What SQL can tell us about 4D

Lincoln Stoller, Ph.D.
Braided Matrix, Inc.
7/9/92

A Little History

SQL, short for Structured Query Language, is a mainframe database access and management language created by IBM engineers in 1974. SQL's sole purpose is data management, it is not an operating system or programming platform. Vendors of large relational database systems who have based their platforms on SQL have done so by adding programming, compiling, interface and multiuser tools as well as developing their own SQL extensions. To compare 4D to SQL we either have to compare 4D with one of these SQL-based platforms like Ingres, Oracle, or Sybase, or compare the data management component of 4D with SQL. In this article I'll focus only on the issue of data access because this brings out some of the most fundamental differences.

A little history will help put SQL in perspective. 20 years ago computer databases ran only on mainframes. They were managed by teams of MIS (Management of Information Systems) specialists who were often the only people able to access the data. Users submitted questions to MIS, MIS wrote programs that printed reports, and users received the reports days later.

MIS managed the database by executing system level commands or by submitting data processing jobs for batch processing. The MIS user was a specialist and the data access cycle involved designing, writing, editing, compiling, debugging and running programs. The problem that faced those developing SQL was to reduce the required level of expertise and to speed up the data access cycle. Their SQL language provided a toolbox of compact statements that freed the user from having to know how or where the data was stored. The user could also get information out of the database as soon as they entered a command. The fact that it still required trained operators and produced output in the least embellished form, namely as text, could hardly be considered a drawback given the circumstances.

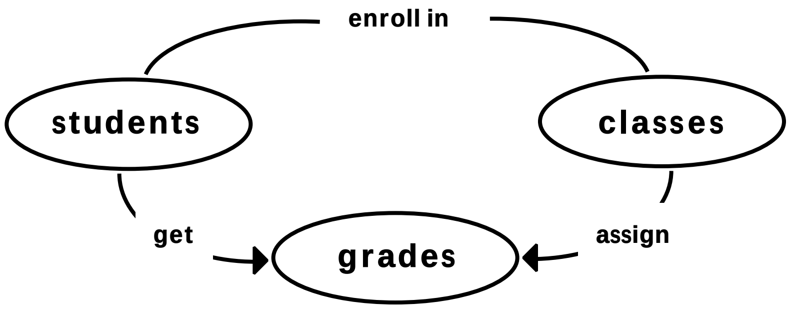
SQL supports the relational database model, as does 4th Dimension. SQL provides tools to procedurally build and destroy tables, columns (the parallel to files and fields in 4D) and indexes with the commands CREATE and DROP TABLE and INDEX; and to add, modify and delete rows (what we call records in 4D) with INSERT, UPDATE and DELETE. However the main purpose of a database is to provide access to data and in this respect SQL's SELECT statement, which handles almost all data retrieval, is used most frequently. In this article I'll focus exclusively on this statement.

The SELECT statement simply requests information that is returned in a table consisting of columns (fields) and rows (records), however SELECT is not a simple command. It uses a host of modifiers, conditionals and subclauses to return grouped, sorted and formatted information with column headings, subtotals, averages and other calculated results. There are introductory SQL textbooks devoted to the command and it's complexity makes it impossible to compare with any single command in the 4th Dimension language.

To accomplish the same effect as the SELECT statement 4th Dimension uses a variety of less complicated commands such as: SEARCH, SEARCH BY INDEX, SEARCH BY FORMULA, SORT, and SORT BY FORMULA for retrieving information; looping structures and set manipulation to screen data; and DISPLAY, MODIFY and PRINT SELECTION along with their layouts for displaying information. The two languages are best compared by way of an example.

A Sample Database and Question

Consider a database that manages students and their grades in various classes. In this model students can have one or more classes, classes can have one or more students and different students can take different classes. Every student also gets a grade in each of their classes. These relations are shown in figure 1.

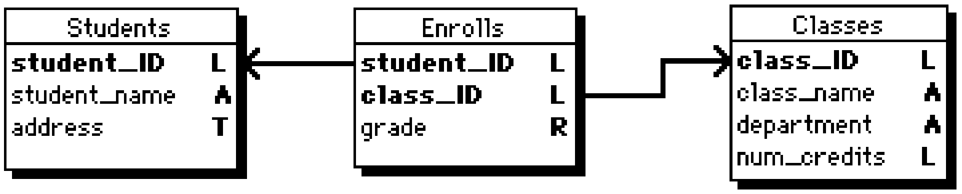


In this context we ask: --- Figure 1 : elements of the sample database. ---

"What students received above average grades in one or more of their classes?"

The 4D Answer

To model the information in 4D we'll use three files, on to track students, another for classes and a third to record a student's grade in a given class. This many-to-many file structure shown in figure 2.



--- Figure 2: 4D file structure. ---

Finding the students that meet our criteria requires a special procedure such as the one given in figure 3. This procedure has two loops, an explicit outer loop for classes and an implicit inner loop, effected by the Average command, that computes the average grade for each class. Finally, the procedure accumulates the above average students in each class using sets in the [Student] file. The use of sets removes any duplicate mention of students that could occur if we simply displayed a selection of records from the [Enrolls] file.

```
`Procedure FindAboveAvrg
CREATE EMPTY SET    ([Enrolls];"AboveAverageGrade")
ALL RECORDS   ([Classes])

`Find those enrolled whose grades are above the class average.
For  ($j;1;Records in selection     ([Classes]))
  RELATE MANY   ([Classes]class_ID)
  $ClassAvrg:=Average  ([Enrolls]grade)

  SEARCH  ([Enrolls];[Enrolls]class_ID=[Classes]class_ID;*)
  SEARCH  ([Enrolls]; & [Enrolls]grade>$ClassAvrg)

  CREATE SET   ([Enrolls];"AboveAverageThisClass")
  UNION  ("AboveAverageGrade";"AboveAverageThisClass";"AboveAverageGrade")
  NEXT RECORD   ([Classes])
End for

USE SET  ("AboveAverageGrade")

`Find students using sets so that duplicates are eliminated.
CREATE EMPTY SET    ([Students];"AboveAverageStudents")
FIRST RECORD   ([Enrolls])
For  ($k;1;Records in selection     ([Enrolls]))
  RELATE ONE   ([Enrolls]student_ID)
  ADD TO SET   ([Students];"AboveAverageStudents"))
  NEXT RECORD   ([Enrolls])
```

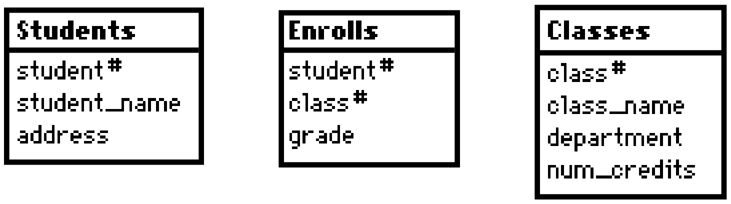
```
End for
USE SET   ("AboveAverageStudents")

CLEAR SET   ("AboveAverage")
CLEAR SET   ("AboveAverageThisClass")
CLEAR SET   ("AboveAverageStudents")
DISPLAY SELECTION      ([Students];*)
```

--- Figure 3: 4D querying procedure. ---

The resulting selection of [Student] records is displayed using the current [Student] output layout.

The SQL Answer:

In SQL the data structure is pictured in figure 4 as a set of tables, similar to the files of figure 2. Since the SQL user never needs to know which fields are indexed or the relations between fields these elements are not shown.



--- Figure 4: SQL tables. ---

The SQL query is given by the following single command which I've written over several lines for greater clarity:

```
SELECT DISTINCT      Students.student_name
FROM   Students, Enrolls Enrl
WHERE   Students.student# = Enrl.student#
AND  Enrl.grade > (       SELECT AVG   (Enrolls.grade)
                          FROM   Enrolls
                          WHERE   Enrolls.class# = Enrl.class#)
```

The SELECT statement is a combination of special syntax and quasi-natural language that's fairly easy to unravel. The first line indicates what is to be returned, the second indicates the tables to be used, and the remaining lines establish the conditions to be met. I'll consider these lines one at a time.

“SELECT DISTINCT Students.student_name” indicates that a list of distinct student names are requested.

“FROM Students, Enrolls Enrl” says that the Students and the Enrolls table will be involved. The “Enrl” after Enrolls indicates that references to quantities in the Enrolls table at the outer level of the Select command will be abbreviated as “Enrl.” This is analogous in 4D to establishing a current selection from the [Enrolls] file and referring to it as a selection in the [Enrl] file.

SQL essentially allows you to define more than one current selection for a given table and when you do this you must give each selection a distinct label. We’ll see the purpose of this construction when we look at the embedded SELECT clause.

“WHERE Students.student# = Enrl.student#” establishes a link between the Students table and the Enrolls table, now referred to as Enrl. Doing this indicates that for any Enrolls record found, the corresponding Students record is the one with the matching student#.

“AND Enrl.grade > (SELECT AVG(Enrolls.grade)...” establishes the condition that for an Enrolls record to meet the selection criteria the grade value must exceed the quantity returned by the subquery that’s within the parenthesis. In this case the subquery is the average of a selection of grades FROM the Enrolls table.

“WHERE Enrolls.class# = Enrl.class#” says that this secondary selection of grades, the ones used to compute the class average, consists of all those grades for Enrolls records that match the class number currently being examined as part of the outer selection of Enrolls records.

Putting this all together we have two loops, one at the outer level where we’re examining the grades for all the Enrolls records, and then for each record we’re entering an inner loop to compute the average grade from those records with the same class number. We then test the grade of the outer Enrolls record against the class average.

The analogy I made above to a current selection in 4D is only true in a limited context. What SQL does is to construct a temporary table used in processing the SELECT statement. This might better be likened to an array than to a current selection. The array is erased after use and by itself the SELECT command does *not* establish any “current selection” of records. SELECT simply returns a stream of text and then disappears.

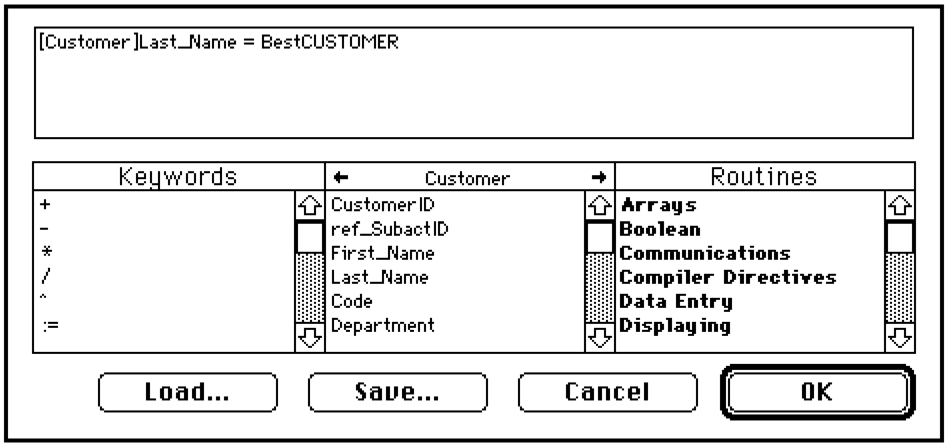
SQL does support a construct called a cursor which is a pointer to the current row in a selection of rows that can be returned by the SELECT command. A cursor is defined for a particular SELECT statement and it enables you to procedurally step through the selection. SQL’s concept of a selection of rows is more limited than 4D’s current selection and, in any event, is unavailable when SQL is run interactively. We won’t use cursors but I wanted to mention them for completeness.

The Structure of the Two Languages Compared

How is it that what required 24 lines of 4D code can be written in one SQL statement, albeit one *complicated* statement? How is it that in SQL we only specify what needs to be done without any reference to how the task is to be carried out?

The answer is that SQL and 4D are fundamentally different languages. SQL is a “nonprocedural” language that frees us, more or less, from having to know how the data are stored and what steps should be followed to arrive at an answer. SQL can support this higher level of abstraction because it’s built on a parsing, optimizing and compiling “engine” that automates all the work that we did by hand in 4D. The SQL engine knows the size of the data files and the data and index structure and it automatically converts our SELECT statement into a series of commands optimized for performance. SQL’s query building environment is itself a user interface, which should come as no surprise since when SQL was invented there was no other more modern interface.

The SELECT statement it is more that a static command embedded in procedure code, it is an interactive query building tool. We can compare it with 4D’s own query building and reporting tools, most importantly the SEARCH, SORT and REPORT commands called from the Runtime environment and the Search Editor, Search by Layout, Search by Formula and Quick Report Editor called from the User environment. The Search by Formula editor is shown in figure 5.


[Customer]Last_Name = BestCUSTOMER

--- Figure 5: Search by Formula using a custom procedure.---

4D's query and reporting tools have a graphical user interface that's "friendly" enough for end-users whereas SQL's syntax is too complex for end-users except in the simplest cases. On the other hand 4D's tools are not nearly as powerful. The question we just considered, for example, is too complicated to be constructed interactively in 4D.

Comparing SQL's SELECT statement with elements in the 4D language shows how different two "relational" database languages can be. In Table I I summarize SQL's SELECT, 4D's programming language and 4D's interactive query tools according to the categories of User Interface, Audience, Scope, Method, and Result.

	4D's interactive editors (Search, Sort ,Report)	4D Programming Language	SQL's SELECT statement
User interface	Graphical.	Written in the language editor, embedded in a procedure.	Run interactively via a command-line interface or embedded in a procedure.
Audience	Sophisticated end-user.	DB designer and 4D programmer with full knowledge of structure.	DB designer and SQL programmer, little knowledge of db structure is required.
Scope	Relatively simple tasks that can be extended by calling custom procedures.	All noninteractive tasks that can be hard coded ahead of time.	All tasks embedded (hard coded) and interactive. Provides for simple expression of complex conditions.

Method	Some index use, not optimized, interpreted execution.	Embedded in compiled procedures that can be run interactively. Optimized by designer.	Automatic assembly and optimization of structure specific query. Compiles code before executing.
Result	A current selection useful for further analysis or reporting.	Same as 4D's interactive editors.	Text, composed of rows and columns, that can be stored or displayed but serves no further function. A current selection and record can be defined but only for embedded SQL code.

--- Table I: comparison of the query languages. ---

SELECT is more powerful than 4D in its scope and method, that is in the extent of what can be done (especially interactively) and the ease of query building.

SQL SELECT's special powers include:
• a wider scope: it can answer complex questions more easily;
• a more powerful method: a lot of the work is automated.

The Shape of Things to Come

In theory 4D can do anything that SQL can, the problem is that in 4D it's harder. This shortfall is being addressed through the addition of new query building tools. Query building externals for 4D are being developed by third parties and we can expect to see new tools provided within the 4D environment in subsequent releases.

When it comes to removing the drudgery from writing procedural code SQL's automatic query building offers a huge advantage. While we may not get a code generator to write our 4D procedures for us, we will see new and more elaborate built-in procedures. For example, one command scheduled for the next release of 4D establishes a selection in a related file based on a selection in a given file, that is it automates the relation I refered to as a Group-to-Group relation in my previous article. There are also improvements to the SEARCH command for better use of indexes, a new type of set that remembers the ordering of its contents as well as other improvements.

Areas where 4D is most likely to be extended include:

• more complex and powerful interactive query tools, like expanded Search and Report editors;

• additional built-in procedures to encapsulate common looping and relational constructions.

Improvements in scope and method are important targets for 4D and meeting these targets will bring the 4D engine up to par with the most sophisticated database systems. From a developer’s point of view the more the language does for us, the faster we’ll generate problem-free code that enables our clients to be more productive.