



The IDUG Solutions Journal October 1998 - Volume 5, Number 3

The editors of IDUG Solutions Journal have invited two DB2 experts to duel peacefully on key issues facing DB2 today. In this regular column, [Willie Favero](#) of BMC Software and [Craig Mullins](#) of PLATINUM technology will face off. Sometimes they even agree.

SQL for the Next Century: Something Old, Something New, Something . . .

By Willie Favero

SQL is now over 25 years old. Since the development of System R (which also celebrated its 25th anniversary just a few years ago), SQL has come a long way. Today, more than 100 products use some form of SQL. That number will only increase in the future. During its life, SQL has seen many changes; it will likely see even more. While each of us would like to see some changes, here are a few of my favorites.

SQL was born of humble beginnings with IBM Research in the mid 1970s. Back then, SQL was actually referred to as Sequel, and later Sequel/2. Later on, the name was changed to SQL for legal reasons. For some reason sequel is still around, although SQL is what it should be called. DB2, and for that matter IBM, was not the first to actually release a commercial product that used SQL. As Don Chamberlin pointed out in his book *A Complete Guide to DB2 Universal Database*, a small company called Relational Software beat IBM to market in 1979 with a product you know as Oracle. IBM did not release a product that used SQL until a few years later. That product was still not DB2; it was SQL/DS. DB2 finally entered the market in 1983.

SQL standards

SQL was around for years before a standard for it was developed. Not until 1986 was the concept of an SQL standard, known as SQL/86, approved by the American National Standards Institute (ANSI). The same standard was approved by the International Organization of Standards (ISO) the following year. When SQL/86 was later enhanced, it became SQL/89. With additional enhancements, it eventually became SQL/92 (or SQL2). DB2 for OS/390 is now SQL/92 Entry Level standard compliant. SQL3, with its eight parts, is currently being voted on. Why review SQL's history? To show how long DB2 existed before the standard began to affect it. The "standard" would have a great

impact on SQL for years to come. To see what is in store for SQL in the 21st century, we need only look at the latest standards. To see the future of SQL in DB2 for OS/390, we need look no further than DB2 UDB Version 5.2.

(I digress here to register a complaint: IBM has once again made it almost impossible for us to talk about DB2 and differentiate the platforms to which we are referring. We now have Universal Database (UDB) available to us everywhere. In this article, I will refer to the mainframe flavor of DB2 UDB, the one near and dear to my heart, as simply DB2 for OS/390. All other flavors of DB2, the baby DB2s on NT, OS/2, UNIX, etc., will be referred to as DB2 UDB. This should temporarily put the naming convention back to where it was prior to the announcement of DB2 UDB for OS/390 Version 6.)

The impact of standards on the direction of SQL is viewed as a mixed blessing: either helping or stifling the SQL language. Many fear, for example, that standards may force conformity. Conformity can be good, for example, in reducing a company's cost when developing new SQL based applications. Regardless of which database an individual developed her expertise on, the SQL she uses is always the same. New SQL dialects need not be relearned because the relational database manager changes. How then does conformity stifle SQL's growth? By forcing a database manager to use only SQL that conforms exactly to the standard.

Improvements may never occur, and performance may be hindered. By preventing a vendor from developing SQL that takes advantage of unique features in its database engine, maximizing SQL strengths may never come about. Is there a problem with this logic? Absolutely. Nothing in the standard tells a database vendor that it cannot add more to its SQL. The standard is nothing more than a common starting point. The next question should be "how can we have portability while adhering to a standard?" I believe portability is highly overrated. As suggested above, it is great for making people portable. But how many times is an application developed for one database system, then suddenly moved to a different database manager without the need to rewrite the code? Performance may be the reason for the database manager switch. Moreover, to achieve that new predicted performance, it may be necessary to take advantage of features unique to the new relational database.

IBM needs to continue the active role they play today in defining and developing the SQL standard. They should proceed by influencing the SQL standard to DB2's, and therefore IBM's advantage. However, there is more to a standard than being something everyone likes. It is even more important that IBM continue to enhance DB2's Optimizer. SQL optimizations is one of DB2's indisputable strengths. And with ODBC, and now JDBC, playing an ever greater role in the development of applications that access DB2, it is important for DB2 to continue to improve the way it processes dynamic and static SQL. The growth of ERP applications demonstrates the need to improve dynamic SQL's performance. Whether the standard is good or bad, it is DB2's responsibility to ensure that it always performs as well as possible.

SQL enhancements

What SQL enhancements should be implemented in DB2 for OS/390? First, a few disclaimers. I have no influence on database vendors, I'm only speculating about the future, my comments are based current standards, and I acknowledge a bias towards the wonderful world of application programming. I have studied much of the SQL/92 standard and a large portion of the SQL3 standard (I actually have some of the parts from ANSI). I have also heard numerous DB2 users discuss what is missing from DB2's SQL implementation and what they wished they had to make their job easier. Armed with this information, I have selected features I thought DB2 for OS/390 should consider. With my disclaimers out of the way, here are some enhancements I'd like to see added to DB2 for OS/390.

Domains. DB2 implemented CHECK constraints in DB2 Version 4 and many described this feature, inappropriately I think, as domain processing. When creating a domain for a column, the data type, and if necessary the length, default values, and check constraints are all defined for that domain name. Then, when creating a table, the domain name can be used as the data type in the table definition. This guarantees that all tables using the domain of that column are defined in the same way. Check constraints, on the other hand, are defined only to a single table. If the same column were added to multiple tables, a different check constraint could be defined in each table and DB2 would have no way of verifying that the constraint was accurate. A domain eliminates that problem.

Constraints. The referential constraints defined in DB2 are also somewhat limited. For example, ON DELETE has no option to set a default value as allowed in the SQL standard. In addition, there is no ON UPDATE constraint. The RI constraint should be extended in DB2 to allow the same actions on an update to the primary key as allowed for delete processing. DB2 also does not take advantage of another interesting constraint feature in the standard: controlling when a constraint is checked. The standard allows a choice between immediate and deferred constraint checking. Finally, to accompany the immediate/deferred checking, DB2 needs a new special register, SET CONSTRAINT, to manipulate when checking should occur for an SQL statement.

Cursors. These constitute an extremely interesting SQL programming issue. They are a necessity if you want to process sets of data in a program that has only limited available storage. In addition, cursors are so useful. You can open a cursor, close a cursor, process a cursor forward, and process a cursor forward. Heaven forbid you need the previous row again, the last row in the set, a specific row, or any other type of cursor positioning other than forward. Of course, the SQL standard allows for all of these options and more. It is DB2 for OS/390 that does not implement that support. To be fair, though, I should mention that DB2 for OS/390 Version 6 does have the new data type called ROWID. DB2 supplies a unique row identifier for a column with the data type ROWID at insert. This identifier stays with that row forever, even through a reorganization of the table space. If the ROWID is extracted from the row and saved in the application, a subsequent SQL statement can go directly to that row by specifying the saved identifier. This is a nice step in the right direction, but it is still not what is allowed by the standard.

In the SQL standard, the FETCH statement has the potential to support, as scrolling keywords, NEXT, PRIOR, FIRST, and LAST. In addition, the standard also allows for ABSOLUTE and RELATIVE cursor positioning. With ABSOLUTE positioning, a specific row from the beginning of the set of rows associated with that cursor (or from the end if the absolute value is negative) can be accessed. With RELATIVE positioning, a row some number of rows from the cursors current position can be accessed. Yet another desirable cursor option is the ability to define a cursor as insensitive when writing a program. Insensitive cursors will not see changes made to the table the cursor is opened against within the same transaction. The standard, however, does not have a definition of a sensitive cursor. If INSENSITIVE is not specified, the cursor is indeterminate. The standard does not allow INSENSITIVE and FOR UPDATE specified for the same cursor. They are mutually exclusive.

The extensions to the GROUP BY clause offer another example of some interesting stuff available in DB2 UDB Version 5 that has not yet been added to DB2 UDB for OS/390. ROLLUP and CUBE add OLAP capabilities to DB2's everyday SQL. ROLLUP allows the collection of data at multiple levels in a single dimension. CUBE, on the other hand, allows the collection of data in more than one dimension.

Explain. Although this does not really describe an SQL feature missing from DB2, it does describe a feature that is desperately needed to improve the SQL processing we already have. We need a better Explain. The Explain facility currently in DB2 has not changed since its implementation in Version 1.2 in 1986. DB2 for OS/390 needs to look once again to its younger sibling, DB2 UDB, for an example of how to improve Explain. Making Explain graphical is not the answer. Perhaps DB2 for OS/390 could someday externalize and document how to use some of the hidden explain tables that already exist in DB2 for OS/390. I believe explaining (no pun intended) how to use the hidden explain table and making the information in those tables usable is essential. Most customers already know about them. Some customers are trying to use this information on their own, in some cases guessing at what it means. Knowing about the existence of these tables but not how they are populated or how to interpret the information in them may be worse than having no information available at all.

Summary Standards are wonderful. But individual database vendors, in our case IBM, have a duty to place their own spin on SQL. Characteristics in the database engine need to be addressed. Simple things like "OPTIMIZE FOR n ROWS" may not be part of the standard, but they are necessary to achieve the performance that is needed to make SQL usable in a DB2 environment. Learn about SQL3 and SQL/92 to discover what SQL changes to expect in the next millennium.

Willie Favaro has been a database professional for more than 20 years, the last 14 years primarily with DB2. He has been a software consultant for BMC Software, Inc. and a senior DB2 instructor for IBM. Favaro is the author of numerous articles, a contributor to

several IBM Redbooks, and a regular speaker at regional, national, and international conferences. He can be reached via email at willie_favero@compuserve.com.

The Future of SQL

By Craig S. Mullins

I believe that the future of SQL is bright; it is the present of SQL that I am worried about - but first, some background.

Structured Query Language, better known as SQL (and pronounced "sequel" or "ess-cue-el"), is the de facto standard query language for relational database management systems (RDBMSs). SQL is used not only by DB2, but also by other leading RDBMS products such as Oracle and SQL Server. Indeed, every relational database management system - and many non-relational DBMS products - support SQL as the method for accessing data.

Why is SQL so successful?

Why is SQL pervasive within the realm of relational data access? What benefits are accrued by using SQL rather than another, more computationally complete language?

There are many reasons for SQL's success. Foremost is that SQL is a high-level language that provides a greater degree of abstraction than do procedural languages. Third-generation languages (3GLs), such as COBOL and C, and even fourth-generation languages (4GLs), require that the programmer navigate data structures. Program logic must be coded to proceed record-by-record through the data stores in an order determined by the application programmer or systems analyst. This information is encoded in the high-level language and is difficult to change after it has been programmed.

SQL, on the other hand, is designed to allow the programmer to specify what data is needed. It does not, indeed it cannot, specify how to retrieve it. SQL is coded without embedded data-navigational instructions. The DBMS analyzes the SQL and formulates data-navigational instructions "behind the scenes." These data-navigational instructions are called access paths. Forcing the DBMS to determine the optimal access path to the data removes a heavy burden from the programmer. In addition, the database can have a better understanding of the state of the data it stores; it can thereby produce a more efficient and dynamic access path to the data. The result is that SQL, used properly, provides a more rapid application development and prototyping environment than is available with corresponding high-level languages.

Another feature of SQL is that it is not merely a query language. The same language used to query data is used also to define data structures, control access to the data, and insert, modify, and delete occurrences of the data. This consolidation of functions into a

single language eases communication among different types of users. DBAs, systems programmers, application programmers, systems analysts, systems designers, and end users all speak a common language - namely, SQL. When all the participants in a project are speaking the same language, a synergy is created that can reduce overall system development time.

Arguably, though, the single most important feature of SQL that has solidified its success is its capability to retrieve data easily using English-like syntax. It is much easier to understand a query such as:

```
SELECT LASTNAME  
FROM EMP  
WHERE EMPNO = '000010';
```

than it is to understand pages and pages of COBOL, C, or PL/I source code, let alone the archaic instructions of Assembler. Because SQL programming instructions are easier to understand, they are easier also to learn and maintain - thereby making users and programmers more productive in a shorter period of time.

SQL is, by nature, a flexible creature. It uses a free-form structure that gives the user the ability to develop SQL statements in a way best suited to the given user. Each SQL request is parsed by the DBMS before execution to check for proper syntax and to optimize the request. Therefore, SQL statements do not need to start in any given column and can be strung together on one line or broken apart on several lines. For example, the following SQL statement:

```
SELECT LASTNAME FROM EMP WHERE EMPNO = '000010';
```

is equivalent to the SQL statement previously depicted. Another flexible feature of SQL is that a single request can be formulated in a number of different and functionally equivalent ways. Of course, this feature can also be very confusing to SQL novices. Furthermore, the flexibility of SQL is not always desirable because different but equivalent SQL formulations can produce sharply differing results in performance.

Finally, one of the greatest benefits derived from using SQL is its ability to operate on sets of data with a single line of code. Multiple rows can be retrieved, modified, or removed in one fell swoop using a single SQL statement.

This provides the SQL developer with great power, but it is this very feature which also limits the overall functionality of SQL. Without the ability to loop or step through multiple rows one at a time, certain tasks are impossible to accomplish using only SQL. Of course, as more and more functionality is added to SQL, the number of tasks that can be coded using SQL alone is increasing.

The origins of SQL The original version of SQL was called SEQUEL (Structured English QUery Language) and it was designed by IBM research in San Jose, California in the early 1970s. The first commercial implementation of SQL occurred in 1979 by Oracle Corporation (then known as Relational Software, Inc.).

In October 1986, ANSI approved a basic version of SQL as the official standard. Most SQL implementations since have included many non-standard extensions to the ANSI standard. ANSI updated the standard in 1989 to include data integrity enhancements and again, more substantially in 1992, to a new standard commonly known as SQL2. Further enhancements have been made to the SQL standard to include support for a Call Level Interface (CLI) and stored procedures.

The next iteration of the SQL standard, commonly known as SQL3, is in the works and will extend SQL to provide object/relational capabilities. There is no clear consensus as to when SQL3 will be fully agreed upon and delivered.

The threat of the present If you believe some industry pundits, reliance on SQL for relational data access is on the wane. New Internet technologies such as Java and XML are being touted as the "next big thing" for accessing databases.

The promise of these new technologies is intriguing. Take XML, for example. If you believe everything you read, then XML is going to solve all of our interoperability problems, completely replace SQL, and possibly even deliver peace on earth. Okay, that last one is an exaggeration, but you get the point.

In actuality, XML stands for eXtensible Markup Language. The need for extensibility, structure, and validation is the basis for the evolution of the Web towards XML. XML, like HTML, is based upon SGML (Standard Generalized Markup Language) which allows documents to be self-describing, through the specification of tag sets and the structural relationships between the tags. HTML is a small, specifically-defined set of tags and attributes, enabling users to bypass the self-describing aspect for a document. XML, on the other hand, retains the key SGML advantage of self-description, while avoiding the complexity of full-blown SGML.

But XML does not do what SQL does and, hence, cannot replace it. And the same can be said for Java. Willie and I discussed Java in this column in the last issue of IDUG Solutions Journal, so I will not rehash the subject here again.

Suffice it to say: Java cannot and will not replace SQL either. The API to relational databases remains SQL - any other data access or presentation technology necessarily must communicate with relational data by means of SQL. XML and Java can provide benefit to organizations by extending the capability of the Internet to include data access and modification through SQL, and this is goodness. But don't be fooled into believing they (or anything else) will be able to completely replace SQL any time soon.

Even so, object-oriented programmers tend to resist using SQL. The set-based nature of SQL is not simple to master and is anathema to the OO (object oriented) techniques practiced by Java developers. All too often the manner in which data is accessed is not planned out and designed in a thoughtful manner. In fact, sometimes it is not thought of at all until performance suffers.

Object orientation is indeed a political nightmare for those schooled in relational tenets. Proponents of OO are almost always the enemy of those who practice sound data management policy. All too often organizations experience political struggles between the OO programming team and the data resource management group. The OO crowd espouses programs and components as the center of the universe; the data crowd adheres to the tenets of normalized, shared data with the RDBMS as the center of the universe.

Thanks to all of the hype, the OO crowd tends to win many of these battles, but the war will eventually be won by data-centered thinking. The notion of data normalization and shared databases to reduce redundancy provides too many benefits in the long run to be abandoned. As the focus blurs away from data management and sound relational practices, data quality will deteriorate and productivity will decline. It is then that a new era of SQL vitality will arise.

The glow of the future SQL remains a growing and very adaptable language. Depending on the version and flavor of DB2 you are using, new functionality such as CASE statements, outer joins, and nested table expressions have increased the number of tasks we can perform using SQL alone. But these features are not the end of how SQL will adapt to change.

Recursive SQL, available in DB2 Universal Database Server in Version 5, and soon to be ported to other members of the DB2 Family, further expands the functionality of SQL. Recursion enables a table expression to refer to itself. With recursive SQL, even more tasks become possible using only SQL without embedding it in a 3GL or 4GL. To help you understand recursion better, think of the following metaphor. Consider what happens when a camera films someone watching himself on live TV. You can see the person multiple times in the picture because it is recursive. It is possible to traverse hierarchies such as a bill of materials or an organization chart using a single recursive query.

Traditionally, DBMS products stored data and nothing else. But all the major RDBMS products of today support (or are moving to support) procedural logic in the form of triggers, functions, and stored procedures, sometimes referred to as Server Code Objects (or SCOs for short). The ability to store procedural logic in the database is increasingly common because it enhances performance of client/server applications, eases security implementation, promotes reusability, and makes databases active.

DB2 for OS/390 Version 6 will add features such as triggers, user-defined functions, and more built-in functions that will enable active databases with increased data integrity. An

active database can take actions automatically and implicitly by virtue of an event occurring. For example, a database trigger can be specified on a table that automatically calculates derived data whenever any value it is derived from changes. The trigger can even INSERT the derived data into another column automatically, thereby preserving data quality and integrity. And all of this happens just because a single SQL statement was issued to change data. So even more tasks will be able to be accomplished using only SQL.

DB2 Universal Database, and recently DB2 for OS/390, provide SQL CLI support. The SQL CLI (Call Level Interface) is an alternative binding style for executing SQL statements. Instead of embedding SQL in an application program, you use routines that allocate and deallocate resources, control connections to the database, execute SQL statements using similar mechanisms to dynamic SQL, obtain diagnostic information, control transaction termination, and obtain information about the implementation. Basically, the CLI issues SQL statements against the database using procedure calls instead of via direct embedded SQL statements. An SQL CLI is useful for enabling SQL access to more languages and programmers than before. This is goodness.

The long term future is even brighter And the long-term future of SQL is even brighter. More and more features will be available using just SQL. Large object support (available in UDB today is OS/390 in Version 6) enables SQL statements to access very large text and multimedia objects. As these features become available more and more applications will use them in novel ways to gain a competitive advantage.

Additionally, SQL will eventually be extended to incorporate new and exciting technology such as fuzzy logic and temporal data. These features are further in the future, but promise to provide exciting new capabilities.

The addition of temporal qualities to relational databases, and thus SQL, will make your databases time-sensitive. Temporal extensions will enable you to query the database not just on its current state, but on its past state as well. For example, temporal capabilities could help you to answer questions when data changes over time, such as:

1. The actual price of a particular product on a given date
2. The commission rate assigned to a sales rep on a given date
3. Monthly revenue changes
4. The status of a project at a particular moment in the past
5. Inventory at any given point in time
6. Sales on the day before Christmas over the past ten years.

And these are just a few of the examples. I'm sure you can think of more using your own systems and needs. There are many possible ways to extend SQL to support temporal databases. Here is one possible example of a simple temporal SQL query:

```

SELECT      PROD_ID
FROM        PROD
WHERE       PRICE > 100,00
FROMDATE    ?1997-03-01?
TODATE      ?1998-09-30?;

```

Of course, you will have to assign a time granularity to the PROD table before such a query could be issued. And there are many different time granularities that could be assigned: YEAR, MONTH, WEEK, HOUR, etc. You get the idea!

Fuzzy logic is also a discipline that will eventually find its way into relational database technology and SQL. Fuzzy logic relies on Eastern philosophy more than Western. Its simple theory is that everything is true, just to different degrees. In the Western world we are used to absolutes: YES and NO; TRUE and FALSE; ON and OFF; 1 and 0. Eastern philosophy is less rigid, teaching that all things are shades of gray, instead of black or white.

Why is fuzzy logic useful?

Consider the following:

1. When eating an apple, when does it stop being an apple and become an apple core instead?
2. If you wish to control room temperature to be 70 degrees F, do you really want to tell the air conditioner to shut off immediately when the temperature is 60.9 and turn back on at 70.1?
3. When querying your data looking for the products earning highest amount of revenue, what is the cut off point inclusion? If it is \$6 million, do you really want to ignore the products that earned \$5.99 million?

Fuzzy logic helps to alleviate these problems by imposing degrees of truth on everything. A bank might want to identify bank accounts that are a drag on earnings. Someone who makes many transactions but maintains a low balance is probably not earning the bank a lot of revenue, so it might wish to levy fees on these accounts. A fuzzy SQL query to help identify these accounts might look something like this:

```

SELECT      NAME, ACCT_NO
FROM        ACCT
WHERE       BALANCE IS LOW
AND         ACTIVITY IS HIGH;

```

Now wouldn't that be simpler than trying to set an arbitrary cut off for what constitutes a low BALANCE or high ACTIVITY? Of course, you will need to set up the fuzzy vocabulary up front. This usually involves defining terms such as LOW and HIGH, FEW and MANY, or YOUNG and OLD. Additional second order terms can be defined to

augment the first order fuzzy terms, such as USUALLY, ALWAYS, and VERY. So, we could augment our query above to add second order terms such as:

```
SELECT      NAME, ACCT_NO
FROM        ACCT
WHERE       BALANCE IS USUALLY LOW
AND         ACTIVITY IS VERY HIGH;
```

In general, fuzzy logic can help to improve the results of SQL and will enable us to do even more with a single SQL statement.

Synopsis

The future of SQL is indeed bright, but challengers lurk around every corner. What should you do as OO, Java, and XML encroach on the world of SQL and threaten data integrity? Well, use common sense. Understand any new technology before implementing it at your company. Know what it can and can't do by practicing with the technology - not by listening to the hype. Be sure to maintain and develop expertise in SQL in your organization. Be sure to know how the new technologies interact with SQL (JDBC, JSQL) and implement them appropriately. And keep on promoting SQL and data resource management. I mean, deep down inside, we know these must be the basis for our IT infrastructure or that infrastructure will crumble around us.

Craig S. Mullins is vice president of marketing and operations for the database tools division of PLATINUM technology, inc. He is also the author of the popular book, DB2 Developers Guide, now in its third edition; the book includes tips, techniques, and guidelines for DB2 through Version 5. He can be reached via email at cmullins@platinum.com.

[About IDUG](#) | [Conferences](#) | [DB2 Resources](#) |
[Regional User Groups](#) | [Solutions Journal](#) | [Vendor Directory](#)

[Home](#) | [Contact Us](#)

International DB2 Users Group
401 N. Michigan Ave.
Chicago, IL 60611
(312) 644-6610