A Teaching System for SQL

R. Kearns
reba@cs.su.oz.au

S. Shead
stephen.shead@reuters.com

A. Fekete
fekete@cs.su.oz.au

Basser Dept. of Computer Science
University of Sydney

March 7, 1997

Abstract

Skill in writing SQL queries is a fundamental outcome expected by industry from any university course in Database Systems. Students often have difficulty with fundamental SQL concepts such as multi-table joins, aggregation and grouping. This paper describes teaching software, which supports students learning these concepts. The system, called esql, is similar to a normal query interface, except that the response to a SELECT statement is not merely to show the result, but instead to show a sequence of images, giving a step-by-step account of how the query result is determined.

1 Introduction

A course on Database Systems is a common part of the later years in a university Computer Science curriculum. Such a course can be taught from many different viewpoints, such as information systems (emphasis on data modelling), software systems (emphasis on performance analysis and tuning), or programming languages (emphasis on first-order logic and optimisation). No matter what approach is followed, students need to develop a facility in expressing queries in SQL. Employers will certainly expect any university graduate who claims knowledge of DBMS, to be able to understand and create queries in SQL.

There are several aspects of SQL that cause difficulties for students. Of course by this stage of their education, students can easily assimilate any syntax rules; however, years of exams give evidence that many students do not understand the fundamental concepts that drive the language. Common student errors include missing the linkage clause from a multi-table query. For example many will express "find all books written by an Australian" by

```
SELECT book.title
FROM book, author
WHERE author.nationality = 'Australian'
```

Similarly it is common for them to use non-aggregated, non-grouped attribute in a grouped query. For example

```
SELECT book.title
FROM book
GROUP BY book.authorname
```

A major thrust in recent innovative teaching is the use of visual presentation. For example, several systems allow the teacher to animate the execution of an algorithm, so students can develop better mental models of what is happening. [2, 4, 7, 9, 10, 11]. This paper reports on a system which aims to enhance students understanding of the semantics of SQL, by showing them a step-by-step display of how the result of any query can be calculated. This is much more informative than merely displaying the *result* of the query, as in commercial SQL systems or other teaching systems [3]. The third author has often taught the third year course on Database Systems at Sydney University; it is usually taken by about 150 students. The course covers a range of topics including SQL queries, file structures and query processing, database design and elements of normalisation theory. An Oracle system is available for practical work. The first two authors were Honours students in the Department, and (over two years), they designed and built a teaching system which displays the conceptual stages in evaluation of a query [6, 12]. This was used (on a voluntary basis) by students in the undergraduate course, and also in a Professional Development external course.

To the user, esql resembles a conventional query interface. Students can call a WYSYWIG editor, and then submit the query to the system, where it is evaluated against the current instance of the database. For UPDATE, INSERT etc, the current instance is modified; for CREATE TABLE etc, the schema is modified. The innovation in esql is that SELECT queries can run in "Help mode" where a sequence of displays is produced, showing conceptual steps in evaluation of the query. The esql system is built with a simple data store, parser etc, all coded in C, and a GUI coded in Tcl. As the system is intended for teaching rather than production use, no attention has been paid to sophisticated file structures, buffering, or transaction management. Instead, our focus is entirely on the assistance that esql can provide to the novice learning SQL.

In Sect. 2, we outline the structure of the system. Sect. 3 presents the detailed design of the step-by-step display for query explanation. In Sect. 4, we discuss the algorithm used to keep the size of the display manageable for multi-table joins. Sect. 5 explains how aggregate and grouping queries are treated. Finally, Sect. 6 gives our conclusions and future plans.

2 Overview of the System

To create esql, a complete database management system has been implemented, handling files, database schemas and the execution of SQL commands. The application is designed and controlled at the highest level in Tcl, with the graphical elements constructed using Tk widgets to create a Motif-like GUI and most of the underlying database parsing and execution code written in C.

The main window in esql has, in addition to menus, an area displaying the full path names of the current schema and file of SQL commands, a text-editing area where commands may be typed interactively, and a message area for the system to display messages relating to the status of execution of various operations. The menus have items allowing the user to — perform operations on database schemas such as loading, creating, viewing and deleting; load, save, delete and run files of SQL commands, as well as run commands typed into the editing area; and, most importantly, turn on and off help mode execution of queries (explained in Section 3.1).

When designing the GUI (and the system as a whole), we had to keep in mind the main purpose of the project, which was to develop the best possible environment for students to learn how SQL works. For example, it has been assumed that the system will only ever be used as a teaching system, not for practical real-world databases, so that the tables will never exceed thirty or forty rows. This means that many issues relating to speed and efficiency, which are vital in most database systems, are not nearly so important in esql. The top priority in all design decisions was to tailor the system to student needs.

The display of the schema shows the schema name at the top, and below this shows the definitions of the tables in the schema. Each table definition is enclosed in a box, and shows the table's name, then two columns containing the names and types of the columns in the table. The name of the schema and names of the tables are also highlighted. This provides a clear way of depicting schemas which is easy for beginners to understand.

The information required for the display of a table is the table's name, the names of the columns, and the rows themselves. It was essential that the design for displaying tables should be good. The result of a SELECT statement would need to display a similar table, and the handling of the SELECT statement forms the heart of the educational facility. The design for table display therefore needed to be flexible, so that it would also be able to be used in viewing query results. In addition to the elements detailed above, the query result also shows which query was executed. This is so that if several results are on display, it is easy to remember the queries which produced them.

3 Educational Query Execution

If an SQL statement is parsed without error, the parser passes it to the execution engine. If an error occurs during parsing or execution, an error message is generated, including the line number of the statement being executed and a description of the error to be displayed in the message area of the main window.

Esql currently parses and executes common forms of the following SQL statements: CREATE, DROP, ALTER, DELETE, INSERT, and SELECT. All of these except SELECT simply attempt to make the requested change to the database and return a message to indicate success or failure.

The result of a SELECT statement is displayed graphically, using a similar mechanism to the one used for displaying tables in the schema. The result is displayed in a top-level window and under the table is a copy of the executed SELECT statement.

After the result of a SELECT statement is put on the screen, it is left there and the system continues running. The SELECT result's window may be left for as long as the user wishes, and is destroyed when the "OK" button is clicked.

This describes the execution of a SELECT statement when help mode is off. With help mode on, execution is rather more involved, and this is described in the next section.

3.1 Help Mode Execution

Thus far, most aspects of the system described have been standard components which would be present in any SQL implementation, and all these had to be implemented for the system to be a practical SQL interpreter. However, the purpose of the project was to make the system educational. This has been achieved by providing an optional Help mode which can be turned on and off from the Esql menu and affects the execution of SELECT statements.

The basic form of an SQL query is:

SELECT *column list*
FROM *table list*
WHERE *restrictions*

Help mode attempts to aid the student in understanding SQL queries by taking the user step-by-step through the execution of the statement. Instead of simply displaying the result of a query, esql shows several intermediate tables which are examined during the execution of the query, describing to the user how these tables are processed to achieve the final result. These intermediate tables look very similar to the final table, but have extra text above the table. At each stage in the execution, esql inserts an explanatory message into this text area, to spell out what is being done, and underlines the clause of the query currently being executed in the text area below. In addition, there are two buttons in the window. If the "Step" button is clicked, the next stage in the execution of the query is shown. If the "Continue" button is clicked, the final result is shown immediately. Note that while the system is stepping through a query in this way, the rest of the system becomes disabled. The user cannot do anything else in esql until the "Continue" button is clicked and the final result displayed.

The aim of the project was to design a system which would be used in conjunction with more conventional learning methods, so that students would learn more effectively. The educational features would therefore need to be novel and unique to the system, and would have to provide help which is not available in textbooks or lectures. It was therefore decided to implement a mechanism allowing the user to step through the execution of a statement, showing what steps are taken and how each step is executed, rather than the more usual "static" forms of help such as pre-fabricated demos. This allows the user to obtain help on any statement, instead of being limited to a set of examples.

Although this method could be used with all types of statements, it was decided that the focus should be the SELECT statement. This was because the main aim of the system is to help students to understand the concepts in SQL, and queries cause the most conceptual difficulties for students. In order to do this, the main steps taken in the execution of a query needed to be identified, and for each of these steps, something needed to be designed which would explain to the user how the step is executed.

Conceptually, there is a clear-cut series of steps which define the meaning of a SELECT statement, and esql follows these steps. In the simple queries handled by esql, there are three main steps:

1. Perform a join operation on the tables in the table list.

2. If the query contains a WHERE clause, determine which rows satisfy this restriction, and discard the rest of the rows.

3. Retrieve the selected columns from the table, and if necessary, perform any set functions on them.

3.1.1 Stepwise Help

The simplest way of helping the user to understand each of these steps is to give a textual explanation, briefly describing what each step is and how it is executed. This has been done in esql by placing descriptive text above the partial result of a query. However, this text is the same for each query, and is much like the type of material which would appear in a textbook. Although students need this sort of explanation to understand the main concepts, extra features were needed which would show how these steps apply to the query being executed.

Each of the steps shown above corresponds to a different part of the query — the first deals with the FROM clause in the table expression, the second with the WHERE clause (if it exists), and the third with the SELECT clause. Esql conveys this concept by displaying the query below the partial result, and at each step the relevant portion of the query being executed is highlighted. This enables the user to see what part is being executed, which will make each step a little clearer. The method designed to display each of the three steps is described below.

1. Students often struggle with the concept of the relational join, and they find it hard to visualise what joined tables look like. However, when there are multiple tables in a query, the result of a join is usually very large, and would be far too slow and cumbersome to display. This problem was solved by showing only a sample set of "typical" rows from the result of the join.

2. To show how the execution of the WHERE clause affects the current query, the user must be shown which rows satisfy the WHERE condition and which do not. Again, this need only be done for the sample row set, to give the user a feel for how the WHERE clause acts. Esql does this by highlighting the rows in the sample which satisfy the condition, because this method is very simple and clear. This is shown in figure 1. To emphasise further that the rows which do not satisfy the condition are discarded at this point, an intermediate step is then shown, in which the rows which do not satisfy the condition are removed from the sample.

   To execute these first two steps, an algorithm was needed to choose a sample set of rows from the joined table. The algorithm would need to take into account which rows satisfy the WHERE clause and which do not. A good algorithm would produce a sample containing some rows which would be highlighted in this step and some which would not. The design of this algorithm is described in the next section.

3. The third step is fairly similar to the second, in that the user needs to be shown which columns

are going to be selected by the query. Again, highlighting the columns provides a clear and simple mechanism for this. Another intermediate step is also added, showing only the selected columns, before any set functions are performed on them, since set functions are not executed until the very end.

The overall result of this design is a system which gives detailed help on any query entered by the user. The explanatory text and visual aids complement each other to provide a teaching mechanism which is clear, simple and easy to understand.

4 Algorithm for choosing a Sample Row Set

During the step by step execution described above, the full joined table is usually far too big to show all at once, so esql only shows a sample set of "typical" rows from this table. To do this, an algorithm for choosing a "typical" set of rows had to be devised.

First, it was decided that a good size for this sample would be twenty rows. This would not be too large and slow, but could give a fair cross-section of the rows in the full table.

The next step in designing this algorithm was to decide on a set of criteria which a typical set of rows would have to satisfy. The following are the criteria which were used to determine what would constitute a typical row set from the joined table in a query:

1. The rows must not all come from the same part of the table.

2. If possible, the sample must include several rows which will satisfy the WHERE clause in the query, and several which will not.

3. The third criterion involves the ratio of rows which satisfy the WHERE clause to the total number of rows. As far as possible, the algorithm should try to keep this ratio the same in the sample as it is in the full table. However, an upper limit of 0.75 and a lower limit of 0.25 is placed on this ratio in the sample set.

The algorithm designed satisfies these criteria, and is shown below. (Note: in the following algorithm, the phrase "selected row" is used to mean "row which will satisfy the WHERE clause", and "non-selected row" is used to mean "row which will not satisfy the WHERE clause".)

1. Determine the number of rows in the sample table — it will be either 20 or the number of rows in the full table, whichever is less.

2. Determine the ratio of the number of selected rows to total rows in the full table.

3. Determine the ideal value for the ratio of selected rows to total rows in the sample, which is given by truncating the ratio for the full table to lie between 0.25 and 0.75.

4. Determine the number of selected and non-selected rows in the sample. To do so, work out the ideal number of selected and non-selected rows in the sample according to the ideal ratio. If the ideal number of selected rows in the sample is greater than the actual number of selected rows in the full table, then decrease the number of selected rows and increase the number of non-selected rows in the sample, until the number of sample selected rows is the same as the number of actual selected rows. Similarly, if the ideal number of non-selected rows in the sample exceeds the actual number of non-selected rows in the full table, then decrease the number of non-selected rows and increase the number of selected rows in the sample, until the number of sample non-selected rows is the same as the number of actual non-selected rows.

5. Create a list of all the selected rows in the full table. Divide this list into near-even groups, with as many groups as there will be selected rows in the sample table. (The difference between the number of rows in one group and the number of rows in another should never exceed one.)

6. The list of rows which will be selected in the sample is created by taking the first row from each group.

7. Repeat steps 5 and 6, but with the non-selected rows in the full table. This will result in the list of sample rows which will not be selected.

8. Merge these two lists to create the final sample set of rows.

Steps 1-4 are simply to determine sensible values for the number of selected and non-selected rows in the sample, satisfying the second and third criteria. Steps 5-8 choose a sample set which gives an even distribution of rows through the table, to satisfy the first criterion.

5 Aggregate Queries

Aggregate queries are a special form of the SELECT statement. Aggregate queries extend the capabilities (and complexity) of the simple SELECT statement. The form of an aggregate query is:

```
SELECT column list
FROM table list
WHERE restrictions
GROUP BY column list
HAVING restrictions
```

Like the WHERE clause, both the GROUP BY and HAVING clauses are optional. Conceptually, the query is processed exactly the same as a non-aggregate query up to the WHERE clause, then the rows resulting from this are grouped according to the values of the columns specified in the GROUP BY list. If there is a HAVING clause but no GROUP BY clause, the whole table is treated as one group. Each group is represented as a row in the final table. Then the restrictions in the HAVING clause are tested, and only those *groups* which satisfy these appear in the resulting table. The SELECT clause is applied as normal, except to the groups, not to single rows. The HAVING clause is essentially the same as the WHERE clause, except that it may include set function specifications, such as MIN, MAX and AVG.

Only grouping columns may appear standing alone in the SELECT or the HAVING clauses, although any column may appear as the parameter of a set function. This semantic restriction is a consequence of the fact that these queries work on groups, not single rows – the only columns whose values are guaranteed to be the same for all the rows in a group are the grouping columns.

We believed that most of the misunderstanding of these queries by students stems from the transition from single rows to groups of rows, and the representation of each group as a single row in the final table. When one of the optional clauses is missing, this can also be confusing.

5.1 Representing Aggregate Queries in Help Mode

Much careful thought was given to the display of the display of aggregate queries. Not only did it have to blend in with the interface for simple joins, but it had to address those areas of aggregate queries which we believed students had the most trouble with, the main one being the transition from rows in the table to groups represented as rows. We decided that the best way to illustrate this transition was to show the rows representing the groups, and to allow the user to open another window to show how the rows from the previous clause were grouped together to create these group-rows. Of course, as with all other steps, the GROUP BY or HAVING clause (as appropriate) is underlined and explanatory text is displayed above the partial result.

The steps now performed and displayed by the system are the same as for non-aggregate queries, with two additional steps between steps 2 and 3 (see section 3.1.1). These additional steps are:

i. The information displayed here depends upon whether or not there is a GROUP BY clause in the query.

   (a) If there is a GROUP BY clause, the partial result shown is a table containing one row for each group, whose columns are the grouping columns from the GROUP BY clause and set functions from the SELECT and HAVING clauses. A third button appears below it labelled "Display Rows". If the user clicks on "Display Rows", a new window appears containing *all* of the rows satisfying the WHERE clause (not just the rows from the sample displayed up to this point, see section 5.2). These rows are shown grouped together, with a blank row between the groups and the grouping columns highlighted. (See figure 2.) The user returns to the previous window by clicking on the "OK" button in this new window.

   (b) If there is a HAVING clause but no GROUP BY, then the whole table is treated as one group. The SELECT and HAVING clauses can only contain set functions because of the semantic restriction outlined above (Section 5). Therefore only one row containing only set functions is displayed.

   A point to note is that, unlike non-aggregate queries, where set functions in the SELECT clause are not evaluated until the very end, aggregate queries must evaluate their set functions at this stage, since the rows making up a group are essentially discarded at this point. The set functions form part of the values for the row representing the group.

ii. If there is no HAVING clause, then the display moves on to Step 3 (Section 3.1.1). Otherwise, the group-rows which satisfy the HAVING clause are highlighted, in the same way that the WHERE clause highlights rows, and, like the WHERE clause, an intermediate step is shown with only the highlighted rows remaining in the table to emphasise that unselected group-rows are discarded.

5.2 Other Display Considerations

In addition to the semantic restrictions outlined above (Section 5), that only grouping columns may appear standing alone in the SELECT or HAVING clauses, a further restriction was added — that all grouping columns must appear in the SELECT list. Although this is not a part of the official specifications of SQL [8], many commercial DBMS include this restriction, and we felt that it would not be too much of a burden to students to become accustomed to it.

The system could have displayed only those groups associated with the selection of rows used to display the FROM and WHERE clauses, but it was decided from the outset that the display of the groups and grouped rows would use all of the rows that resulted from the WHERE clause — that is all of the rows in the final table had the query not had a GROUP BY or HAVING clause. This was done for two reasons. Firstly, students may wish to check the values of the computations of the set functions; but secondly, and more importantly, the reason for using only a selection of rows no longer applied once the query processing had reached the grouping stage. The reason that only a selection of rows was used was that the virtual table of joined rows resulting from the FROM clause would be too big to display for even moderate queries. But since all of the rows involved in the display of the groups would have appeared in the final table had the GROUP BY or HAVING clauses not been in the query, this reason no longer applied at this stage.

6 Conclusion

We believe that the system we built has met the goal we set ourselves — it provides students with a visualisation tool to help them understand areas of SQL that they often find troublesome. The dynamic stepping-through mechanism which allowed students to see a query *in action* as well as its overall result is both novel and far superior to the usual pen-and-paper explanations given in lectures and textbooks. In particular, the display of aggregate queries can help clear up an area of considerable student difficulty.

In future, we hope to extend the system to cover another area of confusion — that of sub-queries, which is difficult to illustrate using a static medium such as a textbook. We would also like to extend its usefullness to students and tutors alike by incorporating a tutorial system [1, 5] into esql which would allow tutors to set and mark exercises and students to complete and submit them. A prototype of such a system has been incorporated into one version of the esql, and its main focus is to take advantage of the stepping-through mechanism to allow a unique type of exercise to be set and completed, so that students can really test their own understanding of SQL query evaluation.

References

[1] Anderson, J., Boyle, C., Reiser, B., "Intelligent Tutoring Systems", *Science*, 228, 1985.

[2] Brown, M., "Perspectives on Algorithm Animation", *Proceedings CHI 88, Human Factors in Computing Systems*, May 1988, ACM Press.

[3] Dietrich, S., Eckert, E., Piscator, K., "WinRDBI: A Windows-based Relational Database Educational Tool", *28th SIGCSE Technical Symposium on Computer Science Education*, Vol 28, No 1, March 1997, ACM Press.

[4] Guimerães, M., Lucena, C., Cavalcanti, M., "Experience Using the ASA Algorithm Teaching System", *SIGCSE Bulletin*, Vol 26, No 4, Dec 1994, ACM Press.

[5] Johnson, W., *Intention - Based Diagnosis of Novice Programming Errors*, Morgan Kaufmann Publishers Inc., 1986.

[6] Kearns, R., "A Teaching System for SQL", Honours Thesis, Basser Dept. of Computer Science, University of Sydney, 1995.

[7] Lim, B., Hunter, R., "DBTool: A Graphical Database Design Tool for an Introductory Database Course", *23rd SIGCSE Technical Symposium on Computer Science Education*, Vol 24, No 1, March 1992, ACM Press.

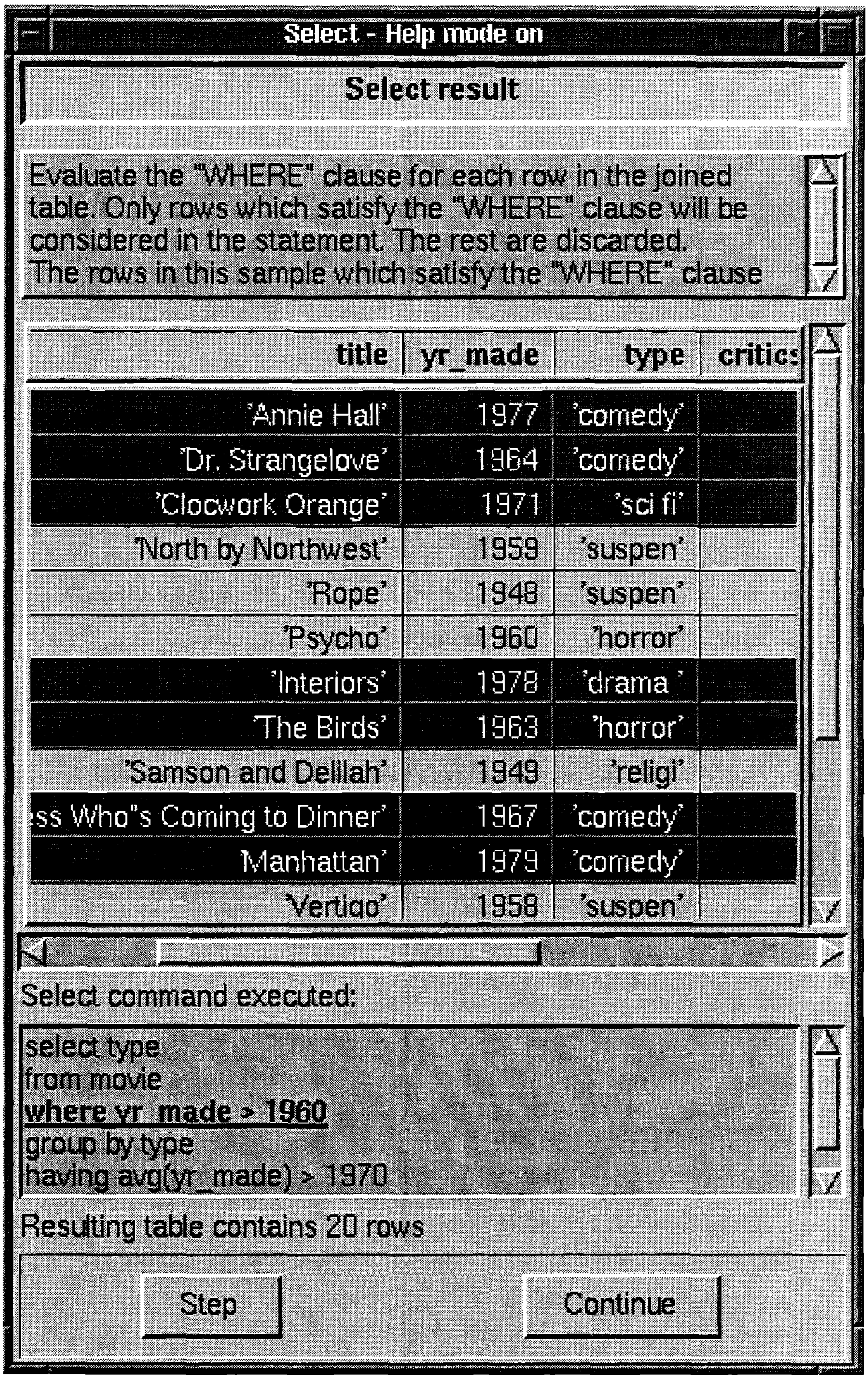
[8] Melton, J., Simon, A., *Understanding the New SQL*, Morgan Kaufmann Publishers Inc., 1993.

[9] Naps, T., "Algorithm Visualisation in Computer Science Laboratories", *21st SIGCSE Technical Symposium on Computer Science Education*, Vol 22, No 1, Feb 1990, ACM Press.

[10] Rasala, R., Proulx, V., Fell, H., "From Animation to Analysis in Introductory Computer Science", *25th SIGCSE Technical Symposium on Computer Science Education*, Vol 26, No 1, March 1994, ACM Press.

[11] Reiser, B., Friedman, P., Gevins, J., et al.,"A Graphical Programming Language Interface For an Intelligent LISP Tutor", *Proceedings CHI 88, Human Factors in Computing Systems*, May 1988, ACM Press.

[12] Shead, S., "A Teaching Environment for SQL", Honours Thesis, Basser Dept. of Computer Science, University of Sydney, 1994.



Select - Help mode on

Select result

Evaluate the "WHERE" clause for each row in the joined table. Only rows which satisfy the "WHERE" clause will be considered in the statement. The rest are discarded.
The rows in this sample which satisfy the "WHERE" clause

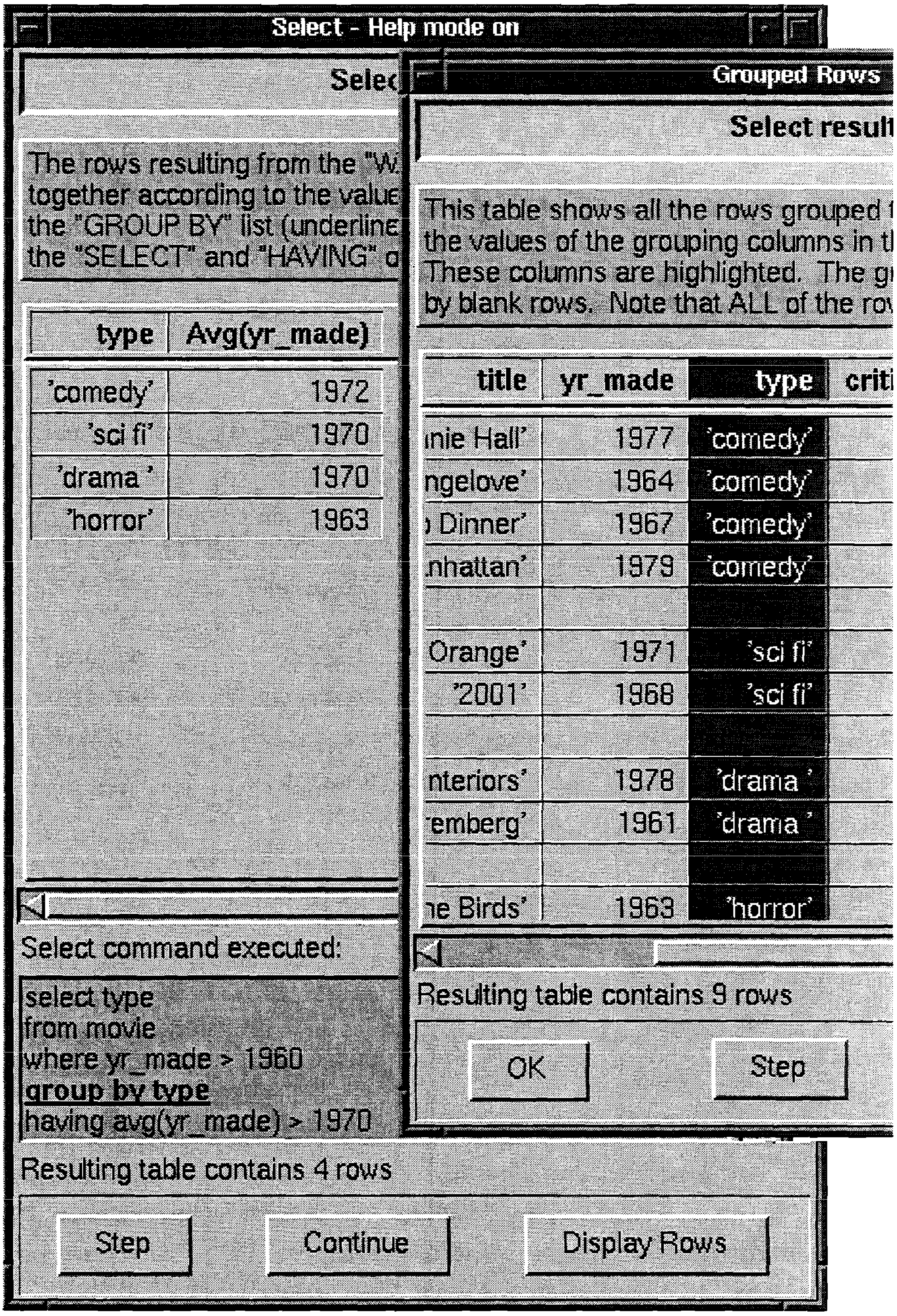
title	yr_made	type	critic
'Annie Hall'	1977	'comedy'	
'Dr. Strangelove'	1964	'comedy'	
'Clocwork Orange'	1971	'sci fi'	
'North by Northwest'	1959	'suspen'	
'Rope'	1948	'suspen'	
'Psycho'	1960	'horror'	
'Interiors'	1978	'drama '	
'The Birds'	1963	'horror'	
'Samson and Delilah'	1949	'religi'	
ss Who"s Coming to Dinner'	1967	'comedy'	
'Manhattan'	1979	'comedy'	
'Vertigo'	1958	'suspen'	

Select command executed:

```
select type
from movie
where yr_made > 1960
group by type
having avg(yr_made) > 1970
```

Resulting table contains 20 rows

Figure 1: The rows selected in the WHERE clause highlighted.



The rows resulting from the "W
together according to the value
the "GROUP BY" list (underline
the "SELECT" and "HAVING" c

type	Avg(yr_made)
'comedy'	1972
'sci fi'	1970
'drama '	1970
'horror'	1963

Select command executed:

```
select type
from movie
where yr_made > 1960
group by type
having avg(yr_made) > 1970
```

Resulting table contains 4 rows

This table shows all the rows grouped
the values of the grouping columns in t
These columns are highlighted. The g
by blank rows. Note that ALL of the ro

title	yr_made	type	criti
nie Hall'	1977	'comedy'	
ngelove'	1964	'comedy'	
Dinner'	1967	'comedy'	
nhattan'	1979	'comedy'	
Orange'	1971	'sci fi'	
'2001'	1968	'sci fi'	
nteriors'	1978	'drama '	
emberg'	1961	'drama '	
e Birds'	1963	'horror'	

Resulting table contains 9 rows

Figure 2: Both windows involved in the GROUP BY display - the first shows each group represented as one row, the second shows the rows from the WHERE clause, with the grouping column(s) highlighted and each group separated by a blank row