# Database in Crisis and Transition:
# A Technical Agenda for the Year 2001

David Vaskevitch
Director of Enterprise Computing, Microsoft Corporation
davidv@microsoft.com

## Abstract

The current paper outlines a number of important challenges that face the database community and presents an agenda for how some of these challenges can be met. This database agenda is currently being addressed in the Enterprise Group at Microsoft Corporation. The paper concludes with a scenario for 2001 which reflects the Microsoft vision of "Information at your fingertips."

## The Database Challenge

Are Databases and Database technology at the center of the information rich computer world of the future, or, ironically, are databases about to become literally irrelevant just as that future arrives? If there is a problem to be addressed by databases, does the solution require more of the same kind of research and development that has characterized the last two decades in database land, or is there a completely different set of questions, in addition to the standard ones, that need to be considered to ensure that databases stay vital, relevant and central to the world? Where are databases going, and how is this different from whence they came?

SIGMOD 94- 5/94 Minneapolis, Minnesota, USA
© 1994 ACM 0-89791-639-5/94/0005..$3.50

Clearly, at one level, databases and database technology are one of the key pillars of the modern computer world. Banks, manufacturing companies, airlines, and organizations of all sizes critically depend on their dbms's every day and in every way. Database products like DB2, Oracle, SQL Server, IMS, and their kindred, like CICS, account for billions of dollars in revenue each year. Finally, no computer science department, no large research group, no serious consulting firm, no systems vendor would be considered complete without at least one major database group.

At the same time, the database industry often seems oddly out of step with the rate of change in the rest of the world; a community caught in some kind of *back to the future* time warp. For instance, most of the same organizations running the biggest relational and non-relational database systems now have more data sitting on desktops than in their db's. This desktop data, often highly structured in nature, is controlled by everything but database management systems. In fact, by any objective measure the number one database in the Western World is, not DB2, Oracle, or even IMS, but rather 1-2-3, with Excel hard on its heels. Yet, where is the database architecture, academic or otherwise that explicitly describes the integration of desktop datastores into latter day federated systems?

Continuing to examine the dimensions of this reality warp, we find a sharp distinction between distributed database, the theory, versus distributed database, the reality. In the theoretical world, distributed databases are completely understood. Techniques like two phase commit, federated database schemas, and partitioned queries represent largely solved problems, and distributed databases aren't even that interesting to talk about. Out in the cold, hard commercial world, not only are distributed databases exceedingly hard to find; most practitioners consider them to not even be feasible. Which is right? Trivial or impossible? Where is the balanced view that pulls these two contradictory perspectives together?

Moving closer to the center of the database universe is the issue of the data model. Here both the academic literature and the commercial vendors are more or less equally out of touch with reality. In one corner of the ring are all the relational vendors, representing a healthy $2B industry, convinced that their products own the world. To attend a client / server seminar is to come away convinced that SQL and RDBMS's are the path to *all* production data, past, present and future. In the other corner of the ring, preparing to slug it out with the reigning champ, is the OODBMS community, strongly committed to the notion of persistent languages, complex data structure and sophisticated navigation. Must we choose? Are we witnessing a generational shift in progress? Perhaps, but what about those two other contenders silently sitting in the other two corners? Which corners, you ask? Well, over on the left are our old friends IMS, IDMS, and that gang. According to Gartner group these old buddies still are responsible for almost half of all the world's production data. Wait there's more . . . In the same corner with IMS is VSAM, RMS, and more recently Btrieve. Between them the ISAM's and pre-relational databases turn out to hold about 75% of the data really used to run organizations. Do we hear reality calling yet? If not, we need only look in the fourth corner of the ring to see those spreadsheets and other desktop database stores seeking our attention. While everybody argues about where the *real* or the *most important* production data is held, the desktops have grown in size and importance to the point where they hold at least as much data as all the servers, mainframes, and minis combined.

The question to ask about all these different data handlers is: *will the real, true database please stand up?* The database community will insist that relational databases are the present, and objects in some form are the future. Does that mean that network and hierarchical dbms's are not databases after all? And, if no database worthy of the name provides the simplicity and functionality of an ISAM, a spreadsheet, or a Paradox, what does that mean? Would the users of a spreadsheet agree that their data is not really data and their database is not one after all?

To see the dilemma we really face in its true perspective, let's consider one last reality confounding conundrum. To most database professionals, distributed data and two phase commit (2PC) go hand in hand. It is a characteristic of large systems built around two phase commit protocols that failures can easily cause the entire system to grind to a halt. That is, if there are many interlocking transactions, and lots of cross node dependencies, then when nodes and communications links go down, major parts of the network will go down until the nodes / links come back up. Of course, 2PC guarantees that the whole system will produce the right results in the end, but along the way, the system can seem very fragile. Various attempts in the industry to provide better approaches for failure in situations like this are not universally accepted. The obvious conclusion is that a centralized system with duplicate hardware would have better uptime characteristics. *Isn't there something deeply counter intuitive about the idea that the distributed solution might be less robust than the centralized one?* Surely there must be some way of building distributed database systems that will be more, not less, robust than centralized ones.

All of these problems suggest that there might be some major new ways of thinking about databases that the database community might consider adopting. The result would be a paradigm shift in the direction of increasing relevance. Relevance?

The core issue behind each of the problems described above is that database technology, as it has evolved, while highly useful, is in danger of becoming irrelevant to the majority of computer users in the next century. Just as computers are ready to truly change society, just as they are on the verge of truly widespread adoption, databases might end up on the sidelines of the resulting picture. How could this be? Simply:

- If most data sits on desktops (and in notebooks) in data bases that are not databases, and
- If most production data sits either in non database stores, or again on desktops, and
- If the highly distributed computers of the future still don't have adequate distributed database technology, and
- If the distributed databases that do exist are not truly robust,

then users and developers will find other ways of managing data. They won't call the result a database, but that's what it will be. And, we, the database profession, will have made ourselves obsolete.

## An Agenda for the Decade

At Microsoft we are working to rethink the very meaning of the term *database*. As part of that redefinition, we have an agenda of important problems that need solution either by us or by partners we can work with. Some of these problems are described below. The reader is referred to the IEEE paper in [ONETAL] for somewhat more detail on Microsoft's technical approach to some of these problems. Many of these problems do not have a current understood approach, and besides being important problems in their own right, they are also a kind of challenge to the database community as a whole.

- **Component Databases** Ironically, databases are the last major preserve of monolithic, closed design. A decision to use a particular dbms is also a decision to accept a way of managing disk space, buffers, an access method, a security scheme, a query language, an API, and more. In short, every database, relational, object oriented, or otherwise, is its own self contained world.

  The first challenge for the decade is to redesign databases around the concept of layered, cooperating components.

- **Interoperable Databases** Component databases, with published interfaces, are, by definition, interoperable databases. A query processor can retrieve data from record providers of all kinds. Many kinds of query providers can be written. A spreadsheet can masquerade as a database by acting like the right kind of component. A geographical query processor can retrieve data from an underlying store just as fully as a relational query processor.

  In an interoperable database environment, individuals can create and maintain budgets in spreadsheets, running as spreadsheets. Yet, a CFO can consolidate data across many spreadsheets (and project managers and databases), using a classical relational query tool, and seeing the whole thing as truly a database running as a database.

  In an interoperable database world, displaying data on a map, and handling geographical queries is just as easy as handling hierarchical and navigational queries, which in turn is as easy as handling hypertext queries. What you see depends only on the query tool, the underlying data is equally accessible no matter which tool you pick.

- **Distributed Databases** How many databases will there be in the year 2001? How many computers? More than millions, actually hundreds of millions. This implies that databases have to be highly distributed. This in turn means first, databases must be completely self installing, self managing. Secondly it means that coordination between databases must be automatic and highly robust. But most of all, it means the distributed infrastructure must scale extremely well.

- **Processes, More Than Tasks** Classical databases and TP Monitors handle transactions and tasks that occur in real time. By definition, a transaction is viewed as an atomic activity, a single event, that either occurs in its entirety or is made to not occur at all. The real world though, is built out of *sequences of tasks* that occur over very long periods of time. (See for example [GARSAL].) Databases and the infrastructure that surrounds them must be designed to handled long running sequences of transactions.

A network designed entirely around coordinated transactions is, indeed, less robust than a centralized system. A network, on the other hand, designed around sequences of tasks, is far more robust than a centralized system. The key is to have infrastructure that makes those sequences easy to build, reliable, and robust.

- **Rich Data Models** Normalized data is fine, when the design calls for normalization. Often though, more complex record structures are both more natural and more efficient (as seen in object-extended relational database models, with products such as UniSQL and Montage). In the same way, representing many-to-many relationships often has risks, but it often has benefits too. Twenty years of real experience teaches us that sometimes normalized tables are right and sometimes not; databases in the future must offer that choice.

- **Databases, Not Languages** One consequence of the component database model is that the underlying database becomes a distinct and separate component from any higher level language environment. Today most modern databases are tightly bound to either SQL or some object oriented language like C++ / SmallTalk. This type of binding has strong advantages for many applications, but there are other cases where the developer simply wants the use of a database manager without being forced to pick a particular language, object model or development framework. In the component world of the future, this kind of separation becomes possible and natural. Once consequence is that a whole new class of record providers can be offered, each inheriting all the higher level environments. Thus the developer of a new type of project manager, for instance, by exposing appropriate methods can be a record providing component. The higher level language environments, whether SQL based or object oriented, can then tap into this record provider just as well as they do into any other.

- **Navigation *and* Queries** Today object oriented databases support one style of navigation; network dbms's and ISAM's support another. Relational databases, on the other hand, provide queries and set based operations. A direct consequence of the component database model is that the developer (and user) no longer has to make a choice. Lower level database components provide the same navigational capabilities as ISAM's. Higher level query processors can skew in either the set oriented or the pointer navigational direction or both. The user can choose.

  Just as importantly though, part of our agenda for the next decade has to be to recognize that both element by element, navigational style processing *and* set oriented, query based computation are equally valid. Often, the query based approach is the best way to specify a set of records in the first place, while at the same time, navigational operation is the only way to then work with the resulting data in a fashion sufficiently rich to meet the needs of complex applications. The sooner we give up on the idea of forcing a choice, the better.

- **Server, Desktop, Laptop** Hundreds of millions of servers describes only a small part of the distribution model for the future. Each of the multitudinous servers will support dozens of desktops. And, many of the desktops, will really be computers that are often disconnected to become notebooks and laptops. Even today, as much data sits on desks, laps, and under arms, as on servers; in the future this ratio will shift even further away from the server. In a world with billions of databases, how do we think about replication, distributed transactions, processes and the like? What does the administrative model need to look like, and how does it operate in a totally decentralized environment?

One tempting, but wrong view is to think of these personal databases as somehow simple, small, or trivial. Features like on-line backup, real transactions, and so on might not be required, right? Take on-line backup . . . Do you backup your computer regularly? Would you be willing to do so if the process was completely automatic and didn't prevent you from working while the backup was in process? Is that possible without on-line backup? How about transactions and recoverability? Do we really believe users *want* to lose data? Can we guarantee they won't without such facilities? Yet, the whole thing has to be so simple that even a garage mechanic or taxi driver can install the database and run it without getting help ever. Quite a challenge? Absolutely, and now's the time to start thinking about it.

- **Thanks for the Memory** Imagine a really big server supporting several hundred desktop computers. Perhaps the server has 500M of memory. How much does each workstation have? If the answer is 25M - 50MB, then how much memory do the workstations in the aggregate have? Here's a situation where the aggregate personal computer memory, at, say, 5,000MB totally dwarves the server's memory. How do we design database systems to really take advantage of this situation? OODB's do some of this, quite well actually, but how well do they do at managing large queries where the work could be divided up across several machines? And, what about relational databases; how much advantage of a two level memory architecture do they make?

  Another way of thinking about this problem is to ask: where do applications keep their private data structures? Certainly not in any classical database: too slow and rigid. Perhaps in an OODB. A goal of the nineties should be to have all applications shift a major part of their currently private data structures to the stewardship of a database manager. Why bother? To simplify queries, management of concurrency, provide recoverability, and so on.

A more detailed explanation of some of these problems is given in [VASK1, VASK2].

## 2001: A Database Odyssey

A salesman is about to take a day trip to another city. As he undocks his notebook computer, it refreshes his database one last time before disconnecting. On the plane, the salesman completes a territory analysis using a spreadsheet, develops an action plan using a project manager, and then decides on the top twenty accounts to visit, using a classical graphical query tool. Each of these tools works directly with the underlying database sitting in his machine.

On landing, his geographical query processor puts up a street map, shows where the top twenty prospects are located, and highlights the best route for making it through the day. Although the mapping program is intensely navigational (no pun intended), the salesman sees it as just another tool accessing his data in a very natural fashion. During the course of the day, the salesman makes several presentations, enters some orders, and updates a few customer records.

Throughout the day, as he rents a car, buys meals, and completes other transactions, his wallet computer (nee credit card) tracks all the transactions for him. Communicating with the notebook computer, the wallet computer also keeps the salesman's expense report constantly up-to-date.

Returning home, the salesman docks his notebook so that it can talk to his house server. Sorting through mail, he finds that his daughter was invited to a birthday party which conflicted with a dentist appointment. The server, talking to the dentist's office computer moved the appointment to eliminate the conflict, and confirmed the daughter's appointment based on its knowledge of the close nature of the friendship. All of the underlying transaction coordination was, of course totally invisible. Reviewing monthly expenses, before having dinner, the salesman finds that his wallet computer has already updated the house server, and all his expenses are already reflected in his personal accounting system.

Finally, the next morning, the salesman returns to the office, docks his notebook computer and starts working. The orders and customer changes entered the previous day are sent to the server which in turn communicates changes it has received back to the notebook. Along the way, the salesman receives the results of a historical marketing analysis he had launched two days ago, which involved collecting data from all over the world, collating and then massaging it. The server, data in hand, sends the final result to the notebook for subsequent analysis.

2001 is only seven years away. Is there any part of this scenario we would want to not have be true by then? Can we build it now? Clearly, if this scenario comes true, 2001 is a world where databases are truly ubiquitous, highly relevant, and quite different from those we know today. That is our challenge.

## References

[GARSAL] H. Garcia-Molina and K. Salem, "Sagas," ACM SIGMOD Proceedings, 1987.

[ONETAL] Patrick E. O'Neil, Mohsen Al-Ghosein, David Vaskevitch, Rick Vicik, Laura Yedwab, "Transaction Processing at Microsoft: Present and Future," IEEE Bulletin on Data Engineering, March 1994

[VASK1] David Vaskevitch, Position Paper for Workshop on High Performance Transaction Systems, Sept. 1993, Asilomar CA.

[VASK2] David Vaskevitch, "Microsoft's Vision for the Transaction Environment," OTM Spectrum Reports, v. 8, no. 1, February 1994. Spectrum Reports Ltd., MCI Mail: 313 8708