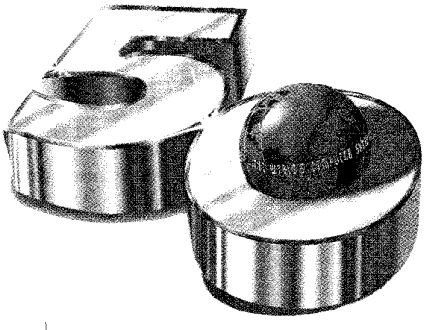


Evolution of Data Management

Jim Gray
Microsoft Research

Early data management systems automated traditional information processing. Today they allow fast, reliable, and secure access to globally distributed data. Tomorrow's systems will access and summarize richer forms of data. Multimedia databases will be a cornerstone of cyberspace.

Computers can now store all forms of information: records, documents, images, sound recordings, videos, scientific data, and many new data formats. Society has made great strides in capturing, storing, managing, analyzing, and visualizing this data. These tasks are generically called data management. This article sketches the evolution of data management systems.

There have been six distinct phases in data management, as shown in Figure 1. Initially, data was manually processed. The next step used punched-card equipment and electromechanical machines to sort and tabulate millions of records. The third phase stored data on magnetic tape and used stored-program computers to perform batch processing on sequential files. The fourth phase introduced the concept of a database schema and on-line navigational access to the data. The fifth step automated access to relational databases and added distributed and client-server processing. We are now in the early stages of sixth-generation systems that store richer data types, notably documents, images, voice, and video data. These sixth-generation systems are the storage engines for the emerging Internet and intranets.

MANUAL RECORD MANAGERS, 4000 BC-1900

Record keeping has a long history: The first known writing describes the royal assets and taxes in Sumeria (Figure 2 shows a Sumerian tablet). The next six thousand years saw great advances, but throughout this era information processing was still done manually.

PUNCHED-CARD RECORD MANAGERS, 1900-1955

The first practical automated information processing began circa 1800 with the Jacquard Loom, which produced fabric from patterns represented on punched cards. Player pianos later used similar technology.

In 1890, Herman Hollerith applied punched-card technology to perform the US census. Hollerith formed a company to produce equipment that could record data on cards and sort and tabulate the cards.[1] His system had a data record for each household. Each data record was represented as binary patterns on a punched card. Machines tabulated counts for blocks, census tracts, congressional districts, and states. Hollerith's business eventually became International Business Machines, a small company that prospered by supplying "unit-record equipment" to business and government from 1915 onward.

By 1955, many companies were dedicating entire floors to storing

0018-9162/96/$5.00 © 1996 IEEE

punched cards, much as the Sumerians had archived clay tablets. Other floors contained banks of card punches, sorters, and tabulators. These machines were programmed by rewiring control panels (patch-boards), which managed accumulator registers and selectively reproduced cards onto other cards or paper. Large companies processed and generated millions of records each night. This would have been impossible with manual techniques. Still, it was clearly time for a new technology to replace punched cards and electromechanical computers.

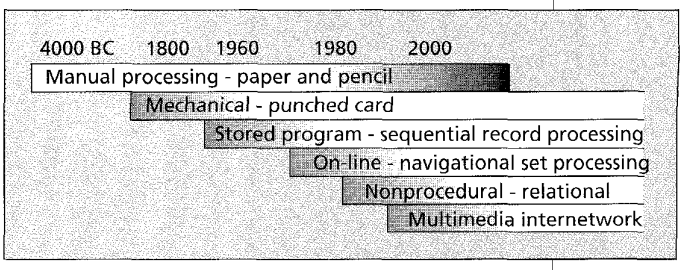


Figure 1. Six generations of data management.

PROGRAMMED RECORD MANAGERS, 1955-1970

Stored-program electronic computers had been developed in the 1940s and early 1950s to perform scientific and numerical calculations. By 1950, Univac had developed a magnetic tape that could store as much information as 10,000 cards, a product that saved space and time and was much more convenient and reliable. The 1951 delivery of the Univac I to the US Census Bureau started the next generation of data management systems, much as the 1890 census began the punched-card era. These new computers could process hundreds of records per second, and they could fit in a fraction of the space occupied by the unit-record equipment.

Birth of high-level programming languages

Software was key to this new technology because it made these computers relatively easy to program and use. High-level programming languages like Cobol made it easy to write computer programs that read, analyzed, and transformed individual records. Indeed, standard software packages began to emerge for common business applications like general-ledger, payroll, inventory control, subscription management, banking, and documentation.

The software of the day provided a *file-oriented record processing* model. Typical programs sequentially read several input files and produced new files as output. Cobol and several other languages were designed to make it easy to define these record-oriented sequential tasks. Operating systems provided the file abstraction to store these records, a job control language to run the jobs, and a job scheduler to manage the workflow.

The response to these new technologies was predictable. Large businesses recorded even more information and demanded faster and faster equipment. As prices declined, even medium-sized businesses began to capture transactions on cards and use a computer to process the cards against a tape-based master file.

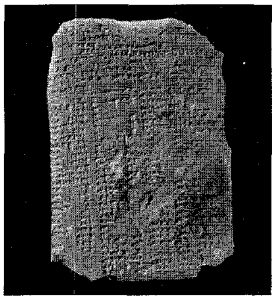
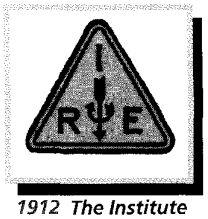


Figure 2. The first known writing was done in ancient times by the Sumerians.

Batch processing

Batch transaction processing systems captured transactions on cards or tape and collected them in a batch for later processing. Once a day, these transaction batches were sorted. The sorted transactions were merged with the much larger database (typically called a "master file") stored on tape, to produce a new master file. This master file also produced a report that was used as the ledger for the next day's business. Batch processing used computers very efficiently, but it had two serious shortcomings. First, transaction errors were not detected until the daily run against the master file, and then they might not be corrected for several days. More significantly, the business did not know the current state of the database because

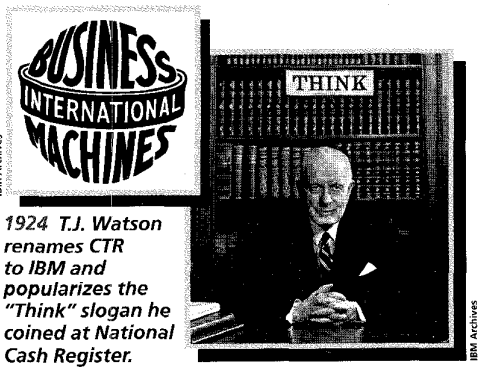


1912 The Institute of Radio Engineers, which will eventually merge with other organizations to form the IEEE, is established.

1915 Use of microchips is foreshadowed as physicist Manson Benedicks discovers that the germanium crystal can be used to convert alternating current to direct current.

1919 Eccles and Jordan, US physicists, invent the flip-flop electronic switching circuit critical to high-speed electronic counting systems.

1920-1921 The word "robot" (derived from the Czech word for compulsory labor) is first used by Karel Câpek in his play RUR (Rossum's Universal Robots).



1924 T.J. Watson renames CTR to IBM and popularizes the "Think" slogan he coined at National Cash Register.

1912 — 1924

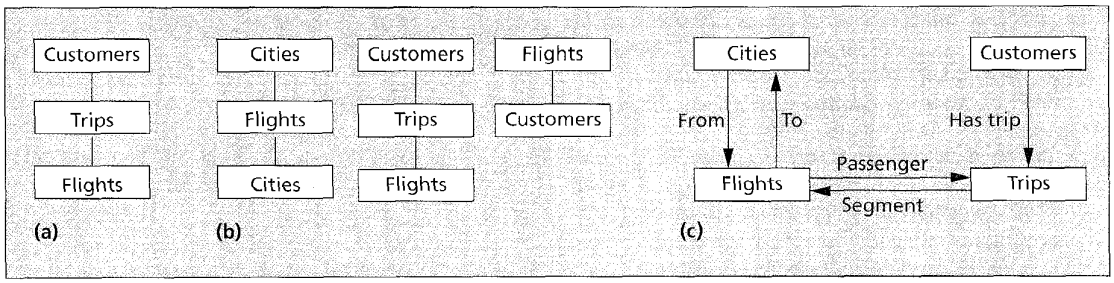


Figure 3. Evolution of data models. (a) Pure hierarchical model with records grouped under other records. (b) As the application grows, different users want different views of the data expressed as different hierarchies. (c) Bachman diagram showing the record sets and relationships among record types.

changes were only recorded in the overnight batch run, so a transaction made one day was not really processed until the next morning.

Solving these two problems required the next evolutionary step, to on-line systems.

ON-LINE NETWORK DATABASES, 1965-1980

Applications like stock-market trading and travel reservation could not use the day-old information provided by off-line batch transaction processing—they needed immediate access to current data. Starting in the late 1950s, the leaders of several industries began innovating with on-line transaction databases, which interactively processed transactions against on-line databases.

Several technologies were key to on-line data access. First, such systems required hardware to connect interactive terminals to a computer. Today's intelligent terminals evolved from the teletypes of that era. *Teleprocessing monitors* provided the specialized software to multiplex thousands of terminals onto the modest servers of the day. These TP monitors collected request messages from a terminal, quickly dispatched server programs to process each message, and then dispatched the response back to the requesting terminal. *On-line transaction processing* augmented batch transaction processing, which performed background reporting tasks.

Indexed sequential records

On-line databases stored on magnetic disks or drums provided subsecond access to any data item. These devices and data management software allowed programs to read a few records, update them, and then return the new values to the on-line user. Initially, the systems provided simple record lookup, either indexed lookup using a record key value, or a direct lookup using a record number.

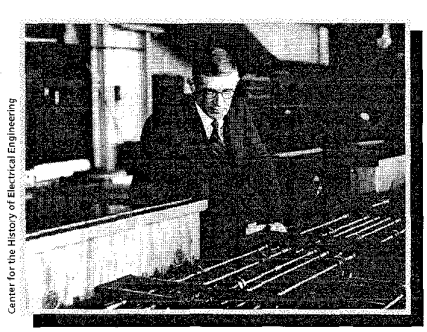
This simple indexed-sequential record organization soon evolved to a more powerful *set-oriented record model*. Figure 3 illustrates this approach, which allows applications to relate two or more records. Figure 3a shows some record types of a simple airline reservation system and their relationships: Each city has a set of outgoing flights, each customer has a set of trips, and each trip consists of a set of flights, and each flight has a set of passengers. Figure 3b shows how this information can be represented as three set-hierarchies. Each hierarchy answers a different question. The first gives the flight schedule by city. The second hierarchy gives the customer's view of his flights. The third hierarchy tells which customers are on each flight. A travel reservation application needs all three of these data views.

The hierarchical representation in Figure 3b has a major shortcoming: it stores data redundantly. This is not only expensive but also creates update problems: New or changed flight information must be updated in all three

1927 Herbert Hoover's face is seen on screen during the first demonstration of television in the US. Accompanying voice transmission uses telephone wires.

1928 The quartz crystal clock makes possible unprecedented time-keeping accuracy.

1929 Color television signals are successfully transmitted.



1930 The Differential Analyzer, devised by Vannevar Bush and colleagues at MIT, solves various differential equations.

1931 Reynold B. Johnson, a high school teacher in Michigan, devises a way to score multiple-choice tests by sensing conductive pencil marks on answer sheets. IBM later buys the technology.

1934 In Germany, Konrad Zuse seeks to build a better calculating machine than those currently available.

1927 — 1934

places (all three hierarchies). To solve these problems, the information could be represented with a *network data model,* shown in Figure 3c. Here a single database stores each record once and a relationship connects it to a set of other records. For example, all the flights involved in a specific customer's trip are related to that trip. A program can ask the database system to enumerate those flights, and new relationships can be created as needed. Figure 3c is variously called a Bachman diagram[2] or an entity-relationship diagram.[3]

Data independence

Managing associative access and set-oriented processing was so common that the Cobol community chartered a Data Base Task Group (DBTG), which defined a standard data definition and data manipulation language. Charles Bachman, who had built a prototype data navigation system at General Electric, received the Turing Award for leading this DBTG effort. In his Turing lecture, Bachman described the evolution from flat-file models to the new model, in which programs could navigate among records by following the relationships among them.[2] Bachman's model is reminiscent of Vannevar Bush's Memex system[4] or the pages-and-links navigational model of today's Internet.

The DBTG model crystallized the concept of *schemas* and data independence. Schemas hide the physical details of record layouts: Programs should see only the logical organization of records and the logical relationships among records. This allows programs to continue to work even as the data layout evolves over time. Records, fields, and relationships not used by the program should be hidden, both for security and to insulate the program from the inevitable changes to the database design.

These early databases supported three kinds of data schemas:

- a *logical schema*, which defines the global logical design of the database records and relationships among records;
- a *physical schema*, which describes both the physical layout of the database records on storage devices and also defines the files and indices needed to support the logical relationships; and
- a *subschema,* which exposes just the subset of the logical schema used by the application.

This logical-physical-subschema mechanism provided *data independence.* Indeed, many programs written in that era are still running today using the same subschema they started with, even though the logical and physical schemas have evolved to completely new designs.

Concurrent access

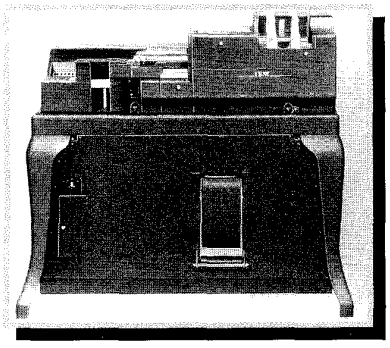
These on-line systems had to solve the problem of running many concurrent transactions against a database shared among many terminal users. Prior to this, the single-program-at-a-time, old-master/new-master approach prevented concurrency and recovery problems. The early on-line systems pioneered the concept of *transactions* that lock just the records they access. Transaction locking allows concurrent transactions to access different records. These systems also kept a log of the records that each transaction changed. If the transaction failed, the log was used to undo the effects of the transaction. The transaction log was also used for media recovery. If the system failed, the log was reapplied to an archive copy of the database to reconstruct the current database.

The logical-physical-subschema mechanism defined by Buchman's DBTG provided data independence.

By 1980 set-oriented network (and hierarchical) data models were very popular. Cullinet, a company founded by Bachman, was the largest and fastest-growing software company in the world.

RELATIONAL DATABASES AND CLIENT-SERVER COMPUTING, 1980-1995

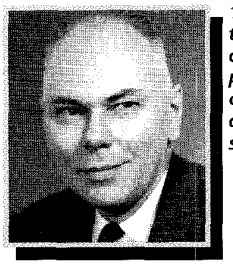
Despite the success of the network data model, many software designers felt that a navigational programming interface was too low-level because programmers needed to use very primitive and procedural database operations to navigate these databases. In 1970, E.F. Codd outlined the relational model,[5] which offered an alternative—it



IBM Archives

1935 IBM introduces not only the 601 multiplying punch-card machine but also an electric typewriter.

1936 Konrad Zuse realizes that programs composed of bit combinations can be stored, and he files a patent application in Germany for the automatic execution of calculations, including a "combination memory."



IBM Archives

1937 Howard Aiken submits to IBM a proposal for a digital calculating machine capable of performing the four fundamental operations of arithmetic and operating in a predetermined sequence.

1937 Claude Shannon publishes the principles for an electric adder to the base two.

1937 George Stibitz develops a binary circuit based on Boolean algebra.

1935 — 1937

gave database users high-level set-oriented data access operations.

Uniform representation

The *relational model* represents both entities and relationships uniformly. The relational data model has a unified language for data definition, data navigation, and data manipulation, rather than separate languages for each task. More importantly, the relational algebra deals with record sets (relations) as a group, applying operators to entire record sets and producing record sets as a result. The relational data model and operators allows much shorter and simpler programs. For example, our airline database would be represented by five tables, as Figure 4 shows. Instead of implicitly storing the relationship between flights and trips, a relational system explicitly stores each flight-trip pair as a record (the "Segment" table in Figure 4).

The relational model had some unexpected benefits beyond programmer productivity and ease-of-use.

The following SQL query would find all segments reserved for customer Jones going to San Francisco:

```
Select Flight#
From   City, Flight, Segment, Trip, Customer
Where  Flight.to = "SF"
  AND  Flight.flight# = Segment.flight#
  AND  Segment.trip# = trip.trip#
  AND  trip.customer# = customer.
       customer#
  AND  customer.name = "Jones"
```

In other words, "Find the flight numbers for flights to San Francisco which are a segment of a trip booked by any customer named `Jones.' Combine the City, Flight, Seg-ment, Trip, and Customer tables to find this flight." This program may seem complex, but it is vastly simpler than the corresponding navigational program.

Given this nonprocedural query, the relational database system automatically finds the best way to match up records in the City, Flight, Segment, Trip, and Customer tables. The query does not depend on which relationships are defined, and so it will continue to work even after the database is logically reorganized. Consequently, it has much better data independence than a navigational query based on the network data model. In addition to improving data independence, relational programs are often five or 10 times simpler than their corresponding navigational programs.

Inspired by Codd's ideas, researchers in academe and industry experimented throughout the 1970s with this new approach to structuring and accessing databases, promising dramatically easier data modeling and application programming. The many relational prototypes developed during this period converged on a common model and language. Work at IBM Research led by Ted Codd, Raymond Boyce, and Don Chamberlin and work at UC Berkeley led by Michael Stonebraker gave rise to the SQL database language. Since it was first standardized in 1985 there have been two major additions to the standard.[6] Virtually all database systems provide an SQL interface today. In addition, all systems provide unique extensions that go beyond the standard.

Unexpected benefits

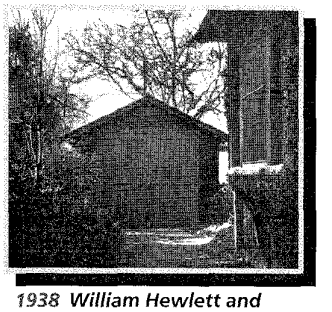
The relational model had some unexpected benefits beyond programmer productivity and ease-of-use. The relational model was well-suited to client-server computing, parallel processing, and graphical user interfaces.

CLIENT-SERVER. Client-server designs divide applications in two parts: The client is responsible for capturing inputs and presenting outputs; the server is responsible for storing the database, processing client requests against a database, and responding with a summary answer. The relational interface is especially convenient for client-server computing because it exchanges high-level SQL requests and responses. Today, many client-server tools are built around the Open Database Connectivity (ODBC) protocol, which provides a standard way for clients to make high-level requests to relational database servers.

Iowa State University

1937 Alan Turing's paper "On Computable Numbers" presents the concept of the Turing machine.

1937 John Vincent Atanasoff spends the winter devising the principles for an electronic-digital computer.

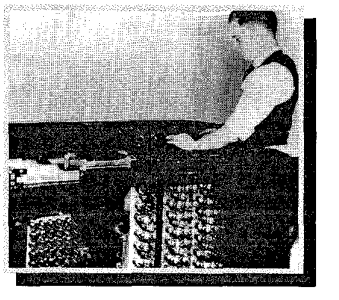


HP Company Archives

1938 William Hewlett and David Packard form Hewlett-Packard in a garage in Palo Alto, California.

1939 Working from October through November, John Vincent Atanasoff, with help from graduate student Clifford E. Berry, builds a prototype electronic-digital computer that uses binary arithmetic.

1938 Zuse completes the Z1 electromechanical binary computer and refines the design with the Z2.



Iowa State University

1937— 1939

PARALLEL PROCESSING. The relational model consists of operators closed under composition: Each operator takes relations as inputs and produces a relation as a result. Consequently, relational operators naturally support pipeline and partition parallelism. Pipeline parallelism is achieved by piping the output of one operator to the input of the next. It is rare to find long pipelines, but relational operators can often be partitioned so that each operator is cloned *N* ways and each clone processes 1/*Nth* of the input relation. These ideas were pioneered by academe and by Teradata Corporation (now NCR). Today, it is routine for relational systems to provide hundredfold speedups using parallelism. Data mining jobs that might take weeks or months to search multiterabyte databases complete within hours when done in parallel. This parallelism is completely automatic—designers simply present the data to the database system, and the system partitions and indexes it. Users present queries to the system (as ODBC requests) and the system automatically picks a parallel plan for the query and executes it.

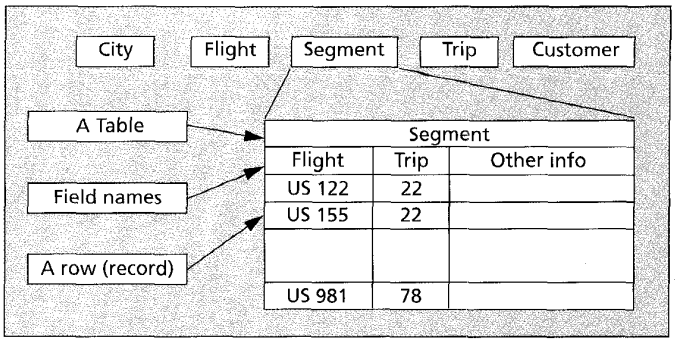


Figure 4. The same information as in Figure 3c represented using the relational model, in which all data and all relationships are explicitly represented as records. The relations are at top. Some details of the Segment relation are shown at the lower right; it has a record for each flight (segment) in any passenger's itinerary.

GRAPHICAL USER INTERFACES. Relational data is also well-suited for GUIs. It is very easy to usually render a relation because relations fit the spreadsheet metaphor—each table row becomes a row of the spreadsheet. Users can create spreadsheet-like relations that they can visually manipulate. Indeed, there are many tools that move relational data between documents, spreadsheets, and databases. It is the relational model's uniform representation of data, relationships, and metadata that makes this possible.

Relational systems combined with GUIs allow hundreds of thousands of people to pose complex database queries each day. The combination of GUIs and relational systems has come closest to the goal of automatic programming. GUIs allow very complex queries to be easily constructed. Given a nonprocedural query, relational systems find the most efficient way to execute that query.

SUMMARY. By 1980, Oracle, Informix, and Ingres had brought relational database management systems to market. Within a few more years, IBM and Sybase had brought their products to market. By 1990, relational systems had become more popular than the earlier set-oriented navigational systems. Meanwhile file systems and set-oriented systems were still the workhorses of many corporations. These corporations had built huge applications over the years and could not easily change to relational systems. Rather, relational systems became the key tool for new client-server applications.

MULTIMEDIA DATABASES, 1995-

Relational systems offered substantial productivity improvements. Nonetheless, in about 1985 the research community began to look beyond the relational model for more powerful ways to represent and process data.

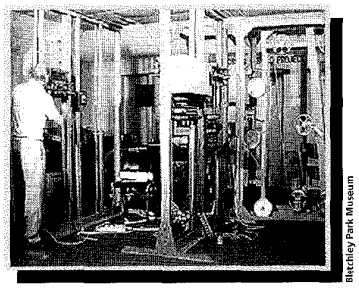
Richer data types

Traditionally, there had been a clear separation between programs and data. This worked well when the data was just numbers, characters, arrays, lists, or sets of records. As new applications appeared, the separation between programs

1940 Konrad Zuse completes the Z2, which uses telephone relays instead of mechanical logical circuits.

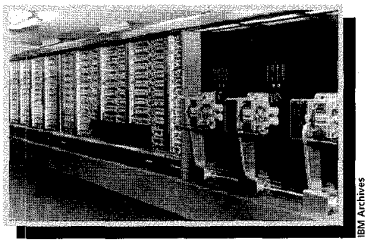
1941 Zuse completes the Z3, the first fully functional program-controlled electromechanical digital computer.

1943 On May 31, 1943, construction begins on the ENIAC at the Moore School of Electrical Engineering in Philadelphia.


Bletchley Park Museum

1943 In December, Colossus, a British vacuum tube computer, becomes operational at Bletchley Park through the combined efforts of Alan Turing, Tommy Flowers, and M.H.A. Newman. It is considered the first all-electronic calculating device.

1944 The Harvard Mark I (a.k.a. IBM Automatic Sequence Controlled Calculator [ASCC]), produced by Howard Aiken, is dedicated at Harvard University on August 7, 1944.


IBM Archives

1940 — 1944

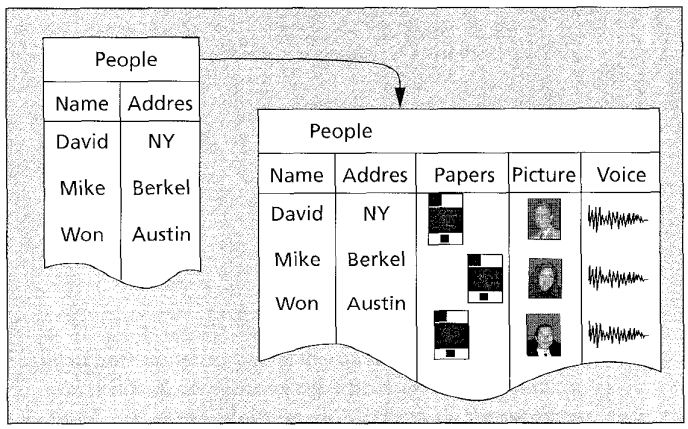


Figure 5. The transition of traditional databases storing numbers and characters into an object-relational database where each record can contain data with complex behavior. These behaviors are encapsulated in the class libraries that support the new types. In this model, the database system stores and retrieves the data and provides relationships among data items, but the class libraries provide the item behavior.

and data became problematic. The applications needed to give the data behavior. For example, if the data was a complex document, then the methods to search, compare, and manipulate the data were peculiar to the document, image, sound, or map data type, as Figure 5 illustrates.

The traditional approach was to build data types right into the database system. SQL added new data types for time, time intervals, and two-byte character strings. Each of these modest extensions was a significant effort. When they were done, the results were not appropriate for everyone. For example, SQL time cannot represent dates before the Christian era, and its multicharacter design does not include Unicode (a universal character set for almost all languages). Users who wanted Unicode or pre-Christian dates had to define their own data types. This experience convinced the database community that database systems must allow domain specialists to implement their own data types. Geographers should be allowed to implement maps; text specialists, text indexing and retrieval; and image specialists, images.

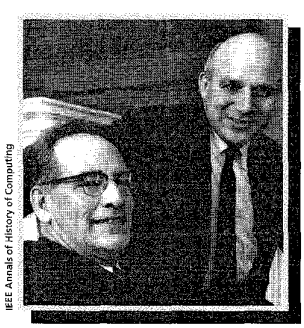
For example, a data time series is a common object type. Rather than building this object into the database system, it is recommended that the type be implemented as a class library with methods to create, update, and delete a time series. Additional methods summarize trends and interpolate points in a series and compare, combine, and difference two series. Once this class library is built, it can be plugged into any database system. The database system will store objects of this type and will manage the data (security, concurrency, recovery, and indexing), but the data type will manage the contents and behavior of time-series objects.

OO databases

Object-oriented programmers see the problem clearly: Data type design requires a good data model and the unification of procedures and data. Indeed, programs encapsulate the data and provide all the methods to manipulate the data. Researchers, start-ups, and established relational database vendors have labored long and hard since 1985 to either replace the relational model or unify the object-oriented and relational systems. More than a dozen OO database products came to market in the late 1980s, but customers were slow to accept these systems. Meanwhile, the traditional database vendors extended SQL to embrace OO concepts, while preserving the benefits of the relational model.

There is still heated debate on the outcome of this "evolution versus revolution" in data models. There is no debate that database systems must store and retrieve objects managed by class libraries. The debate revolves around the role of SQL, around the details of the object model, and around the core class libraries the database system should support.

The rapid evolution of the Internet amplifies these

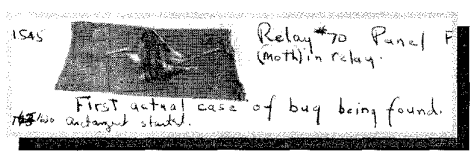


1945 J. Presper Eckert and John Mauchly sign a contract to build the EDVAC (Electronic Discrete Variable Automatic Computer).

1945 By spring of the year, ENIAC is up and running.

1945 John von Neumann introduces the concept of a stored program in a June 30 draft report on the EDVAC design.

1945 Zuse's Z4 survives World War II and helps launch postwar development of scientific computers in Germany.



1945 Working on a prototype of the Mark II, in the summer Grace Murray Hopper finds the first computer "bug," a moth that had caused a relay failure.

1945 In July, Vannevar Bush's As We May Think is published in the Atlantic Monthly.

debates. Internet clients and servers are being built around applets and helpers that capture, process, and render one data type or another. Users plug these applets into a browser or server. The common applets manage sound, image, text, video, spreadsheets, and graphs. Each of these applets is a class library for its associated types. Desktops and Web browsers are ubiquitous sources and destinations for much of the data. Hence, the types and object models used on the desktop will drive the server class libraries that database systems must support.

Unifying procedures and data

Today databases are being called upon to store more than just numbers and text strings. They are being used to store the many kinds of objects we see on the World Wide Web and the relationships among them. The distinction between the database and the rest of the Web is being blurred. Indeed, each database vendor is promising a "universal server" that will store and analyze all forms of data (all class libraries and their objects).

Unifying procedures and data extends the traditional client-server computing model in two interesting ways:

- First, it facilitates the development of *active databases*, which autonomously perform tasks when the database changes. The idea is that a user-defined trigger procedure will fire when a database condition becomes true. Using the database procedure language, database designers can define preconditions and trigger procedures. For example, if a reorder trigger has been defined on an inventory database, then the database will invoke a reorder procedure on an item anytime the item's inventory falls below the reorder threshold. Triggers simplify applications by moving logic from the applications to the data.
- Second, it generalizes the typical request-response model of computing into a *workflow*. A workflow is a script of tasks that must be executed. For example, a simple purchase agreement consists of a six-step workflow: buyer request, bid, agree, ship, invoice, and pay. Systems to script, execute, and manage workflows are becoming common.

Projects that push the limits

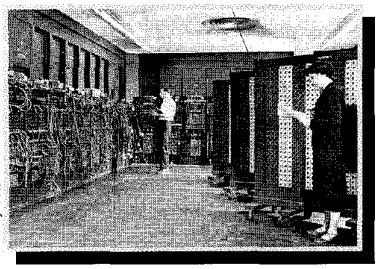
Two large data management projects typify the limits of current data management technology.

The Earth Observation System Data/Information System is being built by NASA and its contractors to store all the satellite data that will start arriving from the Mission to Planet Earth satellites in 1997. The database, consisting of remote sensor data, will grow by 5 terabytes a day (a terabyte is a million megabytes). By 2007, the database will have grown to 15 petabytes, a thousand times larger than the largest on-line databases today. NASA wants this database to be available to everyone, everywhere, all the time for searching, analyzing, and visualizing the data.

Today databases are being used to store the many kinds of objects we see on the World Wide Web and the relationships among them.

Building EOS/DIS will require advances in data storage, management, search, and visualization. Most of the data has both spatial and temporal characteristics, so the system requires substantial advances for storing those data types, as well as class libraries for the various scientific data sets. For example, this application will need a library to recognize snow cover, vegetation index, clouds, and other physical features in LandSat images. This class library must easily plug into the EOS/DIS data manager.

The emerging worldwide library is another challenging database. Many institutional libraries are putting their holdings on-line, and new scientific literature is being published on-line. On-line publishing poses difficult societal issues about copyrights and intellectual property, but it also poses deep technical challenges. The information appears in many languages, in many data formats, and in huge volumes. Traditional approaches to organizing this information (author, subject, title) do not exploit the power of computers to search documents by content, to link documents, and to cluster similar documents together.

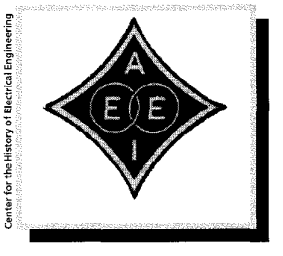


US Army Photo

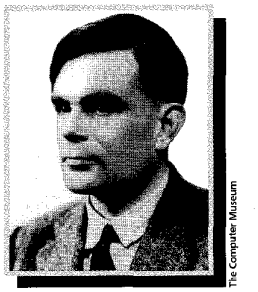
1946 ENIAC, designed by J. Presper Eckert and John Mauchly, is unveiled at the University of Pennsylvania on February 14.

1946 Arthur Burks, Herman Goldstine, and John von Neumann write "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument."

1946 The American Institute of Electrical Engineers establishes a Subcommittee on Large-Scale Calculating Devices—the origin of today's IEEE Computer Society.



Center for the History of Electrical Engineering



The Computer Museum

1946 Alan Turing publishes a report on his design for ACE (Automatic Computing Engine), featuring random extraction of information.

1946

Information discovery—the process of finding relevant information in the sea of text documents, maps, photographs, sounds, and videos—poses an exciting and challenging problem.

REFLECTIONS AND PREDICTIONS

Advances in computer hardware have enabled the evolution of data management from manual processing to automated information search engines. This progress in hardware is expected to continue for many more years.

Data management software has advanced in parallel with these hardware advances. The record and set-oriented systems gave way to relational systems that are now evolving to object-relational systems. These innovations give one of the best examples of research prototypes turning into products. The relational model, parallel database systems, active databases, and object-relational databases all came from the academic and industrial research labs. The development of database technology has been a textbook case of successful collaboration between academe and industry.

Inexpensive hardware and easy software have made computers accessible to almost everyone. It is now easy and inexpensive to create a Web server or a database. Millions of people have done it. These users expect computers to automatically design and manage themselves. These users do not want to be computer operators or programmers. They expect to add new applications with almost no effort: a plug-and-play mentality. This view extends from simple desktop systems to very high end servers. Users expect automated management with intuitive graphical interfaces for all administration, operations, and design tasks. Once the database is built and operational, users expect simple and powerful tools to browse, search, analyze, and visualize the data. These requirements stretch the limits of what we know how to do today.

MANY DATA MANAGEMENT CHALLENGES remain, both technical and societal. Large on-line databases raise serious societal issues. The technical challenges are more tractable. There is broad consensus within the database community on the main challenges and the research agenda. Every five years, the database community does a self-assessment that outlines this agenda. The most recent self-assessment, the "Lagunita II Report,"[7] emphasizes the following challenges:

- Defining the data models for new types (such as spatial, temporal, and image) and integrating them with the traditional database systems.
- Scaling databases in size (to petabytes), space (distributed), and diversity (heterogeneous).
- Automatically discovering data trends, patterns, and anomalies (data mining, data analysis).
- Integrating (combining) data from multiple sources.
- Scripting and managing the flow of work (process) and data in human organizations.
- Automating database design and administration.

These are challenging problems. Solving them will open up new applications for computers both for organizations and for individuals. These systems will allow us to access and analyze all information from anywhere at any time. This easy access to information will transform the way we do science, the way we manage our businesses, the way we learn, and the way we play. It will both enrich and empower us and future generations.

Perhaps the most challenging problem is understanding the data. There is little question that most data will be on-line—because it is both inexpensive and convenient to store it in computers. Organizing these huge data archives so that people can easily find the information they need is the real challenge. Finding patterns, trends, anomalies, and relevant information from a large database is one of the most exciting new areas of data management.[8] Indeed, my hope is that computers will be able to condense and summarize information for us so that we will be spared the drudgery of searching through irrelevant data for the nuggets we seek. The solution to this will require contributions from many disciplines. ▮

References

1. J. Shurkin, *Engines of the Mind: A History of the Computer*, W.W. Norton & Co., New York, 1984.
2. C.W. Bachman, "The Programmer as Navigator," *Comm. ACM*, Nov. 1973.
3. C.J. Date, *An Introduction to Database Systems, Sixth ed.*, Addison-Wesley, Reading, Mass., 1995
4. V. Bush, "As We May Think," *The Atlantic Monthly*, July 1945.
5. E. F. Codd, "A Relational Model of Data for Large Shared Databanks," *Comm. ACM*, June 1970.
6. J. Melton and A.R. Simon, *Understanding the New SQL: A Complete Guide*, Morgan Kaufmann, San Francisco, 1993.
7. "Database Research: Achievements and Opportunities Into the 21st Century," A. Silbershatz, M. J. Stonebraker, and J.D. Ullman, eds., *ACM SIGMOD*, Mar. 1996, pp. 52-63.
8. U.M. Fayyad et al., *Advances in Data Mining and Knowledge Discovery*, MIT Press, Cambridge, Mass., 1995.

Jim Gray *is a senior researcher at Microsoft Research. His interests are in scalable and fault-tolerant database and transaction servers. He is editor-in-chief of* VLDB Journal. *Gray received a PhD in computer science from UC Berkeley. He is a member of the National Academy of Engineering and an ACM fellow.*

Contact Gray at Microsoft Research, 301 Howard St., San Francisco, CA 94105; gray@microsoft.com; http://www.research.microsoft.com/research/barc/gray.