FUTURE TRENDS IN DATA BASE SYSTEMS

Michael Stonebraker

Department of Electrical Engineering
and Computer Sciences
University of California
Berkeley, CA 94720

Abstract

This paper discusses the likely evolution of commercial data managers over the next several years. Topics to be covered include:

Why SQL has become an intergalactic standard.
Who will benefit from SQL standardization.
Why the current SQL standard has no chance of lasting.
Why all data base systems will be distributed soon.
What new technologies are likely to be commercialized.
Why vendor independence may be achievable.

The objective of this paper is to present the author's vision of the future. As with all papers of this sort, this vision is likely to be controversial. Moreover, the reader will detect many of the author's biases and is advised to react with the appropriate discounting.

1. INTRODUCTION

This paper is written from the perspective of a researcher who has had some opportunities to observe the commercial marketplace over the last several years. From this exposure I would like to comment on some of the current trends in this marketplace. In addition, I would also like to speculate on some of the likely trends in the marketplace over the next several years.

Due to the position of IBM, the importance of SQL in this evolution cannot be discounted. Others have pointed out the numerous serious flaws in SQL [DATE85]. Consequently, this paper will not discuss the technical problems in the language; rather, it will focus on the impact of SQL standardization. It will briefly discuss why the standard came about. However, more importantly, it will make a case that very few organizations will benefit directly from the standardization effort. Consequently, considerable effort is being spent to construct a standard, and organizations may not reap the benefits which they anticipate.

Then, the paper will turn to the current collection of prototype data base systems that are being constructed in research labs around the world. In particular, the characteristics that make these systems noticeably better than current commercial systems are identified. Unless some dramatic slowdown in technology transfer takes place, these ideas will quickly move from prototypes into commercial systems. The paper then argues that this movement of new features will spell the doom of a standardized version of SQL.

The paper then considers important technological trends. The most significant one appears to be distributed data bases, and the paper turns to this phenomenon and explains why all commercial systems are likely to become distributed data managers. It also comments on what important problems remain to be solved to facilitate ''industrial strength'' distributed data base systems.

In addition, I will comment on other technological and research trends which are likely to be significant in future data base managers. These comments are in the areas of data base machines, high transaction rate systems, main memory data base systems, and new storage devices.

Lastly, the paper will address a very serious problem which most users of data base systems struggle with. Namely, they are constrained to coping with ‘‘the sins of the past,’’ namely a large amount of application code written in COBOL or other third generation languages which accesses previous generation data managers (such as IMS and other ‘‘tired technology’’ systems as well as ‘‘home-brew’’ data managers). This accumulated baggage from the past is usually an impediment to taking advantage of future hardware and software possibilities. Consequently, the paper closes with a step-by-step procedure by which any user can migrate over a period of years into an environment where he is not constrained to the iron of any particular hardware vendor. At the end of this paper, I will revisit the issue of standardization in light of the proposed migration path and indicate what sort of standardization activity might assist this process.

2. WHY SQL

About 1984 the tom-toms started beating very loudly for SQL. The message was conveyed first by hardware vendors (iron mongers) in search of a data manager. In brief the message said ‘‘IBM will make a big deal of DB 2 and SQL. I want to be compatible with IBM.’’ A similar message was conveyed by so-called value-added resellers (VARS) who said ‘‘I want application code that I write to run both on your data manager and on DB 2’’. Discussions with VARS or iron mongers concerning **exactly** what they meant by SQL and exactly what they wanted in terms of compatibility usually evoked an answer of ‘‘I don’t know’’. Hence the early tom-toms were being beaten by people who were not exactly sure of what they wanted.

Later, the tom-tom pounding was picked up by representatives of large users of data base services. Usually, the message they delivered was:

> ‘‘I need to run my applications on IBM iron and on the iron of vendors X, Y, and Z. I plan to move to DB 2 as my IBM system and I want to ensure that the DB 2 applications I write can be moved to the iron of these other vendors. SQL is the mechanism that will allow me to achieve this objective.’’

The vendors of commercial data managers are not stupid. They listen to the tom-toms and react appropriately. Consequently, all vendors of data base systems have put in place plans to support SQL. Moreover, all other query languages (e.g. QUEL, Datatrieve, etc.), regardless of their intellectual appeal, will become ‘‘sunset’’ interfaces, i.e. they are likely to slowly fade away and become a thing of the past. I wish to make two other points in a bit more detail.

First, there is less interest in standardized SQL outside the USA. In fact, offshore DBMS users seem **much** more inclined to use fourth generation languages and thereby are less sensitive to the SQL issue. This point is further discussed in the next section.

A second point is that data base system vendors were immediately divided into two camps; those that already had SQL and those that had to spend a large number of man-years to retrofit SQL into their systems. Clearly, this presented a significant advantage to vendors in the first category, and helped reshape the competitive positions of various DBMS suppliers. In addition, one interesting measure of vendor responsiveness is the date of SQL introduction by vendors in category 2. Responsive vendors had SQL in 1986, others followed at later times.

3. WHO WILL BENEFIT FROM STANDARD SQL

We turn first to a definition of three possible levels of SQL standardization that might make sense and indicate the level at which ANSI activity has taken place. Then, we consider the classes of users who might benefit from the current ANSI standardization.

3.1. Levels of Standardization

There are three possible ways of interpreting SQL:

1) SQL, the data definition language
2) SQL, the query language
3) SQL, the embedding in a host language

Using the first interpretation, one would standardize CREATE, DROP, ALTER and any other commands that involve storage management and schema creation or modification. This portion of SQL is used by data base administrators (DBAs) and standardization of SQL in this area might benefit this class of persons.

Using the second interpretation, one would standardize SQL, the query language. This would entail adding SELECT, UPDATE, INSERT, and DELETE to the list of standard commands. In this way an end user of standard SQL could expect his SQL commands to run on any DBMS supporting the standard.

The third interpretation would standardize SQL as it is executed from a host language. This interface includes the DECLARE CURSOR, OPEN CURSOR, FETCH, UPDATE, and CLOSE CURSOR commands. In this way, a programmer could expect his host language program to work across multiple DBMSs that adhered to the standard.

Loosely speaking we can call these three levels:

level 1: the DBA level
level 2: the end user level
level 3: the programmer level

It should be clearly noted that the ongoing ANSI standardization effort is at level 3. However, vendors often mean something else by standard SQL. For example, Sybase has chosen only to implement level 2 SQL, while INGRES, Oracle and DB 2 all implement level 3. Consequently, the purchaser of a data base system should carefully inquire as to what ‘‘SQL support’’ really means when he is contemplating an SQL-based data manager.

The last point to note is that level 3 ANSI SQL, DB 2 and SQL/DS are **all** slightly different versions of SQL. Hence, the concept ‘‘standard SQL’’ must be carefully tempered to reflect the fact that all level 3 SQL systems are different in at least minor ways. This corresponds closely to the current UNIX marketplace where the UNIXes offered by various vendors also differ in minor ways.

3.2. Who Will Benefit From SQL Standardization

3.2.1. Introduction

As mentioned earlier, ANSI has standardized a level 3 SQL interface. Such standardization might be of benefit to:

data base administrators
end users
application programmers
vendors of 4th generation languages
vendors of distributed data base systems

In the next several subsections we indicate which of these groups are likely to benefit from SQL standardization.

3.2.2. Data Base Administrators

Clearly, a level 3 standard includes a level 1 standard as a subset. Consequently, a DBA who garners experience with schema definition on one data manager will be able to leverage this experience when designing data bases for a second DBMS. Hence, a DBA should benefit from the ANSI standardization effort. However, there are several caveats that must be noted.

First, most relational DBMSs have nearly the same collection of level 1 capabilities. Hence, except for minor syntactic variations, level 1 was already effectively standardized and there was no need for ANSI machinery in this area.

Second, differences exist in the storage of data dictionary information (the system catalogs). A DBA usually wishes to query the system catalogs to retrieve schema information. The current ANSI standard does not address this aspect, and each standard SQL system will have a different representation for the dictionary. Lastly, differences exist in the exact form of indexes and the view support facilities, which may influence the details of data base design. The current SQL standard does not address these differences, and they limit the leverage a DBA can expect.

In summary, all current relational systems are standard in that they allow a user to construct and index relations consisting of named columns, usually with nearly the same syntax. Consequently, data base design methodologies appropriate to one system are nearly guaranteed to be appropriate for other systems. At this level, current relational systems are already standard, and nothing additional need be done.

There are also differences between the various systems in the areas of types of indices, storage of system catalogs, and view support. These aspects are not yet addressed by the ANSI standardization effort. As a result, I don’t perceive that DBAs will benefit greatly from the ANSI effort, relative to what they will automatically gain just by using relational systems.

3.2.3. End Users

Since the ANSI standardization effort includes a level 2 SQL facility as a subset, one could claim that end users will benefit because they can learn the SQL for one data base system and then be able to transfer that knowledge to other standard data base systems. However, this claim is seriously flawed.

First, end users are not going to use SQL. Human factors studies and early usage of relational systems has shown clearly that end users will use customized interfaces appropriate to their application, usually of the ‘‘fill in the form’’ variety [ROWE85]. Such customized interfaces will be written by programmers. Consequently, end users will not benefit from SQL standardization because they won’t use the language.

Even if end users **did** use SQL, they are still subject to widely divergent presentation services. For example EASE/SQL from ORACLE is very different from IBMs QMF, yet both allow a human to interactively construct and execute SQL commands. These differences will limit the leverage obtainable.

3.2.4. Programmers

One could argue that programmers will benefit from standardization of the level 3 SQL interface because the programs that they write for one SQL system will run on another standard DBMS. Moreover, once they learn the define-open-fetch cursor paradigm for one system, they will immediately be able to write programs for another DBMS. This argument is very seriously flawed.

First, this argument only applies to vendors who have chosen to support the level 3 standard. It clearly does not apply to Sybase, and any other vendor who has chosen to implement the standard only at level 2.

Second, and perhaps of supreme importance, programmers are not going to use the level 3 interface. Most DBMS vendors offer so-called fourth generation languages (4GLs). Such products include Natural, Ramis, Adds-online, Ideal, INGRES/ABF, and SQL-forms. In general these products allow a programmer to:

define screens
define operations to be executed as a result of user input into screens
interactively call subsystems such as the report writer

Application programmers familiar both with 4GL products and with the level 3 style application programming interface report that there is a factor of 3-10 in leverage from using a 4GL. Consequently, a client of DBMS technology is generally well advised to use a 4GL wherever possible and to foresake the level 3 programming interface. This advice is nearly universally true in business data processing applications. In engineering applications, on the other hand, 4GLs may be less advantageous.

In summary, application programmers are going to use 4GLs because of their software development leverage, and not the level 3 SQL interface. Moreover, every 4GL is totally unique, and there is no standardization in sight. The only company who could drive a 4GL standardization activity would be IBM. However, most data base professionals do not believe that IBM has a 4GL (not withstanding IBMs marketing of CSP as a 4GL). Consequently, it will be several years before there is any possible standardization in this area.

On the other hand, suppose a user decides **not** to use a 4GL because he is concerned about portability or alleged poor performance in older products. His applications **still** require screen definition facilities, report specifications, and graph specifications. These facilities are unique to each vendor and not addressed in any way in the SQL standard. To move from one standard DBMS to another, one must relearn the facilities in each of these areas. As a result, only perhaps 10-20 percent of the total specification system is covered by ANSI SQL, and the remainder must be relearned for each system. To avoid this retraining, a user must either write and port his own facilities in these areas, an obviously distasteful strategy, or he must depend on some specific vendor to provide a standard collection of facilities on all platforms important to him. SQL is clearly of no assistance in this dimension.

3.2.5. 4GL Vendors

One could argue that vendors of 4GL products will benefit from standardization because they will be able to easily move their products onto a variety of different data managers. Although this argument has merits, it is also somewhat flawed.

First, as noted before there is no standard for information in the system catalogs. All 4GLs must read and write information in the dictionary, and this will be code unique to each target DBMS. Second, I have asked a variety of 4GL users which target DBMSs are of greatest interest to them. They typically respond with the following three priority requests:

1) IMS
2) DB 2
3) some ‘‘home-brew’’ data manager

To satisfy these requests, a 4GL vendor must develop complete custom interfaces for systems 1 and 3. Only the interface for system 2 would be assisted by standardization. Third, most DBMS vendors have (or are developing) capabilities which superset the ANSI standard. The reasons for this are discussed at length in the next section. A 4GL vendor who wishes to interface to such an extended DBMS has two choices. First, he can restrain his use to the subset which is standard. Consequently, there will be underlying DBMS capabilities which he does not exploit, and he will be at a relative disadvantage compared to 4GLs (such as the one from the DBMS vendor in question) which take full advantage of underlying facilities. The second choice is to do non standard extensions for each target DBMS. Both choices are unattractive. Lastly, any 4GL that is marketed by a hardware vendor is unlikely to take advantage of any opportunity for portability provided by SQL because such a vendor will likely resist providing a migration path for applications off of his particular hardware.

Hence, standardization on SQL clearly helps a 4GL vendor who wishes to make his code portable. However, not all of them will wish to, and there is substantial effort in the areas of system catalogs, non-standard extensions and coupling to non SQL data bases which is required to make this portability occur.

3.2.6. Vendors of Heterogeneous Distributed DBMSs

One could argue that distributed data base systems should have so-called ‘‘open architectures’’ and be able to manage data that is stored in local data managers written by various vendors. Hence, vendors of open architecture products might benefit from SQL standardization, since foreign local data managers will be easier to interface to.

Basically, a distributed DBMS vendor sees the world in exactly the same way as a vendor of a 4GL. Hence, the above section applies exactly to this class of user.

3.2.7. Summary

We can summarize the possible groups who might benefit from standardization of SQL as follows:

DBAs	This group will benefit from the fact that all relational systems use essentially the same data definition language definition language, regardless of the query language supported.
end users	This group will not use SQL and will be unaffected by standardization.
programmers	This group will primarily use 4GLs and consequently will be unaffected by standardization.
4GL vendors	This group may benefit from standardization if they choose to try to interface to a variety of data managers. However, they still have a lot of work to do, and some of them will resist exploiting this portability.
Distributed DBMS vendors	They are in the same position as 4GL vendors.

One draws the unmistakable conclusion that the large amount of effort that is being poured into SQL standardization may **not** pay handsome dividends. A user of DBMS technology will only benefit if he chooses 4GL and distributed DBMS products from vendors committed to open architectures. He will then benefit indirectly from the efforts of these vendors to make their products run on a variety of SQL engines.

However, the situation is much worse than has been portrayed so far because standard SQL, as currently defined, stands **no chance** of lasting more than a few years. The next section shows why SQL will not ‘‘stick’’.

4. WHY STANDARD SQL IS DOOMED

4.1. Introduction

All relational DBMSs were designed to solve the needs of business data processing applications. Specifically, they were designed to rectify the disadvantages of earlier hierarchical and network data base systems. Most DBMS professionals agree that they have succeeded at this task admirably. However, equally well understood are the needs of other users of DBMS technology in the areas of spatial data, CAD data, documents, etc. There is a renaissance of research activity building ‘‘next generation prototypes’’ which attempt to rectify the drawbacks of current relational systems. Consequently, one could say that there are three generations of systems:

generation 1: Hierarchical and Network Systems
generation 2: Relational Systems
generation 3: Post-relational Systems

The following research prototypes are all examples of prototype post-relational systems:

EXODUS [CARE86]
GEM [TSUR84]
IRIS [FISH87]
NF2 [DADA86]
ORION [BANE87]
POSTGRES [STON86a]
STARBURST [LIND87]

Although they are exploiting various ideas, one can make the following observation:

> Essentially all ideas that are being exploited by the above prototype systems can be added to current commercial relational data base systems by extending or reworking their capabilities.

Hence, it is obvious that aggressive vendors will quickly extend their current SQL engines with relational versions of the successful capabilities of these prototypes. In this way, vendors will create systems that are substantial supersets of SQL. Since each vendor will do unique extensions, they will all be incompatible. Moreover, IBM will be the slowest to provide extensions to DB 2.

These extensions will solve problems that are so important to large classes of users that they will gladly use the extended capabilities. In this way, any application that a user writes for vendor A’s system will not run without substantial maintenance on vendor B’s system and vica-versa. This will ensure that application portability will not be achieved through SQL.

The rest of this section indicates two areas in which seductive next generation capabilities are expected.

4.2. Management of Knowledge Bases

I wish to discuss knowledge bases first with regard to expert systems and then with regard to conventional business data processing. I conclude this subsection with a discussion of why it is essential that knowledge management become a data base service.

Expert systems typically use **rules** to embody the knowledge of an expert, and I will use interchangeably the concept of a knowledge base and a rule base. One important application area of expert systems is in surveillance systems. The object to be monitored could be a physical object, such as manufacturing line, an oil refinery, or a stock market. It might also be an area of real estate, such as a battlefield. In either case, an expert system is desired which watches the state of the object and alerts a human if ‘‘abnormal’’ events occur. Such surveillance applications fundamentally involve the data base for the monitored object. Moreover, abnormal events are typically defined by a rule base, developed by consultation with human experts. Hence, such applications require a large data base (the monitored object) and a large set of rules (the events to watch for).

In conventional business data processing applications there is also substantial use for a rule base. For example, consider the processing of purchase orders. The following rules might well apply in a typical company:

All POs over $100 must be signed by a manager
All POs over $1000 must be signed by the president
All POs for computer equipment must be signed by the MIS director
All POs for consultants must have an analysis of need attached

Similar rule systems control allocation of office furniture (e.g, only vice presidents can have wood desks), commission plans for salespersons (e.g, commission is paid only on non discounted POs), vacation accrual, etc.

The possible techniques available to support such composite rule and data base applications are:

1) Put the rules in an application program and the data in a data base.
2) Put the rules and the data in an expert system shell.
3) Put the rules in an expert system shell and the data in a data base.
4) Put both the rules and the data in a composite data/rule base.

I now argue that only option 4 makes any long term technical sense. Option 1 is widely used by business data processing applications to implement rules systems such as our purchase order example. The disadvantage of this approach is that the rules are buried in the application program and are thereby difficult to understand and tedious to change as business conditions evolve. Moreover, if a new program is written to interact with the data base, it must be coded to enforce the rules in a fashion consistent with the previously written application programs. The possibility for error is consequently high. In summary, when rules are embedded in an application program, they are hard to code, hard to change, and hard to enforce in a consistent fashion.

The second alternative is to put both the data and the rules in an expert system environment such as Prolog, OPS5, KEE, ART, or S1. The problem with this approach is that these systems, without exception, assume that facts available to their rule engines are resident in main memory. It is simply not practical to put a large data base into virtual memory. Even if this were possible, such a data base would have no transaction support and would not be sharable by multiple users. In short, current expert system shells do not include data base support, and option 2 is simply infeasible.

Option 3 is advocated by the vendors of expert system shells and is termed **loose coupling.** In this approach rules are stored in main memory in the shell environment which contains an inference engine. Whenever necessary, this program will run queries against a data base to gather any needed extra information. Hence, rules are stored in a rule manager and data in a separate data manager. A layer of ‘‘glue’’ is then used to couple these two subsystems together. An example of this architecture is KEE/Connection from Intellicorp.

Unfortunately loose coupling will fail miserably on a wide variety of problems, and a simple example will illustrate the situation. Suppose one wanted to monitor a single data item in a data base, i.e, whenever the data item changes in the data base, it should change on the screen of a monitoring human. Many investment banking and brokerage houses are building automated trading systems that are much more sophisticated versions of this simplistic example.

The expert system can run a query to fetch the data item in question. However, it will become quickly out of date and must be fetched anew. This repeated querying of the data base will needlessly consume resources and will always result in the screen being some amount of time out of date. Loose coupling will fail badly in environments where the expert system cannot fetch a small, static portion of the data base on which to operate. Most problems I can think of fail this ‘‘litmus test’’.

The fourth alternative is to have a single data/rule system to manage both rules and data, i.e. to implement **tight coupling.** Such a system must be **active** in that it must perform asynchronous operations to enforce the rules. This is in contrast to current commercial DBMS which are **passive** in that they respond to user’s requests but have no concept of independent action.

An active system can tag the data item being watched by our simplistic application and send a message to an application program whenever the data item changes. This will be an efficient solution to our monitoring example. Such a data manager will automatically support sharing of rules, the ability to add and drop rules on the fly, and the ability to query the rule set.

Tight coupling can be achieved in a variety of ways. Extensions to the view definition facility can be utilized as well as extensions to the SQL language directly [STON87]. In the case that the resulting queries are recursive, processing algorithms have been investigated in [ULLM85, IOAN87, ROSE86, BANC86].

Without a doubt many of these ideas will lead to commercial implementations, and I expect that many will be successful. The bottom line is that rules and inference will almost certainly move into data base systems over the next few years. It appears feasible to support this feature by supersetting the query language, and this will certainly be the method of choice for SQL vendors.

4.3. Object Management

If I hear the phrase ‘‘everything is an object’’ once more, I think I will scream. Peter Buneman expressed this frustration most concisely in [BUNE86]: ‘‘Object-oriented is a semantically overloaded term’’. Moreover, in a panel discussion on Object-Oriented Data Bases (OODBs) at VLDB/87, six panelists managed to disagree completely on exactly what an OODB might be.

In any case, there are a class of applications which must manage data that does not fit the standard business data processing world where objects are character strings, integers, floating point numbers and maybe date, time, money and packed decimal. Non-business environments must manage data consisting of documents, three dimensional spatial objects, bitmaps corresponding to pictures, icons for graphical objects, vectors of observations, arrays of scientific data, complex numbers, etc.

In general these applications are badly served by current data base systems, regardless of what data model is supported. This point is discussed in detail in [STON83], and we present here only a very simple example. Suppose a user wishes to store the layout of Manhattan , i.e. a data set consisting of two-dimensional rectangular boxes. Obviously, a box can be represented by the coordinates of its two corner points (X1,Y1) and (X2, Y2). Consequently, a reasonable schema for this data is to construct a BOX relation as follows:

BOX (id, X1, Y1, X2, Y2)

The simplist possible query in this environment is to place a template over this spatial data base and ask for all boxes that are visible in the viewing region. If this region corresponds to the unit square, i.e. the box from (0,0) to (1,1), then the most efficient representation of the above query in SQL is:

```
select *
from BOX
where X1 <= 1 and
      X2 >= 0 and
      Y1 <= 1 and
      Y2 >= 0
```

Moreover, it generally takes a few tries before a skilled SQL user reaches this representation. Consequently, even trivial queries are hard to program. In addition, no matter what collection of B-tree or hash indexes are constructed on any key or collections of keys, this query will require the run-time execution engine to examine, on the average, half of the index records in some index. If there are 1,000,000 boxes, 500,000 index records will be inspected by an average query. This will ensure bad performance even on a very large machine.

In summary the box application is poorly served on existing relational DBMSs because simple queries are difficult to construct in SQL and they execute with bad performance. To support the box environment, a relational DBMS must:

1) support ‘‘box’’ as a data type. In this way, the BOX relation can have two fields as follows:

```
BOX (id, description)
```

2) Support && as an SQL operator meaning ‘‘overlaps’’. In this way, the query can be expressed as:

```
select *
from BOX
where description && ‘‘(0,0), (1,1)’’
```

3) Support a spatial access method such as R-trees [GUTM84] or K-D-B trees [ROBI81]. This will ensure that the above extended SQL command can be efficiently processed.

In addition, examples can be easily constructed which emphasize the need for multiple inheritance of data and operators, efficient storage of very large objects, objects which are composed of other objects, etc. Proposals addressing various of these ideas are contained in [STON86b, CARE86, BANE87, FISH87, LIND87], and these should move into commercial systems in the near future. The aggressive vendors will be include such capabilities as extensions to SQL.

4.4. Summary

ANSI is currently preparing a draft of SQL 2, its proposed future extension to the current SQL standard. However, it contains **no** capabilities in the areas of knowledge management and object management. Since these capabilities are perceived to be **extremely** useful in a wide variety of situations, aggressive vendors will move ahead in these areas with vendor-specific capabilities. As a result SQL 2 will contain only a subset of available commercial functions. In a time of rapid technological change, the standard will substantially lag the industry leaders and will be doomed to instantaneous technological obsolescence.

To clearly see the reason for this dismal state of affairs one need only look at the philosophy of standardization that is being pursued. There are two successful models, the **resolution** model and the **beacon** model. In the beacon model one assembles the vendors of existing similar but not quite compatible products in a committee with interested users and charges them with resolving their differences by a political negotiation. This model of political resolution by a large committee works well when:

1) there are many implementations of the object being standardized
2) dramatic changes are not happening in the object being standardized
3) resolution of differences is a political problem

The ongoing standardization efforts in Cobol and Fortran clearly fit into the resolution model.

On the other hand, Ada is a good example of the beacon model. Here a new standard was invented with no commercial implementations preceding it. In this case, DOD wisely generated the standard by charging several small teams with designing languages and then picked the best one. In this case, design of the standard was accomplished by a small team of very gifted language designers decoupled from any political process. The beacon model works very well when:

1) there are no implementations of the object being standardized
2) dramatic changes are contemplated
3) a small team of technical experts does the actual design

We can now examine SQL standardization in this light. Clearly, the previous activity (which we call SQL-1) is an example of the resolution model. Moreover, the process has converged a collection of slightly incompatible versions of SQL to a political resolution. However, SQL-2 is a proposal to extend the language onto virgin turf, i.e. to include capabilities which no vendors currently have implementations for. Moreover, relational DBMSs are an area where dramatic technical change is happening. Hence, now capabilities would be best worked on by a small team of experts, and not by a large group of politicians.

In summary, there are two defendable choices open to ANSI at the current time. First, they could follow the resolution model. In this case, they have accomplished their initial objective of coalescing the initial versions of SQL. They should now adjourn the committee for a couple of years while the aggressive vendors do substantial supersets. Later they should reconvene the committee to resolve the differences by political negotiation. On the other hand, if they choose the beacon model, they should subcontract two or more small teams of experts to do proposals and then pick the best one. The problem with ANSI SQL is that they did SQL-1 according to the resolution model. Now with no change in structure, they are trying to switch to the beacon model. As a result, I feel they are guaranteed to fail.

5. DISTRIBUTED DATA BASES

5.1. Why Distributed DBMSs

There is considerable confusion in the marketplace concerning the definition of a distributed DBMS. At the very least it must provide a ‘‘seamless’’ interface to data that is stored on multiple computer systems. For example, if EMP is stored on a machine in London and DEPT is placed on a machine in Hong Kong, then it must be possible to join these relations without explicitly logging on to both sites and assembling needed data manually at some processing location. Instead one would want the notion of ‘‘location transparency’’ whereby one could simply state the following SQL:

```
select name
from EMP
where dept in
      select dname
      from DEPT
```

where floor = 1

There are several vendors who are marketing software systems as distributed data bases which cannot run the above query but instead provide only remote access to data at a single site or provide only a micro-to-mainframe connection. Such systems are **NOT** distributed DBMSs and the marketing hype on the part of such vendors should be immediately discounted.

Moreover, location transparency can be provided either by:

a network file system (NFS) or
a distributed DBMS

A user should **very carefully** check which technique is being using by any vendor who claims to sell a distributed data base system. Consider a user in San Francisco who is interacting with the above EMP and DEPT relations in London and Hong Kong. To find the names of employees on the first floor using an NFS solution, both relations will be paged over the network and the join accomplished in San Francisco. Using a distributed DBMS a heuristic optimizer will choose an intelligent accessing strategy and probably choose to move the the first-floor departments to London, perform the join there, and then move the end result to San Francisco. This strategy will generally be orders of magnitude faster than an NFS strategy. As Bill Joy once said:

think remote procedures not remote data

Put differently, one should send the queries to the data and not bring the data to the query.

A lazy vendor can quickly implement an NFS-based distributed data manager that will offer bad performance. Distributed DBMSs with heuristic optimizers are considerably more work, but offer much better performance. A client of distributed data managers must develop the sophistication to be able to distinguish the lazy vendors from the serious ones.

Distributed data base systems will find universal acceptance because they address all of the following situations. First, most large organizations are geographically decentralized and have multiple computer systems at multiple locations. It is usually impractical to have a single ‘‘intergalactic’’ DBA to control the world-wide data resources of a company. Rather one wants to have a DBA at each site, and then construct a distributed data base to allow users to access the company resource.

Second, in high transaction rate environments one must assemble a large computing resource. While it is certainly acceptable to buy a large mainframe computer (e.g. an IBM Sierra class machine), it will be nearly 2 orders of magnitude cheaper to assemble a network of smaller machines and run a distributed data base system. Tandem has shown that transaction processing on this architecture expands linearly with the number of processors. In most environments, a very efficient transaction processing engine can be assembled by networking small machines and running a distributed DBMS. The ultimate version of this configuration is a network of personal computers.

Third, suppose one wants to offload data base cycles from a large mainframe onto a back-end machine, as typically advised by data base machine companies including Britton-Lee and Teradata. If so, it will make sense to support the possibility of more than one back-end CPU, and a distributed DBMS is required. In fact, Teradata includes one on their machine already.

Fourth, as will be discussed presently, I expect more and more users to have workstations on their desks, replacing standard terminals. I also expect most workstations will have attached disks to ensure good I/O performance. In such an environment, one will have a large number of data bases on workstations that may be of a personal nature (such as appointment calendars, phone directories, mail lists, etc.) Even such personal data bases require a distributed DBMS, because such tasks as electronically scheduling as meeting require them.

Lastly, virtually all users must live with the ''sins of the past'', i.e. data currently implemented in a multitude of previous generation systems. It is impossible to rewrite all applications at once, and a distributed DBMS which supports foreign local data managers allows a graceful transition into a future architecture by allowing old applications for obsolete data bases to coexist with new applications written for a current generation DBMSs. This point is further elaborated in Section 7.

I expect everybody to want a distributed data base system for one or more of these five reasons. Hence, I believe that all DBMS vendors will implement distributed DBMSs and it will be hard to find vendors who offer only a single site DBMS in a few years.

5.2. Research Issues in Distributed DBMSs

There has been a mountain of research on algorithms to support distributed data bases in the areas of query processing [SELI80], concurrency control [BERN81], crash recovery [SKEE82] and update of multiple copies [DAVI85]. In this section, I indicate two important problems which require further investigation.

First, users are contemplating **very large** distributed data base systems consisting of hundreds or even thousands of nodes. In a large network, it becomes unreasonable to assume that each relation has a unique name. Moreover, having the query optimizer inspect all possible processing sites as candidate locations to perform a distributed join will result in unreasonably long optimizer running times. In short, the problems of ''scale'' in distributed data bases merit investigation by the research community.

Second, current techniques for updating multiple copies of objects require additional investigation. Consider the simple case of a second copy of a person's checking account at a remote location. When that person cashes a check, both copies must be updated to ensure consistency in case of failure. Hence, at least two round trip messages must be paid to the remote location to perform this reliably. If the remote account is in Hong Kong, one can expect to wait an unreasonable amount of time for this message traffic to occur. Hence, there will be no sub-second response times to updates of a replicated object. To a user of DBMS services, this delay is unreasonable, and algorithms that address this issue efficiently must be developed. Either a lesser guarantee than consistency must be considered, or alternatively algorithms that work only on special case updates (e.g, ones guaranteed to be commutative) must be investigated. The work reported in [KUMA88] is a step in this direction.

6. OTHER TECHNOLOGIES

In this section I discuss a collection of other interesting trends that may be significant in the future.

6.1. Data Base Machines

It appears that the conventional iron mongers are advancing the performance of single chip CPUs at nearly a factor of two per year, and that this improvement will continue for at least the next couple of years. Bill Joy quotes single chip CPU performance as:

MIPS = 2 ** (year - 1984)

Therefore, in 1990 we can expect 64 MIPS on a chip. Not only is this prognosis likely to happen, but also, machines built from the resulting chips are guaranteed to be extremely cheap, probably on the order of $1K - $4K per MIP. Moreover, nothing stops aggressive system integrators from coupling such CPUs into shared memory multiprocessors to achieve very powerful multiprocessor machines. Earlier examples of this approach include the DEC 62xx series of machines and the Sequent Symetry. In light of these advances in general purpose machines, it seems unlikely that a hardware data base machine vendor can develop cost effective CPUs. Because such a vendor makes machines by the 10s, he is at a significant disadvantage against a conventional iron monger who makes machines by the 10,000s. It is generally agreed that a factor of 3, at a bare minimum, is required in the custom architecture before a custom machine is feasible. Personally, I don't see where to get such a number. As a result, I see hardware data base machines as a difficult business in the coming years.

6.2. High Transaction Rate Systems

It is clear that relational data base systems will be used for production applications which generally consist of repetitive transactions, each of which is a collection of single-record SQL commands. The goal is to do 100, 500, even 1000 such transactions per second. Most relational systems are getting increasingly nimble and should continue to do so over the next couple of years. Moreover, all commercial systems have essentially the same architecture, so that any tactic used by one vendor to increase performance can be quickly copied by other vendors. Hence, the ‘‘performance wars’’ tend to be a ‘‘leapfroging’’ sort of affair, and the current winner is usually the vendor who came out with a new system most recently. Moreover, all systems are expected to converge to essentially the same ultimate performance.

The bottom line is that **all** vendors are addressing high transaction rate environment because that is where a significant number of customer applications reside. All will offer similar performance in this marketplace. The ability of any specific vendor to claim this arena as his ‘‘turf’’ is guaranteed to fail.

6.3. Main Memory Data Bases

Not only are CPU prices per MIP plummeting, but also main memory prices are in ‘‘free fall’’. Prices are currently under $500 per megabyte in most environments where competition exists, and are continuing to drop. Moreover, the maximum amount of main memory that can be put on a machine is skyrocketing in a commensurate manner. This increasingly allows a client of data base services to contemplate a data base entirely (or mostly) resident in main memory. Current DBMSs have been typically designed under the assumption that all (or most) data is on disk. As a result, they must be changed to efficiently handle very large buffer pools, to implement hash-join processing strategies [SHAP86], and to deal efficiently with log processing (which may be the only I/O which remains in this environment).

The opportunity of using **persistent** main memory is also enticing. One idea would be for the memory system to automatically keep the before and after image of any changed bits as well as the transaction identifier of the transaction making the change. If the transaction gets aborted, the memory system can automatically roll backwards. Upon commit, the before image can either be discarded or spooled to a safe place to provide an additional measure of security. With error correcting codes and alternate power used in the memory system, this will provide a highly reliable main memory transaction system. My speculation is that it is neither difficult nor expensive to design such a system.

Such techniques will hopefully become part of commercial iron in the not to distant future.

6.4. New Storage Architectures

Besides persistent main memory, there are some other ideas that may prove appealing. First, one could construct a high speed, write-only device with arbitrary capacity. Such an ‘‘ideal logger’’ could be constructed out of persistent main memory, an auxiliary processor and a tape drive or optical disk device. Additionally, the log can be substantially compressed during spooling. The CPU cycles for such activity seem well worth the benefit that appears possible.

Optical disk drives have received considerable attention, and they may well play an important part in future memory systems for data managers. Lastly, the most intriguing idea concerns the availability of very cheap 5 1/4'' and 3 1/2'' drives. Rather than using a smaller number of 14'' disks (such as the 3380), it seems plausible to construct a large capacity disk system out of an larger number of small drives. It appears that such a disk system could offer the possibility of a **large** number of arms and modest (if any) higher cost per bit compared to 3380 style technology. A step in this direction direction is the work reported in [PATT88]. Moreover, how to construct file systems for such devices is an interesting area of research. For instance, should one stripe blocks from a single file across all the disks. Alternately, should one retain the sequential organization of most current file systems whereby a single file is stored in large extents on a single drive.

7. HOW TO ACHIEVE VENDOR INDEPENDENCE

The current software and technological environment may allow an astute client of data base services to achieve vendor independence. What follows is a step by step algorithm by which any user can achieve freedom from his current hardware vendor. Since the most common vendor to which clients are firmly wedded is IBM, we use an IBM customer as an example and show in this section how that client can become vendor independent. We assume that the hypothetical client begins with his data in an IMS data base and his application programs running within CICS.

7.1. Step 1: Get to a Relational Environment

The first step is for the client to replace his data manager with a relational DBMS. Many companies are already considering exactly this sort of migration, and there are several strategies available to accomplish this step. In this subsection we discuss one possible approach. Consider the purchase of a distributed data base system that allows data in local data bases to be managed by a variety of local data managers. Such ''open architecture'' distributed data managers are available at least from Relational Technology and Oracle and without doubt, will soon be available from others. Consequently, the example client should consider purchasing a distributed DBMS that manages local data within both IMS and the target relational data manager. With this software, he can recode his old application programs one by one from IMS to his target relational DBMS. At any point in time, he has some old and some new programs. The old ones can be run directly against IMS, while the new ones can be run through the distributed DBMS. After the entire application has been recoded to make SQL calls, he can discard the distributed DBM, move the data from IMS to the target DBMS and then run his programs directly against the target DBMS.

Hence, a client can obtain a distributed DBMS and then slowly migrate his application and data bases from IMS to the target environment. This code and data conversion can be done at the client's leisure over a number of years (or even decades). At some point he will finish this step and have all his data in a modern DBMS.

7.2. Step 2: Buy Workstations

It is inevitable that all ''glass teletype'' terminals will be replaced by workstations in the near future. Hence, 3270-style terminals are guaranteed to become antiques and will be replaced by new devices which will be Vaxstation 3000, Sun 3, PC/RT, Apollo, Macintosh, or PS 2 style machines. Clients will replace their glass teletypes with workstations for two reasons:

1) to get a better human interface
2) cost

It is obvious to everybody that bitmap-mouse-window environments are much easier to use than 3270 style systems. For example, a user can have multiple windows on the screen and his application can take as many interrupts as needed since a local CPU is being used. There is no need for the cumbersome ''type to the bottom of the screen and then hit enter'' interfaces that are popular with 3270s. Already, knowledge workers (e.g, stock traders, engineers, computer programmers) are being given workstations. Later, data workers (e.g, clerks, secretaries, etc.) will also get workstations. The basic tradeoff is that a workstation translates into some quantifiable improvement in employee productivity. The cost, of course, is the purchase and maintenance of the workstation. This tradeoff will be made in favor of workstations for high priced employees and not for lower paid ones. Over time, as workstations continue to fall in price, it will be cost effective to give one to virtually everybody.

The second reason to give employees a workstation is that it enables one to move an application program from a mainframe (a 370 in our example) which costs more than $100,000 per MIP to a workstation which costs perhaps $1000 per MIP. The overall cost savings can be staggering. Hence, over the next decade I expect workstations to essentially replace glass teletypes completely.

Whether one chooses to move to workstations for human interface reasons or cost considerations does not matter. To take advantage of either, one must move application programs from a 370 to a workstation. Moreover, the only sensible way to do this is to rewrite them completely to change from a ‘‘type to the bottom of the screen’’ to a ‘‘menu-mouse-bitmap-window’’ style interface. During this rewrite, one must also move the program from CICS to some other programming environment (e.g. Unix, OS 2) This leads to step 3.

7.3. Step 3: Rewrite Application Programs

Whatever the reason chosen, clients must migrate application programs from CICS to the workstation. Of course, a client can run a window on a workstation that is simply a 3270 simulator connected to CICS. In this way, a client can slowly migrate his applications to the new environment while the old ones continue to run in CICS through workstation simulation of a glass teletype interface. At some point, all CICS applications will have been rewritten, and only a relational DBMS remains running on the 370 machine. Of course, this migration may take years (or even decades). However a persistent client can move at a rate appropriate to his resources. This will lead ultimately to step 4.

7.4. Step 4: Move to a Server Philosophy

At this point the example client has application programs running on a workstation and a relational data base system running on a shared host. These machines communicate over some sort of networking system. Moreover, the applications send SQL commands over this network to the shared host and receive answers or status back. In this environment, one should move to the following thinking:

workstations are application servers
shared hosts are SQL servers

Moreover, SQL servers should be thought of as a commodity product. To the extent that a client remains within the standard SQL defined by ANSI, it should be possible to replace an SQL servers built by one vendor (in this case IBM) with an SQL server bought from another vendor (say DEC) or even by a collection of servers running a distributed data base system (say a network of Suns). Vendor independence has been facilitated since it is now fairly easy to buy SQL cycles from the vendor who offers the best package of price/performance/ reliability. If the vendor of choice fails to remain on the performance curve compared to his competitors, there is little difficulty in unhooking that vendor’s machine and replacing it with one built by one of his competitors which offers superior cost effectiveness.

Similarly, one should think of workstations as application servers. If one is careful, one can write applications which run on a variety of workstations. If the current vendor ceases to offer price competitive iron, the client can simply replace his workstations by those built by one of his competitors. In this way **iron independence** is achieved.

7.5. Summary

During these four steps, a client will choose at least the following:

a relational DBMS
a workstation Operating System
a window manager
networking hardware and software
an application programming language

An IBM customer will, of course, be guided by his IBM salesman to choose the following:

relational DBMS: DB 2 plus the DBMS in the extended edition of OS 2
workstation OS: OS 2
window manager: IBM Presentation Manager
networking software: SNA
application programming language: COBOL (?)

In addition, he will be sold on the virtues of SAA as part of his solution. If the client moves in this direction, he will achieve iron independence to at least some degree. He can buy workstations from any of the clone manufacturers and can use SQL services that run on the various instantiations of IBM iron (e.g, PS 2, AS 400, 370, etc.).

However, the client can also make an alternate collection of choices:

relational DBMS: one from an independent vendor
workstation OS: Unix
window manager: X Window System
networking software: TCP/IP
application programming language: 4GL from an independent vendor

With these choices he can be assured of buying application servers and data servers from at least the following companies: DEC, DG, IBM, HP, Sun, Apollo, ICL, Bull, Siemans, Sequent, Pyramid, Gould, and the clone makers.

This section has pointed out a path by which one may obtain iron independence. Along this path, a collection of options must be chosen. These can be the ones suggested by the salesperson of a particular hardware vendor or the set that will maximize iron independence. This choice can be made by each client.

7.6. Standards Revisited

We close this paper with some comments on what can be done to assist a user in achieving vendor independence. Clearly, a user can buy an open architecture distributed data base system. In this scenario the client will have available the extended SQL implemented by that vendor. Statements in extended SQL will run on a local data base that is managed by the local data manager provided by the vendor. Standard SQL will be executable on foreign local data managers. Such distributed data base software will provide a seamless interface that hides data location and allows data to be moved at will as business conditions change without impacting application programs.

A second possibility is that a user will remain within standard SQL and build location information into his application programs. In this way, he will expect to send SQL commands onto a network for remote processing by some server. The server must accept the remote request and send back a reply. To facilitate being able to replace one server by a different one, it is **crucial** that a standard format for communication of SQL commands and the resulting responses over a network be developed. Standardization of remote data base access (RDA) is being pursued by

ISO but appears not to be an important ANSI activity. In my opinion, remote data base access will be more important than local data base access from an application program. I would encourage standards organizations to budget their resources accordingly.

REFERENCES

[BANE87] Banerjee, J. et. al., ‘‘Semantics and Implementation of Schema Evolution in Object-oriented Databases,’’ Proc. 1987 ACM-SIGMOD Conference on Management of Data, San Francisco, Ca., May 1987.

[BANC86] Bancilhon, F. and Ramakrishnan, R., ‘‘An Amateur’s Introduction to Recursive Query Processing Strategies,’’ Proc. 1986 ACM-SIGMOD Conference on Management of Data, Washington, D.C., May 1986.

[BERN81] Bernstein, P. and Goodman, N., ‘‘Concurrency Control in Database Systems,’’ Computing Surveys, June 1981.

[BUNE86] Buneman, P. and Atkinson, M., ‘‘Inheritance and Persistence in Database Programming Languages,’’ Proc. 1986 ACM-SIGMOD Conference on Management of Data, Washington, D.C., May 1986.

[CARE86] Carey, M., et. al., ‘‘The Architecture of the EXODUS Extensible DBMS,’’ Proc. International Workshop on Object-Oriented Database Systems, Pacific Grove, Ca., September 1986.

[DADA86] Dadams, P. et. al., ‘‘A DBMS Prototype to Support NF2 Relations,’’ Proc. 1986 ACM-SIGMOD Conference on Management of Data, Washington, D.C., May 1986.

[DATE85] Date, C., ‘‘A Critique of SQL,’’ SIGMOD Record, January, 1985.

[DAVI85] Davidson, S. et. al., ‘‘Consistency in Partitioned Networks,’’ Computing Surveys, Sept. 1985.

[FISH87] Fishman, D. et. al., ‘‘Iris: An Object-Oriented Database Management System,’’ ACM-TOOIS, January, 1987.

[GUTM84] Gutman, A., ‘‘R-trees: A Dynamic Index Structure for Spatial Searching,’’ Proc. 1984 ACM-SIGMOD Conference on Management of Data, Boston, Mass. June 1984.

[IOAN87] Ioannidis, Y. and Wong, E., ‘‘Query Optimization Through Simulated Annealing,’’ Proc. 1987 ACM-SIGMOD Conference on Management of Data, San Francisco, Ca., May 1987.

[KUMA88] Kumar, A. and Stonebraker, M., ‘‘Semantics Based Transaction Management Techniques for Replicated Data,’’ Proc. 1988 ACM-SIGMOD Conference on Management of Data, Chicago, Il., June 1988.

[LIND87] Lindsay, B., ‘‘A Data Management Extension Architecture,’’ Proc. 1987 ACM-SIGMOD Conference on Management of Data, San Francisco, Ca., May 1987.

[PATT88] Patterson, D. et. al., ‘‘A Case for Redundant Arrays of Inexpensive Disks (RAID),’’ Proc. 1988 ACM-SIGMOD Conference on Management of Data, Chicago, Il., June 1988.

[ROBI81] Robinson, J., ‘‘The K-D-B Tree: A Search Structure for Large Multidimensional Indexes,’’ Proc. 1981 ACM-SIGMOD Conference on Management of Data, Ann Arbor, Mich., May 1981.

[ROSE86] Rosenthal, A. et. al., ‘‘Traversal Recursion: A Practical Approach to Supporting Recursive Applications,’’ Proc. 1986 ACM-SIGMOD Conference on Management of Data, Washington, D.C., May 1986.

[ROWE85] Rowe, L., ‘‘Fill-in-the-Form Programming,’’ Proc. 1985 Very Large Data Base Conference, Stockholm, Sweden, August 1985.

[SELI80] Selinger, P. and Adiba, M., ‘‘Access Path Selection in a Distributed Database Management System,’’ PROC ICOD, Aberdeen, Scotland, July 1980.

[SHAP86] Shapiro, L., ‘‘Join Processing in Database Systems with Large Main Memories,’’ ACM-TODS, Sept. 1986.

[SKEE82] Skeen, D., ‘‘Non-blocking Commit Protocols,’’ Proc. 1982 ACM-SIGMOD Conference on Management of Data, Ann Arbor, Mich., May 1982.

[STON83] Stonebraker, M., et. al., ‘‘Application of Abstract Data Types and Abstract Indexes to CAD Data,’’ Proc. Engineering Applications Stream of 1983 Data Base Week, San Jose, Ca., May 1983.

[STON86a] Stonebraker, M. and Rowe, L., ‘‘The Design of POSTGRES,’’ Proc. 1986 ACM-SIGMOD Conference on Management of Data, Washington, D.C., May 1986.

[STON86b] Stonebraker, M., ‘‘Inclusion of New Types in Relational Data Base Systems,’’ Proc. Second International Conference on Data Engineering, Los Angeles, Ca., Feb. 1986.

[STON87] Stonebraker M. et. al., ‘‘The Design of the POSTGRES Rules System,’’ Proc. 1987 IEEE Data Engineering Conference, Los Angeles, Ca., Feb. 1987.

[TSUR84] Tsur, S. and Zaniolo, C., ‘‘An Implementation of GEM -- Supporting a Semantic Data Model on a Relational Back-end,’’ Proc. 1984 ACM-SIGMOD Conference on Management of Data, Boston, Mass., June 1984.

[ULLM85] Ullman, J., ‘‘Implementation of Logical Query Languages for Data Bases,’’ Proceedings of the 1985 ACM-SIGMOD International Conference on Management of Data, Austin, TX, May 1985.