Predictions and Challenges for Database Systems in the Year 2000

Patricia G. Selinger

Data Base Technology Institute, IBM Almaden Research Center, San Jose, CA 95120,
pat@almaden.ibm.com

Abstract We have passed the two decade mark for the relational model, and it has been more than 15 years since the first relational prototypes and products. I would like to share with you my views on where I believe DB technology is, and make some personal predictions about where, technically, DB systems will be in the year 2000.

What we know today is that many factors work together to shape database products. These include:

- Application requirements
- Platform capability and cost
- Demands of our customers' businesses
- I/O device capability and cost
- Standards and architectures such as X/OPEN and ISO SQL

1. Introduction

With those factors shaping how database systems evolve, you might have the impression that databases will become the center of the universe! Not true!

Let's face facts -- you and I are focussed on database so that our database customers and the people who build applications on top of our database systems don't have to be. They shouldn't need to know what's inside the engine and they shouldn't have to look under the hood.

In fact, people should think about database systems very much like the way they think about telephones. So why is a database like a telephone? Or more precisely, why are database systems ***going*** to be like telephones?

1. First, they are going to be increasingly intuitive and easy to use. Ideally, no manual is required in order to use one.

2. Next they will connect to anything anywhere across the world.

3. They will connect to one another using recognized standards, as telephone systems do.

4. They will be ubiquitous, in homes, stores, warehouses, doctor's offices, airplanes, ... in almost every country on earth.

Proceedings of the 19th VLDB Conference
Dublin, Ireland 1993

5. And, they will have industrial strength -- almost never break, will be reliable, consistent.

6. Finally they will have growing functionality in dramatically different marketplaces than in the past -- just as phone systems add call-waiting, phone mail, interactive television viewing, and so on.

Let's go through each of these areas individually, discussing my views of where the database industry is in general and where I see it going.

2. Ease of use

Ease of use has many aspects, from convenient, intuitive user interfaces to use of standards for portability and ease of vendor selection. Performance is also an ease of use issue -- if it allows you to ask more powerful questions then ever before and get the answer in what the customer thinks is a reasonable amount of time.

And finally and also very important, no system can be easy to use if it is full of errors, loses your data, and it takes forever for service people to help you with the problem. So quality and service are important to ease of use as well.

What does ease of use mean in the year 2000? Ease of use involves user interfaces that are intuitive and easy to learn for novices as well as efficient for advanced users. Every successful database system must have a rich set of tools and applications. And these applications will be what sells database engines. Furthermore, tools will make using that database engine efficient.

It's unlikely that "one size fits all" for user interfaces to databases so we should expect that many end users will access database systems not through SQL directly but through query applications with menus and point-and-click selection mechanisms and through specialized, vertically integrated applications.

This leads to my first prediction:

Prediction 1: By the year 2000, databases will feature a layer of middleware on top of what we think of today as the database engine, providing the look and feel of a single advanced function database across many data sources.

This prediction has its basis in history. In the past, each decade has added a higher layer on top of previous database layers, and we should expect by the year 2000 that yet another layer will be added.

In the 1960's applications stored data in file systems whose main purpose was to store data for specific applications. In the 1970's, applications began to share their data files using an integrating layer, thus making

the first true databases such as IMS [IMS]. These databases were navigational in nature and applications explicitly followed the data layout of records using commands such as "get next under parent". These databases provided not only centralized storage management, but also transaction management, recovery facilities in the event of failure and system-maintained access paths.

In the 1980's another layer on top of this navigational functionality was added in commercial systems using the relational data model [Codd71]. The advantages brought by this layer were data independence and productivity through use of a non-procedural language such as SQL [ASTR75]. Generally speaking, the relational language layer was established on top of a more navigational layer similar in function to those of the 1970's navigational database systems. The mapping from a non-procedural language such as SQL to the lower level navigational interface is provided by a relational compiler or interpreter which performs parsing, semantic checking, access path selection, and maintains a catalog of the database contents. This added relational layer then drives the underlying navigational layer to execute the user request.

As we move through the decade of the 1990's, we can expect to see a layer of middleware on top of these 1980's databases engines. What problem is this layer solving? What kind of performance and functionality will it have?

In my opinion, this added middleware layer is a natural evolution of the relational database compiler technology invented during the decade of the 1980's. As this compiler technology matures, it can expand to play several new roles:

- It will provide **relational extensions** to satisfy more diverse application needs. This will open up opportunities to use databases for applications that traditionally have not exploited databases. Image, medical, document management, engineering design, and multimedia are examples of such applications. Both performance and functionality will play key roles in determining how useful these relational extensions will be in these non-traditional areas. The functionality provided at the user interface from this new layer can be signficantly extended above what the underlying data sources can provide. For example, such a layer is capable of supporting recursion or it can apply user-supplied aggregation functions such as standard deviation that might not be understood at lower levels.

- Just as the 1970's databases provided an integrated view of data for many applications, this 1990's layer can provide yet a further degree of integration -- this time what we might call **data source transparency**. By data source transparency, I mean that the request can be independent of the data-providing source in terms of the language syntax and semantics of that data. This middleware layer will have knowledge of the semantics and interfaces of a variety of data sources, e.g. DB2, Oracle, IMS, and will provide the semantic and syntactic mapping of the portion of the original request that refers to data at a particular data source. So a single original user request will be accepted by this new layer and divided into possibly multiple individual requests to many heterogeneous data sources using intelligence about the capabilities and data stored at those sources. After the individual data source requests execute, the new layer will also "re-integrate" the results from all the contributing sources back to the user. In other words, the layer provides the look and feel of a single virtual database across all these data sources.

- This layer will also provide **high performance** for the execution of user requests. Sophisticated optimization will be used to separate the original query into pieces targeted for individual data sources whose content and order of execution are optimal. Furthermore, according to customized profiles of users and workloads, the layer will determine if it is valuable to execute the request in parallel. If so, it will apply smart semantic processing in order to determine if parallel processing is feasible for that request.

- This layer can provide **translations among platform data representations and national language character sets**. For example, a German client might request data from a French document database to be joined with Swiss document catalogs in a separate database, with the answer to be returned to the client so that the client can display the results in its native (German) character set. The new layer will minimize number of data representation conversions in order to achieve this.

- Finally the new layer will provide the linkage and toolkit interfaces to allow for **cross-platform, cross-product system management**. Not only will it be necessary in the year 2000 to provide multi-data source system management (two phase commit [Gray78] and global deadlock detection for example) but also management across a variety of operating systems, transaction monitors, and applications. One example is workload balancing that will require information and participation not only from the database engines running on a platform, and the transaction monitors driving the workload, but also the applications and operating system(s) running on those platforms. Any workload balancing that does not include information and participation from all these sources will be inferior to one that does.

I predict that in the year 2000, such capabilities will be found in this kind of **middleware** layer above the database engine. This layer will shelter users from the functional capabilities of data sources and from the language dialects of the interfaces to such data sources. Applications using such a middleware layer can be portable and relatively independent of the functionality of the data sources, since the middleware can supply any functionality that is missing in the lower levels (possibly with less performance). The challenge, of course, is to provide such a middleware layer with extremely low overhead for requests that can be mapped directly to lower levels with no added semantics or translation.

3. Standards

What will not change by the year 2000 is that all these software layers will continue to transform these ever fancier and more complex user requests into standard interfaces supported by data sources.

By use of standard interfaces both at the database interface, (that is SQL89, and the various levels of SQL92) and also between clients and servers (such as CLI), it will be possible to achieve a rational freedom of choice of components. Think about it -- we're going to have millions of applications, tens of databases using tens of communication protocols running on tens of hardware and software platforms. No one wants to ***have*** to buy these all from one vendor because multi-vendor solutions won't work with one another. No customer ***wants*** to have only a limited choice of applications, and no application vendor ***wants*** to limit his marketplace to only a single database or single hardware platform. Being able to mix and match through standard interfaces will provide tremendous freedom of choice and market opportunity.

As these tools and applications become the principal means for end users to access the database, I predict that several things will happen:

1. **Prediction 2: SQL will remain the "Esperanto" for relational database systems.**

   That is, regardless of how rich the user interface of the application is, it will still map down to a version of an SQL database programming interface. Further, to allow for freedom of choice to mix and match applications and database systems, customers will ***demand*** that those database system vendors who also offer applications such as query managers ***decouple*** those applications from their databases. This can most easily be done not by proprietary gateways but by mapping the applications to Standard SQL.

2. **Prediction 3: the SQL interface itself must grow richer over time to accommodate increasingly complex database requests, database structures, and database operations.**

3. Processing these complex high function requests efficiently will demand highly intelligent cost-based SQL compilers. This in turn leads to the following prediction:

   Prediction 4: Database vendors will compete with one another on the quality of their SQL optimizers and compilers.

4. Performance

As database systems provide more function and performance, it simply entices people to use a database for more applications, which in turn increases their appetite for MIPS. We've all seen the numbers -- uni-processor MIPS are growing at 40% per year, and hardware packaging with SMPs multiplies this effect. At the same time, DASD access times (which are the only parameter that is relevant from a database system performance view) are only improving at 6% per year. So, while databases are still primarily CPU-bound, with the exception of artificially small transactions of the TPC-B class, this may soon no longer be true.

How are database vendors going to address this growing gap?

Prediction 5: Database systems in the year 2000 will have a strong repertoire of techniques to defer or avoid I/O in favor of significantly more in-memory processing.

To achieve this, first we need to exploit higher capacity memory chips that bring us very large main memories. IBM's ES9000 expanded store is one example; they can be exploited by database techniques such as IMS Fast Path. The more data you hold in memory, the better.

Next you can expect to see database systems adding many more in-memory processing techniques [KolWe93]. One such example is the addition in 1989 of a third join technique to DB2 on MVS. This new hybrid join [CHHIM91] exploits in-memory sorting, large memories, sequential prefetch I/O into large buffer pools and defers I/Os. Another example of new processing techniques is parallel asynchronous I/O.

Besides providing new processing techniques, database systems will also have to defer I/O's or avoid them altogether. One example is in IBM's DB2 on MVS, where a technique called index ANDing [MHWC90] is used. If a single table has 5 predicates matching 5 indexes, all the record-ids can be retrieved from the 5 indexes and only the records appearing in all 5 lists need to be retrieved. So no I/O's to data pages are done until the record has been qualified through all 5 indexes. Furthermore, such a record list can be sorted into an order that makes the eventual I/O's to data pages as efficient as possible. Of course, this requires special locking techniques to ensure that records don't change values between when the index is checked and when the data is fetched. These are the kind of techniques that must become more common in order to bridge the increasing MIPS to I/O gap.

We can also expect to have improved buffer management techniques to defer or avoid I/Os [TeGu84]. Besides having larger buffer pools, I expect to see much more sophisticated buffer manager intelligence -- e.g. detection of sequential processing which in turn triggers prefetching, tuning of prefetching amounts according to system behavior, extra-wide merge streams using parallel I/O for sorting, dynamic partitioning of tables to narrow the range for table scans [CrHT90], hints from the optimizer to the buffer pool manager to predict whether a given page in the buffer pool is likely to be used again.

What about architectural solutions to the MIPS to I/O gap? One obvious example is parallel architectures. In addition, we might see substantial use of non-volatile memory -- perhaps for main memory databases, also increasingly larger disk controller caches to reduce read and write latency for disk I/O's. Another alternative is offloading function to the disk controller -- doing predicate processing and index searching outboard of the main processor -- these are really special forms of parallelism.

5. Parallelism

Parallelism typically comes in two forms -- shared disk or shared nothing. With shared disk, every processor can directly access data on every disk. This architecture has many advantages in terms of workload balancing and system availability. Shared disk is somewhat complex in that locking must be global, and a naive implementation of global locking can have considerable overhead. Furthermore since pages can be in the buffer pools of many different processors at the same time, coherency protocols [MoNa91] (rather like MP cache coherency) must be used to make sure that every application always sees the newest consistent data. Fast inter-processor communication and specialized hardware can help achieve efficient implementations for these requirements.

A second popular architecture for parallelism is shared nothing -- where every processor has a portion of the entire database, and operations on that data must be executed on that processor. This architecture is very scalable, but needs hard work to do a good job of capacity planning, database design, and system management.

Different customers see different advantages to parallel architectures.

Some see price/performance as an advantage, by exploiting microprocessor MIPS. IBM has already indicated that they expect to be building 390 processors using CMOS, which will put them on the same technology and cost curve as workstations. Because of this, we can expect that in the future, price/performance will not be a significant advantage of parallel systems.

But there will still be many other reasons to be interested in parallel architectures. One of these is capacity or throughput, applying more MIPS to the same data than can be found in a single box. This kind of parallelism is ***inter-query*** -- allowing you to cash millions of checks simultaneously. In this usage of a parallel architecture, an application runs on one box, and if the parallel database architecture is shared disk, then the database operations all run on that same box as well. This is the architecture of choice for heavy duty OLTP applications, such as banking, insurance, inventory management, retail sales....

If, however, the workload is complex query or decision support, or monthly summary reports, or database mining, ... then response time for a given request is more important than throughput. In this case parallel database systems can break up (through intelligent optimization) a user request into many pieces and each piece can be sent to a processor to be executed in parallel. The response time is then the time for the slowest piece plus the time to put the answer back together for the user. The database optimizer must be very sophisticated in order to be able to break up requests into approximately equal pieces. This ***intra-query*** parallelism is also very useful for batch processing, particularly if the time available for batch processing (the so-called batch window) is shrinking.

A fourth advantage of parallel architectures is availability. If one processor fails in a shared disk complex then all processing can continue, with slightly longer response times. If one processor fails in a shared nothing architecture, then only the data attached to the failed processor is inaccessible, and there are a number of techniques such as multi-ported disks and declustering that can make this data available at the cost of some system complexity.

6. Parallel Challenges for the Year 2000

Prediction 6: By the year 2000, success in parallel databases will require solutions to the following problems:

1. **Workload balancing, both static and during execution**
2. **Optimization that is query-formulation independent**
3. **Coexistence of batch, complex query, and OLTP**
4. **"Just in time" execution plan scheduling**
5. **Single system image for system management and service**

Let's discuss what I believe is needed for success in each of these areas.

One particularly difficult area is that of load balancing. To do a good job, you need not only cost-based optimization, but also sophisticated data statistics collection and semantic processing to know all the alternative ways of processing the data. Further, with complex queries, there are so many parallel alternatives, you will need efficient ways to prune the unreasonable choices quickly. Techniques like simulated annealing, the AB technique [Swly93], and iterative improvement will be essential. All of these techniques must, to be successful, must outperform exhaustive search optimization above 10 or 15 way joins in selecting access paths while still being within a few percent of the optimal plan. Extremely sophisticated techniques will be needed to estimate evenly sized pieces of work for N parallel processors.

Having done the best job you can in chopping up the work into equal pieces, you will not always succeed. So during execution, the database system needs to detect when things are imbalanced, and institute fixes (like combining small pieces of work and further chopping up pieces that were too big). And doing this on the fly by redistributing data and re-routing messages. Also, if a query runs for an hour, many of system parameters change during that time, so the workload imbalance detector needs to know how to adapt to those changes from both the database and system environment viewpoint. I see the beginnings of some of these techniques today but it is likely to take 5-10 years of improvement to get them right.

Another challenge is to have a query compiler that chooses the best plan independent of how the query was expressed. I have seen examples where the same query expressed two different ways resulted in more than an order of magnitude difference in execution time. This is not just an issue for parallelism but is true for

any type of complex query which has many possible execution plans. Parallelism just brings ***additional*** and more ***visible*** opportunities for error.

Another very hard problem for parallelism is making complex query and batch processing coexist on the same data with OLTP, which has very different ideal system operating points and different demands on system tuning. Again, this challenge is not unique to the parallel environment, but the demands of parallelism will exaggerate the problem. Solutions include invention of new concurrency techniques such as unlocked reads and versioning.

Sophisticated optimization techniques will be needed to chop up a user's request into an execution tree of parallel operations. Think of tilting this execution tree on its side, with the leaves indicating the first operations and the root of the tree being the "return to user" operation. Naively you would start all the leaves at the same time, and at each operation wait until all the input operations are done. Instead, you can be very clever and do what "just in time" manufacturing operations do -- know enough about the work being done at each step of the plan to do "just in time" operation scheduling. With this scheduling, all the start times for the inputs to an operation are staggered so that they all complete at exactly the same time, which is just the right start time for the next operation.

Simply coupling together a number of processors is not enough to make a parallel database system. Such a system should ideally be as easy to install, run, backup, restore, diagnose problems, add applications, do database design, perform service and maintenance, and so on as a single system. This requires automation -- so parallel systems will be challenged to offer a single point of control. Processor dropout and rejoining and adding new processors are just one example of required parallel system management.

Providing solutions to these challenges will be required for success in parallel database systems.

7. Changing Database Environments

Let's return for a moment to our telephone analogy. Telephones are found everywhere. They offer world-wide connectivity regardless of national language or the individual architecture or brand name of the phone on the other end. They also offer increasingly rich functionality. Telephones also take advantage of standards to be able to interoperate with any other telephone in the world.

Similarly, databases will be exploited in many environments in the year 2000 and will offer many new functions and features. The combination of new environments and new application areas will pressure database technology to invent new functionality. Some examples are massive data support, heterogeneous connectivity, and high availability. At the same time, the natural evolution of database technology will provide the technology push to make databases usable for a much broader range of applications.

Prediction 7: Application "pull" and technology "push" together will shape database environments in the year 2000 on systems ranging in size from palmtops to teraflops.

I expect that within the next five years or so, we will see the frequent use of personal mobile databases that work disconnected -- in delivery vans, used by business people on airplanes, for example. These will "catchup" to the mother database when reconnected -- e.g. updating calendars, exchanging email, reconciling inventories, and so on. Sophisticated merging techniques and reconciliation policies will be needed when versions conflict.

Standalone databases, will likely continue, but many of them will eventually be connected in workgroups and become clients to LAN and WAN-based servers.

If we make databases sufficiently cheap, easy to use, reliable, and high performance, we will find them everywhere. They will increasingly replace file systems in enterprise use across a variety of networks, machines, and communication protocols. We should also expect increasing use of inter-enterprise database processing -- even today manufacturers can monitor levels of parts and automatically invoke transactions at suppliers to trigger deliveries of parts that are running low. I don't expect this to go beyond triggering transactions at other databases. It is my opinion that we will not see multi-enterprise joins or interactive query between companies. I don't imagine that Sears is going to allow Bank of America to browse its customer and billing databases. It ***will*** however, allow the bank to invoke the PAY THE BILL transaction across a wide area multi-enterprise network.

Multi-enterprise distributed databases will need more sophisticated transaction models -- both nested transactions, federated databases come into play as part of what some people call workflow management. This workflow is a linked series of separate transactions to several enterprises, with logic to drive their invocation and compensation transactions (such as "cancel order") to achieve an overall meta-transaction ("have enough umbrellas to sell on rainy days").

The technologies needed are multi-enterprise transaction models, common protocols, authorization, authentication, application control distributed among independent enterprises, mechanisms for coordinating activities across this scope of work that spans many of today's short transactions. Compensation transactions will be required to cancel the effects of previous short transactions under workflow manager control. Mechanisms for verification, problem diagnosis, and recovery of these workflow transactions will also be essential. It should also be possible to track the the status of these meta-transactions. For example, an inventory clerk should be able to ask "what is the status of today's umbrella order?"

8. Connectivity

What kind of connectivity do database customers need?

The answer is access to whatever data the customer's business needs in order to compete and make a profit.

Generally, this means that the database needs to interoperate with all other databases and platforms and

applications the customer's business needs. And it needs to do this in a timely way.

Timeliness is key to success. A classic example is women's fashions. They are seasonal (3 months), the goods have a short shelf-life, a high turn-over and are very profitable. It makes a big difference to catch on to trends quickly. So if you could predict exactly what clothing people will buy during a certain week at a certain store, then you can reduce unwanted store inventory and lower capital outlay for the same profit. Analyzing store sales data by product on a daily basis can make a 2-3% difference in margin -- and in a business where the margin is typically 4%, this is a significant business success factor.

What technology is needed to achieve timely interoperability as described above? Real-time access to production point of sale information, database mining for analysis to detect trends immediately, high performance, and multi-vendor database connectivity, cooperation among heterogeneous clients and servers. This kind of interoperability is best done using standard interfaces such as ANSI SQL, and standard transaction management, such as XA, communications and distributed data access protocols.

Another example illustrates interoperability. Mechanical CAD design automation in certain heavy machinery industries is done with world-wide teams wherever the experts with certain skills reside. So world-wide groupware is needed, with realtime changes of parts -- not three month integration intervals. Concurrent re-engineering with one-day turnaround for updating parts is what's required. What technologies are needed? They include realtime remote data access, high bandwidth interactions to remote databases, lots of MIPS to do the engineering analysis, and support of a variety of cooperating heterogeneous databases.

Prediction 8: In the year 2000, what we database vendors need to do is deliver value with information -- the right information to the right place from any source at the right time.

9. Industrial Strength

It's very clear what value customers receive from connectivity, but what about industrial strength? In the telephone system, industrial strength means that

- Whenever you want to use it, it's there. 24 hours a day, 7 days a week. No one ever takes the phone system down for maintenance between 2 AM and 5 AM.
- It almost always works even if pieces of it are broken. When the Loma Prieta earthquake hit, the phone system still worked.
- You can install a new telephone without major disruption.
- When someone else adds new facilities and new systems, yours still work. There is no world-wide upgrade that everyone needs to install on Dec. 31 at midnight GMT in order to allow the old facilities to be upgraded to communication with the new ones.
- And finally to use a telephone, you don't need to know what kind of complicated technology is being used. The complex system design and engineering are hidden from end users.

10. Industrial Strength Challenges in the Year 2000

Prediction 9: The challenges database systems in the year 2000 will have to meet to be successful are:

- **Reliability** -- getting the right answer all the time, always achieving total integrity -- indexes matching data, catalogs accurate, data pages recovered properly after media failures. This also includes getting predictable, consistent results, and preserving transaction consistency regardless of system and network architecture.

  To achieve this what is needed? -- defect-free database systems, and beyond that no error-prone operator or command interfaces. New consistency techniques -- yet to be invented -- are needed. Systems of the future will need self-checking and self-repairing techniques -- e.g. the AS/400 [AnCo88] knows when it's not feeling well and it automatically captures symptoms of its problem and phones up the service center to transmit the problem description, and often can receive the fix electronically. We database folks have to do similar things -- not just for database problem diagnosis but for vertically integrated solutions. Some facilities exist now; more are going to be expected. That is a challenge for us -- automatic fault detection and correction.

- **Continuous operations** -- never having any planned shutdowns for anything -- not to re-organize the data, not for installing new versions of database software, not for catastrophic disk media failures -- nothing. What customers are challenging us to do is to support their mission critical applications. My children and yours are riding in cars and airplanes that are designed with databases I helped write; we database vendors need to make our database systems as bullet-proof as possible. And if some pieces break, the system needs to degrade gracefully.

- **Automated system management.** System management needs to be on-line, and non-disruptive. On-line incremental backup facilities are an example. But again more is needed. For example, feedback systems to allow databases to be self-tuning -- I gave the example earlier of how DB2's buffer pool decides prefetching quantities by looking at recent buffer behavior. On-line automatic checking and repair of data and index pages are other examples. What about self-reorganizing databases at fine levels of granularity?

- **High concurrency mechanisms that allow query and OLTP on the same data** (not prior versions) at the same time without interference.

Generally speaking mainframe systems are ahead of workstation systems in providing industrial strength, clearly every product needs to focus on these issues.

11. High Availability

Customers who have mission critical data and applications, whose business depends on their database working correctly and continuously, focus resources on disaster recovery. Banks, insurance companies, the US Federal Reserve, and anyone who has spent time waiting in line because "the computer is down" all want to have continuous service and no loss of data. Ideally, if it were free, all companies would want to have immediate recovery after an outage, with no loss of data, and no loss of service. Ideally, the person at the ATM machine never notices that the bank's data center just blew up. This is on customers minds a lot after the Loma Prieta earthquake, Hurricane Andrew, the World Trade Center, and similar disasters [Gray86] [BuTr90]. In reality, this doesn't come for free. In fact, there is a range of requirements -- you would like to specify that for example, it is OK to lose the $10 check but not the one million dollar check -- even in the same database on the same day. I'm told that if the Fed can't complete the reconciliation in the US banking system that happens between 4 and 5 pm, it actually affects the US GNP.

Prediction 10: Databases in the year 2000 will have to provide the following capabilities for high availability:

- Support for **long distance fiber communications**, far enough away to reach a different power grid.
- **Remote backup site** to receive another database's log data and do one of several things:
  - Low budget: catalog and store the log -- hours to days to recover
  - Moderate: apply the log to a copy of the production database as it arrives in real time (30-60 min. to get back in production)
  - Deluxe: above plus keeping backup communication links to clients so that production can resume in a few minutes
- **Worst case failure recovery** -- being able to handle media failures at remote site, link failure between production and remote sites (lots of old log data being transmitted once link back in service), and then a failure of production site
- **Fast catchup** -- processing (probably parallel) of arriving log data at the remote site to "catch up" quickly, especially after a link outage. -- to keep the remote site primed and ready.
- **Broad range of hardware architecture support** at both the production and remote site(s) -- parallel, datasharing, hot standby, -- with possibly many logs for the database, not just one.
- **Knobs to adjust data loss and service loss parameters** -- the ability to establish rules like if $ amount > 10K, then immediately send the log off-site; if $ amount <= 10K, then let the log buffer fill up before sending it to the remote site.

12. Functionality

Over the last 3 decades, databases have moved from the back office doing payroll and inventory to the front office -- hotel reservations, retail point of sales, pharmacy prescription records, and so on. Front office data looks a lot like back office data -- short character strings for names and addresses, credit card numbers, account numbers, item numbers, and creeping in every once in a while was text and image data with very simple functionality -- store it away as bunch of bits, give the bits back to me.

More recently, we see customers using databases in previously non-traditional ways -- long-running design applications like mechanical CAD, storing multi-megabyte document files, and demanding greater functionality on not only this new kind of data, but on all data. In fact we will be seeing dramatically new types of data and applications -- medical imaging, video services, document libraries, archaeology, petrochemical exploration ...

Today database vendors offer this functionality in a very layered approach -- applications of many varieties issuing requests to databases which run on file systems in some cases and use services of the operating system. We have database language standards like SQL 92 [MELT93] that are very rich in expressive power on the near-term horizon and SQL3 standards being defined to add more object-oriented features to SQL. It's my guess that database vendors will be offering object-oriented extensions even before they have implemented all the functionality of all levels of SQL92.

Today, added functionality is being defined in product-specific ways -- customer experts are defining the contents of event monitors, triggers, alerts, stored procedures for each of their databases. The languages for these are not necessarily standardized and there is no easy mechanism for sharing definitions between applications on different database instances, particularly if those instances come from different vendors. The SQL3 standards work on object-oriented extensions is a very active area now for many vendors, including IBM, Oracle, and DEC.

Prediction 11: By the year 2000, I expect that today's ad hoc situation will have evolved in much the same way that programming languages have done. In the year 2000 databases will have standardized general mechanisms for providing higher level data abstractions, and more complex operations or methods -- all of these giving the database increased power to model real-world entities and events.

Furthermore, it will be possible to define these real world objects and operations and use them to construct other objects in a type hierarchy with inheritance and subtyping -- and to do in a way that allows sharing between database instances and even between database vendors. For a model of how this might possibly work, look at class libraries sold for C++ systems or Fortran subroutine library packages for matrix manipulation.

We need to be able to do this for extended relational systems in a way that does not limit functionality like set-oriented query and sophisticated access path opti-

mization that bring customers so much value and productivity. And this added capability cannot make our systems slower either.

Expect to see complex structured objects -- made up of value-based relationships between multiple other objects. For example, parts and sub-parts that can be queried separately or as complex assemblies. Queries can be recursive, even on cyclic objects, and can involve aggregation -- e.g. all assemblies from the wing that use more than 5 bolts and no rivets. Queries might also involve reachability on these complex structures -- e.g. all people who are descendants of France's Louis the XIV.

This kind of capability will provide building blocks for user-definable types and operations that can be sold independently of the basic database system and tailored along with constraints and policies for specific application areas like office and engineering.

To keep this added functionality from slowing down systems, I expect to see functions move lower down in today's layered database system environment. The policies and abstractions that today are in individual customer applications will move down into the database system itself. Database systems will increasingly take over functionality previously done only by file systems -- directly manipulating data on disks. Database systems will also take over functionality typical of operating systems -- direct device control, doing its own paging and threads support.

Prediction 12: We are heading towards a year 2000 environment where at every level of the system functionality is being pushed down to achieve higher performance -- resulting in a blurring of the boundaries and a higher performance, higher function, more complex to manage system.

13. Massive Data

Prediction 13: Databases in the year 2000 will need to provide support for massive amounts of data, upwards of 100 terabytes.

As databases support larger and larger amounts of data, not all data will necessarily be on magnetic disks. But customers still want that data ***managed*** by the database system and they want it ***"near-line"*** if not on-line [StFGM93]. The data and applications fall into three categories:

- **Traditional data** -- e.g. a single table of retail sales where the most recent 14 months are on-line on magnetic and the older sales data is on an optical library. Most requests and all OLTP work are done on the recent data on magnetic disk. Auditing, billing disputes, and database mining applications to determine buying trends may access data in both magnetic and optical. Every month another chunk of the magnetic data spills over to be stored in the optical library. What does it take to do this well? An extremely clever access path selector that takes into account what data is stored on what media (in fact, which platter) and makes its plans accordingly. Also, database design tools that recommend how much should be on magnetic and how much on optical for a given workload -- e.g. should indexes be on magnetic and data on optical? Should optical libraries use non-volatile caches?

- **Multi-media.** A typical example is a record which logically consists of some number of coded data fields, e.g. for a hospital patient and then some non-coded data such as the patient's MRI scan. Should the non-coded data be stored in the database -- maybe yes, maybe no, depending on the kind of processing that might be done on it. Today, the answer is typically no -- just store a pointer to the file containing the data. In the year 2000, with spatial data representation and processing techniques such those found in the QBISM project [ACFRT93], the answer is probably yes.

- **Statistical data collections**. Another kind of massive data is today typically not stored in a database at all -- this is data that is almost exclusively used for marketing or statistical analysis. A very good example is census data or customer history files. Here's how an application on this data works today -- suppose you are offering a new product -- a ski parka and want to send out advertisements to the "best 10 thousand prospects" -- you have the customer history files for 10 million people. You sit down with an expert market analysis person and make a profile of best prospects (between 22 and 40, lives in Colorado, bought sweaters from us last year, always pays their bills). You save up similar queries, possibly for different products and run them, perhaps once a week against the terabytes of tapes containing all the customer buying histories. Every record is evaluated for "best fit" against each query. If you had a bad profile -- too many prospects or too few, wait until next week and try again.

  If we database people want to do this job instead of custom applications doing file I/O, we need to support the following:

  - Massive amounts of data -- tera- and petabytes
  - Simultaneous multi-statement optimization in a way that is better than the brute force tape method. This kind of optimization would include finding common sub-expressions to evaluate once and generalizing several queries into a common superset query whose results can be further filtered to get answers to the original queries.

14. Summary

The decade of the 90's will be a very exciting one for databases. Emphasis on performance and ease of use will continue, and there will be growing competition between database vendors on the level of industrial strength and system management they can provide and on the quality of their SQL optimizers and compilers.

There will be a dramatic increase in connectivity and functionality as well, fostered by the evolution of relational database compilers into a layer of middleware that supplies a universal set of functionality across many data sources. This middleware will also provide applications with independence from language and platform dependencies on the individual data sources.

15. References

ASTR75 Astrahan, M. M. and Chamberlin, D. D. *Implementation of a Structured English Query Language*, **Communications of the ACM**, Vol. 18, No. 10, October 1975.

ACFRT93 Arya, M., Cody, W., Faloutsos, C., Richardson, J. and Toga, A. *QBISM: A Prototype 3-D Medical Image Database System*, **Bulletin of the IEEE Computer Society Technical Committee on Data Engineering**, March, 1993.

AnCo88 Anderson, M., Cole, R. *An Integrated Data Base*, In **IBM Application System/400 Technology**, Document Number SA21-9540, IBM, June 1988.

CHHIM91 Cheng, J., Haderle, D., Hedges, R., Iyer, B., Messinger, T., Mohan, C., Wang, Y. *An Efficient Hybrid Join Algorithm: a DB2 Prototype*, **Proc. 7th International Conference on Data Engineering**, Kobe, April 1991.

BuTr90 Burkes, D. and Treiber, R.K. *Design approaches for real-time transaction processing remote recovery*, **Proc. of IEEE Compcon**, IEEE, 1990.

Codd71 Codd, E. F. *A database sublanguage founded on the relational calculus*, **Proc. of the ACM SIGFIDET Workshop on Data Description, Access and Control**, ACM, New York, 1971.

CrHT90 Crus, R., Haderle, D., Teng, J. *Method for Minimizing Locking and Reading in a Segmented Storage Space*, **U.S. Patent 4,961,134**, IBM, October 1990.

Gray78 Gray, J. *Notes on Data Base Operating Systems*, In **Operating Systems - An Advanced Course**, R. Bayer, R. Graham and G. Seegmuller (Eds.), Lecture Notes in Computer Science, Vol 60, Springer-Verlag, 1978.

Gray86 Gray, J. N. *Why do computers stop and what can be done about it*, **Proceedings, 5th Symposium on Reliability in Distributed Software and Database Systems**, January, 1986.

IMS **IMS/ESA V4 General Information Manual**, IBM Corp., GC26-3068-00.

KolWe93 Kolodner, E. and Weihl, W. *Atomic Incremental Garbage Collection and Recover for a Large Stable Heap*, **Proc. SIGMOD 93**, Washington, D.C., 1993.

MELT93 Melton, J. **Understanding the New SQL: A Complete Guide**, Morgan Kaufmann Publishers, 1993.

MHWC90 Mohan, C., Haderle, D., Wang, Y., Cheng, J. *Single Table Access Using Multiple Indexes: Optimization, Execution, and Con Control Techniques*, **Proc. International Conference on Extending Data Base Technology**, Venice,

MoNa91 Mohan, C., Narang, I. *Recovery and Coherency-Control Protocols for Fast Intersystem Page Transfer and Fine-Granularity Locking in a Shared Disks Transaction Environment*, **Proc. 17th International Conference on Very Large Data Bases**, Barcelona, September 1991.

StFGM93 Stonebraker, M., Frew, J., Gardels, K. and Meredith, J. *The Sequoia 2000 Storage Benchmark*, **Proceedings, SIGMOD 1993**, Washington, D.C., 1993.

Swly93 Swami, A., Iyer, B. *A Polynomial Time Algorithm for Optimizing Join Queries* **Proceedings, Data Engineering Conferece**, IEEE Computer Society, April, 1993.

TeGu84 Teng, J., Gumaer, R. *Managing IBM Database 2 Buffers to Maximize Performance*, **IBM Systems Journal**, Vol. 23, No. 2, 1984.