From: Bill Gates [/o=microsoft/ou=northamerica/cn=Recipients/cn=1648] on behalf of Bill Gates
Sent: Monday, January 15, 2001 5:34 PM
To: Jim Allchin (Exchange); Steve Ballmer
Subject: FW: The Fifth Database Revolution

We need to get someone very technical to pull together our platform story.

Jim could do it but its probably best for him to delegate it to a small group with a leader.

The leader could be Eric Rudder or Rick Rashid or someone I am not thinking of. Some good work was done during the NGWS days that needs to be carried through.

Eric tells me that currently there is some progress on this stuff but not a clear direction from management.

It is as a key advisor to this group that David's input would become important. The key stuff is under Paul Flessner and Yuval Neeman but neither of them is right to drive it directly. It does touch on other pieces like WMI and Office extensibility.

This is one of the bigger items in my memo and its waiting there. I am not saying its easy work to do.

Lets pick how this is going to be driven.

I need to discuss that with both of you for a number of items in the memo but this is perhaps the most urgent.

Here is the latest on this from the memo:

Applications platform

Our applications platform message is quite confused today. Pieces like CLR, WMI, MSMQ, XML runtime, Biztalk, MTS, IIS, ASP+, Load Balancing, Message bus, SOAP, UDDI and Yukon are not consistent and reinforcing. Basic standards like eventing, logging, and filtering have to be established. The disconnection of these products make our message when trying to win back the developers who like JAVA and J2EE very difficult especially when we have the limitation of being only on Windows and those technologies are supported on many platforms by many companies. Although we have waited a long time for the shipment of VS with the URT that doesn't give us anywhere near a complete consistent platform story.

The most consistent platform in the industry is Oracle. They have used their database as the center of gravity to drive a very strong story. We need to integrate more capabilities like email and directory and workflow and file system where Oracle has done very little. In the basic intrastructure area though there are some lesssons to learn from them.

We have talked about many of these problems but not pulled things together. MSMQ is a bit of an orphan. Our transaction strategy isn't getting any traction while BEA has established an $800M per year business around that technology. We did a good job on MSMQ and MTS but they couldn't thrive on their own. Our decision to make Yukon the center of gravity and to connect Yukon to the URT should give us the clear starting point. We may need to be able to package Yukon so that it doesn't feel like a database if all you want is a Message bus. We may need to create some subset implementations of things like Queuing for size and speed reasons. However the API set should be consistent. We may need to be compatible with some of the J2EE apis.

Our application platform for the server and the client need to be the same. The strength of our approach is that code should be able to run Offline. This highlights again the importance of a Distributed Application Architecture where code can determine what it needs to execute on a different server or down on the client. ASP+ has to be made reasonable as a client side API set which it is not today.

We have to take a hard look at our tools and consider how to be a better high end solution. We have to spend a lot of money to make sure the openness of C# is well understood and that it is accepted at a level that allows our innovations to have traction.

I think that between Paul, Yuval and Eric's group with leader from Rick Rashid we should be able to go through another iteration on this (like we did with NGWS) and come up with some clear answers.

The strength of this platform and the innovation around it is the key element in preventing commodization by Linux, our installed base and Network Appliance vendors. We are in the best position to define the distributed application model that allows work to be moved out into the Network. We don't have enough research our product group people pushing this agenda but we have the best opportunity. This is what it takes to seize leadership in caching, load balancing and protocols. I think between Management/Setup and a vision of how our platform is Distributed we give ourselves a chance to lead in all the Level 7 networking pieces. I almost included this as a separate item but executing on these two technical pieces will give us what we need except for packaging, marketing and sales force.

There is a major packaging question once we get architectural coherence. To what degree should we package or charge for the rich so called middleware pieces separately from the rest of the platform? Are there advanced forms of some of these pieces that cost extra? Most of the API set we want supported in the base server with understandable advanced services costing extra.

We are discussing with IBM a joint effort to agree on most of the Application server pieces so that companies have a choice of our two implementations. Although this would be an unexpected partnership I see a lot of advantages for both companies. I think they can help with parts of the architecture. The current view is that we do not share any code between the companies.

We also need to drive Microsoft to use the new platform to prove it out and show it off. Our Services need to use these architectures so that our tools make them easy to extend.

-----Original Message-----
From: **David Vaskevitch**
Sent: Sunday, January 14, 2001 6.12 PM
To: Bill Gates
Cc: Jim Allchin; Steve Ballmer
Subject: The Fifth Database Revolution

A while ago I promised Bill that I would write down in some detail what has to happen next in database land. It's also come up in conversation with Steve. So, here are two papers. There are also two papers dating back about two years that supply some of the more intricate underlying technical details. The second paper is more technical, more pointed, and better written. The first paper is more motivational, kind of, and, because I switched to the second paper before finishing the first one, the first one runs out of steam near the end.


The Fifth Database Revolution....


The Structure of the Fifth Dat...

Having now sent these I have to admit I also feel pretty weird sending them. Weird and conflicted. On the one hand, I feel pretty deeply that if we don't do what is described in these papers, and some of the others I've been writing, we will either a) not achieve our long term goals (platform adoption, business growth, developer wins, etc), or b) get into relatively serious trouble (never catch up with Oracle, not have the platform the biggest apps are written on, miss key changes). All of that makes me want to write these papers, want to see them acted on. Then there's the "on the other hand" . .

On the other hand I am now totally disconnected from pretty much everything to do with our platform. These papers are hard to write in a wide variety of ways: time consuming, energy draining, etc. And, being so disconnected from the platform, it means that most of what gets written in papers like this is just not going to happen. True of storage. True for distributed app support. True for things in general. So, I'm saying out loud, that I'm trying to figure out whether to even keep writing this stuff. Besides the fact that it might well not have much effect, chews up time, etc, it must be annoying for the people actually having to build this stuff, to have people off in other areas writing this kind of stuff down for them . . .

The next one I would have written was going to drill into the whole "distributed" and "application server" mess. But, I'd really appreciate feedback on whether it is good, bad, or indifferent, and why, to be writing in this vein . . .

The Fifth Database Revolution

Changing the Rules of the Game

Do to Oracle what they did to Cullinet in the 80's. Make databases the storage for applications of all kinds everywhere. Create the center for the next generation operating system, application platform, and for the computing world in general. All of that and more follows for creating the fifth database revolution. That means developing a very clearly focused plan for Yukon, a plan that is more ambitious that we are currently thinking about, a plan which aims for more fundamental innovation, and innovation at a more fundamental level than we have ever done before

Fundamental Forces

- Need for Sea Change
- Huge Disks
- PC / Server Symmetry
- Pictures and Sounds
- Multimedia Web
- Transactional Internet
- XML
- Huge Memory

Technically, do we aim for a fundamental new data model, a fundamental new algebra, a new strongly typed system that lies at the base of all our storage? The alternative – which is plan of record – is to keep the relational model, essentially intact, and glue on side pieces to handle XML shredding and retrieval, memory oriented caching, stream oriented files, and a bunch of other stuff, none of which will lead to a fundamental revolution.

This is the first of two papers: this one describes the motivation of the revolution, the other paper[1] described the technical features required to make it happen.

More, Far More, than a DB

We are going to transform the very concept of what it means to be a "database" to the point where every application, every user, every organization will just by default assume they want and need a database on every machine. The fact that, by the time we do this, the database will have become central to the operating system and the basic application platform will be both a consequence and a driver of this change The key thing is we have to really be fixed on – have an incredibly clear vision – of what it is we are transforming the database into.

Core Principles

1 XML to the Core
2. Relational File System
3. XML-Relational Data Model
4. In Memory Data Base
5. Object Relational Mapping
6 World Models

Today a database is a slightly exotic industrial artifact at the central of heavy-duty applications. Today databases only store records – basically – are only seen by programmers – basically – and are not particularly relevant to either most applications or most users. The decision we get to make – whether explicitly or unconsciously – is whether we want to transform the very notion of a database – or not – into something totally mainstream

There is a set of fundamental transformations to the computing world around us that both enables and drives such a redefinition of databases; I lay those out. Then there is a new, challenging and complete, model for what the database can and must look like to be the center of the new world. That is what we need to sign up to build.

The new database finally becomes what databases were always supposed to be: the place where essentially all data can be stored More important, as databases were always supposed to do, ***the new database not only stores but organizes and gives meaning to all the data in it.*** It is this "giving of meaning", this "providing a structured framework" that truly distinguishes the new world databases from the file systems and amorphous stores that we have today.

To succeed the new world database cannot be an accidental creation. For example, just gluing XML and, perhaps a relational file

[1] "The Structure of the Fifth Database Revolution"

system. onto the database, totally begs the question of what the new world data model needs to be And, if we beg the hard questions, all we will build is a more complex engine – databases are already complex enough – that no one will want. And, worse it will not support new classes of applications, because it won't provide a solid architectural foundation for building them on.

So, there is our challenge: redefine the database world. Sign up to the big engineering project. But most of commit to thinking through, down to the foundations, what that new database world really looks like, so what we propose is a new world order. Not a new world mess, or a new world patchwork quilt, but truly a new world order

The Need for a Sea Change

Why even bother trying to create a revolution? After all it is hard work technically, entails a fairly large degree of risk, and may cause us to just look weird. Why not just get back to basics, make our database faster and faster, add on new features on the same basic base, and aim for clustering in the release after? Isn't that safer strategy?

Without a basic sea-change, and without us leading the sea-change, actually causing the sea-change, we will never overtake Oracle. They start with a technically impressive product that they, too, keep improving. The product runs on several platforms, including ours. Many of the other platforms, as hardware environments, offer more scalability than we do, and coupled with the surrounding operating systems, the other environments offer more manageability too. The core Oracle database starts out ahead of ours in several areas, including embedded Java, stream file system, object relational, abstract data types, queuing, parallelism, and more. As we add stuff, so are they. If we do pick a front, like clustering, to focus on, it is hard to believe that they won't be able to get there just about as fast as us, making any advantage we can target short lived at best. And, all of this is before we start to think about their overwhelming marketshare, their customer relationships, their perceived position as market leader, and the fact that they run on our platform and the platforms we don't run on offering customers the safe choice So, all of this says that competing with Oracle on their own turf is a losing proposition: that they only way to win is to take the big risk and cause a revolution.

The world both needs and is ready for a revolution in data storage. If we don't do it, somebody else will, and then we will have yet another first mover to catch up to. Worst of all, if we don't do it, it is quite likely that, in a slightly more incremental fashion than we would, Oracle will be the one to do it first, and then we might as well pack our bags and go home. The question is: "why does the world so much need a data storage revolution?".

Huge Disks

Start with huge amounts of very cheap storage, on notebooks and desktops, as well as servers. Being able to carry around 10-20GB's under the arm is routine today. What in the world would an individual put into a 20GB hard drive? Not Office, not a life time of memos, not a shelf full of books, not budgets or even charts of accounts for even a pretty big business, not a year's worth of presentations – even all of those put together don't amount to even 1 GB. Beyond words and numbers, pictures and sounds, make even the largest disk seem small.

Imagine the web without pictures and sounds. Try to picture a web of pages consisting of only words and number. Beyond just images and soundtracks, more and more, animation is starting to be key too.

Where is the single place where I can store documents, records, transactions, sounds, pictures, videos, web pages, links, tracking data – everything? That place doesn't exist, the need for it to exist is one driver for the revolution. Be clear, the need is a personal need, a workgroup need, an MIS need, an ISV need, and a webfarm need. All users, all operators, all application developers will need a single, consistent, safe, and secure place to store, find, update, and work with all their data. Yes, this is the old "universal database" dream, but guess what, the time has finally come . . .

PC / Server Symmetry

For years we have talked about "the day" when it would make sense to have the same DB running on the PC as runs on the server. That day is here It is here from a need perspective, a features perspective, and, at least as important a hardware perspective.

At one level the need for a serious, industrial strength database on personal computers has been both completely obvious and completely preposterous If we could ever get there users could finally have security, "safety", robust features, and key issues like replication, backup, and so on would be vastly simplified. On the other hand, three major obstacles have always stood in the way: cost, applicability, and need

Until about two years ago, stuffing a real database into a notebook or desktop PC would have been just too much of a burden. Even today, though, with 96M, 128M or 192M, let alone even bigger memories, not to mention 1GHz processors, and 10GB disks, most users won't even notice if SQLserver is (pre) installed on their machines So, from an affordability perspective, the day has come. What about applicability?

Imagine giving SQLserver as a birthday present; what would any normal person do with it? A database that only stores business records is of only limited use on most PC's. Now, even as a record holder, if we ever built a version of Outlook that really had a data model and was built on a local SQLserver, every user in the world would immediately have a use for the db. Every CRM vendor - guaranteed. 100% -- would immediately integrate with that version of Outlook, and suddenly tens of millions of sales and support people would not only use, but actually mildly stress their client databases. And, modulo our willingness to get Outlook there, such a prospect is now eminently practical. So, even without a change to the definition of databases, cheap, huge hardware, has brought the data of the client db without our reach and within customers expectations. And that is even with today's still limited applicability. Now suppose that changed.

Suppose the database could store files, applications, songs, photographs, contacts, email, reminders, videos, web pages, links, and more. Suppose the database brought with it robust back, transactional integrity, and configurable security. Most of all, suppose that database ***came with a built in data model that organized all those different kinds of information***. Now you can store photographs without "naming" them, yet you can find them in a way nobody can today. Now links are all part of a database structure that understands annotations, private webs, and my links versus your links. Now documents, trip reports, emails, are automatically associated with the trips, meetings, tasks, and budgeted projects they should be connected to. Songs are automatically classified by album, author, performer; you can have many playlists, when you hear a song on the radio, your system knows how to find it and link it in to this structure. Who wouldn't want a store that can do all that? That is a store that everybody would want on every notebook, desktop, and pocket PC. That is a solution to the applicability problem. When we build that database, that database will be one that would make a great birthday present – although the present will likely be the computer on which it arrives already installed So, we are coming up on an "applicability watershed"; when we cross it, everybody will want what we have built. Which brings us to need.

Whether we build it or not, the need to organize all that data at a personal level is there Today. Finding documents was hard. Finding emails is harder. But finding photographs is almost impossible. And, songs are not a lot easier. Think about it this way: the same world in which high-schoolers have notebooks with 20GB hard disks, is that world in which those school kids also have a 20GB sized data management problem. Remember for a moment that just ten years ago 20GB was a big corporate database. Now it's a kid sized database. That's a real need, multiplied by the many data types, multiplied by the flow of information across the Internet and into people's homes, dorms and offices. The hardware is there, the applicability is easy to see, and, most of all, the need is here.

Somebody will fill that need, and whoever does it first, will be the winner in the next database revolution

The point of this section is not just that PC db's are important. The point really is about symmetry. The point is as much a server point as it is a client point And, in the end, it is truly a db revolution point

Why Clients Matter So Much

For the first time, database design has to be driven as much by the desktop as by the server It still has to be driven by the server, and many scalability, manageability, and distribution requirements will be server unique. However, for the first time, database design will be equally -- however you define "equally" – driven by requirements that emerge from the client and user side of the equation. This is why the point is not "ubiquituous clients", so much as "PC / Server Symmetry". Why?

First, now that clients are so powerful, have such big disks, and so much memory, they are capable of running industrial strength software. Second, as users amass large collections of information, they will develop their own set of demanding requirements. Third – and most important of all – the very data types that are driving the next revolution – sounds, pictures, XML records -- are equally at home on the client and on the server. And that will drive us to need and build symmetrical databases.

In the past all the data, all the records originated in, stayed in, and were the property of large organizations. That leads to server centric databases. Now that we are tapping into all the more personal data too, into data that doesn't only originate in large orgs, our focus expands to the whole world, and the design of our db changes along the way This change is so counter-intuitive that it is hard to even know how to think about it.

Factoring the Client In

There are two keys here: 1) keep the client in mind all the time, and 2) think about user datatypes and operations all the time. This affects every aspect of the database, and, if taken to heart, will drive a design that will last for a long time. Why a long time? Because if we get the organizational needs, as we always have, and now get the individual and end user needs, that really is pretty much the whole world. And, that's the central point: finally databases are going to be relevant to the whole world.

Pictures and Sounds

What does it mean to really do a good job with pictures and sounds? Of course it means storing and retrieving streams, including very large streams, quickly and reliably New quality of service considerations arise – delivering a sound or video stream in a stuttering fashion is almost as bad as not delivering it at all. But just doing a great job with blobs is not enough After all records could be stored as blobs, but then nobody would think we had done a good job.

Compression and storage formats need to be part of our database work. For example fractal technology allows pictures to be stored in a lossless fashion while providing very interesting interpolation characteristics when the pictures are scaled up for printing. If scalable fonts – which once was a big deal for the Microsoft Corporation – was a big deal, then scalable pictures is ten times as big a deal. Compression is a hot topic for sounds too. MP3 for example does a pretty good job on the size front, but at a real cost in sound quality. Even native CD formats are, in some ways, not as good as old LP's, because the sampling rates and algorithms are not up to what the human ear can hear.

Indexing and organization are just as challenging as compression. For example, an Israeli company has developed software that, with modest training, can recognizes a few faces; that means your computer can pick out pictures according to which of your children, friends or wives (!) are in them. Beyond indexing, the structure of collections – collections of songs, pictures, classical pieces, art pieces – ought to be an intrinsic part of the database itself.

Quality of service takes on a whole new meaning when the storage system is delivering continuous sounds and videos; having "Cliffhanger" pause is just not an option

Supporting pictures and sounds is critically important for individuals, families, and organizations, but it takes on a whole new dimension in the context of the Web. Then the question becomes: do we want Yukon to be the backing store only for non-Web applications, or for Web applications, too.

The Multimedia Web

The Web of the future is defined by three characteristics: 1) intrinsically multi-media, 2) very high concurrency with high peak user loads, and 3) intrinsically transactional The point about the multimedia web is that it is not just about streams If the database is to be useful it has to deal with the multimedia data in its native format, decomposed into its element, so that web servers can project personalized pages to users.

This all places a huge new demand on the new type system: at some point sites will be dealing with millions of objects, creating animated pages on the fly, and the question is will we be there to support them.

The Transactional Web

If you take away updates, then building a read-only system simplifies the job a lot. Many of the features called for here might still be required, but they might not be required in the context of a database It is because we want to support sites that create rich content on the fly *and* support serious transaction loads that we need a database at the core

Putting It All Together

The Revolution – which is what we need to cause – comes from putting this all together. This paper outlines a series of changes in the computing environment which will create a huge requirement for a new sophisticated data store of some kind I suppose it could evolve from some other direction than a database, but that means we are missing our main opportunity.

The opportunity is to recognize first how big the sea change is. Second to recognize how much it is in our interest to change the rules of the game anyway. And, then to do it.

Are we doing it?

The Structure of the Fifth Data Revolution

A Solid Foundation to Build On

The fifth data revolution, in reality, advances a radically new data model, algebra, and theoretical foundation on which databases of the future are built. This new foundation is based on six fundamental substructures, described in the box at right. The key is to not stop with the first two bullets, and in fact to realize that the XML-Relational Data Model and the "world models" are really the bottom most elements on which everything else is built. On that note, I jump immediately to describing the foundation elements.

Core Principles

1. XML to the Core
2. Relational File System
3. XML-Relational Data Model
4. In Memory Data Base
5. Object Relational Mapping
6. World Models

XML to the Core

XML erases the distinction between documents and records, essentially blurring that distinction away to the point where we have to consider documents and records as one smooth continuum. It is this aspect of XML that most forces us to re-examine the underlying data model of the database, because, this aspect means dealing with amorphous and loosely structured data within the same framework that previously only recognized data with completely fixed and repetitive structure.

XML also has two other key characteristics: 1) hierarchically oriented, and 2) language independent. The second characteristic is easy to underestimate – in many ways, it is XML that, for the first time, allows complex data structures to be described, transmitted, and worked with in a way that is not tied to VB, C, Java, SQL, or any other language.

Because so much of the Internet's – even more the new economy's – commercial content will be sent around in XML form, it has already become necessary for any serious database to be able to work with XML records in a high performance fashion. However, the implications of XML run much deeper, and the real win is to proceed from simple shredding and construction of XML documents, to having an underlying type system that can deal with complex records directly. In the chicken-egg sweepstakes, it is the complex records that come first as a fundamental need.

XML has undone 20 years of relational orthodoxy in one feel swoop. Since the early 80's we have been trained, indoctrinated, even brainwashed to believe that databases – both the business entity and the underlying engine – should revolve around fully normalized rows and tables. In this overly simplified world, purchase orders cannot all be represented in one place, hierarchies are handled in a round about way, and any real complexity in data structure is banished forever. All of this has created a real hunger at two levels

The database cognoscenti – most ISV's and MIS shops – have learned to work around the limitations of rows and tables. To be fair, fully normalized representations have a lot of advantages including very high degrees of concurrency and superb update consistency behavior. On the other hand, when SAP, for example, finally gave in and fully normalized their database, it grew by a factor of three and slowed down by a factor of seven. Unfortunately the alternative was storing the data in the database in a form opaque to the underlying engine (eg not rows and tables) and therefore giving up on queries, reports, and all the other advantages of the modern database. So, the inner circle, which is huge in size, will also be hugely relieved when they finally regain the freedom to express complex data structures directly, as they were able to in the Codasyl days It is XML that they will have to thank for regaining their design freedom. The fact is, though, that the win for all those who aren't in the inner circle is even larger.

Far too many people believe that relational databases actually cannot even handle

David Vaskevitch

version of: January 13, 2001

hierarchies and other complex structures at all. That is this is patently – and ridiculously – false is most demonstrated that literally even serious commercial application built around a database, revolves around hierarchies, and many of them: reporting relationships, charts of accounts, bills of materials, and on and on. Yet, after twenty years of arguing, and not really winning the arguments, surely the better part of valor, is to simply add complex data structures, including hierarchies, back into databases.

So, XML to the core, means building databases that natively consume, emit and work with XML. This means query processors that can find XML documents, transaction engines that allow sequences of changes to be completed as a whole or not at all, sophisticated backup, everything a database is about. Most of all though, it means not just tacking XML on, but having it permeate to the very center of the database That permeating has three basic implications binary representation, document storage, and a rich new type system.

Of course XML can, and has to be able to be, represented in text form. However, the idea that, as a consequence of moving to XML we should start storing all records, natively, in text form, is ridiculous. XML is a self-description format, and it can work equally well in a text-only world, and in a binary-only world. If we are to be serious about XML we need to automate, in a transparent fashion, all the mappings from text to binary and back This of course points back to the type system.

XML is about not just complex documents, but highly variable, even amorphous documents. Being XML to the core means a database which smoothly handles the full continuum from documents with no repeating structure, to database collections in which the structure is both fully known and completely repetitive This means both having an XML cache, but even more important, extending the database to handle collections of documents as well as collections of records – this is the "relational file system".

Relational File System

The RFS starts with incredible BLOB support, which allows arbitrary documents, pictures, streams, to be stored in the database. BLOB support is mostly about plumbing – backup, transactions, Quality of Service guarantees (to avoid jerky music or videos). Just BLOB's is not enough though.

Win32FS compatibility is the second major requirement and one worth laboring long and hard over. There are two basic ways to get there 1) leap back out to the file system itself, and 2) implement the compatibility in the database itself. I favor the second because once the work is done, most other integration related aspects will work so transparently. It is a lot of work though, in terms of both programming and testing.

In the process of implementing document / stream support, we really need to think through the kinds of streams people will be saving and what indexing, compression, backup, and retrieval capabilities are required. For pictures, ideally lossless compression, with fractal based interpolation for scaling up of prints is both an opportunity and a requirement. For sounds, the whole area of compression and fidelity is wide open for innovation. Indexing, too raises some interesting challenges, some examples are described in the companion paper. The point is, are we even serious about supporting pictures, sounds, videos, and other documents, in a way that will really take us a whole step – or two – past the file system.

It is the handling of *directories* that really makes it the "relational file system". At the simplest level, it is absolutely, totally key that all directories be nothing more nor less than database tables. It is here though that life really gets interesting. The simple way to handle directories is to represent them in the database, the catalog, and stop there. Even at this level this implies an ability to query against combinations of document meta-data and commercial data. However, given that we will not ship for at least a year or two, stopping with mere tables will be a sin. Our metadata model

needs to reflect the web world our users now live in.

The RFS must model links – as first class and queryable objects -- annotations, again really through from a data model perspective, private webs, and more. It is this thinking through of the larger world that the relational file system lives in that will really set it apart. Are we doing this?

XML-Relational Data Model

If there is one core, defining feature of the fifth data model revolution, this is it Just as IMS was defined by hierarchies, Cullinet by graphs, Oracle by rows and tables, ODI by object graphs – if we cause a new revolution we will be associated with a rich new data model that for the first time, keeps the simplicity and power of tables, while reaching out accommodate the complex data structures the world is so hungry for.

Adding just five new features to the underlying relational algebra, extends it to a new algebra, which is provably complete enough to handle arbitrary data structures and graphs:

1. **GUID's** allow all records and documents to have a unique id. At one level we have already implemented this in Sphinx; the trick is to finish adding all the new kinds of indexes that allow a record to be found as quickly as possible, given only its GUID. Are we doing this?

2. **Pointers** start with GUID value fields, but for efficiency require swizzling, link fixup, and other optimizations to be implemented. What we really need is a complete theory, we are committed to, and implementing around, of "marks, identifiers, bookmarks, and paths" Pointers, and pointer based structures live in four worlds:
   a. Database based pointers which point only within the confines of a structured set of records.
   b Mapped Structures allow a set of database records to accurately represent the same graph as represented across a set of objects or in a memory based set of data structures.
   c. XML Pointers extend XML past hierarchies into networks and graphs.
   d. Links are also represented in the meta data / table environment of the surrounding database This means links have two representations.

3 **Entity Valued Attributes** are also often called hierarchically embedded tables. They allow hierarchies and other recursively embedded structures to be represented directly in the database. In theory it is possible to represent EVA's logically, physically, or both. I believe both representations are essential, but XML makes that whole issue go away; it certainly requires a physical representation

4. **Abstract Data Types** (optional). Years ago everybody thought that ADT's were the center of rich data type support in database. It turns out though that most developers only need a small number of atomic level extra data types, places are at the head of any list. So, while for completeness, at some point we will want to implement ADT's, if we defer it by one or two releases, that should not be an issue given the richness introduced the rest of the type system.

Implementing the XML-Relational type system fully is a big job. It is, however the foundation for representing rich data structures, once and for all, and by being the first to focus on really extending the type system we get a variety of advantages including performance, flexibility and thought leadership.

Implementing the type system means a new storage manager, a new query syntax (or more than one?), substantial improvements to the query processor (transitive closure to begin with); it touches a great deal of the core of the database. It really is doing to Oracle what they (and IBM) did to Cullinet in the early 80's Are we doing it? Soon? Seriously. Completely?

In Memory Database

Are we serious about XML, the XML-relational model, and the new type system? If so, then we better be damn serious about imdb. Rich data structures, all pointing to each other, call out for in memory representation. And, as memory gets essentially free, as machines have unbelievable amounts of memory it will become essentially just incompetent to not implement imdb behavior and features. All of a sudden most of the database can be in memory most of the time. Why not be able to access all the memory-resident data directly, without intervening api's, at least for reads? A write barrier is required for updates to ensure security and transactional semantics, but that too can be made transparent.

Once upon a time, before memory and processors became quite what they are today, the way to do an imdb was as a separate engine; a lot of work and a lot of complexity achieving synchronization. However, suppose we start with the assumption that there is a SQLserver in every node of a network[1] – every server, every notebook, every desktop.

Each SQLserver has a cache, a buffer manager – generalize it in three ways. First have the cache support 65 bit addressing and very large caches; this is a requirement for performance anyway. Second provide a mechanism so that "external" applications can access the data in the cache directly, natively, with no api's. In an ideal world this would all happen with and through the CLR / URT, but we should not link the two things because we can't wait too long to get this done. Finally think through a variety of synchronization mechanisms, based around souped up replication, for keeping many caches in synch.

The first benefit of this strategy is that, by definition, both applications and the QP operate against the one and same imdb. Second, integrity of local updates is automatically guaranteed because the local SQLserver (there always is one, remember) has a transaction log, backing store, etc. Finally, piggybacking on replication, even is significant new modes are required, represents a huge synergy.

IMDB, as we look out two years, is more than an optional feature. It is a key performance optimization, which we can't afford not to have. But, most of all, it is the engine feature that makes the rich data type system really fully useable.

Object Relational Mapping

For all of our talk about disconnected datasets – certainly a valuable feature – talk to GPSI or to our own Pace developers, brings out a single and immediate truth: they work with the database pretty directly, and one way or another, use it through their own data structures. As the finishing touch to the XML-relational type revolution, we need to implement rich object relational mapping between CLS / URT objects and structures and the underlying "extended tabular" representation that will appear in the database.

Even in the world of the XNL-relational model, databases will still consist largely of collections of records with repeating structure. In our new world the repeating structures can be far more complex; for example entire orders can be represented in a single record. Normalization will still be common, but now it will be a design decision rather than a limitation imposed by the underlying engine. However, even in that new world, there will be two fundamental representations of data, **both of which will generally exist in the same application and often the same computer**: collections of records, and graphs of objects. The point of the mapping system is to allow both representations to exist without the programmer having to write all the mapping code.

The mapping system is the first and most obvious place where the CLS /URT can really come together in a big way that supports all our languages. Are we doing this?

World Models

For all of its vaunted ability to organize data, a database doesn't really do diddley-squat to organize data for us. It's all left to the

[1] At least for me, original credit for this idea goes to Dave Lomet.

application programmer. So, for example, even today, our hard disks are a mess with dozens of three letter file extensions that nobody knows Our willingness to implement world models gets to the core, the essence, of whether SQLserver stays "just a database" or becomes more of the application platform Essentially, start by asking: do we really want most people to store all their data, all their information in our database? If the answer is yes, we better provide more than slightly better backup.

The completion of the five elements already introduced here is a meta model that describes the structure and relationships of the most core data that goes on every disk, the most core data that is central to all applications. This means two basic models: social and operating environment.

The social model includes basic concepts about people, addresses, places, times, and events If we are implementing a new version of Outlook, not to mention a CRM system on this, then we should have our heads examined. Equally though, the social model, the work required to develop the social model, is much of the work required to "migrate" Exchange and AD to SQLserver in a way that makes sense.

Be clear: if we simply port AD and Exchange to SQLserver without radically rethinking their data models – hard work indeed – we will kill AD, kill Exchange, kill SQLserver and subtract instead of adding advantage to customers. There is a pony under there, but the cost really is a rethinking of the data model which in turn will lead to replacing quite a lot of code.

The operating environment model finally allows us to really organize all the documents on our disk Thousands of photographs, hundreds of songs, thousands of mail messages, hundreds of applications components? The operating environment model, with its built in notion of "collections" provides default order to all this chaos.

The operating environment model goes on to define rich notions of links, private webs, annotations, discussions, discussion threads. All of this should be an intrinsic part of storage; now we will make it so. But, again, the question comes up. are we doing this?

Becoming An Application Platform

The challenge we face is finally make databases truly relevant to everybody and every application. This means transforming, morphing the database from a technical engine focused on transactional and commercial applications, to a far more general engine that, among others things, can be at the center of a next generation operating system. More to the point, it means setting out to have the database really play a larger role in organizing information of all kinds and in all situations.

It's our decision. Do we want to create the next revolution, fundamentally change the definition of the term database? So, others can start keeping up with us? Or do we want to stick to improving databases as we all know them today, and continue slowly catching up with everybody else?

One thing is for sure, without the rich type system, and powerful extensions described here, it just will not be possible to have the database become the place where all data is stored. We can try. but without the right underlying fundamentals, all that will happen is that we will become discouraged. And, since the fundamentals are so exciting, the answer is: let's just do them.

Are we doing this?