A Graphical Filter/Flow Representation of Boolean Queries: A Prototype Implementation and Evaluation

Degi Young
Ben Shneiderman *

Human-Computer Interaction Laboratory &
Department of Computer Science
University of Maryland
College Park, MD 20742

February 1993

Abstract:

One of the powerful applications of Boolean expression is to allow users to extract relevant information from a database. Unfortunately, previous research has shown that users have difficulty specifying Boolean queries. In an attempt to overcome this limitation, a graphical Filter/Flow representation of Boolean queries was designed to provide users with an interface that visually conveys the meaning of the Boolean operators (AND, OR and NOT). This was accomplished by implementing a graphical interface prototype that uses the metaphor of water flowing through filters. Twenty subjects with no experience with Boolean logic participated in an experiment comparing the Boolean operations represented in the Filter/Flow interface with a text-only SQL interface. The subjects independently performed five comprehension tasks and five composition tasks in each of the interfaces. A significant difference ($p < 0.05$) in the total number of correct queries in each of the comprehension and composition tasks was found favoring Filter/Flow.

* address correspondence to Ben Shneiderman.
Ben Shneiderman is also a member of the Systems Research Center.

1 Introduction

"Much research and experience with Boolean retrieval systems indicates clearly and repeatedly that Boolean search formulation syntax and retrieval techniques are not very effective in search performance and not very usable or efficient search methods for end users" [Hildreth, 1988].

Many attempts have been made to overcome the "Boolean bottleneck" in the use of Fourth Generation Languages. These attempts include a linear text such as that used in Structured Query Language (SQL), tabular structures such as those used in Query By Example (QBE) and various other graphical representations [Reisner, 1988; Jarke and Vassiliou, 1986]. However, despite these attempts, extracting information using a Boolean query is often too abstract for novice users and problematic for many experienced users.

1.1 Literature Review

In order for users to form queries using the linear text approach, they must be familiar with Boolean logic; (i.e., the logical operator AND which is the intersection of two conditions, the logical operator OR which is the union of two conditions and the logical operator NOT which is the negation of a condition). Equipped with this knowledge users can extract information simply by typing the query. Indeed, in an experiment using both secretaries and college business administration students, the subjects preferred SQL to QBE, because of its structural approach and because fewer steps have to be made when forming a query on the computer [Boyle, Bury and Evey, 1983].

The difficulty of using the linear text approach lies in attaining the knowledge to form the query. One of the reasons for this difficulty is that novice users use terms that they are familiar with (indeed the terms *and* and *or* are used often in natural language); but these terms take on different meanings when used to form a query. Thus, when constructing queries in SQL, users tend to make errors because they resort to their knowledge of English. This result was noted in experiments conducted by Greene, Devlin, Cannata and Gomez [1989], Michard [1982], and Boyle et al. [1983]. One of the common mistakes users made was to substitute the AND logical operator for the OR logical operator when translating an English sentence to a linear text query.

Reisner, Boyce and Chamberlin [1975] further noted that the subjects would insert words from the English sentence into the SEQUEL or SQUARE query in place of the correct table name, column name, or data value.

Another requirement of SQL further complicates the formation of queries. In order for users to form complex queries, they must be familiar with the rules of precedence and the use of parentheses. Michard, Greene et al., Boyle et al., and others have indicated that novice users have difficulty using parentheses, in particular nested parentheses. Greene et al. went further to note that an increase in the complexity of a query should not mean an increase in the difficulty of the method of specification.

The tabular representations of Boolean queries make some attempts at either reducing the use of Boolean logic (QBE) or eliminating the use of Boolean logic (Truth-table Exemplar-Based Interface (TEBI)). Users can form simple queries in QBE by placing either variables or constants on various sections of the table. Initial studies conducted by Thomas and Gould showed that subjects made accurate decisions about simple "AND/OR" constructions and seldom used the wrong values when they translated English sentences to form queries in QBE. Follow-up studies conducted by Greenblatt and Waxman [1978] not only reaffirm these findings but go a step further to indicate that QBE is a superior language to either SQL or the algebraic language that they used, in terms of learnability and application ease. Unfortunately, when forming complex queries in QBE, users must invoke a condition box and express the complex query as a Boolean expression. The disadvantage of using Boolean expressions, in particular parentheses, that existed in the linear text still exists in QBE. Furthermore, as indicated by an experiment conducted by Boyle et al. [1983], subjects have difficulty knowing how and when to use the condition box.

TEBI was designed and developed by Greene and others. It is a tabular representation of a query language that is void of the logical operators, parentheses or fixed syntax. In order to form a query in TEBI, users must be able to recognize, rather than generate, the form of the query that they are interested in. This method serves users who are unsure of the query that they want to form and have the time to establish an interactive form of communication with the computer.

However, if users know what query they want to form and their difficulty lies only in expressing the query then this form of representation may not be desirable.

The advantages of using a graphical interface are numerous. For example, users tend to complete their tasks quickly and accurately, feel less frustrated and show fewer signs of fatigue. Unfortunately, as mentioned by Lundh and Rosengren [1990], there exists some difficulty in expressing Boolean expressions graphically.

An attempt was made by Michard [1982], who used Venn diagrams to show the operations on sets. In this interface, each chosen attribute is represented by a circle and users form queries by pointing at the desired portion of the intersecting circles. A human factors comparison was done between a linear text language called "TEST" and his Graphical Query Language. He used twelve college students who had experience with basic Boolean algebra and the use of Venn diagrams. In his study he found that subjects performed better when they used the Graphical Query Language interface.

A graphical Boolean interface based on the relative positions of tiles within a block structure was developed by Anick, Brennan, Flynn, Hanssen, Alvey and Robbins [1990]. Their interface deals only with intersection and union operations -- negation and clustering are not addressed. Furthermore, the implied semantics of horizontal and vertical relationships between the tiles is sometimes confusing and ambiguous, making it difficult to form certain queries.

2 The Filter/Flow Interface

Filter/Flow was designed to present the Boolean operators using a metaphor that graphically conveys the meaning of the operators. This concept originated from Shneiderman [1991]. Similar to TEBI, this interface is void of the logical operators, parentheses or any form of fixed syntax. However, unlike TEBI, this interface can be used to generate queries and is graphical. In order to use Michard's Graphical Query Language, users must be familiar with Venn Diagrams and the operations that they represent. Filter/Flow provides a novel conceptual model of queries and was designed to help users formulate and express queries.

2.1 Description Of Filter/Flow

Initially, the screen contains two scrolling lists. The database list is placed on the extreme left hand side of the screen and the result list is on the extreme right hand side. The database list consists of the tuples of the initial database, while the result list consists of the tuples that satisfy the query formed.

At the top of the screen, arranged horizontally, are buttons that are used to represent the attribute names. In this interface, they are LOCATION, MANAGER, SALARY, TITLE and MORE. The MORE button is used to indicate that additional attribute names may exist.

At the bottom of the screen are the action buttons of the Filter/Flow interface; they are labeled as NEW QUERY, CLEAR FLOW, FLOW and QUIT (Figure 1).

Users form queries by clicking on the attribute name buttons. This causes the attribute menus to be displayed on the screen. The attribute menus may be dragged and placed at various positions on the screen. Users click on the attribute value to select the appropriate values. By selecting the FLOW action button, users may view the logical representation of the query that they formed in the form of water (displayed in blue) flowing through filters. The NEW QUERY button clears all the attribute menus and the flow that are displayed on the screen. The CLEAR FLOW button clears only the flow displayed on the screen and leaves the attribute menus intact. The QUIT button enables users to exit Filter/Flow.

2.2 The Filter/Flow Operators

INTERSECTION OPERATOR (AND):

In order to specify an intersection two or more conditions are needed. A condition is indicated by selecting an attribute menu and selecting an attribute value within that menu. These attribute menus must be in different columns and there must be a flow that passes through both of the menus.

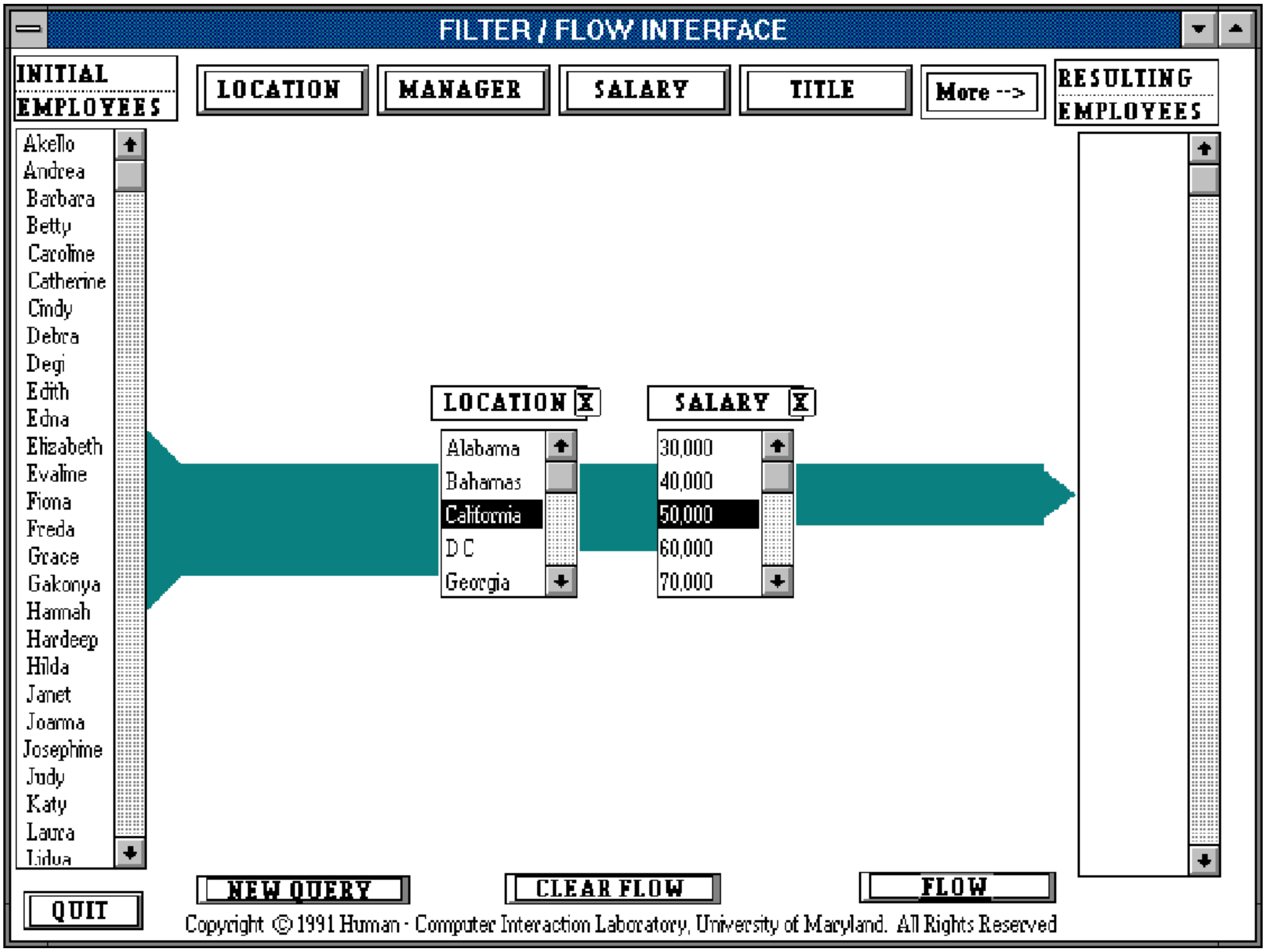


Figure 1: Intersection Operator

Find the employees who are located in California that earn exactly fifty thousand.

For example, in Figure 1, both of the attribute menus are on the same row but in different columns. Graphically, the attribute menus are filters that restrict the flow of water. As the stream of water passes through the first attribute it loses its volume; the width of the water flowing out of the filter is less than the width of the water flowing into the filter. This representation indicates that only the tuples that satisfy the specified condition were allowed to pass through. The tuples that result in the first condition are applied as input to the second condition. From those tuples, only the tuples that satisfy the second condition are of interest. The result consists of the tuples that have satisfied both the first condition and the second condition. This process can be extended to multiple conditions.

UNION OPERATOR (OR):

There are two ways of specifying a union operation. The first method is across attribute menus and can be used in all union specifications. The second method is within attribute menus and can only be used for two or more attribute values within the same attribute menu.

1) Across Attribute Menus

A union operation between attribute menus can be specified by placing the attribute menus on different rows and ensuring that there is a parallel flow to both conditions (Figure 2).

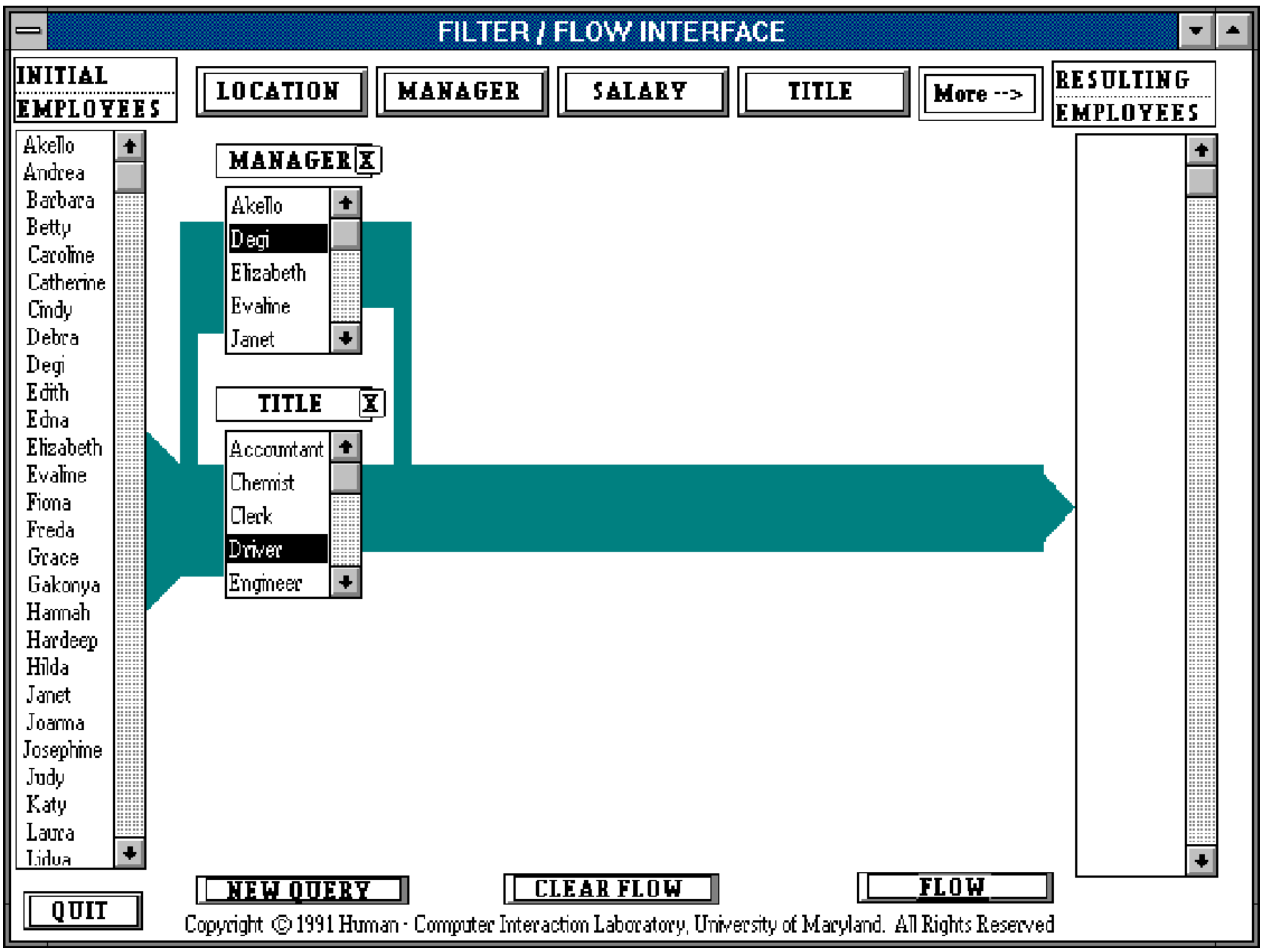


Figure 2: Across Attribute Menus - Union Operator

Find the employees who are either managed by Degi or are Drivers.

The flow from the initial database enters each attribute menu with the same width. This indicates that the same number of tuples enter the attributes. As the water flows out of each of the attribute menus, the volume of the water has decreased. Similar to the intersection operator, this process is used to indicate the relative number of tuples that have satisfied the condition. In this case there is a direct flow to the result list. Thus, each condition has been satisfied independently.

2) Within An Attribute Menu

A union can also be specified by two or more attribute values within the same attribute menu (Figure 3). This indicates that the filter allows two or more specifications to be satisfied, each independent of the other. A tuple can, therefore, satisfy any of the specified alternatives in the attribute menu.

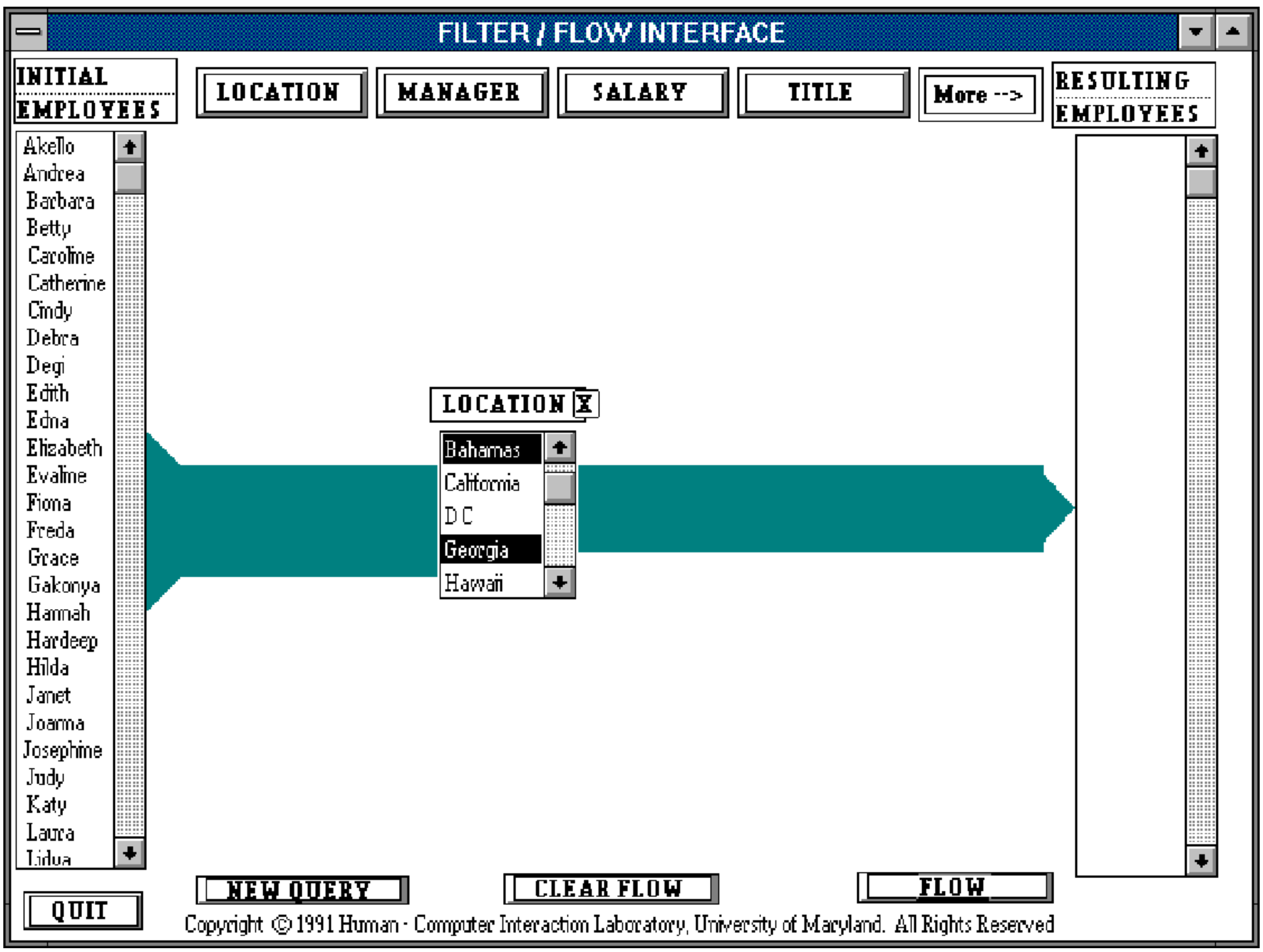


Figure 3: Within an Attribute Menu - Union Operator

Find the employees who are located in either the Bahamas or Georgia

NEGATION OPERATOR (NOT):

Users can specify negation by selecting an attribute menu and initially choosing the attribute value that they do not want to satisfy. The "X" located on the upper right hand side of the attribute menu deselects the selected attribute values and selects all values that were not selected; tuples that did not satisfy the original query will satisfy the negation query and vice versa (Figure 4).

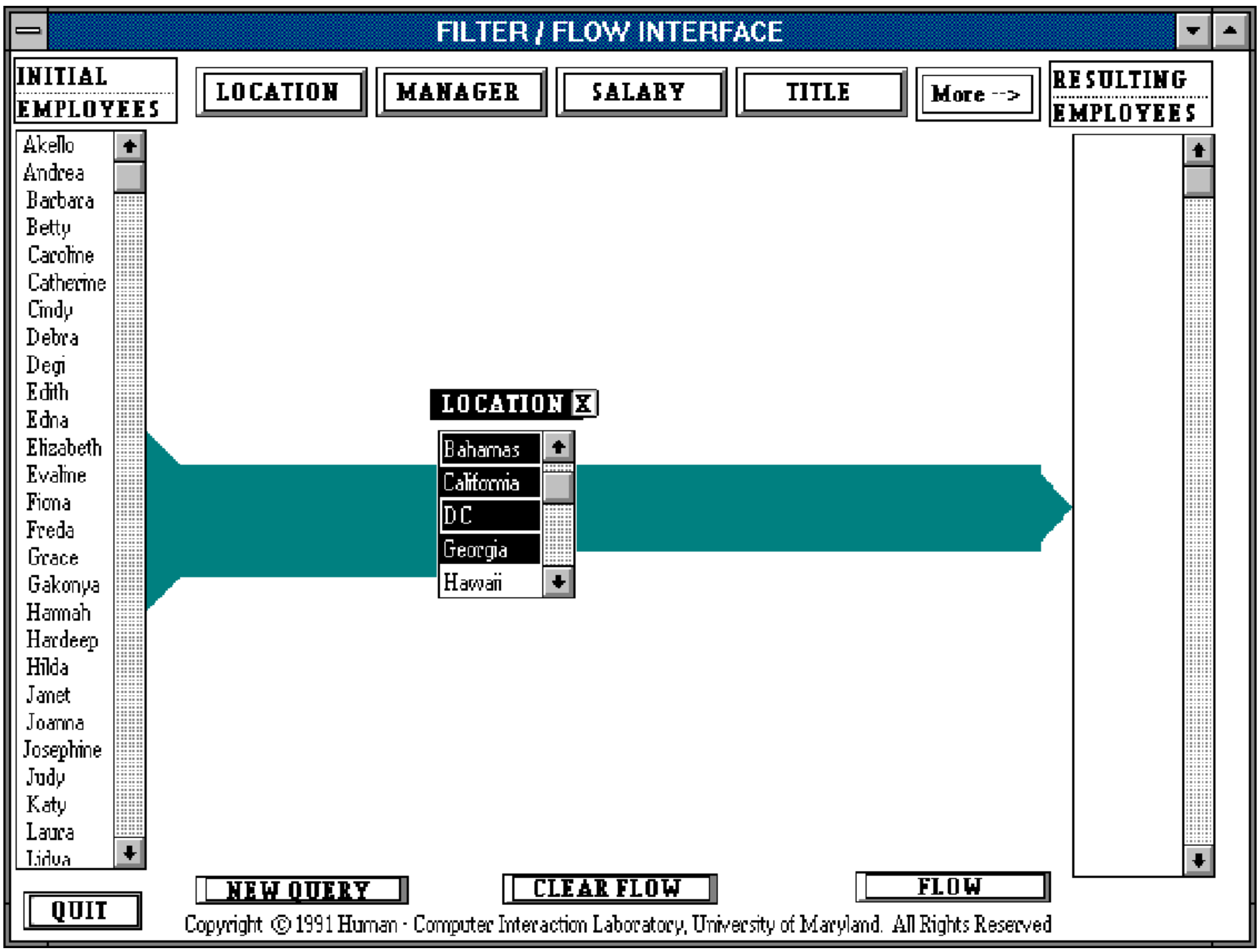


Figure 4: Negation Operator

Find the employees who are not located in Hawaii

COMBINATIONS:

A combination of the above operators may be used to form a complex query (Figure 5).

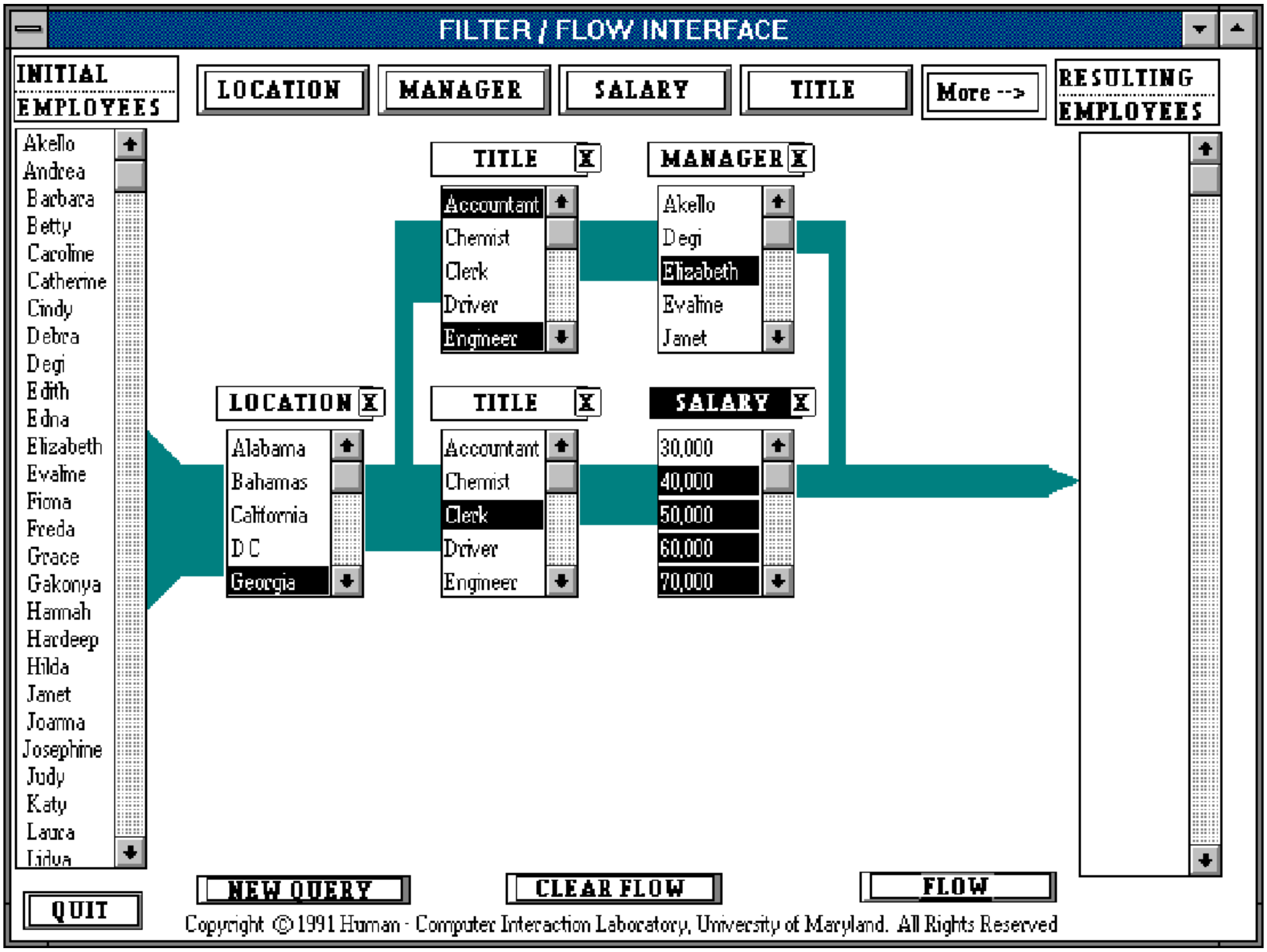


Figure 5: Example of a Complex Query

Find the accountants or engineers from Georgia who are managed by Elizabeth, or the clerks from Georgia who make more than thirty thousand.

CLUSTERS:

Users may form more complex queries by clustering attributes on a menu This results in an attribute menu that graphically displays a reduced view of the section that was selected. In so doing the relative position of each of the attribute menus in the cluster are maintained. Furthermore, attribute menus within the cluster that are negated are displayed in black. Users may create labels for the attribute menu representing the cluster and use it in the same manner as any other attribute menu. It is possible, for example, to negate the clustered attribute menu. The query represented in Figure 6 is a cluster representation of the query in Figure 5. The icons within the attribute menu represent the union between the accountants or engineers managed by Elizabeth and the clerks who earn more than 30,000.

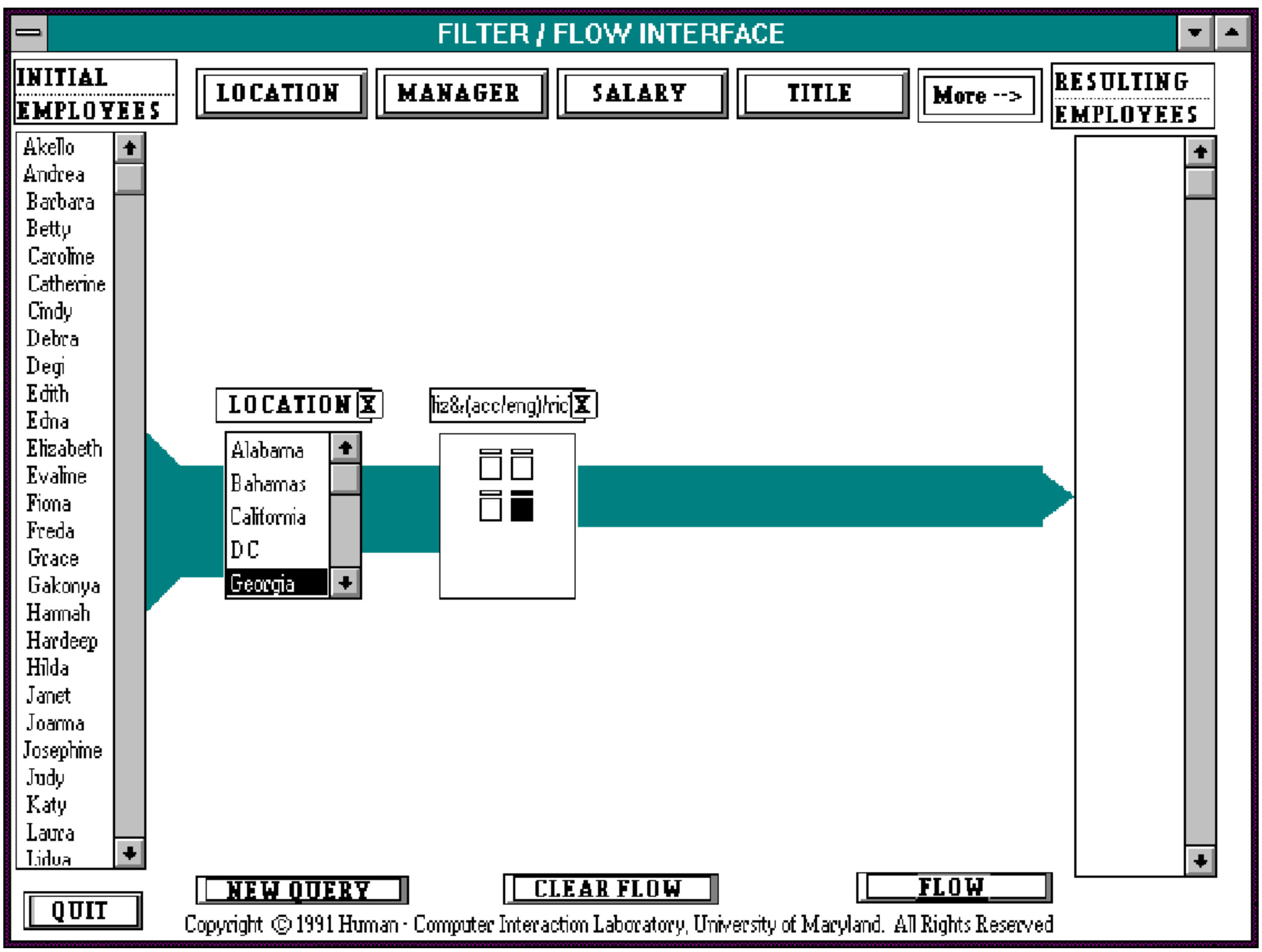


Figure 6: Clustered Attribute Menu

Find the accountants or engineers from Georgia who are managed by Elizabeth, or the clerks from Georgia who make more than thirty thousand.

2.3 Conventions of the Interface

This interface is a prototype; it is not based on a functioning relational database. However, we believe that an interface based on the Filter/Flow model that supports all the features of a relational database model can be implemented. In the design of the Filter/Flow prototype some restrictions were made on the database schema as well as on the visual representation. In a commercial implementation:

1) the Filter/Flow model would have to accommodate joins between multiple relations. Our mockup is based on a one relation database. Several approaches have been demonstrated to work effectively [IBM's DIS, 1990; Clark and Wu, 1992].

2) extended filters could apply graphical techniques to specification of attribute values (see Section 5.1.2 for suggestions). In our mockup, the retrieval of tuples are based on selections from scrolling menus.

3) operations such as transitive closures and aggregate functions should be addressed [Clark and Wu, 1992].

4) computation of values within an attribute menu or equality across menus should be addressed.

5) The representation of the width of water flowing through the filters should be dependent on the values from the actual database. In our mockup, we used a heuristic to determine the width.

Clearly, these features would lead to a more complete representation of database queries. We believe that these limitations did not interfere with our objectives which were to explore an alternative method of forming Boolean queries.

3 Experimental Evaluation

Previous researchers emphasized the goal of discovering what features facilitate the understanding of Boolean queries. This orientation leads to a focus on the type of errors made and their cause. In order to collect this evidence, an experiment was run to compare the Boolean operators used in SQL with the implemented version of Filter/Flow.

3.1 Hypothesis

The analogy present in the Filter/Flow interface is expected to enable users to predict the outcome of a query. This is due to the fact that they can relate their prior knowledge of the domain (water flowing through the filter(s)) to that of the target (operation of Boolean queries). It is conjectured that using a Filter/Flow interface would enable users to interpret a query correctly with higher frequency than they would if they were to read the same query in SQL.

SQL is a language equipped with definite syntax rules, such as in-order application of "AND" and "OR" and prefix order application of "NOT". It also requires parentheses to specify the order of operation. In addition there is often an inconsistency in the use of the natural language "and" and "or" and the logical AND and OR as discussed in the literature review. Filter/Flow is a language that replaces a linear structure with a two-dimensional visual image. There is no referral to natural language words and users do not use parentheses. Furthermore, the visual feedback should facilitate an accurate formation of the query. It is conjectured that users should be able to form more accurate queries using a Filter/Flow interface.

3.2 Subjects

Twenty subjects (ten male and ten female) responded to advertisements posted at the University of Maryland College Park campus. The requirements for the subjects were that they had some experience using a mouse and very little or no programming language experience. In addition to this, subjects who had used Boolean logic in any of their course work were excluded. The ages of the subjects ranged from eighteen to forty, the majority of the subjects being early twenties.

3.3 Method

The subjects used an AT&T computer with a 80386 processor running at 25 Megahertz with a 17 inch VGA monitor. Both the SQL and Filter/Flow interfaces were created using ToolBook on the same computer, thus ensuring the same rate of response.

The independent variable in this experiment is the type of interface (Filter/Flow or SQL). Subjects performed two types of tasks (comprehension and composition). The dependent variables are the number of correct queries and the number of errors made in each task. Subjects were given two similar query sets for each interface. The order in which the subjects viewed the two query sets varied. The order in which the subjects viewed the two interfaces was also alternated.

3.4 Training

A fifteen to twenty minute training period was given for each interface. In the first five minutes the subjects were given specific tasks to perform on the computer, thus enabling the subjects to become familiar with the medium that they were about to use. For example, in SQL subjects were asked to type in statements, and in Filter/Flow, subjects were asked to use the mouse to select and drag boxes on the screen. The remaining ten to fifteen minutes of the training period were spent going over the operations of Boolean logic. The subjects were given two sheets of paper: a question sheet consisting of five queries and an employee data sheet. The first query consisted of a simple extraction, the second query introduced the concept of the intersection operator, the third query introduced the union operator, the fourth query introduced the negation operator coupled with the intersection operator (in the case of SQL, parentheses were introduced in this question) and finally the fifth query consisted of all the above operators.

The experimenter formed and interactively interpreted the first four questions and asked the subjects to form and interpret the final question. Since the final question contained all the Boolean operators, it served as an indication as to whether or not the subjects had a basic understanding of the material presented. Subjects were free to ask any questions throughout the training period.

3.5 Tasks

The first task was Query Comprehension. In this task subjects were given an employee database. The employee database indicates the location, salary, title and manager of thirty employees. The experimenter either typed in or constructed a query (depending on the interface used) and the subjects were required to indicate on a sheet of paper the results of the query. In order to do this task, subjects must be familiar with the connotation of the operators used in the query.

The second task was Query Composition. In this task, the subjects were given an English sentence and they had to form a query using either the SQL or Filter/Flow interface. The query was written on a piece of paper allowing the subjects to refer to the query at any time. When using SQL, subjects were provided with the stem of the query and were asked to type in the "WHERE" condition. This task is usually difficult, because it entails creating an expression that represents the query. Two examples of the composition queries that were asked are given below:

> 1) I'd like to find the employees who are managed by Nancy. Please ensure that these employees don't earn exactly 50,000 but they are lawyers.
>
> 2) Nancy, with Degi's help, wrote a paper about Alabama women in history. Degi talks to all her employees about this paper. Nancy gave a seminar about this paper to her employees who are from Alabama. Nancy felt that this paper would be of interest to the historians she employs so she told them, too. Could you list all the people who have heard about this paper.?

The subjects always performed the comprehension task (consisting of five queries) followed by the composition task (consisting of five queries) using one of the designated interfaces and one of the designated query sets. After completion of the last composition query, subjects were asked whether they wanted a break or not. If so, a five minute break was taken. After that the subjects repeated the two tasks using the alternate interface and the alternate query set.

After completion of all the tasks, subjects were asked for their opinion of the two interfaces and the reasons for their preferences.

3.6 Experimental Results

3.6.1 Scoring

The total number of correct queries made by each subject was collected and errors were noted. A 2x2 ANOVA was done to compare the scores obtained in the Filter/Flow with that obtained in SQL and study the order effect. The null hypothesis was that there is no difference between the two interfaces.

Comprehension task:

If the subject did not write down the correct list of tuples then the answer was considered to be wrong. The number of correct answers for each query with each interface was noted (Table 2). The types of errors made were of interest; thus the incorrect result lists were examined. A break down of the types of errors was noted for each question (Table 4).

The Boolean errors were of special interest, so the number of Boolean errors was noted (Table 3). The Boolean errors included:

1) Interpreting the logical operators incorrectly: For example, interpreting the union for an intersection or the intersection for the union operation

2) Ignoring the order of precedence.

Composition task:

In general syntactical errors were not counted as wrong answers. For example, a missing quote or a misspelled attribute name or value were considered accepted when the subject used SQL. There was an error check procedure in Filter/Flow that would indicate that the flow was incorrectly specified, so the subjects had a chance to change this when they were forming their query. Since this was not the case in SQL, subjects were not penalized for those types of errors. It is, of course, conceivable that a program could be implemented to correct syntactical errors in SQL [Welty, 1985].

Every error made was counted. Thus, if a subject repeated a mistake, that error was counted the number of times it was repeated. For example in SQL, if the query called for an “A OR B OR C” structure and the subject wrote “A AND B AND C”, then two errors were made and the error type was AND instead of OR (Table 5).

In noting the Boolean errors (Table 7), the definition of the Boolean errors had to be redefined. In this case the Boolean errors consisted of:

1) Typing “AND” instead of “OR” or typing “OR” instead of “AND” in SQL. Placing the attribute menus on the same row instead of different rows or placing the attribute menus on different rows instead of the same row in Filter/Flow.

2) Placing the parentheses in the wrong place or omitting the parentheses in SQL. Placing an attribute menu in the same column instead of the preceding column in Filter/Flow.

3) Forgetting to apply the negation as a pre-order specification in SQL, or forgetting to specify the negation filter.

3.7.2 Statistical Results

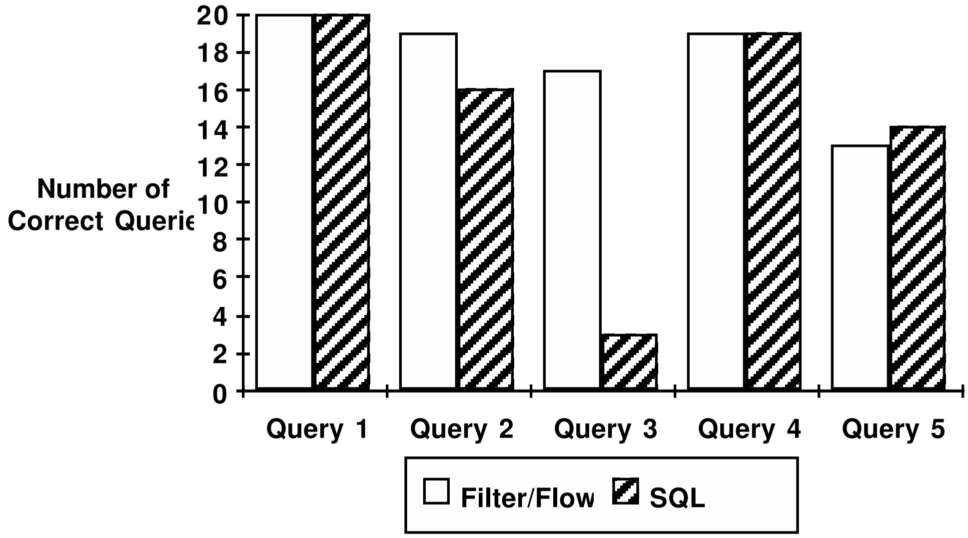
Comprehension task:

There was a significant difference in the number of correct answers for this task favoring Filter/Flow($F(1,18) = 12.9$, $p < 0.05$) (Table 1).

Filter/Flow	SQL	F	P
4.3 (0.87)	**3.5 (0.95)**	**12.9**	**0.02**

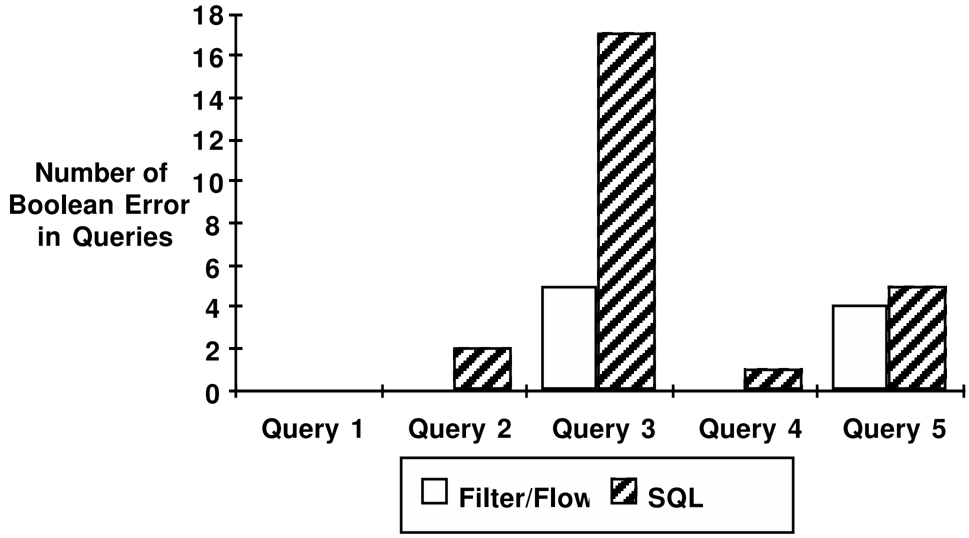
Table 1: Means and Standard Deviations of Number of Correct Queries for Comprehension Task

Fewer Boolean errors were made in Filter/Flow than were made in SQL. The total number of errors made in Filter/Flow was 12, of which 9 were Boolean errors. In SQL, 28 errors were made, of which 25 were Boolean errors (Table 2 and Table 3).



Query	Correct Queries		Query Expression
	Filter/Flow	SQL	
1	20	20	$A \cap B \cap C$
2	19	16	$(A \cup B) \cap C$
3	17	3	$A \cup B \cap C$
4	19	19	$A \cup \neg(B \cup C)$
5	13	14	$((A \cap B \cap \neg(C \cup D)) \cup (E \cap \neg F)) \cap G$

Table 2: Number of Correct Queries in Comprehension Tasks



Query	Number of Boolean Errors		Query Expression
	Filter/Flow	SQL	
1	0	0	$A \cap B \cap C$
2	0	2	$(A \cup B) \cap C$
3	5	17	$A \cup B \cap C$
4	0	1	$A \cup \neg(B \cup C)$
5	4	5	$((A \cap B \cap \neg(C \cup D)) \cup (E \cap \neg F)) \cap G$

Table 3: Boolean Errors in Comprehension Task

The most common error occurred in the third question, where five subjects wrote the wrong answer in Filter/Flow and seventeen subjects wrote the wrong answer in SQL (Table 3). The main type of error for this question was that subjects did not use the rules of precedence to evaluate the results. Seventeen subjects did not use the rules of precedence in SQL while only four subjects did not use the rules of precedence in Filter/Flow (Table 4).

Type of Error	Interface	Query 1	Query 2	Query 3	Query 4	Query 5	Total
Did not use rules of precedence	Filter/Flow	0	0	4	0	0	4
Did not use rules of precedence	SQL	0	0	17	0	0	17
Attribute menu on same row instead of on a different row	Filter/Flow	0	0	1	0	0	1
Used logical "AND" instead of logical "OR"	SQL	0	2	0	0	0	2
* Not applicable	Filter/Flow	0	0	0	0	0	0
Order of parenthesis	SQL	0	0	0	0	4	4
Inverted meaning of Negation operation	Filter/Flow	0	0	0	1	3	4
Applied Negation to one Condition instead of Two Conditions	SQL	0	0	0	1	1	2
Misread Sheet	Filter/Flow	0	1	0	1	1	3
Misread Sheet	SQL	0	0	0	0	3	3
Other errors	Filter/Flow	0	0	0	0	3	3
Other errors	SQL	0	0	0	1	1	2

Table 4: Comprehension Error Types

The most incorrect answers per query in Filter/Flow were in the fifth query (7 incorrect answers), where three subjects inverted the negation operation (i.e. allowing tuples to go through rather than restricting them) (Table 2 and Table 4). Six incorrect answers were found in the fifth query in SQL. Nine errors were made in the fifth query in SQL and four of those nine errors were made when subjects misinterpreted the order of the parentheses (Table 2 and Table 4). Four Boolean errors were found in the fifth query in Filter/Flow and five Boolean errors were found in the fifth query in SQL (Table 3).

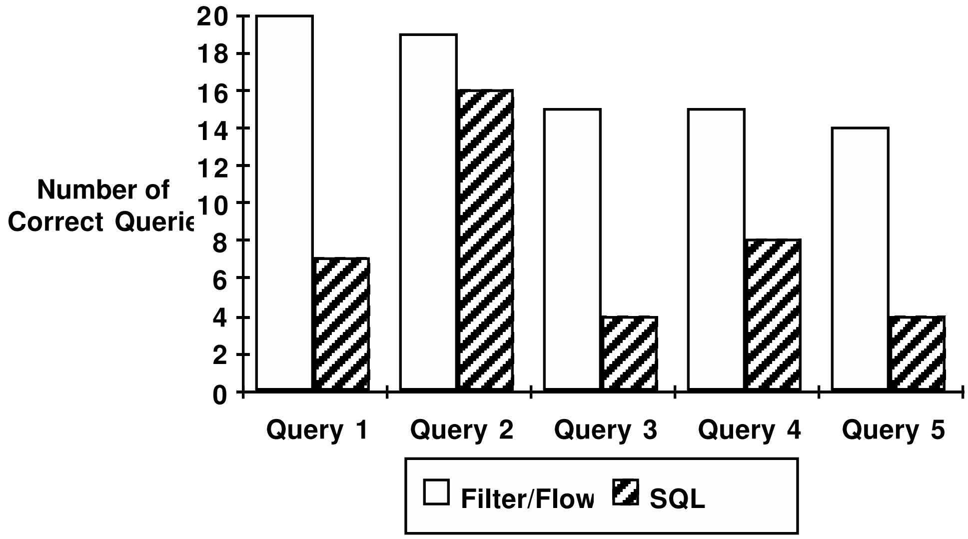
Composition task:

There was a significant difference between the number of correct queries in SQL and Filter/Flow, favoring Filter/Flow ($F(1,18) = 65.8$, $p < 0.05$) (Table 5).

Filter/Flow	SQL	F	P
4.2 (0.98)	**1.9 (1.37)**	**65.8**	**0.00**

Table 5: Means and Standard Deviations of Number of Correct Queries in Composition Task

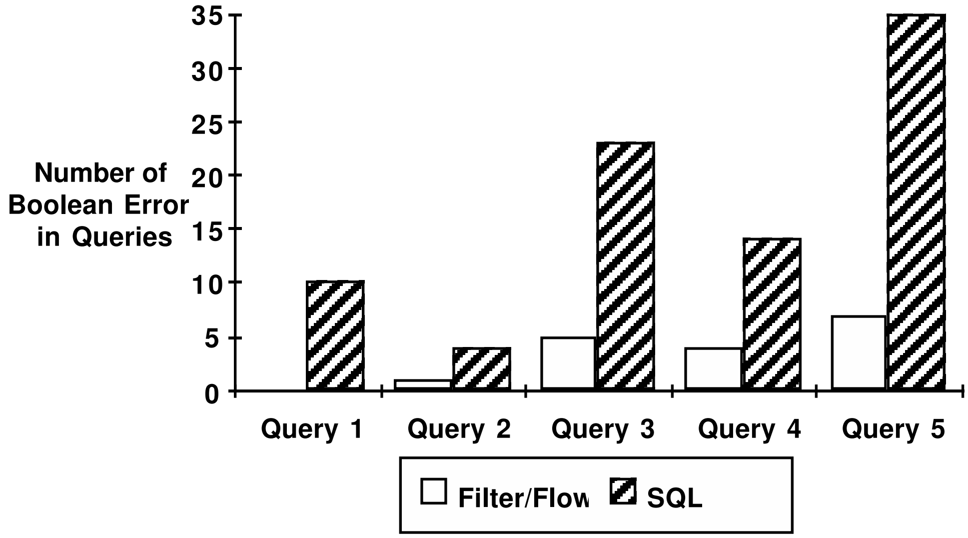
The number of correct queries was greater for every query in Filter/Flow than it was in SQL (Table 6).



Query	Correct Queries		Query Expression
	Filter/Flow	**SQL**	
1	20	7	$A \cap (B1 \cup B2 \cup B3 \cup B4)$
2	19	16	$A \cap B \cap \neg C$
3	15	4	$(A \cup B) \cap \neg C$
4	15	8	$(A \cap (B \cup C)) \cup D$
5	14	4	$A \cap (B \cap (C \cup D) \cup (E \cap F))$

Table 6: Correct Queries in Composition Task

Subjects made the most Boolean errors in the fifth query. Thirty-five errors were made in SQL and only seven errors were made in Filter/Flow. Subjects made the least number of Boolean errors in the first query in Filter/Flow and the least number of Boolean errors in the second query in SQL (Table 7).



Query	Number of Boolean Errors		Query Expression
	Filter/Flow	SQL	
1	0	10	$A \cap (B1 \cup B2 \cup B3 \cup B4)$
2	1	4	$A \cap B \cap \neg C$
3	5	23	$(A \cup B) \cap \neg C$
4	4	14	$(A \cap (B \cup C)) \cup D$
5	7	35	$A \cap (B \cap (C \cup D) \cup (E \cap F))$

Table 7: Boolean Errors In Composition Queries

The most frequent error type was the use of parentheses. A total of fifty parentheses errors were made in SQL, while only two errors were made in specifying the order of the attribute menus in Filter/Flow (Table 8). The most frequent error made in Filter/Flow was the placement of attributes on the same row as opposed to different rows; this error was made nine times. However, this was less than the number of times the comparable error was made in SQL. In this case, the AND Boolean operator was used instead of the OR operator; this error occurred twenty-two times (Table 8). Another rather frequent error type that occurred in SQL was that subjects did not specify the attribute name. This occurred twenty-four times. One subject did not negate an attribute menu in Filter/Flow, while four subjects used the negation as a binary operator (applying to two operands) rather than a unary operator (applying to one operand) in SQL. There were nine unidentified errors in SQL, these errors were unique to the individual subject. For example, some subjects introduced terms that are not used in the SQL. One subject tried to use the within attribute menu specification incorrectly when specifying a union (Table 8).

Type of Error	Interface	Query 1	Query 2	Query 3	Query 4	Query 5	Total
Precedence of Filter/Flow	Filter/Flow	0	0	2	0	0	2
Missing parentheses or parentheses in the wrong place	SQL	6	3	12	7	22	50
Attribute menu on same row instead of on a different row	Filter/Flow	0	0	2	2	5	9
Used logical "AND" instead of logical "OR"	SQL	3	0	7	7	9	26
Attribute menu on different rows instead of the same row	Filter/Flow	0	1	1	1	1	4
Used logical "OR" instead of logical "AND"	SQL	1	0	2	1	1	5
Did not negate attribute menus	Filter/Flow	0	0	1	0	0	1
Did not use negation as a unary operator	SQL	0	1	0	0	3	4
Missing attribute menu	Filter/Flow	0	0	1	1	2	4
Didn't specify attribute label	SQL	9	0	2	4	9	24
Used Within specification of the union operator instead of between	Filter/Flow	0	0	0	0	1	1
Other errors	SQL	1	2	3	1	2	9

Table 8: Composition Error Types

Subjective Satisfaction

At the end of the experiment, one subject told his friend: “It’s fun man, go for it.” That, coupled with the fact that only two subjects wanted to take a break after the first interface, showed some indication that the subjects enjoyed the experiment in general. When asked to compare the two interfaces, all twenty subjects claimed that Filter/Flow was easier to use than SQL. Indeed, when subjects attempted the training task in Filter/Flow, comments included “this is fun” and "this is groovy". In SQL, when some of the subjects were told that they would have to type, they made apologetic claims about not being great typists.

When asked which interface they preferred, a few of the subjects looked as if this was a rather stupid question and stated that it was Filter/Flow, of course. In fact some subjects did not need any prompting. After completing the Filter/Flow interface and in the middle of the SQL task, one subject stated that she could see why SQL was left until the end, it was much harder. Another subject said that using SQL reminded him of using the IBM where everything has to be typed in but when he used Filter/Flow, it was like using the Mac where you can use the mouse to get a result. When forming a complex query in SQL, one of the pilot subjects stated: "Gosh, I'm lost in bracket city." Overall this subject felt more confident when she used Filter/Flow because she had visual feedback when she used SQL she was not sure that what she had done was correct.

A few subjects felt that SQL challenged them and was therefore fun to use. In comparison to SQL they expressed the opinion that Filter/Flow was definitely easier to use and recommended it as a training tool. One subject stated that he preferred the visual feedback and it was "neat" to watch the water flow. In one extreme case, a subject started with Filter/Flow and as he was doing the tasks, he seemed very enthusiastic about the tasks, with an occasional smile as soon as he completed a query. He then switched to SQL and half way through the task, he asked what the operation of the "OR" was. He was then told that no further explanations could be given because he had finished the trial task and had started the experimental task. As he continued the tasks he began to show his displeasure by swearing. Gone was the occasional smile and in its place was an occasional shrug of acceptance. It was almost as if he was saying, "I don't know if this is correct or not but this task has to be performed so I'm just going to do it". As he was reaching the end of the SQL task, it dawned on him that if he used the concept of Filter/Flow, he might be able to form better queries. So he started drawing the structural flow on a piece of paper and turned around and said "I'm using the Flow thing to figure this thing out." He decided to use the knowledge he had attained from Filter/Flow to structure his answers in SQL.

4 Discussion of the Results

4.1 Comprehension Results

There were more correct answers in the comprehension task (160 out of 200) compared with the composition task (122 out of 200). When comprehending relatively simple queries such as the first query, which consisted of three conjunctions, subjects had little difficulty in either of the two interfaces. Furthermore, subjects were able to solve queries whose structure was similar to queries that they were shown in the training session. However, when subjects were given a unique query (i.e., a query whose form they had not seen before) they were able to solve the query more accurately in Filter/Flow than in SQL. When using SQL, subjects often failed to apply the rules of precedence; they would evaluate the queries on a left to right basis. Filter/Flow encourages this process; the flow is from left to right and the subjects evaluate the information from left to right and this representation results in the correct answer.

When interpreting complex queries, the major error made by SQL users was the incorrect interpretation of the order of parentheses. Some subjects did not know which of the parenthesized expressions to solve first, especially when the parentheses were nested.

4.2 Composition Results

Our findings suggest that one of the simpler structures to form in both interfaces was the combination of three intersection operators and a negation of one of the terms. This coincides with Michard's finding that the AND operator is the easiest operator to use [Michard, 1982]. In some cases when using SQL, subjects would place the two conditions next to each other, assuming that if they were together it meant that these conditions were automatically intersected.

Subjects had little difficulty using the "within an" attribute representation of the union operator in Filter/Flow. However, they did have difficulty in forming a union between several attribute values within an attribute in SQL. They made several attempts to shorten the specification of the query, including: not specifying the attribute names (or in one case, using a different attribute name); not using the OR operator between the values and using commas to list all the values.

The correct use of parentheses is essential if users want to change the rules of precedence that apply to the Boolean operators. Users must be able to use parentheses effectively if they want to express complex queries. The major error made with the SQL interface was the improper use of parentheses. In some cases subjects would not use parentheses; logical operators with a higher order precedence were thus bound together when the subjects expected the logical operators of lower order precedence to be bound together. In the Filter/Flow experiment, subjects were able to form complex queries easily by placing attribute menus in the preceding column and indicating the flow appropriately. Using Filter/Flow, the majority of the subjects were able to form queries that were two parentheses levels deep but were unable to do so in SQL.

Errors made in SQL seem to indicate that if the subjects are not sure how to express themselves, they will try to relate SQL to an English construct. This was evident by the Boolean operators and by the syntax that they used. For example, in the first composition query the subjects used a comma to indicate that they wanted to take the union of the attribute values. Another common error was the use of "AND" instead of "OR" which coincides with previous studies [Boyle et al., 1983; Reisner, 1988].

4.3 Subjective Satisfaction

There is good evidence that the subjects preferred using Filter/Flow to SQL. They may have felt more confident with their answers, because they had a more effective visual feedback to indicate when they were on the right track. The Filter/Flow representation appears to have helped these novices by providing a means to explore.

4.4 Experimental Limitations

The Filter/Flow interface is an implemented prototype that was used to explore some of the issues that affect the representation of Boolean queries. Thus, in the evaluation of the Filter/Flow model some limitations and conventions were made. For example, the experiment conducted was based on the comparison of specific Boolean queries in the Filter/Flow model with the SQL model. This experiment concentrated on novice users performing two specific task types (comprehension and composition) on a particular single relation database (i.e., the employee database based on the attributes: title, manager, salary and location). Further studies are warranted with alternative tasks, more experienced users, and different databases.

5 Future Research Directions

5.1 Interface Implementation

Our prototype for a Filter/Flow interface was static, that is, the width of the flow was reduced by a fixed amount at each attribute menu. In other words, to indicate intersection the width was reduced a certain amount while to indicate a convergence between two resulting operations, the width of the largest operation was used as the resulting width. A design that is consistent with the data evaluated would vary the width of the flow dependent on the number of tuples that satisfy the query.

The graphical representation of the union between attribute menus was an inverted “L” structure. A better representation would have been a diagonal block or rounded corners going from the initial attribute menu to the resulting attribute menus.

We conjecture that if the inversion of the values selected within the attribute menu were blue (the same color as the water flow) instead of black, fewer subjects would have been confused about whether or not tuples were allowed to pass through. Furthermore, in this prototype the resulting list of tuples is not shown.

5.1.2 Graphical Filters to Extend Attribute Menus

The representation of the attribute menus in this interface was based on a scrolling list of menu items. More effective forms of representation for different values exist [Weiland and Shneiderman, 1991].

<u>Specification of numerical values</u> When specifying numerical values, users may want to specify a

range of values rather than an exact value; using sliders as a representation of a filter may be more suitable than a scrolling list. Users may specify a lower bound by placing the right arrow marker on the designated value. Similarly, a left arrow marker may be used to specify a higher bound (Figure 7). Alternatively, users may want to specify computations of values within an filter.

Specification of non-numeric values Check boxes or radio buttons may be used to specify a limited amount of fixed values. For example, there are only a limited number of options in the Scholarship attribute (Figure 7), and these are easily displayed as radio buttons. An alternative method of specification may allow users to type in the appropriate value after receiving a prompt. For example, in the Interests menus, users can type in their interests (Figure 7).

Specification of values within a geographical region Users may select a specific map region with a mouse or touch screen. Alternatively, multiple regions may be chosen by specifying the appropriate areas.

Display of results on a geographic region Figure 7 shows how it might be applied to help students choosing colleges. Users can select from the set of attributes and get an appropriate filter widget (type-in for interest areas, sliders for cost, and buttons for scholarships) which is placed on the screen with flow lines showing ANDs (sequential flow) and ORs (parallel flows). The X in each filter widget could be selected to negate the filter values.

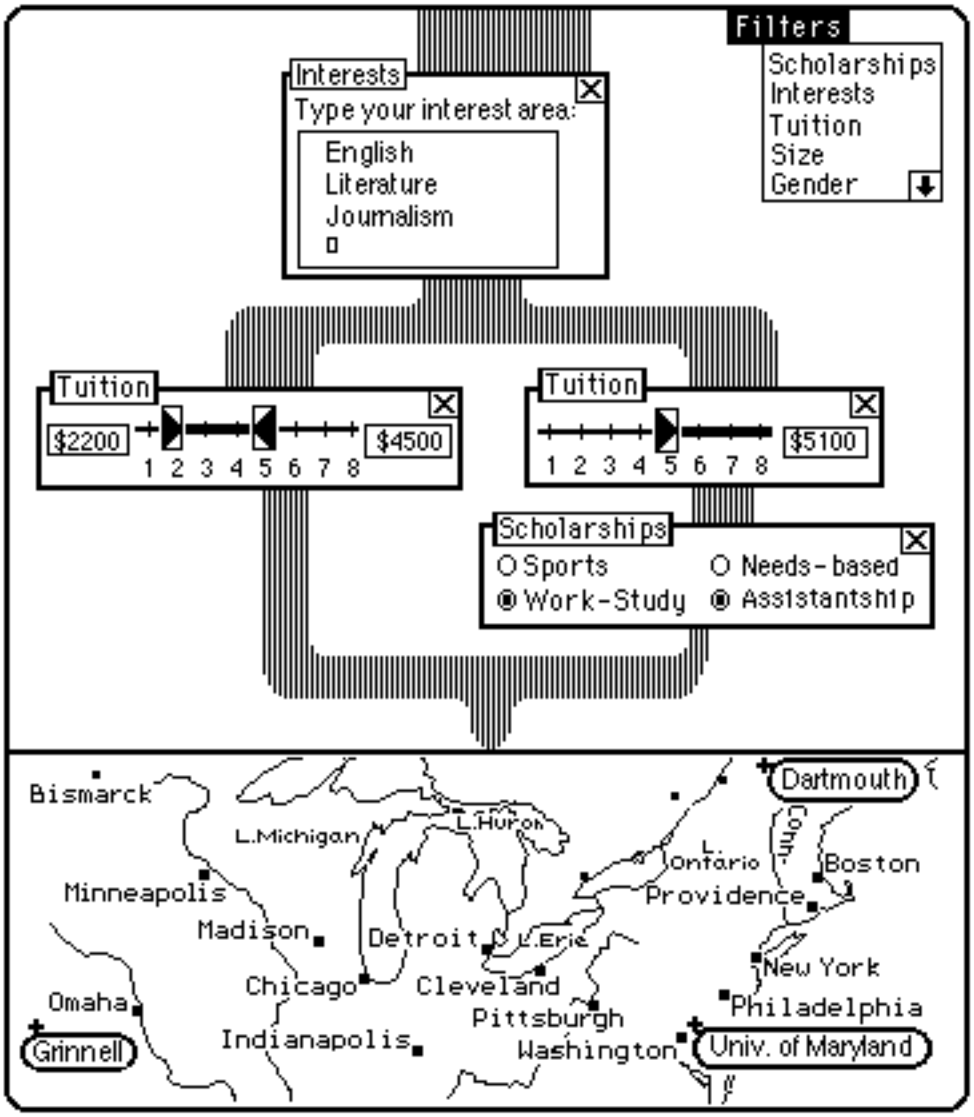


Figure 7: Mockup of a Filter/Flow Boolean Query

Find ((Interests = English or Literature or Journalism) AND ((Tuition greater than or equal to $2200 or less than or equal to $4500) OR ((Tuition greater than or equal to $5100) AND (Scholarships are available by Works-Study or Assistantship))))
combined with map output to show the result (Dartmouth, Grinnell, and the Univ. of Maryland).

5.2 Experimental Design

It would have been interesting to spend additional time on the more complex queries. Unfortunately, in order to get the subjects to the level in which they could form complex queries, they had to be guided through the simpler queries first. A more thorough study of more complex queries may result in showing just how much the subjects understood the Filter/Flow interface.

The complexity of queries did not reach a level where subjects were forced to use the implemented clustering. It might be interesting to see whether the use of clustering leads to better organization in the formation of queries with more experienced users.

Acknowledgments: We appreciate the financial support of General Electric Information Services and express our thanks to our proof readers: Dave Turo, Catherine Plaisant, Andy Sears, Dave Carr, Scott Gilkeson, Richard Chimera, Richard Potter, Jerry Floyd and Kurt Nemes.

REFERENCES

Anick, P. G., Brennan, J. D., Flynn, R. A., Hanssen, D. R., Alvey, B., Robbins, J. (1990). A direct manipulation interface for Boolean information retrieval via natural language query. *Proceedings of the ACM SIGIR Conference 1990, 135-150.*

Boyle, J., Bury, K. and Evey, R. (1983). Two studies evaluating learning and use of QBE and SQL. *Proceedings of the Human Factors Society 27th Annual Meeting,* 663-667. Santa Monica, CA: Human Factors Society.

Bury, K., and Boyle, J. (1982). An on-line experimental comparison of two simulated record selection languages. *Proceedings of the Human Factors Society 26th Annual Meeting,* 74-78. Santa Monica CA: Human Factors Society.

Clark, G. J., and Wu, C. T., (1992). DFQL: Dataflow Query Language for Relational Databases, Naval Postgraduate School, Department of Computer Science, Monterey, CA.

Greene, S., Devlin, S., Cannata, P. and Gomez L. (1990). No IFs, ANDs, or ORs: A study of database querying. *International Journal Man-Machine Studies*, **32** , 303 - 326.

Greenblatt, D., and Waxman, J. (1978). A study of three database query languages. In B. Shneiderman, (Ed.) *Databases: Improving Usability and Responsiveness,* New York: Academic Press.

Hildreth, C. (1988). Intelligent Interfaces and Retrieval Methods For Subject Search in Bibliographic Retrieval systems. *Research, Education, Analysis & Design,* Springfield, IL, 48.

IBM Data Interpretation System (DIS) User's Guide, IBM White Plains, NY, 1990.

Jarke, M., and Vassiliou, Y. (1986). A framework for choosing a database query language. *Computing Surveys*, **17**, 313-340.

Lundh, J. and Rosengren, P. (1990). Hybris - An ER-based graphical query tool with an integrated data dictionary. SISU, Swedish Institute for Systems Development, Kista, Sweden.

Michard, A. (1982). A new database query language for non-professional users: Design principles and ergonomic evaluation. *Behavioral and Information Technology,* **13**, 279-288.

Reisner, P., Boyce, R. and Chamberlin, D. (1975). Human factors evaluation of two database query languages - Square and Sequel. *National Computer Conference*, Anaheim, CA. AFIPS, 447-452.

Reisner, P. (1988). Query Languages. In Helander, M., (Ed.) *Handbook of Human-Computer Interaction.* Elsevier Science Publishers B.V. (North - Holland).

Shneiderman, B. (1991). Visual user interfaces for information exploration, *Proceedings of the 54th Annual Meeting of the American Society for Information Science*, **28,** 379-384. Learned Information Inc., Medford, NJ.

Thomas J. and Gould J. (1975). A psychological study of query by example. *National Computer Conference*, Anaheim, CA. AFIPS,**44**, 439-445.

Weiland, W. and Shneiderman, B. (1991). *A Graphical Query Interface Based on Aggregation/Generalization Hierarchies*, Department of Computer Science Technical Report CS-TR-2702, University of Maryland, College Park, MD.