Chamberlin and Boyce - SEQUEL: A Structured English Query Language

Comments on:

Chamberlin, D. D. and Boyce, R. F. (1974). SEQUEL: A Structured English Query Language. *Proceedings of the 1974 ACM SIGFIDET Workshop on Data description, Access and Control*, May 1974.

The authors say that SEQUEL is intended for non-computer specialists. Do you have an idea of how popular SEQUEL or SQL has been among non-computer specialists?

I do not have precise information available, but I would say it is very popular for people who need to work with data.

"The concepts of structured programming have been introduced in order to simplify programming and reduce the cost of software". What exactly do the authors mean by structured programming and in relation to declarative languages, has it really been shown to reduce the cost of software, or is that an assumption they're making?

Structured programming incorporates most of the techniques we've seen repeated again and again in the papers from the late 1960's and early 1970's: Goto-less control structures, problem decomposition (typically top-down, but also bottom-up), increasing levels of abstraction. Structured programming is used primarily for imperative programs. From my perspective, structured programs are easier to understand (which simplifies maintennnce, if not programmimng itself). I don't know about formal studies of cost. So, it's an assumption they are making, but it was a widely held assumption.

Where does SQUARE come from? It is so ugly it makes me want to cry.

They told you. They wrote it. It looks to me like they're trying to use two-dimensional formatting to clarify relationships. In the time period of the paper, that was a common technique. History has shown it to be a bad idea. It is, however, quite concise :-)

Is this really the best they can do for readability? I keep seeing in articles about how a particular language is more and more like English and therefore easier and easier to use and understand. When they started talking about this in Ruby I didn't buy it, some of the "sentences" still look like gibberish. SEQUEL certainly does a better job of simplifying commands but I am highly critical of the claim that it is either natural or particularly easy to use.

You need to balance readability with unambiguity. I think SEQUEL does a decent job, and it is certainly more readable than, say the corresponding C or Scheme might be. Part of the problem in these examples comes from the relationships of the way data were arranged into tables to the kinds of queries that they have as samples. But yes, I agree with you, this is not particularly English-like once you get an intersting query.

One of the goals of SEQUEL is to enable non-professional programmers to be able to manipulate databases (pp. 250). While I largely agree with this I have a bone to pick with the statement: the language gives too much power to the person issuing the queries especially where permissions are not well set. For example:

```
DELETE * FROM STUDENTS WHERE ADVISOR='REBELSKY';
```

is very different from

```
DELETE * FROM STUDENTS;
```

There should be some mechanism to ensure that such mistakes by the programmer do not happen.

While I agree with you that access mechanisms are important for changes to the database, you'll note that this paper focuses exclusively on how one gets data from an existing database. Changes to the database are left as a separate issue. One might note that security issues also come into play for read-only access (e.g., you should not be able to read someone else's data), but that is also something of an independent issue.

The SELECT-FROM-WHERE block seems to do a pretty good job when it comes to simplicity and ease of understanding. But how does SEQUEL handle errors? Although declarative languages do not require the user to know the implementation of a procedure, I guess that they need to know how to deal with errors. However, the paper does not specify what happens when an error is encountered. For example, when someone uses outer and inner mapping (pg255), what happens if the table for inner mapping does not exist? Or somehow it cannot return any value to be applied to the outer mapping?

If a table or column does not exist, I expect that you get an error message that the table or column does not exist. If you try to select data and no data meet the criteria, then presumably you get no results. But that issue happens even if there is no inner mappings - given the tables they show, selecting all employees whose name is Sam would also give you no result.

In addition, you'll note that they often envision an interactive system that guides the user. I expect that such a system could help ensure that only valid table and column names are chosen.

On page 253, Chamberlin mentions that SEQUEL and SQUARE are equivalent in power. Why would anyone want to use a language that is less natural to read. Are there other benefits of SQUARE that aren't mentioned?

SQUARE is much more concise. Many people value concision over novice readability. (I expect that for expert SQUARE programmers, SQUARE was equally readable to SEQUEL.)

Also, Q5.2 on page 257 doesn't make sense to me. I understand how they got John and Fred as the result but it doesn't seem like it's a method you would want to use. Fred (Toy) and Fred (Shoe) should be different. I guess this means that when the intersection is applied the strings are compared, not pointers to their location in memory. Is this a disadvantage?

I agree with you that the result is confusing at first glance, but that the result does make sense once you realize that there are two Freds. I think they're making a subtle point about database design, which is that you need to be careful with what you select as a key. We shouldn't use non-unique values as keys. We

could probably rewrite this with an EMPID field, intersect the EMPIDs from the two selections, and then select names based on EMPID.

The article states that SEQUEL users describe the relevant data to be accessed by set expressions rather than by row-at-a-time iteration. This tends to make things more concise, clearly expressed, easy to write , maintain and modify. What implications does this sort of access have on runtime and what are the disadvantages of access by set expressions?

One implication is that the implementors get a bit more freedom in how to handle the query. They could parallelize it, precompile some results, and so on and so forth. What are the disadvantages? If you only want one value that meets some criterion, rather than all values, it may be more difficult to express that query. To many programmers, the inability to know how your queries are actually implemented is also a disadvantage.

I find it surprising that declarative languages are not more widely used. For most people, it seems the natural way ot interact with a machine. While I realize that most declarative languages are pertty slow, most computers should be fast enough now that it is not really an issue. So why else have declarative languages not been popular?

It depends on the application. There are a lot of declarative languages that are relatively popular in restricted domains. SQL is certainly very popular, and many people succeed in building interesting stuff with SQL without drawing upon the more imperative languages that often accompany SQL. Some consider Excel a declarative language, and an immense amount of data-directed "programming" goes on in SQL. It's also the case that for many problems, unless you have some careful control over how data are processed, the processing can be wildly inefficent.

Claim from paper (approximately): "Declarative languages are easier and more efficient to program in" The author shows that SEQUEL queries are clear and easy to write, this makes me curious as to why people might still choose to use square or any other language based on predicate calculus. That is, are there any disadvantages to using SEQUEL?

SQUARE and the ilk are also somewhat declarative. You'd use a more imperative language because you care about efficiency and don't trust the underlying implementation.

Also, in the first paragraph of page 250, the author claims that there is an evolution from procedural to declarative problem specification. On Monday April 23, we talked about the major classifications of languages, but from the outline, I do not recall us talking about procedural languages. Is this another name for one of the classifications we talked about?

Yes, "procedural" is another name for "imperative" (or vice versa). I don't use "procedural" because it's too close to "functional" (and because we use "procedure" when talking about the functions in Scheme).

The declarative structure of SEQUEL is very clean thanks to many of the abstractions provided to the user, but I am having trouble thinking of situations outside of database querying and make files where writing a rules based program would be easier than writing an imperative, functional or object-oriented program.

Excel is my current favorite example. Lots of people write declarations in Excel (this cell depends on the following cells, some of which depend on other cells, ...). AI is another field that has benefitted from declarative languages.

It seems to be a really roundabout away of writing a program by giving rules rather then steps in an algorithm. Even if a rules based approach did work, it seems it would have to be highly domain specific, making reuse difficult. What I mean by this is that it seems unlikely that programs written in declarative style would be dependent on their domain.

It sounds like you mean that the programs would depend on their domain, rather than wouldn't. The key idea here is that for most users*, writing the algorithm is not an appropriate task, describing desired results is. And even for many programmers, there are some significant advantages to not having to describe every algorithm. (For example, most of us rely on built-in hash tables, vectors, sorting routines, and the ilk.)*

This paper only talks about queries and the relationship between some simple queries and math. It doesn't talk about other application delete, add , update,etc. It introduced the inner mapping or nested query which I did not know before or it uses different grammar from SQL. The invention of SEQUEL obviously connects the end-users and data processing. Two questions: How long will be a query saved in the memory, can we do something cool like the continuation mechanism? Even we can use nested mapping, we can't do recursion, can we?

Let's see ... The focus of SEQUEL is clearly retrieval, not update. I'm not sure how many of the SEQUEL mechanisms made it into SQL, and how they evolved. Certainly there are ways to do things similar to the queries in the paper, although slightly differnet.

It is, of course, up to the implementer how long something is saved in memory. I'm not sure how a continuation would be appropriate here. The most natural place in the SEQUEL syntax to stop is afer completing an inner mapping, and I think we could handle that simply by naming the result of the inner mapping. No, you can't do recursion, but I'm also not sure how recursion would apply in a sitution like this.

How do SEQUEL and MySQL differ?

SEQUEL seems primarily written for retrieval. MySQL handles both. The remainder of the differences are a good topic for class discussion.

The part about free variables (pp. 258) lost me. Is x being defined in terms of itself? How does that work?

No, x isn't being defined in terms of itself. Read it as "x in EMP such that *...".*

Is the function of "GROUP BY" to simply sort outputs?

No. "ORDER BY" sorts outputs. (And they don't provide it.) "GROUP BY" builds groups of items (in some sense, subrelations) with common features. In some sense, "GROUP BY" is a way to undo normalization.

Every pronoun in this reading is masculine. :(

Welcome to the early 1970's.

The authors use "keyword English templates" instead of something more "mathematical" to make the language more accessible to non-programmers, but they never examine whether this claim is true. I would argue that it isn't necessarily. The lack of punctuation or mathematical notation makes the code more difficult to discern without a careful reading. At a glance, it isn't clear which words are keywords and which are table, column, row, or variable names. This lack also confuses order-of-operations groupings, which the authors themselves admit on page 257 when they suggest the use of semicolons; however, even the proposed usage is ambiguous. On page 256, the authors need to explain the difference between using AND and OR within a mapping and using them between mappings-- but if there were some method to logically group statements (perhaps parenthesis), I think that distinction would have been trivial. Generally speaking, the English language is inherently ambiguous, which is why we need highly structured programming languages in the first place. Yes, SEQUEL is structured, but it seems that we wouldn't want to go *closer* to something ambiguous when creating a language for people who do not necessarily think in a highly structured, computational sort of way.

I'm not sure the SEQUEL is more ambiguous than English. (Yeah, it is more ambiguous than SQUARE.) Note that some of the issues you're hitting is audience. The authors are writing for other computer scientists, discussing how they build things for non-computer-scientists. It is, in some sense, a preliminary proposal. These days, one would hope that they'd do usability testing and such, but those were not central ideas in those days. I expect the authors would also tell you that the tool used to build the queries would help ensure that authors could understand what is happening and write unambiguous queries.