current task level.

In Simplify, compounded uncertainty was the least likely because there was immediate feedback. When a table was displayed on the screen, the subjects generally knew very quickly whether or not it was the right table. Subjects did not have to worry about syntax, so there was no uncertainty about forgetting a comma or using the wrong word. The greatest uncertainty in Simplify, as in SQL, was found with joins. For example, subjects were uncertain about whether or not they had joined the appropriate attributes.

5.6. Natural Language Connection

Ogden has noted that the quality of the connection must be considered when evaluating natural language interfaces [20]. An inadequate connection can make a good natural language system unusable. Furthermore, it is unreasonable to expect that actual users will be able to develop as consistently good connections as the authors of a natural language system. In practice, such connections often take a long time to produce, and they require repeated iterations of examining inputs that the system cannot understand and modifying the knowledge base. Experts claim that this process may take as long as 9 months. [16]

We spent several weeks working on the knowledge base. Then, a pilot study was run with several users to debug the experimental design and the connection. Problem areas were discussed with NLI and the changes they suggested were made. Nonetheless, not all problems were fixed.

For example, DataTalker had trouble with the use of the words *location* and *room* for the place of meeting for classes and activities. Classes can only meet in classrooms whereas activities can meet in classrooms or in any other location. During the experiment, references to *locations* were interpreted as referring to activities while references to *rooms* were interpreted as referring to classes. Consequently, when subjects asked about the location of classes, DataTalker attempted to form a join between classes and activities. The closest join was to combine classes and activities who were led by the same teacher. This join, however, was unrelated to the subjects' queries.

It may be possible for some of the problems encountered with the connection to be fixed. Furthermore, new releases of DataTalker may also help. If these problems can be resolved, we expect that subject performance on DataTalker will be better. How much better, however, can only be determined by further experimentation.

6. Experts Experiment

This section describes the results of a small scale experiment that was performed using expert users of each of the interfaces. This experiment serves as a benchmark to compare with the performance of subjects in the performance phase. Only two experts on each interface were included since the experts themselves were not to be studied.

This experiment was performed in exactly the same manner as the other experiment. Subjects worked through the learning phase so they could learn about the database in the same way that the other subjects did. Their performance in the performance phase is presented here.

The same two metrics used in the performance phase of the other experiment were also recorded for the experts. That is, we recorded their success and the amount of time required to complete the queries.

The details of the performance of these experts can be found in [5]. Only the results will be discussed here.

On all systems, experts did better than the best non-experts (programmers and database users) but not by a wide margin. The experts were often faster and more successful though not always.

The biggest difference between experts and non-experts was for subjects using SQL on the untrained tasks. One or both of the SQL experts were able to complete these tasks while no other subjects using SQL could complete them. This result is not surprising since SQL has a great deal of power using features that are hard to learn. And, these tasks are common problems that database users learn to solve. So what is probably happening is that expertise implies being able to solve these types of problems.

Simplify experts did not do much better than non-experts. This result was expected because most of the functionality of Simplify is easy to find. All Simplify features that the experts used were immediately available to non-experts in the menus on the screen. This transparency is in sharp contrast to SQL where the more complex features cannot be discovered without explicit teaching. This result supports the conventional wisdom that graphical user interfaces are easy to learn and use.

DataTalker experts did no better overall than non-experts. Probably the most important experience for DataTalker users is experience on the actual database to be used. Experience using DataTalker in other situations may be helpful, but performance is more dependent on knowledge of the specific words and phrases that work with a specific connection rather than general experience with the interface.

7. Conclusions

The following conclusions can be drawn from this experiment.

7.1. No Interface is Best, Yet

It comes as no surprise that none of these interfaces outperforms the others in all cases. Each does better under certain conditions. Furthermore, each interface can be improved.

7.2. SQL Best for Experts and Novices

It also comes as no surprise that SQL is very effective for those with SQL expertise. If users have the time and ability to learn it, SQL is the best all-around tool. However, both Simplify and DataTalker have shown that they may be able to out do SQL even among experts. For example, three table joins were performed by experts on Simplify and DataTalker much faster than experts on SQL.

The surprise for SQL is that novices did very well on simple tasks. In the performance phase, the success rate was about the same on SQL and Simplify, and the time required to complete a task was slightly greater on SQL than on either Simplify or DataTalker. However, the learning phase and the qualitative data showed that SQL was less complex for these users than either Simplify or DataTalker. Because of this simplicity, novices were able to learn how to use SQL and how to recover from their mistakes relatively well.

Simplify had a simple basic structure, but the use of a three button mouse was difficult throughout the study for the novices.

The basic idea of DataTalker was easy to grasp for the novices, but they had a hard time learning how to adjust to its idiosyncrasies. Consequently, they were less consistent, though they could perform a broader range of tasks.

7.3. Simplify Best for Programmers When it Works

Simplify was very strong in general on the trained tasks, but was best for programmers. These subjects learned Simplify quickly, and were very consistent in their success. Simplify was also very good for end users, but DataTalker was better in that category. One of Simplify's greatest strengths was its ability to help users with the structure of the database. By seeing the list of tables on the screen, subjects could more quickly choose the desired table, and they could verify that they got the right table by checking the attributes that were displayed. Furthermore, subjects could think through their joins by looking at the Simplify query. Subjects could verify their query directly by reading it, instead of having to interpret the query and then verify that interpretation as in SQL.

In contrast, Simplify did very poorly on the untrained tasks. Simplify was incapable of performing one of these tasks, and the other did not fit within the users' understanding of what Simplify queries meant. Furthermore, Simplify did not support disjunctions (i.e., 'or') for multiple restrictions. The greatest need of Simplify is the ability to perform more types of queries. What it does, it does well, but its limits are too narrow.

7.4. *DataTalker Best for End Users, Otherwise Varied*

DataTalker was strongest for end users. These subjects were able to perform a greater variety of tasks, and they did as well as or better than end users on the other interfaces throughout the tasks (aside from level P2). They did not have to concentrate on the structure of the database or the relational operators of the tasks. Tasks that are conceptually similar were similar from the subject's point of view. In contrast, conceptually similar tasks in SQL sometimes had to be solved with very different queries.

DataTalker was generally powerful but inconsistent. Subjects sometimes did very well, but they also did poorly at times. This inconsistency was overlooked by some subjects, but it led to great aggravation for others.

We conclude that both the graphical and natural language interfaces tested show promise as a better interface overall than the artificial language interface. Today, however, the artificial language still has the best general applicability.

As our conclusions state, these results apply specifically to the interfaces tested. However, to the extent that other query interfaces are like these interfaces, the same results can be expected to apply [10]. For example, for other graphical languages that follow a model similar to Simplify (e.g., VQL from Sybase [4]), and for other natural language interfaces that have a similar coverage of English and allow a similar dialog as in DataTalker, the same results can be expected to occur. Actual experimentation is needed, however, to explore the similarity of other interfaces to these systems, and consequently, the applicability of these results.

Many other topics for research in the comparison of interfaces for ad hoc database query also remain. A focused study on each of the user groups, especially end-users, needs to be performed in order to determine a specific and reliable comparison of the three interface styles. Research should also be performed which focuses more on the qualitative aspects such as evaluating user strategy when working with natural language interfaces or developing a model of user knowledge and behavior when working with query interfaces in general. Research should also consider other interfaces such as NLMenu which is a menu-based interface to a natural language system [27]. Finally, similar research should be performed as enhancements are made to these interfaces in order to determine how relative user performance is affected.

References

1. "Natural Language™ Database Retrieval System User Manual." 1988.
2. "SunSimplify™ 2.0 Reference Manual." 1989.
3. "Using INGRES Through Form and Menus." 1989.
4. "VQL." 1989.
5. Bell, J. E. "A Case Study of Ad Hoc Query Interfaces to Databases." Ph.D. Dissertation. University of California, Berkeley. 1990.
6. Bierman, A. W., B. W. Ballard and A. H. Sigmon. "An experimental study of natural language programming." International Journal of Man-Machine Studies. **18**: 71-87, 1983.
7. Boyle, J. M., K. F. Bury and R. J. Evey. "Two studies evaluating learning and use of QBE and SQL." Proceedings of the Human Factors Society 27th Annual Meeting. 1983. Santa Monica, CA. Human Factors Society.
8. Brosey, M. and B. Shneiderman. "Two experimental comparisons of relational and hierarchical database models." International Journal of Man-Machine Studies. **10**: 625-637, 1978.
9. Cattell, R. G. G. "An Entity-based Database User Interface." ACM SIGMOD. 144-150, 1980.
10. Cornfield, J. and J. W. Tukey. "Average values of mean squares in factorials." Annals of Mathematical Statistics. **27**: 907-949, 1956.
11. Damerau, F. J. "Operating statistics for the transformational question answering system." American Journal of Computational Linguistics. **7**: 30-42, 1981.
12. Fink, P. K., A. H. Sigmon and A. W. Biermann. "Computer control via limited natural language." IEEE Transactions on Systems, Man, and Cybernetics. **15**: 54-68, 1985.
13. Greenblatt, D. and J. Waxman. "A study of three database query languages." Databases: Improving Usability and Responsiveness. Shneiderman ed. 1978. Academic Press, Inc. New York.
14. Jarke, M., J. A. Turner, E. A. Stohr, Y. Vassiliou, N. W. White and K. Michielsen. "A field evaluation of natural language for data retrieval." IEEE TSE. **SE-11(1)**: 97-114, 1985.
15. Krause, J. "Natural language access to information systems. An evaluation study of its acceptance by end users." Information Systems. **5**: 297-319, 1980.
16. Limperis, E. Personal communication. 1990.

17. Lochovsky, G. H. "Data base management system user performance." Ph.D. Dissertation. University of Toronto, Canada. 1978.

18. Maier, D., D. Rozenshtein, S. C. Salveter, J. Stein and D. S. Warren. "Toward logical data independence: A relational query language without relations." ACM SIGMOD Conference. 1982.

19. Moher, T. G. "Estimating the distribution of software complexity within a program." CHI '85 Proceedings. 61-64, 1985.

20. Ogden, W. C. "Using Natural Language Interfaces." Handbook of Human–Computer Interaction. Helander ed. 1988. Elsevier Science Publishers.

21. Ogden, W. C. and S. R. Brooks. "Query Languages for the Casual User: Exploring the Middle Ground between Formal and Natural Languages." CHI'83. 161-165, 1983.

22. Ogden, W. C. and A. Sorknes. "What do users say to their natural language interface?" Proceedings of Interact '87 - 2nd IFIP conference on Human-Computer Interaction. 1987. Amsterdam. Elsevier Science.

23. Reisner, P., R. F. Boyce and D. D. Chamberlin. "Human factors evaluation of two database query languages – Square and Sequel." National Computer Conference. 1975. Anaheim, CA. AFIPS.

24. Shneiderman, B. "Improving the human factors aspect of data base interactions." ACM TODS. 3(4): 417-439, 1978.

25. Small, D. W. and L. J. Weldon. "An experimental comparison of natural and structured query languages." Human Factors. 25: 253-263, 1983.

26. Soloway, E., K. Ehrlich and J. B. Black. "Beyond Numbers: Don't Ask 'How Many'...Ask 'Why'." Proceedings of the Conference on Human Factors in Computing Systems. 1983. Boston, MA.

27. Tennant, H. R., K. M. Ross and C. W. Thompson. "Usable natural language interfaces through menu based natural language understanding." CHI 1983 Conference on Human Factors in Computer Systems. 1983. Boston, MA. North-Holland.

28. Thomas, J. C. and J. C. Gould. "A psychological study of query by example." National Computer Conference. 1975. Anaheim, CA. AFIPS.

29. Welty, C. and D. W. Stemple. "Human Factors Comparison of a Procedural and a Nonprocedural Query Language." ACM TODS. 6: 626-649, 1981.

30. Wolcott, H. F. "Ethnographic research in education." Complementary methods for research in education. Jaeger ed. 1988. American Educational Research Association. Washington, DC.

31. Zloof, M. M. "Query by Example." National Computer Conference. 1975. Anaheim, CA. AFIPS.