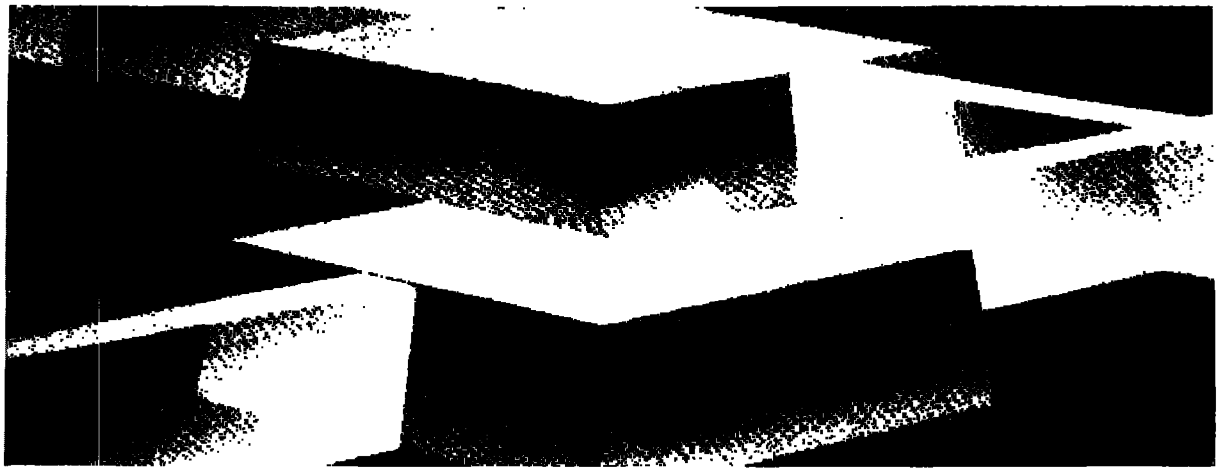

DATABASE SYSTEMS: ACHIEVEMENTS AND OPPORTUNITIES

Avi Silberschatz
Michael Stonebraker
Jeff Ullman
Editors

The history of database system research is one of exceptional productivity and startling economic impact. Barely 20 years old as a basic science research field, database research has fueled an information services industry estimated at $10 billion per year in the U.S. alone. Achievements in database research underpin fundamental advances in communications systems, transportation and logistics, financial management, knowledge-based systems, accessibility to scientific literature, and a host of other civilian and defense applications. They also serve as the foundation for considerable progress in basic science in various fields ranging from computing to biology.

As impressive as the accomplishments of basic database research have been, there is a growing awareness and concern that only the surface has been scratched in developing an understanding of the database principles and techniques required to support the advanced information management applications that are expected to revolutionize industrialized economies early in the next century. Rapid advances in areas such as manufacturing science, scientific visualization, robotics, optical storage, and high-speed communications already threaten to overwhelm the existing substrate of database theory and practice.

In February 1990, the National Science Foundation convened a workshop in Palo Alto, California for the purpose of identifying the *technology pull* factors that will serve as forcing functions for advanced database technology and the corresponding basic research needed to enable that technology. The participants included representatives from the academic and industrial sides of the database research community.[1] The primary conclusions of the workshop participants can be summarized as follows:

1. A substantial number of the advanced technologies that will underpin industrialized economies in the early 21st century depend on radically new database technologies that are currently not well understood, and that require intensive and sustained basic research.
2. Next-generation database applications will have little in common with today's business data processing databases. They will involve much more data, require new capabilities including type extensions, multimedia support, complex objects, rule processing, and archival storage, and will necessitate rethinking the algorithms for almost all DBMS operations.
3. The cooperation among different organizations on common scientific, engineering, and commercial problems will require large-scale, heterogeneous, distributed databases. Very difficult problems lie ahead in the areas of inconsistent databases, security, and massive scale-up of distributed DBMS technology.

This article provides further information about these topics, as well as a brief description of some of the important achievements of the database research community.

Background and Scope

The database research community has been in existence since the late 1960s. Starting with modest representation, mostly in industrial research laboratories, it has expanded dramatically over the last two decades to include substantial efforts at major universities, government laboratories and research consortia. Initially, database research centered on the management of data in business applications such as automated banking, record keeping, and reservation systems. These applications have four requirements that characterize database systems:

- *Efficiency* in the access to and modification of very large amounts of data;
- *Resilience,* or the ability of the data to survive hardware crashes and software errors, without sustaining loss or becoming inconsistent;
- *Access control,* including simultaneous access of data by multiple users in a consistent manner and assuring only authorized access to information; and
- *Persistence,* the maintenance of data over long periods of time, independent of any programs that access the data.

Database systems research has centered around methods for designing systems with these characteristics, and around the languages and conceptual tools that help users to access, manipulate, and design databases.

[1]The workshop was attended by Michael Brodie, Peter Buneman, Mike Carey, Ashok Chandra, Hector Garcia-Molina, Jim Gray, Ron Fagin, Dave Lomet, Dave Maier, Marie Ann Niemat, Avi Silberschatz, Michael Stonebraker, Irv Traiger, Jeff Ullman, Gio Wiederhold, Carlo Zaniolo, and Maria Zemankova.

Database management systems (DBMSs) are now used in almost every computing environment to organize, create and maintain important collections of information. The technology that makes these systems possible is the direct result of a successful program of database research. This article will highlight some important achievements of the database research community over the past two decades, including the scope and significance of the technological transfer of database research results to industry. We focus on the major accomplishments of relational databases, transaction management, and distributed databases.

Today, we stand at the threshold of applying database technology in a variety of new and important directions, including scientific databases, design databases, and universal access to information. Later in this article we pinpoint two key areas in which research will make a significant impact in the next few years: next-generation database applications and heterogeneous, distributed databases.

Accomplishments of the Last Two Decades

From among the various directions that the database research community has explored, the following three have perhaps had the most impact: relational database systems, transaction management, and distributed database systems.

Each has fundamentally affected users of database systems, offering either radical simplifications in dealing with data, or great enhancement of their capability to manage information.

Relational Databases

In 1970 there were two popular approaches used to construct database management systems. The first approach, exemplified by IBM's Information Management System (IMS), has a data model (mathematical abstraction of data and operations on data) that requires all data records to be assembled into a collection of trees. Consequently, some records are *root* records and all others have unique *parent* records. The query language permitted an application programmer to navigate from root records to the records of interest, accessing one record at a time.

The second approach was typified by the standards of the Conference on Data Systems Languages (CODASYL). They suggested that the collection of DBMS records be arranged into a directed graph. Again, a navigational query language was defined, by which an application program could move from a specific entry point record to desired information.

Both the tree-based (called hierarchical) and graph-based (network) approaches to data management have several fundamental disadvantages. Consider the following examples:

1. To answer a specific database request, an application programmer, skilled in performing disk-oriented optimization, must write a complex program to navigate through the database. For example, the company president cannot, at short notice, receive a response to the query "How many employees in the Widget department will retire in the next three years?" unless a program exists to count departmental retirees.
2. When the structure of the database changes, as it will whenever new kinds of information are added, application programs usually need to be rewritten.

As a result, the database systems of 1970 were costly to use because of the low-level interface between the application program and the DBMS, and because the dynamic nature of user data mandates continued program maintenance.

The relational data model, pioneered by E. F. Codd in a series of papers in 1970–72, offered a fundamentally different approach to data storage. Codd suggested that conceptually all data be represented by simple tabular data structures (relations), and that users access data through a high-level, nonprocedural (or declarative) query language. Instead of writing an algorithm to obtain desired records one at a time, the application programmer is only required to specify a predicate that identifies the desired records or combination of records. A query optimizer in the DBMS translates the predicate specification into an algorithm to perform database access to solve the query. These nonprocedural languages are dramatically easier to use than the navigation languages of IMS and CODASYL; they lead to higher programmer productivity and facilitate direct database access by end users.

During the 1970s the database research community extensively investigated the relational DBMS concept. They:

- Invented high-level relational query languages to ease the use of the DBMS by both end users and application programmers. The theory of higher-level query languages has been developed to provide a firm basis for understanding and evaluating the expressive power of database language constructs.
- Developed the theory and algorithms necessary to *optimize* queries—that is, to translate queries in the high-level relational query languages into plans that are as efficient as what a skilled programmer would have written using one of the earlier DBMSs for accessing the data. This technology probably represents the most successful experiment in optimization of very high-level languages among all varieties of computer systems.
- Formulated a theory of *normalization* to help with database design by eliminating redundancy and certain logical anomalies from the data.

- Constructed algorithms to allocate tuples of relations to *pages* (blocks of records) in files on secondary storage, to minimize the average cost of accessing those tuples.
- Constructed buffer management algorithms to exploit knowledge of access patterns for moving pages back and forth between disk and a main memory buffer pool.
- Constructed indexing techniques to provide fast associative access to random single records and/or sets of records specified by values or value ranges for one or more attributes.
- Implemented prototype relational DBMSs that formed the nucleus for many of the present commercial relational DBMSs.

As a result of this research in the 1970s, numerous commercial products based on the relational concept appeared in the 1980s. Not only were the ideas identified by the research community picked up and used by the vendors, but also, several of the commercial developments were led by implementors of the earlier research prototypes. Today, commercial relational database systems are available on virtually any hardware platform from personal computer to mainframe, and are standard software on all new computers in the 1990s.

There is a moral to be learned from the success of relational database systems. When the relational data model was first proposed, it was regarded as an elegant theoretical construct but implementable only as a toy. It was only with considerable research, much of it focused on basic principles of relational databases, that large-scale implementations were made possible. The next generation of databases calls for continued research into the foundations of database systems, in the expectation that other such useful "toys" will emerge.

Transaction Management

During the last two decades, database researchers have also pioneered the transaction concept. A transaction is a sequence of operations that must appear "atomic" when executed. For example, when a bank customer moves $100 from account *A* to account *B*, the database system must ensure that either both of the operations—Debit *A* and Credit *B*—happen or that neither happens (and the customer is informed). If only the first occurs, then the customer has lost $100, and an *inconsistent* database state results.

To guarantee that a transaction transforms the database from one consistent state to another requires that:

1. The concurrent execution of transactions must be such that each transaction appears to execute in isolation. *Concurrency control* is the technique used to provide this assurance.
2. System failures, either of hardware or software, must not result in inconsistent database states. A transaction must execute in its entirety or not at all. *Recovery* is the technique used to provide this assurance.

Concurrent transactions in the system must be synchronized correctly in order to guarantee that consistency is preserved. For instance, while we are moving $100 from *A* to *B*, a simultaneous movement of $300 from account *B* to account *C* should result in a net deduction of $200 from *B*. The view of correct synchronization of transactions is that they must be *serializable;* that is, the effect on the database of any number of transactions executing in parallel must be the same as if they were executed one after another, in some order.

During the 1970s and early 1980s the DBMS research community worked extensively on the transaction model. First, the theory of serializability was worked out in detail, and precise definitions of the correctness of schedulers (algorithms for deciding when transactions could execute) were produced. Second, numerous concurrency control algorithms were invented that ensure serializability. These included algorithms based on

- Locking data items to prohibit conflicting accesses. Especially important is a technique called *two-phase locking,* which guarantees serializability by requiring a transaction to obtain all the locks it will ever need before releasing any locks.
- Timestamping accesses so the system could prevent violations of serializability.
- Keeping multiple versions of data objects available.

The various algorithms were subjected to rigorous experimental studies and theoretical analysis to determine the conditions under which each was preferred.

Recovery is the other essential component of transaction management. We must guarantee that all the effects of a transaction are installed in the database, or that none of them are, and this guarantee must be kept even when a system crash loses the contents of main memory. During the late 1970s and early 1980s, two major approaches to this service were investigated, namely:

Write-ahead logging. A summary of the effects of a transaction is stored in a sequential file, called a log, before the changes are installed in the database itself. The log is on disk or tape where it can survive system crashes and power failures. When a transaction completes, the logged changes are then posted to the database. If a transaction fails to complete, the log is used to restore the prior database state.

Shadow file techniques. New copies of entire data items, usually disk pages, are created to reflect the effects of a transaction and are written to the disk in entirely new locations. A single atomic action remaps the data pages, so as to substitute the new versions for the old when the transaction completes. If a transaction fails, the new versions are discarded.

Recovery techniques have been extended to cope with the failure of the stable medium as well. A backup copy of the data is stored on an entirely separate device. Then, with logging, the log can be used to roll forward the backup copy to the current state.

Distributed Databases

A third area in which the DBMS research community played a vital role is distributed databases. In the late 1970s there was a realization that organizations are fundamentally decentralized and require databases at multiple sites. For example, information about the California customers of a company might be stored on a machine in Los Angeles, while data about the New England customers could exist on a machine in Boston. Such data distribution moves the data closer to the people who are responsible for it and reduces remote communication costs.

Furthermore, the decentralized system is more likely to be available when crashes occur. If a single, central site goes down, all data is unavailable. However, if one of several regional sites goes down, only part of the total database is inaccessible. Moreover, if the company chooses to pay the cost of multiple copies of important data, then a single site failure need not cause data inaccessibility.

In a multidatabase environment we strive to provide location transparency. That is, all data should appear to the user as if they are located at his or her particular site. Moreover, the user should be able to execute normal transactions against such data. Providing location transparency required the DBMS research community to investigate new algorithms for distributed query optimization, concurrency control, crash recovery, and support of multiple copies of data objects for higher performance and availability.

In the early 1980s the research community rose to this challenge. Distributed concurrency control algorithms were designed, implemented and tested. Again, simulation studies and analysis compared the candidates to see which algorithms were dominant. The fundamental notion of a two-phase commit to ensure the possibility of crash recovery in a distributed database was discovered. Algorithms were designed to recover from processor and communication failures, and data patch schemes were put forward to rejoin distributed databases that had been forced to operate independently after a network failure. Technology for optimizing distributed queries was developed, along with new algorithms to perform the basic operations on data in a distributed environment. Finally, various algorithms for the update of multiple copies of a data item were invented. These ensure that all copies of each item are consistent.

All the major DBMS vendors are presently commercializing distributed DBMS technology. Again we see the same pattern discussed earlier for relational databases and transactions, namely aggressive research support by government and industry, followed by rapid technology transfer from research labs to commercial products.

The Next Challenges

Some might argue that database systems are a mature technology and it is therefore time to refocus research onto other topics. Certainly relational DBMSs, both centralized and distributed, are well studied, and commercialization is well along. Object management ideas, following the philosophy of object-oriented programming, have been extensively investigated over the last few years and should allow more general kinds of data elements to be placed in databases than the numbers and character strings supported in traditional systems. The relentless pace of advances in hardware technology makes CPUs, memory and disks drastically cheaper each year. Current databases will therefore become progressively cheaper to deploy as the 1990s unfold. Perhaps the DBMS area should be declared solved, and energy and research efforts allocated elsewhere.

We argue strongly here that such a turn of events would be a serious mistake. Rather, we claim that solutions to the important database problems of the year 2000 and beyond are not known. Moreover, hardware advances of the next decade will not make brute force solutions economical, because the scale of the prospective applications is simply too great.

In this section we highlight two key areas in which we feel important research contributions are required in order to make future DBMS applications viable: Next-generation database applications and heterogeneous, distributed databases.

In addition to being important intellectual challenges in their own right, their solutions offer products and technology of great social and economic importance, including improved delivery of medical care, advanced design and manufacturing systems, enhanced tools for scientists, greater per capita productivity through increased personal access to information, and new military applications.

The Research Agenda for Next-Generation DBMS Applications

To motivate the discussion of research problems that follows, in this section we present several examples of the kinds of database applications that we expect will be built during the next decade.

1. For many years, NASA scientists

have been collecting vast amounts of information from space. They estimate that they require storage for 10^{16} bytes of data (about 10,000 optical disk jukeboxes) just to maintain a few years' worth of satellite image data they will collect in the 1990s. Moreover, they are very reluctant to throw anything away, lest it be exactly the data set needed by a future scientist to test some hypothesis. It is unclear how this database can be stored and searched for relevant images using current or soon-to-be available technology.

2. Databases serve as the backbone of computer-aided design systems. For example, civil engineers envision a facilities-engineering design system that manages all information about a project, such as a skyscraper. This database must maintain and integrate information about the project from the viewpoints of hundreds of subcontractors. For example, when an electrician puts a hole in a beam to let a wire through, the load-bearing soundness of the structure could be compromised. The design system should, ideally, recalculate the stresses, or at the least, warn the cognizant engineer that a problem may exist.
3. The National Institutes of Health (NIH) and the U.S. Department of Energy (DOE) have embarked on a joint national initiative to construct the DNA sequence corresponding to the human genome. The gene sequence is several billion elements long, and its many subsequences define complex and variable objects. The matching of individuals' medical problems to differences in genetic makeup is a staggering problem and will require new technologies of data representation and search.
4. Several large department stores already record every product-code-scanning action of every cashier in every store in their chain. Buyers run ad-hoc queries on this historical database in an attempt to discover buying patterns and make stocking decisions. This application taxes the capacity of available disk systems. Moreover, as the cost of disk space declines, the retail chain will keep a larger and larger history to track buying habits more accurately. This process of "mining" data for hidden patterns is not limited to commercial applications. We foresee similar applications, often with even larger databases, in science, medicine, intelligence gathering, and many other areas.
5. Most insurance firms have a substantial on-line database that records the policy coverage of the firm's customers. These databases will soon be enhanced with *multimedia* data such as photographs of property damaged, digitized images of handwritten claim forms, audio transcripts of appraisers' evaluations, images of specially insured objects, and so on. Since image data is exceedingly large, such databases will become enormous. Moreover, future systems may well store video walk-throughs of houses in conjunction with a homeowners policy, further enlarging the size of this class of databases. Again, applications of this type are not limited to commercial enterprises.

These applications not only introduce problems of size, they also introduce problems with respect to all conventional aspects of DBMS technology (e.g., they pose fundamentally new requirements for access patterns, transactions, concurrency control, and data representation). These applications have in common the property that they will push the limits of available technology for the foreseeable future. As computing resources become cheaper, these problems are all likely to expand at the same or at a faster rate. Hence, they cannot be overcome simply by waiting for the technology to bring computing costs down to an acceptable level.

We now turn to the research problems that must be solved to make such next-generation applications work. Next-generation applications require new services in several different areas in order to succeed.

New Kinds of Data

Many next-generation applications entail storing large and internally complex objects. The insurance example, (5) requires storage of images. Scientific and design databases often deal with very large arrays or sequences of data elements. A database for software engineering might store program statements, and a chemical database might store protein structures. We need solutions to two classes of problems: data access and data type management.

Current databases are optimized for delivering small records to an application program. When fields in a record become very large, this paradigm breaks down. The DBMS should read a large object only once and place it directly at its final destination. Protocols must be designed to *chunk* large objects into manageable size pieces for the application to process. A new generation of query languages will be required to support querying of array and sequence data as will mechanisms for easily manipulating disk and archive representations of such objects. In addition, extended storage structures and indexing techniques will be needed to support efficient processing of such data.

A second class of problems concerns type management. There must be a way for the programmer to construct the types appropriate for his or her application. The need for more flexible type systems has been one of the major forces in the development of object-oriented databases. One of the drawbacks of the systems developed so far is that

Many of the new applications will involve primitive concepts not found in most current applications.

type-checking is largely dynamic, which lays open the possibility that programming errors tend to show up at run time, not during compilation. In order to provide the database application designer with the same safety nets that are provided by modern high-level programming languages, we must determine how we can combine static type disciplines with persistent data and evolution of the database structure over time.

Rule Processing

Next-generation applications will frequently involve a large number of rules, which take *declarative* ("if *A* is true, then *B* is true"), and *imperative* ("if *A* is true, then do *C*") forms. For example, a design database should notify the proper designer if a modification by a second designer may have affected the subsystem that is the responsibility of the first designer. Such rules may include elaborate constraints that the designer wants enforced, triggered actions that require processing when specific events take place, and complex deductions that should be made automatically within the system. It is common to call such systems knowledge-base systems, although we prefer to view them as a natural, although difficult, extension of DBMS technology.

Rules have received considerable attention as the mechanism for triggering, data mining (as discussed in the department store example), and other forms of reasoning about data. Declarative rules are advantageous because they provide a logical declaration of what the user wants rather than a detailed specification of how the results are to be obtained. Similarly, imperative rules allow for a declarative specification of the conditions under which a certain action is to be taken. The value of declarativenes in relational query languages like SQL (the most common such language) has been amply demonstrated, and an extension of the idea to the next generation of query languages is desirable.

Traditionally, rule processing has been performed by separate subsystems, usually called expert system shells. However, applications such as the notification example cannot be done efficiently by a separate subsystem, and such rule processing must be performed directly by the DBMS. Research is needed on how to specify the rules and on how to process a large rule base efficiently. Although considerable effort has been directed at these topics by the artificial intelligence (AI) community, the focus has been on approaches that assume all relevant data structures are in main memory, such as RETE networks. Next-generation applications are far too big to be amenable to such techniques.

We also need tools that will allow us to validate and debug very large collections of rules. In a large system, the addition of a single rule can easily introduce an inconsistency in the knowledge base or cause chaotic and unexpected effects and can even end up repeatedly firing itself. We need techniques to decompose sets of rules into manageable components and prevent (or control in a useful way) such inconsistencies and repeated rule firing.

New Concepts in Data Models

Many of the new applications will involve primitive concepts not found in most current applications, and there is a need to build them cleanly into specialized or extended query languages. Issues range from efficiency of implementation to the fundamental theory underlying important primitives. For example, we need to consider:

Spatial Data. Many scientific databases have two- or three-dimensional points, lines, and polygons as data elements. A typical search is to find the 10 closest neighbors to some given data element. Solving such queries will require sophisticated, new multidimensional access methods. There has been substantial research in this area, but most has been oriented toward main memory data structures, such as quad trees and segment trees. The disk-oriented structures, including K-D-B trees and R-trees, do not perform particularly well when given real-world data.

Time. In many exploratory applications, one might wish to retrieve and explore the database state as of some point in the past or to retrieve the time history of a particular data value. Engineers, shopkeepers, and physicists all require different notions of time. No support for an algebra over time exists in any current commercial DBMS, although research prototypes and special-purpose systems have been built. However, there is not even an agreement across systems on what a "time interval" is; for example, is it discrete or continuous, open-ended or closed?

Uncertainty. There are applications, such as identification of features

Next-generation applications often aim to facilitate collaborative and interactive access to a database.

from satellite photographs, for which we need to attach a likelihood that data represents a certain phenomenon. Reasoning under uncertainty, especially when a conclusion must be derived from several interrelated partial or alternative results, is a problem that the AI community has addressed for many years, with only modest success. Further research is essential, as we must learn not only to cope with data of limited reliability, but to do so efficiently, with massive amounts of data.

Scaling Up

It will be necessary to *scale* all DBMS algorithms to operate effectively on databases of the size contemplated by next-generation applications, often several orders of magnitude bigger than the largest databases found today. Databases larger than a terabyte (10^{12} bytes) will not be unusual. The current architecture of DBMSs will not scale to such magnitudes. For example, current DBMSs build a new index on a relation by locking it, building the index and then releasing the lock. Building an index for a 1-terabyte table may require several days of computing. Hence, it is imperative that algorithms be designed to construct indexes incrementally without making the table being indexed inaccessible.

Similarly, making a dump on tape of a 1-terabyte database will take days, and obviously must be done incrementally, without taking the database off line. In the event that a database is corrupted because of a head crash on a disk or for some other reason, the traditional algorithm is to restore the most recent dump from tape and then to roll the database forward to the present time using the database log. However, reading a 1-terabyte dump will take days, leading to unacceptably long recovery times. Hence, a new approach to backup and recovery in very large databases must be found.

Parallelism

Ad-hoc queries over the large databases contemplated by next-generation application designers will take a long time to process. A scan of a 1-terabyte table may take days, and it is clearly unreasonable for a user to have to submit a query on Monday morning and then go home until Thursday when his answer will appear.

First, imagine a 1-terabyte database stored on a collection of disks, with a large number of CPUs available. The goal is to process a user's query with nearly *linear speedup.* That is, the query is processed in time inversely proportional to the number of processors and disks allocated. To obtain linear speedup, the DBMS architecture must avoid bottlenecks, and the storage system must ensure that relevant data is spread over all disk drives. Moreover, parallelizing a user command will allow it to be executed faster, but it will also use a larger fraction of the available computing resources, thereby penalizing the response time of other concurrent users, and possibly causing the system to thrash, as many queries compete for limited resources. Research on multiuser aspects of parallelism such as this one is in its infancy.

On the other hand, if the table in question is resident on an archive, a different form of parallelism may be required. If there are no indexes to speed the search, a sequential scan may be necessary, in which case the DBMS should evaluate as many queries as possible in parallel, while performing a single scan of the data.

In general, it remains a challenge to develop a realistic theory for data movement throughout the memory hierarchy of parallel computers. The challenges posed by next-generation database systems will force computer scientists to confront these issues.

Tertiary Storage and Long-Duration Transactions

For the foreseeable future, ultra large databases will require both secondary (disk) storage and the integration of an *archive* or tertiary store into the DBMS. All current commercial DBMSs require data to be either disk or main-memory resident. Future systems will have to deal with the more complex issue of optimizing queries when a portion of the data to be accessed is in an archive. Current archive devices have a very long latency period. Hence, query optimizers must choose strategies that avoid frequent movement of data between storage media. Moreover, the DBMS must also optimize the placement of data records on the archive to minimize subsequent retrieval times. Finally, in such a system, disk storage can be used as a read or write cache for archive objects. New algorithms will be needed to manage intelligently the buffering in a three-level system.

The next-generation applications often aim to facilitate collaborative and interactive access to a database. The traditional transaction model discussed earlier assumes that transactions are short—perhaps a fraction of a second. However, a designer may lock a file for a day, during which it is redesigned. We need entirely new approaches to maintaining the integrity of data, sharing data, and recovering data, when transactions can take hours or days.

Versions and Configurations

Some next-generation applications need *versions* of objects to represent alternative or successive states of a single conceptual entity. For instance, in a facilities engineering database, numerous revisions of the electric plans will occur during the design, construction and maintenance of the building, and it may be necessary to keep all the revisions for accounting or legal reasons. Furthermore, it is necessary to maintain consistent configurations, consisting of versions of related objects, such as the electrical plan, the heating plan, general and detailed architectural drawings.

While there has been much discussion and many proposals for proper version and configuration models in different domains, little has been implemented. Much remains to be done in the creation of space-efficient algorithms for version management and techniques for ensuring the consistency of configurations.

Heterogeneous, Distributed Databases

There is now effectively one worldwide telephone system and one worldwide computer network. Visionaries in the field of computer networks speak of a single worldwide file system. Likewise, we should now begin to contemplate the existence of a single, worldwide database system from which users can obtain information on any topic covered by data made available by purveyors, and on which business can be transacted in a uniform way. While such an accomplishment is a generation away, we can and must begin now to develop the underlying technology in collaboration with other nations.

Indeed, there are a number of applications that are now becoming feasible and that will help drive the technology needed for worldwide interconnection of information:

- Collaborative efforts are underway in many physical science disciplines, entailing multiproject databases. The project has a database composed of portions assembled by each researcher, and a *collaborative* database results. The human genome project is one example of this phenomenon.
- A typical defense contractor has a collection of subcontractors assisting with portions of the contractor project. The contractor wants to have a single project database that spans the portions of the project database administered by the contractor and each subcontractor.
- An automobile company wishes to allow suppliers access to new car designs under consideration. In this way, suppliers can give early feedback on the cost of components. Such feedback will allow the most cost-effective car to be designed and manufactured. However, this goal requires a database that spans multiple organizations, that is, an *intercompany* database.

These examples all concern the necessity of logically integrating databases from multiple organizations, often across company boundaries, into what appears to be a single database. The databases involved are heterogeneous, in the sense that they do not normally share a complete set of common assumptions about the information with which they deal, and they are distributed, meaning that individual databases are under local control and are connected by relatively low-bandwidth links. The problem of making heterogeneous, distributed databases behave as if they formed part of a single database is often called interoperability. We now use two very simple examples to illustrate the problems that arise in this environment:

First, consider a science program manager, who wishes to find the total number of computer science Ph.D. students in the U.S. There are over 100 institutions that grant a Ph.D. degree in computer science. We believe that all have an on-line student database that allows queries to be asked of its contents. Moreover, the NSF program manager can, in theory, discover how to access all of these databases and then ask the correct local query at each site.

Unfortunately, the sum of the responses to these 100+ local queries will not necessarily be the answer to his overall query. Some institutions record only full-time students; others record full- and part-time students. Furthermore, some distinguish Ph.D. from Masters candidates, and some do not. Some may erroneously omit certain classes of students, such as foreign students. Some may mistakenly include students, such as electrical engineering candidates in an EECS department. The basic problem is that these 100+ databases are *semantically inconsistent.*

A second problem is equally illustrative. Consider the possibility of an electronic version of a travel assistant, such as the Michelin Guide. Most people traveling on vacation consult two or more such travel guides, which list prices and quality ratings for restaurants and hotels. Obviously, one might want to ask the price of a room at a specific hotel, and each guide is likely to give a different answer. One might quote last year's price, while another might indicate the price with tax, and a third might quote the price including meals. To answer the user's query, it is necessary to treat each value obtained as *evidence,* and then to provide *fusion* of this evidence to form a best answer to the user's query.

To properly support heterogeneous, distributed databases, there is a difficult research agenda that must be accomplished.

Browsing

Let us suppose that the problems of access have been solved in any one of the scenarios mentioned earlier. The user has a uniform query language that can be applied to any one of the individual databases or to some merged view of the collection of databases. If an inconsistency is detected, or if missing information appears to invalidate a query, we cannot simply give up. There must be some system for explaining to the user how the data arrived in that state and, in particular, from what databases it was derived. With this information, it may be possible to filter out the offending data elements and still arrive at a meaningful query. Without it, it is highly unlikely that any automatic agent could do a trustworthy job. Thus, we need to support browsing, the ability to interrogate the structure of the database and, when multiple databases are combined, interrogate the nature of the process that merges data.

Incompleteness and Inconsistency

The Ph.D. student and travel advisor examples indicate the problems with semantic inconsistency and with data fusion. In the Ph.D. student example there are 100+ disparate databases, each containing student information. Since the individual participant databases were never designed with the objective of interoperating with other databases, there is no single *global schema* to which all individual databases conform. Rather, there are individual differences that must be addressed. These include differences in units. For example, one database might give starting salaries for graduates in dollars per month while another records annual salaries. In this case, it is possible to apply a conversion to obtain composite consistent answers. More seriously, the definition of a part-time student may be different in the different databases. This difference will result in composite answers that are semantically inconsistent. Worse still is the case in which the local database omits information, such as data on foreign students, and is therefore simply wrong.

Future interoperability of databases will require dramatic progress to be made on these semantic issues. We must extend substantially the data model that is used by a DBMS to include *much* more semantic information about the meaning of the data in each database. Research on extended data models is required to discover the form that this information should take.

Mediators

As the problems of fusion and semantic inconsistency are so severe, there is need for a class of information sources that stand between the user and the heterogeneous databases. For example, if there were sufficient demand, it would make sense to create a "CS Ph.D. mediator" that could be queried as if it were a consistent, unified database containing the information that actually sits in the 100+ local databases of the CS departments. A travel adviser that provided the information obtained by fusing the various databases of travel guides, hotels, car-rental companies, and so on, could be commercially viable. Perhaps most valuable of all would be a mediator that provided the information available in the world's libraries, or at least that portion of the libraries that are stored electronically.

Mediators must be accessible by people who have not had a chance to study the details of their query language and data model. Thus, some agreement regarding language and model standards is essential, and we need to do extensive experiments before standardization can be addressed. Self-description of data is another important research problem that must be addressed if access to unfamiliar data is to become a reality.

Name Services

The NSF program manager must be able to consult a national name service to discover the location and name of the databases or mediators of interest. Similarly, a scientist working in an interdisciplinary problem domain must be able to discover the existence of relevant data sets collected in other disciplines. The mechanism by which items enter and leave such name servers and the organization of such systems is an open issue.

Security

Security is a major problem (failing) in current DBMSs. Heterogeneity and distribution make this open problem even more difficult. A corporation may want to make parts of its database accessible to certain parties, as did the automobile company mentioned earlier, which offered preliminary design information to potential suppliers. However, the automobile company certainly does not want the same designs accessed by its competitors, and it does not want any outsider accessing its salary data.

Authentication is the reliable identification of subjects making database access. A heterogeneous, distributed database system will need to cope with a world of multiple authenticators of variable trustworthiness. Database systems must be resistant to compromise by remote systems masquerading as authorized users. We foresee a need for mandatory security and research into the analysis of covert channels, in order that distributed, heterogeneous database systems do not increase user uncertainty about the security and integrity of one's data.

A widely distributed system may have thousands or millions of users. Moreover, a given user may be identified differently on different systems. Further, access permission might be based on role (e.g., current company Treasurer) or access site. Finally, sites can act as intermediate agents for users, and data may pass through and be manipulated by these intervening sites. Whether an access is permitted may well be influenced by who is acting on a user's behalf. Current authorization systems will surely require substantial extensions to deal with these problems.

Site Scale-up

The security issue is just one element of scale-up, which must be addressed in a large distributed DBMS. Current distributed DBMS algorithms for query processing, concurrency control, and support of multiple copies were designed to function with a few sites, and they must all be rethought for 1,000 or 10,000 sites. For example, some query-processing algorithms expect to find the location of an object by searching all sites for it. This approach is clearly impossible in a large network. Other algorithms expect all sites in the network to be operational, and clearly in a 10,000 site network, several sites will be down at any given time. Finally, certain query-processing algorithms expect to optimize a join by considering all possible sites and choosing the one with the cheapest overall cost. With a very large number of sites, a query optimizer that loops over all sites in this fashion is likely to spend more time trying to optimize the query than it would have spent in simply executing the query in a naive and expensive way.

Powerful desktop computers, cheap and frequently underutilized, must be factored into the query optimization space, as using them will frequently be the most responsive and least expensive way to execute a query. Ensuring good user response time becomes increasingly difficult as the number of sites and the distances between them increase. Local caching, and even local replication, of remote data at the desktop will become increasingly important. Efficient cache maintenance is an open problem.

Transaction Management

Transaction management in a heterogeneous, distributed database system is a difficult issue. The main problem is that each of the local database management systems may be using a different type of a concurrency control scheme. Integrating these is a challenging problem, made worse if we wish to preserve the local autonomy of each of the local databases and allow local and global transactions to execute in parallel.

One simple solution is to restrict global transactions to retrieve-only access. However, the issue of reliable transaction management in the general case, where global and local transactions are allowed to both read and write data, is still open.

Conclusion

Database management systems are now used in almost every computing environment to organize, create and maintain important collections of information. The technology that makes these systems possible is the direct result of a successful program of database research. We have highlighted some important achievements of the database research community over the past two decades, focusing on the major accomplishments of relational databases, transaction management, and distributed databases.

We have argued that next-generation database applications will little resemble current business data processing databases. They will have much larger data sets, require new capabilities such as type extensions, multimedia support, complex objects, rule processing, and archival storage, and they will entail rethinking algorithms for almost all DBMS operations. In addition, the cooperation between different organizations on common problems will require heterogeneous, distributed databases. Such databases bring very difficult problems in the areas of querying semantically inconsistent databases, security, and scale-up of distributed DBMS technology to large numbers of sites. Thus, database systems research offers

- A host of new intellectual challenges for computer scientists, and
- Resulting technology that will enable a broad spectrum of new applications in business, science, medicine, defense, and other areas.

To date, the database industry has shown remarkable success in transforming scientific ideas into major products, and it is crucial that advanced research be encouraged as the database community tackles the challenges ahead.

Acknowledgments

Any opinions, findings, conclusions, or recommendations expressed in this report are those of the panel and do not necessarily reflect the views of the National Science Foundation. Avi Silbershatz and Jeff Ullman initiated the workshop and coordinated and edited the report. Mike Stonebraker provided the initial draft and much of the content for the final report. Hewlett-Packard Laboratories hosted the workshop. Postworkshop contributors to this report include Phil Bernstein, Won Kim, Hank Korth, and Andre van Tilborg.

The workshop was supported by NSF Grant IRI-89-19556.