Database Systems

Michael W. Blasgen

Summary. Database systems, computer systems that are principally devoted to the management of large amounts of data, are becoming more and more important to the operation of many enterprises. This article surveys the technology of database software and hardware, describing some of the principal issues related to the user's view of data, sharing, concurrent access, security, and integrity.

Computer systems are increasingly used to aid in the management of information, and as a result, new kinds of data-oriented software and hardware are needed to enhance the ability of the computer to carry out this task. This article reviews the rapidly evolving field of database systems—computer systems devoted to the management of relatively persistent data. The computer software employed in a database system is called a database management system (DBMS).

There are more than 15,000 database systems installed in the United States, and these systems are being directly used by hundreds of thousands of people. Indirectly, almost every citizen uses one or more of these systems, since it is not possible today to write a check, use a credit card, make an airline reservation, or pay a bill without causing an action in a database.

Most database systems are oriented toward the modification and retrieval of formatted data—the kind of data associated with the management of inventories, accounts payable and receivable, personnel information, and other administrative applications. Some systems (called information retrieval systems) are oriented toward the search and retrieval (not update) of unformatted text—a scientific abstracts service, for example. There are now approximately 1000 different information retrieval services that can be tapped by subscribers, up from 400 two years ago (*1*).

In addition to these "mundane" database systems, a small but growing number of database systems are employed for more exotic applications such as (i) managing the engineering drawings for a new commercial airliner and using the drawing information to directly drive numerically controlled machine tools; (ii) storing the three-dimensional structures of organic compounds and their pharmacological properties so that proposed drugs can be rapidly compared with other compounds and their properties more accurately predicted; and (iii) storing and manipulating geographic information such as maps, digitized photographs, and Landsat images, which then permit image enhancement and pattern recognition in applications such as oil exploration and land use planning. But it is not necessary to appeal to these advanced applications to understand the technical challenges of database systems. It is sufficient to consider a relatively simple example.

A Banking Example

To understand the objectives of a database system, let us consider an example application in a multibranch bank. The database will contain four files of information on:

1) Accounts, including account number, owner, and balance for each account.

2) Teller stations, including cash position and teller identification for each teller station.

3) Branches, including cash on hand and summary information for each branch.

4) Activities, including transaction type (such as withdrawal), account number, date and time, and amount involved for each transaction.

Bank employees and the bank customers must be able to inquire about the status of the database (for instance, the balance of an account or the date of the last deposit) as well as modify its contents (deposit and withdraw funds, handle a check). Periodic reports (such as monthly statements) must also be obtained. The system must also be prepared to go far beyond these simple illustrative requirements.

To support these needs, the bank will acquire a computer system, an operating system, and a general-purpose DBMS. The bank's data processing department then provides the application software. The relationship of these components is illustrated in Fig. 1.

The initial development of the application software is a large undertaking, requiring the efforts of many programmers over a period of years. Furthermore, this software will require constant maintenance, either because of changes in the underlying hardware and operating system or because of changes in the bank's requirements. The bank has a major interest in reducing both the initial cost of the application software and the cost of its maintenance. To facilitate this, the DBMS should serve both to insulate its users from the idiosyncrasies of the hardware and system software and to provide the application programmer with a simpler and easier to use view of data.

The DBMS must also help maintain the integrity of the data. As more and more of its critical data are put in the computer, the bank becomes more vulnerable to computer-related failures. The DBMS and the application programs must ensure that failures do not affect the correctness or consistency of the database. The DBMS must also help in maintaining the security of the data by ensuring that only authorized "reads" or "writes" take place.

This database system must also be prepared for new applications such as credit card authorization, telephone bill paying, or cashless retailing. These applications must be incorporated by using the same database. This point may seem obvious—since the database is already there and it contains most, if not all, the information to support these new applications, why not use it?—but it took people many years to become aware of the necessity of doing this. Without database technology to ease the sharing of data between applications, each application typically had its own copy of the data. But as the data changed, all the copies had to be updated. This maintenance became such a headache that today no data processing department will

The author is manager of advanced systems technology, IBM Federal Systems Division, Bethesda, Maryland 20817.

0036-8075/82/0212-0869$01.00/0

allow this kind of unnecessary duplication.

The system must do its work within hardware budgets and within prescribed time limits. For example, on-line operations must be carried out within a 5-second response time, and accounts must be reconciled overnight. These performance requirements can be quite onerous. Consider just the throughput requirements: a bank might have 200,000 active accounts, with 50 to 100 transactions per account each month. Using the larger figure, this means approximately 20 million transactions per month, or 20 transactions per second on the average. The peak demand will probably reach 100 transactions per second. If these transactions are performed on-line, then each transaction might require the execution of 100,000 computer instructions as well as four or five accesses to disk storage. A rate of 100 transactions per second thus can demand the execution of 10 million instructions per second and 500 disk accesses per second, requiring a very large computer and 10 to 20 disks.

Objectives of a Database Management System

From this example it should be clear that the objective of the DBMS is to help the bank control its assets at reasonable cost by providing:

1) Data independence: the application programs are protected from changes in the hardware, operating system, data storage devices, and so on.

2) Data sharing: all the applications use one copy of the database.

3) Security: only authorized individuals, terminals, and programs can perform specific functions.

4) Data integrity: hardware and software defects will not make the database inconsistent.

5) Ease of use: the view of data provided to the programmers and other users is clear, straightforward, and easy to use; the DBMS is packaged in such a way that the system programmers find it easy to install and maintain.

6) Performance: response time and throughput requirements are met.

Technology Trends

There has been rapid growth in the use of DBMS's in the past few years. In part this is because people now recognize that information is a critical asset and must be handled as carefully as a more traditional asset such as money. But technological trends have also fostered this growth by lowering the cost of the underlying hardware. Certain economies of scale (in particular, large-scale integrated circuits) have allowed the computer industry to sell more computers at much lower prices, taking advantage of a remarkable elasticity of demand for computing. More and cheaper computers mean more opportunity for DBMS's, which in turn has meant more suppliers of these systems. It is estimated (*2*) that 50 companies market 54 different DBMS packages and that the sale of DBMS software generates $150 million per year in revenues.

The personal computer revolution may also ensure a wider use of databases. The personal computer itself will have only a limited database capability, but to be truly useful it must have access to a large central database. At first this demand may be satisfied by access to stock market information (Dow Jones) and the wire services (New York Times), but later users will want direct access to bank accounts, reservation systems, and order entry systems. This access will force even greater emphasis on ease of use and security in databases.

Database Management System Services

A DBMS offers a number of services, including:

1) User view of data: describing the way data appear to the user of the DBMS, a view that is usually quite different from the way data are stored in the computer (the DBMS maps from one to the other).

2) Data language: allowing the user to retrieve, update, insert, and delete data from the database.

3) Transaction management: providing execution control over the database, determining the level of concurrent access, the recovery options, and specifying the "atomicity" of database operations.

In the sections that follow these three areas are discussed and some of the technical issues that arise are reviewed.

User Views and Data Languages

Database technology has evolved to the point where there are basically two approaches to data representation and manipulation: a network (or hierarchical) view and a relational view.

Network view. This view (*3*) organizes the data records into several record types. In our banking example, accounts is a record type, as is activities. Records having the same record type share the same form; thus an accounts record always consists of an account number field, an account owner field, and so on. If a record of one record type is related to a record of another record type, this fact is explicitly represented by a connection from the first record to the second. Thus the fact that a recent withdrawal (in the activities record type) involves a particular account (in accounts) is represented as a connection from the accounts record to the activities record. The accounts-activities connection is often called a parent-child or an owner-member relationship. An example of such a network organization is shown in Fig. 2A. The parent-child relationship can be extended further: parents can have parents, children can have children. In the bank, an account may belong to a branch, in which case the general diagram is as shown in Fig. 2B.

The network data structure is well suited to represent $1:N$ relationships between parents and children, as for example if an activities record can involve only one accounts record, and thus one account is related to N activities. The $M:N$ relationships—where, for example, an activity record can describe a transfer between two accounts—are more difficult. The usual solution is to define a third record type that relates the other two and that may be empty or may contain data pertaining to both the original record types.

The data language for a network view is navigational in nature; the language permits the user to explicitly traverse the connections, examining the records as he goes. Typical operations involved in printing a monthly statement for account 23 are:

```
POSITION ON ACCOUNTS RECORD
FOR ACCOUNT 23
MOVE TO FIRST CHILD
GET NEXT CHILD WITH SAME
PARENT
```

At all times the user (usually a programmer) is aware of a position in the database. In our example the programmer must be aware that after the execution of these three commands he is positioned on the second activities record under account 23. This position can be used to extract information in the record, to modify or delete the record, or to insert a new record immediately before or after the record.

Relational view. This view evolved in reaction to the apparent complexity of the network view. In 1970, Codd (*4*)

argued that database systems would be both simpler and more rigorous if the data were organized into units modeled after mathematical relations and the data language had the power of the first-order predicate calculus. The relational view has received increasing attention in the years since Codd's paper.

A relational database consists of a number of tables, each representing a relation and each consisting of some number of rows and columns. There are no connections between data items, and the order of the rows is not significant. Instead, all information that one item is related to another must be represented by data values. Figure 3 shows tables corresponding to our activities and accounts data. The fact that account 23 is the "parent" of activity 12 is represented not by a connection, but the common account value.

Because of its mathematical foundation in the predicate calculus, a relational data language is a high-level, set-oriented, nonnavigational language. In SQL, the data language of SQL/DS (*5, 6*), the bank statement of the previous section is generated by

```
SELECT DATE, ACTION, AMOUNT
FROM ACTIVITIES
WHERE ACCTNO = 23
```

A relational data language must support the "join" operator that combines information from two or more tables by relating rows that have the same value in specified columns (this is analogous to traversing the connections in a network). In SQL, to obtain information about the account that was involved in the transaction with activity number 10, the tables are tied together with the predicate ACCTNO = NUMBER.

```
SELECT NAME, BALANCE, ACTION,
AMOUNT, DATE
FROM ACCOUNTS, ACTIVITIES
WHERE ACCTNO = NUMBER
AND ACTIVITYNO = 10
```

SQL supports set-oriented database modifications as well. All activities associated with account 23 can be removed by saying

```
DELETE ACTIVITIES
WHERE ACCTNO = 23
```

A relational data language like SQL can be used as part of a computer program, or it can be used directly by a nonprogramming user. The direct support of ad hoc query is a very useful feature in a DBMS. Without support for query, if a manager wanted to know, say, how many accounts had a balance of $1000 and had no activity in the current month, he would have to negotiate with a programmer and the report would be delivered days or weeks later. If the DBMS has support for ad hoc query, the manager can obtain the answer directly.

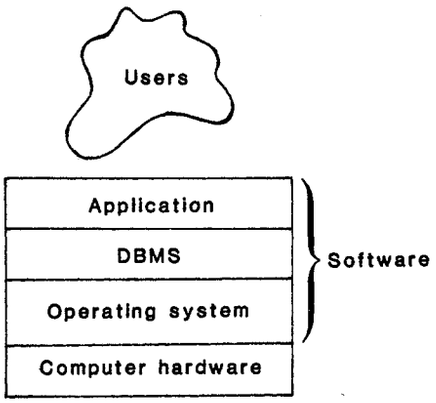

Fig. 1. Relationship of the software components.

Supporting the Data Language

A DBMS contains a component that is responsible for access path selection, that is, producing a plan for the execution of the statements. If the DBMS supports a navigational language, the access path selection component is quite trivial, because the user has already stated what paths are to be followed. In a relational DBMS, however, the access path selection routines may be quite complex since the language statements only specify what is to be done, and not how to do it. The job of the access path selection routine, or optimizer, is then to find the best way to carry out the statements.

For a specific query, the optimizer is capable of generating many different plans. In the relational database management system SQL/DS, a join query that finds (using our banking example) all the accounts that received a deposit and have a high balance can, depending on the tables, generate 64 different plans. To see how this process takes place, consider a join between the account and activities tables based on equal account numbers.

The simple join method scans one table (the outer table) and for each row, say t, in the outer table, scans the second (or inner) table to find all the rows whose account numbers are the same as the account number in t. This method touches every record in the outer table once, and every record in the inner table once for each row in the outer table. Thus if there are N rows in the outer table and M in the inner, the number of rows read by the simple join method is $N + NM$. This method is always feasible.

If both tables are sorted by account number we can use a merge join which operates in the same way as the simple join, but because of the sorting, it avoids unnecessary references to rows in the inner table. As a result, the merge join reads only $N + M$ rows. This method is feasible only for sorted tables.

A final observation is that any table of N rows can be sorted with a number of reads proportional to $N \log N$ (*7*).

Thus the optimizer can use the following decision procedure. If the tables are sorted, use a merge join. Otherwise compare the estimated costs of a simple join and a sort/merge join (that is, first sort the tables, then perform a merge join) and choose the plan that has the lowest estimated cost.

This example decision procedure, which is itself a great simplification of the procedures actually implemented in a DBMS like SQL/DS, can make enormous improvements in performance. If the tables are large and unsorted, plunging ahead with a simple join would be a serious mistake; with 10,000 activities and 500 accounts, a sort/merge join can be 100 times faster than a simple join.

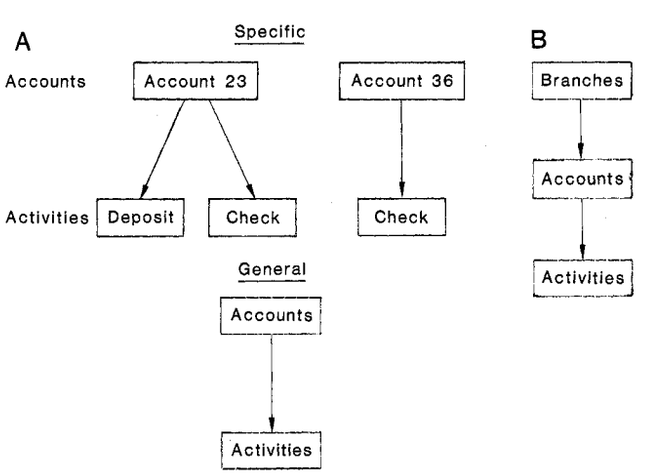

Fig. 2. Network examples: (A) two-level and (B) three-level.

Transaction Management

Quite unrelated to the network versus relational debate there are a number of other issues. Consider a transaction that transfers money from your savings account to your checking account. This transaction will be expressed as a sequence of statements that decreases the balance in one account, increases the balance in another, and inserts some activities records. What happens if the computer fails in the middle of the execution of this sequence? It would be undesirable for the transaction to be left half done—the savings debited, for example, but the checking unchanged. The bank would have many unhappy customers if this happened often.

Instead, what we want is the ability to treat a sequence of actions as a single, atomic action—in the face of various failures, the action is either completed in its entirety or not done at all. Further, if an agent outside the DBMS has been informed that an action is completed, then that action must "stick" (stay completed) in the database in the face of subsequent failures.

A DBMS with these characteristics is said to be failure resilient. Failure resilience is provided by DBMS software that keeps track of the progress of each atomic transaction. As the transaction modifies the database, the DBMS logs the changes onto a storage device that will persist through a failure. If the computer fails, then when the system is restarted the log is examined to determine what recovery must be done—the DBMS must undo transactions that were in progress at the time of failure (and possibly restart them) and must redo transactions that were completed prior to the failure but that for any reason did not stick.

In addition to recovery, transaction management may be concerned with concurrency. The simplest execution model of a DBMS is one in which requests for service are queued up behind a single server that satisfies the requests one at a time. Some DBMS's work this way, but others provide multiple servers so that several transactions are in execution concurrently. Concurrent execution is certainly desirable when the computing system is made up of multiple processors, and it is also desirable in a single processor to improve response time and use the relatively long processor waits associated with disk reads and writes.

Accounts

Number	Name	Balance
23	Blake	235.05
39	Will	40.15

Activities

Activityno	Acctno	Action	Amount	Date
10	39	Dep	20.00	4/15
12	23	With	100.00	4/16
5	23	Dep	5.00	4/10

Fig. 3. A relational database.

Consider two concurrent executions of the transaction that transfers money between accounts, and suppose the two transactions are interlaced so that the first transaction debits the savings account (say from \$300 to \$200) and then, before the first transaction proceeds, the second transaction is executed so that the same account is debited again (say from \$200 to \$150), the checking account credited, and the transaction completed. Now suppose a failure occurs. At restart, the recovery logic is going to discover that the first transaction was incomplete at the failure and undo the actions, thereby placing the old value of \$300 back into the savings account. This scenario, if allowed, has just cost the bank \$50.

To avoid this, a transaction must lock data that are to be modified and keep the data locked until the transaction is complete. Another transaction attempting to lock an already locked data item will be delayed until the first transaction is complete. It turns out (*8*) that appropriate locking protocols can allow a large degree of concurrent execution while providing database changes equivalent to running the transactions serially, one after the other. In other words, even though transactions are running concurrently on shared data, they are in fact logically isolated from one another.

Future Trends

Currently available commercial DBMS's contain many of the features—optimization, transaction management, and concurrency control—that have been outlined in this article. In the area of semantics, Codd (*9*) has extended the relational model, and Chen (*10*) has developed an information modeling concept called the entity relationship approach. Workers in artificial intelligence are making use of specialized knowledge about the data domain in order to improve the utility of database systems.

In the systems area, further progress is being made in failure resilience and in concurrency control. The ultimate in failure resilience is a system that permits continuous operation, so that the database is always available. Much research is being done on distributed systems (*11*) consisting of a number of computers interconnected with communication links.

Tempering all this technology is the knowledge that databases can be used to undermine privacy. Computerized database systems are not qualitatively different from the manual systems that preceded them; the concern is that these new systems make it feasible to record more detail in a centralized data bank that provides for easier retrieval. For this reason, research on secure DBMS's continues (*12*).

Conclusions

The trend toward greater use of database systems will continue. Both relational and network systems will be found, with network systems being employed in applications that are well structured and where efficiency is critical, and relational ones having an advantage in evolving environments where adaptability and ease of change are of primary concern. Research can be expected to resolve a number of outstanding issues in multicomputer distributed systems, in information representation, and in failure resilience. Future database systems will become even more usable, more adaptable, and more widespread.

References

1. *Wall Street Journal*, 10 December 1981, p. 1.
2. P. Krass and H. Weiner, *Datamation*, October 1981, pp. 153–170.
3. Report of the CODASYL Data Base Task Group, Association for Computing Machinery, New York, April 1971.
4. E. F. Codd, *Commun. ACM* **13**, 377 (June 1970).
5. IBM Corporation, SQL/Data System for VSE, G320-6590 (IBM, White Plains, N.Y.).
6. M. W. Blasgen *et al.*, *IBM Syst. J.* **20** (No. 1), 41 (1981).
7. D. E. Knuth, *Art of Computer Programming*, vol. 3, *Sorting and Searching* (Addison-Wesley, Reading, Mass., 1973).
8. J. N. Gray, *Comput. Sci.* **60**, 393 (spring 1978).
9. E. F. Codd, *ACM Trans. Database Syst.* **4**, 397 (December 1979).
10. P. Chen, *ibid.* **1**, 9 (March 1976).
11. I. W. Dreffen and F. Poole, Eds., *Distributed Databases, an Advanced Course* (Cambridge Univ. Press, New York, 1980).
12. E. B. Fernandez, R. C. Summers, C. Wood, *Database Security and Integrity* (Addison-Wesley, Reading, Mass., 1981).