sequential or by specified key). The data segments themselves were linked together in networks or hierarchies, depending upon the anticipated usage patterns. For example, a customer account record might contain the customer name, the billing address, the credit rating, and a pointer to the set of orders made by that customer. The orders record might contain the order number, the salesman's name, the total, the desired delivery date, and a pointer to the set of items ordered. This is an example of data stored as a hierarchy. If the database management system supported the network model[7] of data storage, the salesman record might contain the salesman's name, the commission earned, and a pointer to the orders he took, thereby providing two ways of accessing orders records: from the salesman who took them or from the customer who gave them.

In the mid-1970s, modern database management systems emerged.

Access to data in these second-generation database management systems required two layers (at least) of software: (1) a user-written application program with database calls and application-specific processing logic, and (2) the database management system with general-purpose database management facilities to define, manipulate, and protect the data. The application programs still ran in batch mode and performed processing needed by the "back office" of the enterprise. Some of the processing logic written in first-generation data management applications (e.g., I/O, buffering, concurrency management, recovery from crashes, transaction management, etc.) was taken out of the application and done inside second-generation database management systems. However, applications still had to specify the access path to the data. For example, the application might access the order records by scanning the relevant salesman or customer account records in the example above.

In summary, second-generation data management systems evolved to database management systems by advancing in function and generality over first-generation systems. They featured both sequential data access and access to data stored on disk by key. Application programs issuing database requests, however, still ran in batch mode and specified database requests by navigating among database records. Application programmers using second-generation database management systems gained in productivity by using well-defined, consistent data and also by using the database management facilities to do operations they would have had to code themselves in first-generation data management applications.

Third generation: Database management systems. In the mid-1970s, modern, third-generation database management systems emerged. These systems extended all the features supported by the second generation and also offered interactive access. This interactive capability evolved concurrently with time-sharing operating systems and data communication subsystems. Instead of running in batch mode, an application program can be run interactively and provide immediate answers to a user at a terminal. Several users can run different applications concurrently on the same data, and the database manager must serialize these data manipulations appropriately. Interactive applications consist of one or more screen interactions interspersed with one or more database calls to check credit limits, account balances, etc., and require the database management system to provide fast responses. This makes it possible to automate the "front office" of an enterprise through applications that do on-line reservations, check cashing, and point-of-sale purchases, and then record the transaction in a database.

As one might expect of a maturing technology, third-generation database management systems have also become more specialized for various markets. A user can choose a database management system that makes the right trade-offs (for that user) with respect to function, performance, productivity, and machine resources. Among the IBM database management products, one might choose to use Transaction Processing Facility (TPF),[8] which offers high performance with less flexibility in data storage, or IMS[6] or CICS[9] to access DL/I data (interactive and batch processing, good performance, and navigational access to data), or DB2[10] or SQL/DS[11] (interactive and batch processing, reasonable performance, and nonprocedural access to data). These systems represent three different selections along a continuum of performance versus function. They are also suited to different types of database usage, from repetitive transactions requiring high throughput on sets of records whose format

changes very infrequently, to *ad hoc* queries and data whose format can easily be changed.

IBM's relational database management systems (DB2 and SQL/DS) reflect the new trend in database technology of providing easy access to data for people who are not data processing professionals as well as those who are. Relational databases provide a very simple, tabular view of data. Unlike the earlier hierarchical and network organizations, relational databases derive relationships from the data values rather than having explicit pointers between records.[12] In a relational view of the customer accounts-and-orders example previously given, the order records would contain the account number of the customer who gave them instead of having a pointer in the customer account record pointing to the corresponding orders. Questions like "What is the credit rating of customers in New York who ordered toasters?" are answered by joining together information from the customer account records and the order records by matching the customer account numbers. This simple relational representation of data makes it possible to access data in a nonprocedural way, giving only the names of the files and the search conditions to be checked. IBM's relational database language is called SQL.[13] The SQL version of the query just given is the following:

```
SELECT credit rating, customer name
FROM customer account, orders
WHERE customer account.account number =
        orders.account number
        AND customer account.city = 'New York'
        AND orders.item = 'toaster'
```

This type of nonprocedural database language eliminates the need to specify the access path to the data. The database management system itself chooses the access path by estimating the cost of obtaining the query answer using each possible access path and choosing the cheapest. In our example, the DBMS could find toaster orders, get the associated customer account number, use it to find the relevant customer account record, check whether the city was New York, and return the credit rating field. Or the DBMS could sequentially scan all customer account records, check whether the city was New York, and if so, find the matching orders and check whether the item was a toaster. Depending on the number of customers in New York or the number of toaster orders, one of these methods might be significantly faster than the other. State-of-the-art database management systems keep statistics that help them make a good estimate of the cost of each possible access path. More information on access path selection can be found in Reference 14.

The high-level, nonprocedural access to data offered by relational database management systems relieves one of the major problems facing data processing today: the shortage of data processing professionals and the increasing backlog of applications users would like to implement. It improves the productivity of data processing professionals by simplifying the database calls in the application programs they write. In addition, it makes it possible for non-DP professionals to specify their queries to an interactive query program and receive their answers on a screen or printer, thereby avoiding the development of an application program altogether. The end result is improved productivity and reduced workload for the data processing professional, and increased accessibility to data for end users, including those who are not data processing professionals.

Relational technology has evolved from a research topic to a commercial reality.

State-of-the-art database technology. The third-generation database management systems that are oriented toward high performance are now mature systems, with years of usage and improvements behind them. Their customers have a major investment in applications that use them and in the education of programmers on how to exploit their best features. We can expect usage of these systems to continue to be significant in the future, and we can expect that the systems themselves will continue to evolve and be enhanced.

Relational database technology has been an area for research and development for more than a decade. During this time, the emphasis of research and development work on relational databases has changed from a focus on system internals such as indexing, query optimization, and concurrency to a concentra-

tion on the functional capability of databases. For example, IBM Research has been experimenting with relational database system designs for about fifteen years. These experiments have included the construction of a complete relational data management system, System R,[15,16] a relational database system that is highly available, a relational database management system for storing office and engineering data, a distributed relational database system, and others. A number of other research teams in universities and corporations are also addressing database technology. In summary, relational technology has evolved from a pure research topic to a commercial reality, but its commercial exploitation is still in its infancy, and it remains an active area of research.

Future directions for database technology

What can one expect in the future for database technology? In my view, the evolution of database technology will be influenced by the following three factors:

- Performance-inspired improvements to the basic technology
- New developments in hardware
- Customer requirements for additional function

Performance improvements. One can expect continuing improvements in performance as database technology matures. Performance is always an area where improvements are desired, particularly for frequently executed portions of code. Database management system algorithms and code to retrieve a record, lock a record, compare two fields or a field and a value, log an update, search an index, etc., are always candidates for further optimization. In addition, we continue to invent algorithms that offer better performance (often with additional function besides) than that of the algorithms used by current database management systems. Areas of active research and development of new, improved algorithms include buffer management, recovery, concurrency control, access path management, and query optimization and compilation.

Performance improvements might be achieved with only local changes, or they might require a dramatically different internal database management system architecture. For example, we can improve the performance of relational queries by improving the query optimizer's cost model of a query's buffer utilization. This would be a local change to the query optimizer. Other performance improvements could be attained for all data models by coordinating database algorithms with similar functions in the underlying operating system. Two examples of these more extensive architectural improvements are (1) the provision of "hints" from the access path manager to the I/O subsystem to prefetch the next database pages that will be read, and (2) the coordination of the paging done by the database buffer manager and the operating system virtual memory manager to avoid paging policy conflicts and extra paging I/Os.

Simple database management systems for personal computers will grow in function to rival those of mainframes.

With these and other improvements, one can expect steady and continuing progress in performance because of improved algorithms and techniques. Depending on the maturity of the technology, these performance improvements are more likely to be less than a factor of two, rather than an order of magnitude. I expect larger gains in performance in the less mature relational technology, but improvements are also attainable for other models.

The desire for even better performance in relational database management systems will result in continuing improvement to the basic subcomponents of relational database architectures. IBM has already introduced the concept of compiling queries[16] rather than interpreting them, as is done by most relational and navigational systems. The extra processing done to compile a query is amortized over all the times it executes. Furthermore, the compiled code is so efficient that it is worthwhile to compile even *ad hoc* queries. For the future, we can obtain even better performance by further enhancing the query optimization in relational database management systems. There is still room for more detailed modeling of costs and for enhancing the repertoire of access paths, and the database research and development community is very active in this area.

As functional and performance enhancements are made to existing database management systems, their internal architecture will be modernized and restructured. This will enable them to continue to grow and be enhanced. This will also remove the constraints imposed by previous hardware limitations, such as address space size, limits on the number of terminals that can be connected, and device dependencies on screens or disks. It will position these systems to exploit new I/O devices, multiprocessors, large address spaces, and so forth. This exploitation of new developments in hardware is not being limited to developments prompted by a desire for better performance, as we now discover.

New developments in hardware. Many new functions of database management systems will be motivated by a desire to tolerate or exploit new hardware technologies. Some simple examples are those of restructuring database system architectures to exploit multiple processors, extended addressing, and larger-capacity disks. As described in the preceding section, one can exploit hardware to provide performance improvements. For example, a database management system can process transactions faster by using nonvolatile memory such as random-access memory (RAM) with battery backup to save changes, or by loading parts of the database into very large main memories.

A more complex example of exploiting new developments in hardware is the invention of database management systems for personal computers. Today these systems are rather simple, compared to the rich function of mainframe database management systems. As microprocessors become faster, as memory becomes cheaper, and as small disks grow in capacity, these simple database management systems will grow in function to rival those of mainframes. As local-area networks of workstations become more popular, a requirement for multiple-user database management systems on server workstations will emerge. In addition, mainframe, server, and personal computer databases will need to share data or at least permit data to be extracted from one to the other. Users of personal computers will need access to shared corporate data stored on mainframe or server database management systems, and will want to archive and protect some of their local data on the mainframe. These extraction functions are provided today by a variety of vendors. Most of these systems offer the ability to construct queries, ship them from workstation to host, and receive and store the answer either in a file or in a local DBMS. Some vendors offer the capability of triggering a host DBMS update from a program running on the personal computer.

Virtually no one provides a single-system image of all the data managed by a DBMS running on a personal computer plus all the data managed by a host or server DBMS. Such a system would permit updating both databases in a single transaction and combining in a single query data from both systems, as in a relational join. The issues involved in providing such a single-system image among workstations, servers, and hosts involve challenges in security, naming, query optimization, recovery, and all aspects of control. For example, does it make sense to have a workstation lock items at the host or control when the host part of a transaction commits? What protects data extracted from a securely managed host DBMS onto a personal computer? Can a user on one workstation access data stored on another? Can a user at a host DBMS extract data from a DBMS on a personal computer? What happens if the personal computer is turned off either before or during such an access? Some of these issues are philosophical, and others require technical invention to resolve. These issues are now being addressed in both research and development settings.

New hardware devices and chips can be exploited by database management systems.

New hardware devices and chips can be exploited by database management systems. These include write-once optical disk for low-cost archival storage, read-write optical disk for low-cost storage where performance is not critical, image data storage and retrieval on videodisk or video cassettes, and the utilization of data compression chips.

The invention of microprocessors has made it economically feasible to have computer architectures with intelligent outboard devices such as intelligent disk controllers, database search engines, and special-

purpose, dedicated machines, e.g., for message processing. These intelligent devices will be able to do processing that previously was done by a host database management system. As a result, database architectures will become more modular, with well-defined internal interfaces to accommodate these intelligent devices. Much more performance modeling must be done and experimental prototypes created before we can determine the right functional division between outboard devices and host. One of the key problems is to invent a solution that will not prove obsolete as outboard microprocessors, memory devices, and communications become even faster and cheaper.

Database technology will be most strongly influenced by customer demand.

Database machines are one example of the use of intelligent outboard microprocessors. Many different database machine architectures have been proposed. One promising idea is to ride the cost and performance curves of general-purpose microprocessors and their associated disks, memory, and so forth by linking together multiple microprocessors. Such an architecture could take advantage of the higher volume and, therefore, the lower cost of the microprocessor database machine components. At the same time, a database machine could perform multiple operations in parallel. This parallelism could be exploited by running many small database transactions concurrently or by splitting a single long, complex transaction into many parallel pieces of work. Research in database machine architectures is under way at many research institutions and universities.[17]

Another use for hardware in the database area is special architectures or devices to trade resources for speed. One example is a machine architecture with massive amounts of memory (hundreds or thousands of megabytes) for use as an in-memory database. If such memory had battery backup as well as backup for the backup, one could do away with resource recovery in the database architecture. Even without battery backup, an in-memory database could provide extremely fast database service, because no disk I/O would need to be done for many database requests. The implications of such an architecture are currently under study.

The direction that database technology takes in exploiting developments in hardware is very dependent on the demand for those features from current or new customers, the outlook for which we now discuss.

Customer requirements. Unquestionably, database technology for the future will be most strongly influenced by customer demand. Among the functions that I perceive as needed by customers are storage of nontraditional data, such as engineering or office data, distributed access to data, productivity and usability aids for database management systems, fault tolerance or resilience, continuous operation, and portability. Each of these can place conflicting design requirements on database technology.

Storing nontraditional data. Today, more than one third of all office workers use some form of terminal or workstation to perform their jobs. As more and more office and engineering tasks are computerized, the demand for storing office and engineering data will increase. These data, if automated at all, are usually stored in files, not in database management systems. As workstation and terminal users store more and more of their engineering and office data in a computer, it is likely that they will find it inconvenient to store those data in a file system while their accounting, inventory, and other data processing data are stored in a database management system. These dual storage mechanisms make it difficult to relate the two kinds of data, for example when referring to information in a parts inventory in the database management system while doing an engineering drawing using that part. Users may also wish to query their engineering and office data in the same way as they query their data processing information, rather than learning two separate interfaces. For all these reasons, I expect that database management systems must permit the definition, storage, and manipulation of office and engineering nontraditional data together with traditional data processing data.

This use of nontraditional data significantly affects the architecture and algorithms of database manage-

ment systems. On systems today, data processing records are usually fairly short—tens to hundreds of bytes—whereas engineering drawings might be 25K to 500K bytes, and office documents such as technical reports range in size from very small to very large. Databases today log the old and new values, when a field in a record is updated, but it would be very unwieldy and expensive to store a document as a single very long field and keep an old and a new copy every time a comma is inserted. Storing these nontraditional data requires new recovery and buffering techniques.

End users will be able to create and manage their own databases.

Storing engineering and office data as single long fields also does not provide users with the ability to query portions of the document or drawing. Further, users would like to apply new operations to these data, depending upon the type of data. Some examples are the following: context searching on documents, querying the author field of a document, doing matrix multiplication on engineering data, selecting and updating a segment of a drawing, or querying whether one part of a drawing is "near" or "contains" another. Rather than adding new operations and data types for each new kind of data, current research in database technology[18] is directed toward building an extensible database management system in which users can define new data types and operations and specify new policies for locking, recovering, buffering, and accessing the data.

Distributed access to data. As data processing managers add new equipment to their computer complexes and as users buy and install equipment to manage information, the typical organization of medium or large size will have more than one computing complex with multiple machines per complex, plus a variety of departmental machines and/or individual workstations. Depending on price, different decisions might be made by users or DP managers in any size organization as to whether to buy one large machine or several smaller machines. In each case, an organizational entity might have several machines storing data in database management systems. These data are already distributed. The problem is how users at other sites, other mainframes, or other workstations can both access their local data stored on their local database system and, if suitably authorized, access data stored remotely on similar or different database management systems.

Transparent access to distributed data has been the subject of years of research.[19–22] For mainframe-to-mainframe distributed systems with similar (relational) database management systems, algorithms for distributed query compilation, distributed data definition, distributed deadlock detection, and distributed transaction management have been invented and used in experimental research prototypes. Research is still under way on how workstation database management systems participate in a distributed database management system and how heterogeneous database management systems can cooperate to share data.

Productivity and usability aids. In the future, I expect that many areas of basic database management technology will be enhanced in terms of the functions they provide. Productivity is a key driver for these enhancements. It is easy to predict that interactive query facilities will continue to grow in function and user-friendliness. I expect also, however, that interactive data definition facilities will become easier to learn and use, so that end users will be able to create and manage their own databases.

As more and more persons who do not have data processing skills use database management systems, these systems must become more easy to learn and use for both regular and occasional users. The SQL language is one example of technology to improve both productivity and ease of use. Some vendors offer products that attach very high-level, fourth-generation programming languages to databases for easier application development. More tools are needed, including database design tools, performance tuning aids, on-line tutorial facilities, and helpful on-line coaching in the user's native language when errors are encountered. Also needed are integrated toolkits that package these tools together with report generators, interactive query programs, on-line data dictionaries, special purpose turnkey applications, and other applications, such as spreadsheets that can be customized by casual users.

Fault tolerance and continuous operation. The recovery facilities of today's mainframe database management systems are very sophisticated. When a machine or a subsystem crashes, the database is automatically recovered to the last consistent state, that is, a state with no transactions in progress. These facilities also allow the user or system to request that the changes made by an in-progress transaction be undone. Furthermore, the database log of changes can be reapplied to an old copy of the database to obtain a current version of the database in order to recover from disk media failures.

But what happens if a machine or subsystem fails, and the customer wants to keep on using the database? The solution is to add software and hardware redundancy both inside and outside the database management subsystem to enhance the availability of the data. In this way, subsystem or processor crashes can be detected and applications rerouted to another database system with the same data (or an up-to-date copy of it). Research and development work is under way to improve the technology for both detecting failures and recovering from failures with minimal delay and duplication of resources.[23-25]

This research is aimed at increasing availability in spite of failures. Another issue being addressed is that of planned database shutdowns to archive or reorganize data, to do maintenance on disk drives, etc. One can copy the data on the disk to be maintained or the data to be reorganized, but how can the data simultaneously be copied while normal processing continues? One solution is to prevent normal processing during the copy operation or during the copying needed for archiving the data. This is not acceptable to customers who have automated their front offices and want to continue making sales or reservations 24 hours a day, seven days a week. Clever algorithms are required to permit "fuzzy copies" in which some changes made by a transaction are included and others are not applied to the copy until later. "Catching-up" the off-line disk drive after maintenance also requires similar technology.

Portability. Many customers have several machines from different vendors with different instruction sets running different operating systems. They would like to have the same database management interface in all these environments. Vendors of database management system products would like to implement their systems once and have them run in many different environments. This is not possible in the strictest sense, but it is achieved in practice by a careful database architecture that isolates machine and operating system dependencies to a few code modules that must be rewritten for each new environment. This is achievable, and has been accomplished in newer database management systems. Older systems must be restructured in this way to achieve more portability.

Concluding remarks

We have described the evolution of database technology from the early, tape-oriented, user-written data management systems of the 1950s to the diverse, sophisticated third-generation database management systems of today. Today's systems offer a choice of database management architectures, functions, performance, and productivity characteristics. In the future, we can expect function and performance enhancements driven by customer requirements and new hardware opportunities. Many creative inventions have brought database technology to the place where it is today. Although significant progress has already been made, even more can be expected in the future.

Cited references

1. C. J. Date, *An Introduction to Database Systems*, 3rd Edition (Volumes 1 and 2), Addison-Wesley Publishing Co., Reading, MA (1983).
2. W. C. McGee, "Data base technology," *IBM Journal of Research and Development* **25,** No. 5, 505–519 (1981).
3. J. P. Fry and E. H. Sibley, "Evolution of data-base management systems," *ACM Computing Surveys* **8,** No. 1, 7–42 (1976).
4. *OS/VS2 MVS Data Management Services Guide*, GC26-3875, IBM Corporation (1980); available through IBM branch offices.
5. *OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide*, GC26-3838, IBM Corporation; available through IBM branch offices.
6. *IMS/VS General Information Manual*, GH20-1260, IBM Corporation (1984); available through IBM branch offices.
7. *CODASYL Data Base Task Group, April 1971 Report*, Association for Computing Machinery, New York, 1971.
8. *Airline Control Program/Transaction Processing Facility: An Introduction*, SC20-1909, IBM Corporation (1981); available through IBM branch offices.
9. *Customer Information Control System/Virtual Storage General Information*, GC33-0155, IBM Corporation; available through IBM branch offices.
10. *Database 2 General Information*, GC26-4073, IBM Corporation (1986); available through IBM branch offices.
11. *SQL/DS General Information Manual*, GH24-5012 (for VSE), GH24-5064 (for VM/SP), IBM Corporation (1986); available through IBM branch offices.
12. E. F. Codd, "A database sublanguage founded on the relational calculus," *Proceedings of the ACM SIGFIDET Workshop on*

Data Description, Access, and Control, Association for Computing Machinery, New York, 1971.
13. D. D. Chamberlin, M. M. Astrahan, K. P. Eswaran, P. P. Griffiths, R. A. Lorie, J. W. Mehl, P. Reisner, and B. W. Wade, "SEQUEL 2: A unified approach to data definition, manipulation, and control," *IBM Journal of Research and Development* **20,** No. 6, 560–575 (1976).
14. P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access path selection in a relational database management system," *Proceedings of the ACM SIGMOD International Conference*, Association for Computing Machinery, New York, 1979, pp. 23–34.
15. M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. Griffiths, W. F. King, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, and V. Watson, "System R: Relational approach to database management," *ACM Transactions on Database Systems I*, 97–137 (1976).
16. M. W. Blasgen, M. M. Astrahan, D. D. Chamberlin, J. N. Gray, W. F. King, B. G. Lindsay, R. A. Lorie, J. W. Mehl, T. G. Price, G. R. Putzolu, M. Schkolnick, P. G. Selinger, D. R. Slutz, H. R. Strong, I. L. Traiger, B. W. Wade, and R. A. Yost, "System R: An architectural overview," *IBM Systems Journal* **20,** No. 1, 41–62 (1981).
17. *Advanced Database Machine Architecture*, David K. Hsiao, Editor, Prentice-Hall, Inc., Englewood Cliffs, NJ (1983).
18. R. Lorie and J. J. Daudenarde, *On Extending the Realm of Application of Relational Systems*, Research Report RJ-4973, IBM San Jose Research Laboratory, San Jose, CA (December 1985).
19. M. Stonebraker, "Concurrency control and consistency of multiple copies of data in distributed INGRES," *IEEE Transactions on Software Engineering* **5,** No. 3, 188–194 (May 1979).
20. B. Lindsay, L. Haas, C. Mohan, P. Wilms, and R. Yost, "Computation and communication in R*: A distributed database manager," *ACM Transactions on Computer Systems* **2,** No. 1, 24–38 (February 1984).
21. J. B. Rothnie, P. A. Bernstein, S. Fox, N. Goodman, M. Hammer, T. Landers, C. Reeve, D. Shipman, and E. Wong, "Introduction to a system for distributed databases (SDD-1)," *ACM Transactions on Database Systems* **5,** No. 1, 1–17 (March 1980).
22. C. Mohan, *Tutorial: Recent Advances in Distributed Data Base Management*, IEEE Computer Society Press, Silver Spring, MD, 1984.
23. J. Bartlett, "A non-stop kernel," *8th Symposium on Operating System Principles*, December 1981, pp. 22–29.
24. A. Borr, "Transaction monitoring in Encompass: Reliable distributed transaction processing," *Proceedings of the 7th International Conference on Very Large Databases*, IEEE, September 1982.
25. J. N. Gray, "Why do computers stop and what can be done about it," *Proceedings, 5th Symposium on Reliability in Distributed Software and Database Systems*, January 1986, pp. 3–12.

Patricia Griffiths Selinger *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120-6099.*
Dr. Selinger has worked at the IBM San Jose Research Laboratory since 1975. She was elected to Phi Beta Kappa in 1970, and received her Ph.D. degree in applied mathematics in 1975, her M.A. in 1972, and her B.A. in 1971, all from Harvard University. Dr. Selinger joined IBM Research as a member of the System R research project on relational databases, where she worked on authorization and query optimization. She received an IBM Outstanding Contribution Award for her work on query optimization. From 1978 to 1982, she was the manager of R*, a research project that explored the extending of relational database technology into a distributed database management system. She received an IBM Outstanding Innovation Award for her work on distributed compilation. In 1983, Dr. Selinger became the manager of the Office Systems Laboratory, which does research in the field of advanced technology for the office. She was named to her current position of manager of the Database Technology Institute in 1986. Dr. Selinger has participated on the program committees of the SIGMOD, PODC, and Distributed Database and Computer Networks Conferences, and has served as a member of the CSNET Executive Committee and as vice-chairperson of SIGMOD.

Reprint Order No. G321-5288.