How Data Got its Base: Information Storage Software in the 1950s and 1960s

Thomas Haigh
University of Wisconsin, Milwaukee

Generalized report generation and file maintenance programs were widely used in the 1950s, standardized by the Share user group with 9PAC and Surge. By the 1960s the first recognizable DBMS systems, such IMS and IDS, had evolved to address the challenges of disk drives and MIS projects. Finally, in the late 1960s Codasyl's Data Base Task Group formulated the DBMS concept itself.

The database management system (DBMS) is the foundation of almost every modern business information system. Since the 1950s, the storage, retrieval, and updating of large volumes of stored data has been a key requirement for most computer applications, which is why *data processing* was the common name for administrative computing work from the mid-1950s to early 1980s. Nothing has been more vital to the computer's success as an administrative tool than the development of software to hide the complexities of data manipulation from application programmers and end users. Today, a flurry of database transactions powers each update of a major Web site, literature search, or Internet shopping trip. Yet little research addresses the history of this vital technology or the ideas behind it. Although some attention has been paid to the DBMS as an important product class for the early software industry, professional historians have given little attention to its technological evolution, influence on data processing practice, or intellectual history.[1] When short technical histories feature in texts and essays by database specialists, they are schematic and generally begin with the publication of a series of reports by the data processing standards group Codasyl from 1969 onward.[2]

In this article, I adopt a different perspective, in which Codasyl's specifications are not the starting point but the culmination of 15 years of practice and experience by administrative computing specialists. By framing work during the 1950s on report generators and generalized file maintenance systems such as Surge and 9PAC in the context of the needs and priorities of data processing users, I expose the submerged foundations of the DBMS in these earlier, user-driven projects. Likewise, by interpreting the work of the 1960s, particularly Charles Bachman's efforts on Integrated Data Store (IDS) and other projects, in the context of efforts to build so-called total management information systems (MISs), I connect the creation of early DBMSs with the broader managerial and organizational context that drove these efforts. Today, Codasyl's database work is remembered only for its propagation of the network data model. My final discussion of the work of Codasyl's Data Base Task Group and Systems Committee, however, highlights their role in articulating and disseminating the DBMS concept and a surprisingly modern and ambitious description of its architecture and capabilities.

This article follows recent work in science and technology studies by emphasizing what has been called the ''materiality'' of early computing technologies as an influence on both computing practice and the subsequent development of software technologies.[3] Writing on software tends to emphasize its abstract and ethereal nature, but database tools evolved from the daily work of the data processing staff, grappling with tight schedules and working intimately with simple hardware to tackle ambitious assignments. Software tools encapsulate craft knowledge, working practices, and cultural assumptions. Thanks to the power of

 Published by the IEEE Computer Society 1058-6180/09/$26.00

backward compatibility and installed base (or what historians of technology call *technological momentum*), these encapsulated qualities are reproduced with each new software revision, often enduring for decades.[4] To understand the development, appeal, and form of early DBMSs, we must also understand the data processing cultures, technologies, and practices of the 1950s and 1960s.

Because the database management system exists behind the scenes, invisible to most people, a few definitions are in order. The DBMS is a complex piece of system software able to manage multiple databases, each consisting of many different data tables. It includes mechanisms for application programs to store, retrieve, and modify this data and also lets users query it interactively. One of the most important features of the DBMS is its ability to shield the people and programs using the data from the details of its physical storage. It polices access the stored data, giving access only to tables and records for which a given user has been authorized. Because all access to stored data is mediated through the DBMS, a database can be restructured or moved to a different computer without disrupting the programs using it.

Early data processing, 1954–1957

The DBMS evolved from generalized file maintenance and report generation created within the unglamorous world of corporate data processing to simplify the creation of programs for routine administration. File-maintenance systems were intended to reduce the cost of producing routine administrative programs and make the finished programs easier to change and maintain. Report-generation systems made it easier to produce printed reports based on particular criteria.

Computer use in the early 1950s was dominated by scientific computation. By the end of that decade, the balance had shifted decisively toward business administration applications.[5] Most computer installations were organized around running a handful of automated clerical tasks, most commonly payroll and accounting, over and over again. Data processing jobs were not run interactively during this era. Instead, they were run through the computer one at a time, either manually or with an automatic queue managed by a simple operating system (OS). Programs and the input data they were processing were loaded together into the computer. Input data was entered onto punched cards by specialist key-punch operators. On smaller, cheaper computers such as the IBM 650, the workhorse machine of the late 1950s, these cards would remain the main storage medium. On larger computers, the data punched onto the cards was usually transferred to tape for processing and storage.

None of this seemed, to borrow a phrase from the era, like rocket science. Processing a weekly payroll run ought to be easier than modeling a missile flight path. Yet, the first generation of American data processing installations spent much more and took far longer than expected to get their electronic marvels doing useful work. General Electric's famous 1954 use of a Univac computer to automate payroll processing set the pattern for other firms.[6] Data processing managers were shocked by the complexity of programming work and the inflexibility computerization imposed on areas such as data entry and special-case handling. Standard application packages were rare in the 1950s and 1960s. Instead, code was specially written for each user company, usually from scratch by in-house programmers (though sometimes with the aid of consultants or using a sample program as a base). Data processing applications were usually written with assembly language, although Cobol gradually gained popularity over the 1960s.

Data was stored on tape as a sequence of codes, and efficient processing was possible only when the tape was read from start to end with a minimum of rewinding or searching. Early computers had small internal memories, which limited the complexity of each application program and the amount of data it could read in one go. Memory limitations, coupled with the inflexible, serial nature of tape storage, meant that a single major job might require dozens of programs to be run one after another, each reading and writing information from several tapes. Most of these programs processed intermediate results written during earlier runs, as shown in Figure 1.

The concepts of records, files, fields, special codes to mark the beginning and end of files, and the merging information from one file to another (all ubiquitous in computer systems today) have their origins in electromechanical punched card machine methods dating back to the 1930s.[8] Records using the same basic format were laid out sequentially along the strip of magnetic tape. Additional codes were introduced to provide checks against corrupted data.

Exhibit 3

MANUFACTURING CONTROL SYSTEM

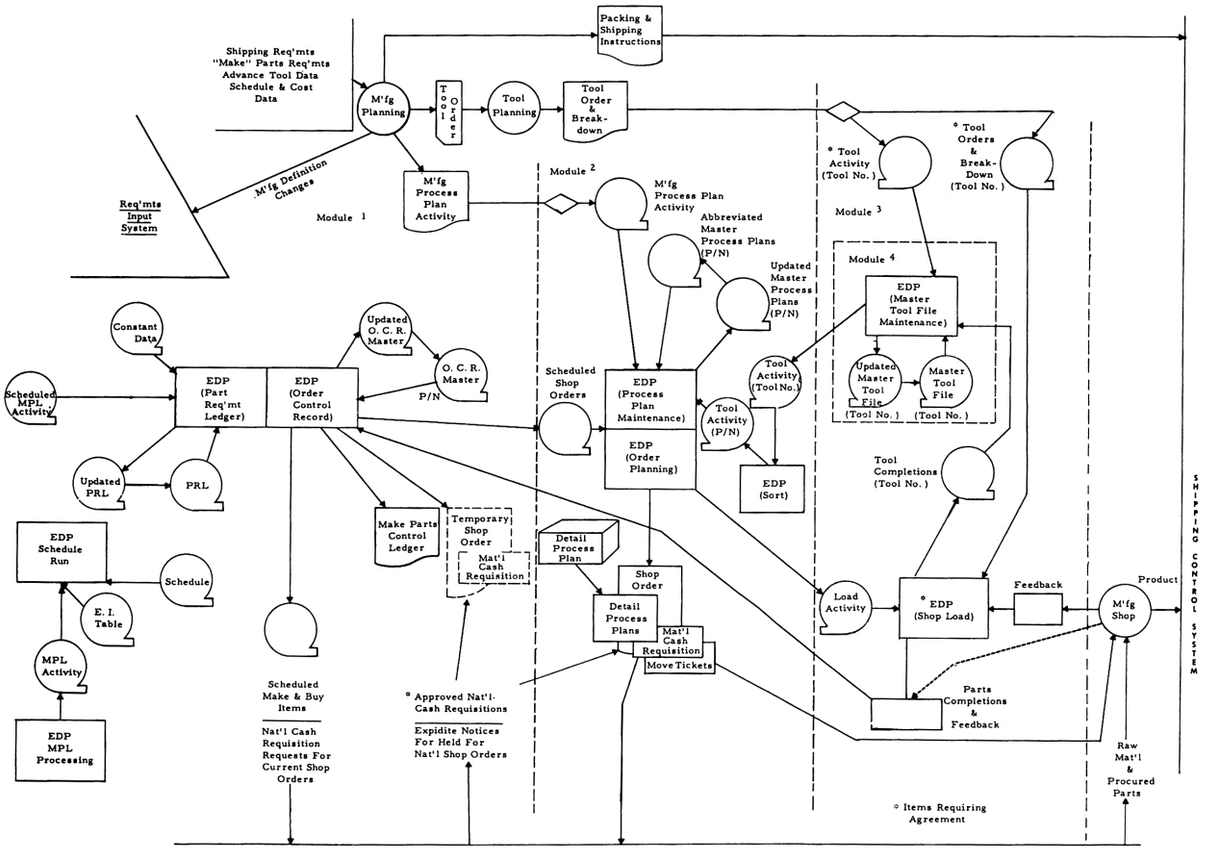

Figure 1. Part of a file-based system for manufacturing control, circa 1962. The system would have involved many runs, evidenced by the large number of programs and intermediate files involved.[7]

Some things were much harder to accomplish with high-end, tape-based computers of the 1950s than they had been with the specialized machines of the punched-card days. Before a file could be used for a particular task, such as generating required output or being used to update a set of master records, its contents had to be sorted into the appropriate order. With conventional punched-card machines, this was easily accomplished by physically reordering the cards in a sorter. In contrast, slicing the magnetic tape into thousands of little pieces and sticking it back together was hardly a viable option. Neither was reading the entire file into memory.

Early sorting practice is captured in IBM's "705 Generalized Sorting Program Sort 51," issued in 1955. The original file was split into two files. These were repeatedly copied from two input to two output tapes. Each sorting run brought the records a step closer to being ordered. When just two ordered sequences remained, these were consolidated into a single output file. In the worst case, sorting a single input tape holding 50,000 records of 100 characters each might require 17 runs through the million-dollar computer, occupying it for approximately an hour.[9] File-sort programs sound trivial, but even in the 1970s, some of the most important independent software products were generalized sort routines.

Early application programs were written in assembly language and ran without a memory resident operating system to handle input and output chores. They thus had to specify the minute details of reading and writing information to and from tape, card, or drum.[10] Programs were closely tied to particular hardware configurations. Changing the tape drive used for temporary storage might require considerable programming work, while adapting a program to make

efficient use of more memory sometimes involved a fundamental rewrite. The problem was compounded as companies attempted to reap the benefits of automation by using the output of one major application as the input to another—for example, by linking their production scheduling to their inventory control, their accounts receivable, and billing. Because one major application might contain dozens of small programs, it could take Herculean efforts on the part of the programming staff to do something as simple as adding an extra digit to the employee number.

The pioneers of the late 1950s and early 1960s evolved new practices as they struggled to contain costs while maximizing flexibility. Data processing programmers spent much of their time crafting code to read records from tapes and print lines on paper, dealing each time with the many read errors, synchronization problems, and other problems that might strike at any time. Data processing teams soon hit on the idea of producing a single set of well-written, reusable subroutines to handle these chores. Standard code was modified slightly to fit the particular situation and then inserted into each application program, often via assembler macro instructions.

Technological change also motivated the shift to standardized I/O routines. Computer manufacturers built ever-more powerful capabilities into their data processing hardware, meaning that the programming work required to read and write records on tape with maximum efficiency became more complex. For example, IBM's tape hardware for the 705 (announced 1954) could read 15,000 characters a second either forward or backward, from a reel holding five million characters. Tape units functioned somewhat independently of the main processor, under the control of a separate tape-control unit. A single read instruction would transfer five characters at a time for improved efficiency. Specified areas of the main memory were used as buffers, so that the computer could process one record while another was transferred. Drives could even read and write simultaneously to the same tape, for efficient file updates. Read and write operations were automatically verified using a parity bit and a special instruction would rewind by one unit record, making it easier to repeat a failed read operation.[11] The application programmer still remained responsible for writing code to carry out operations such as parsing a record into fields, searching for a record with a particular key value, or inserting a record into an existing file.

Substitute "different tapes" for "the same tape" here. I misread the manual.

Report-generation software, 1957

In some areas of data processing a technique known as *generalization* could eliminate the need for a custom program entirely. One such area, quickly recognized, was sorting records. Another was report generation and the closely related area of file maintenance.

Data processing applications produced voluminous printed reports during their scheduled runs (daily, weekly, monthly, or quarterly), but the only way to obtain a special report was to write another program. A manager wishing to tabulate data in a different way, or to include only a subset of the original records in the calculations, could either wait for a programmer to get around to this job or wade through the printout tallying records. By the late 1950s, the more innovative data processing teams had begun to address this by creating report-generation programs. Data processing personnel provided these programs with descriptions of the desired output and the organization of the data inside the relevant master files and were rewarded with the desired reports.

The most influential early system for generalized report creation was designed in 1957 by GE's team at the Hanford Nuclear Reservation on its IBM 702—IBM's first large computer designed for administrative use. Generalized file-sort routines were already in use there, but the idea was to produce a suite of three new generalized programs: the report generator, an improved sort routine, and a file-maintenance program able to make changes to file formats as well as file data. All three worked with a consistent *source file dictionary* that described record formats, which was included as a header at the start of each data file. The report-generator program used this data dictionary and an input file holding details of the desired report format (which could include calculated fields such as totals and subtotals) to produce a customized program that, when compiled and executed, would yield the desired output. An updated version of the report generator, finished in 1958, added integral sort capabilities and expanded facilities for complex calculations.[12,13] Because it was optimized for memory efficiency, the system could produce up to 20 simultaneous reports on a single run through the source data tapes (p. 65).[14]

The objective was to make data stored on tape easily accessible for reporting purposes, just like data in traditional punched-card systems. The urgent need for programs of this kind reflected the computer's complexity as a general-purpose machine. Punched-card machines were specialized for specific tasks such as sorting or tabulating and were configured by setting switches or inserting wires into a patch board. Computers required elaborate programming.

Project participant Russell McGee remembers that the Hanford managers responsible for business programming and computer operations ''all had extensive punched card backgrounds, which had a profound influence on our work.''[15] The ultimate goal was ''a single powerful computer on which all data for the organization ... was stored and available for reporting and processing.''[13] His colleague Fred Gruenberger noted that punched-card experience likewise shaped customer expectations. Managers who requested changes to reports were ''vitriolic'' when told that minor changes required programs to be written ''at the cost of $1,500'' when they had been ''well trained by us in data processing over the years to the idea that all you have to do to change your report is move five wires on the 405'' (pp. 49–50).[14]

The GE effort was not the only one of its kind, and participants quickly realized that companies were wasting effort by producing their own systems. Gruenberger organized a panel for the October 1957 meeting of the IBM Share user group. Noting that development of these systems had so far been a ''100 percent non-cooperative programming venture which curiously enough took place at about four independent installations at almost the same time within a period of six months;'' he expressed hope for future collaboration.[16]

Standardizing file management systems, 1957–1961

For its first few years in the computer business, IBM left systems programming work to its customers. Its main strategy for providing its customers with utility programs was to help them help themselves via collaborative user groups. In 1955, initial deliveries of the IBM 702, the firm's first large administrative computer, were accompanied by nothing more than listings for a symbolic assembler and a sample sort program. The code for both was presented in its programming manual, ''for study and actual application if desired.'' By 1956, there were still only nine IBM programmers working to support its successor, the 705.[17]

User groups filled the vacuum, creating program libraries full of routines contributed by user companies and organizing project committees to design and create ambitious new programs flexible enough to satisfy many companies. Share was for users of IBM's large scientific computers, and a corresponding group Guide was formed for users of its most expensive administrative machines.[18] Initially, IBM's main contribution to these efforts was to handle the logistics of cataloging and distributing the software libraries, although by the late 1950s, its programmers were increasingly involved in implementing the packages specified by Share.

In the mid-1950s, Share's members were dealing with IBM's 704, a machine optimized for scientific calculation that was rarely applied to administrative tasks. Its successor, the 709, was promoted as being equally adept at administrative work and technical computing, pushing Share toward engagement with data processing issues. Share's biggest and best-known early project was the design of a full suite of system software for the 709. This project, the Share 709 system (SOS), included an assembler, monitor, job-control system, and range of other components. One of the projects was a standard I/O system designed to provide assembly subroutines for I/O operations on cards or tape.[19] In 1957, its designers were already distinguishing between logical and physical aspects of data storage in tape files, a key concept for the decoupling of application programs from specific hardware configurations and file formats—although in fact the SOS input, output, and buffering macros provided little abstraction from the underlying hardware.[20] Similar macros were developed by Guide.

Companies interested in using the 709 for administrative work founded the Share Data Processing Committee, which met for the first time on 2 October 1957 under the chairmanship of Charles W. Bachman of Dow Chemical.[21] Members spent the first half of the inaugural meeting trying to standardize terms such as ''card record'' and ''tape file.''[22] In the second half, a report on the Hanford effort was distributed to members.[23] They ''extensively discussed'' this system and resolved to use it as a prototype for an ''input generator'' able to parse input data into a

specified file format. Gruenberger's plea for collaborative efforts had fallen on friendly ears.

Two important projects followed. Surge, under the leadership of Fletcher Jones (then of North American Aviation, soon to cofound Computer Sciences Corporation), was intended to be a short-term project to build something called the ''704 Data Processing Sort Routine'' to provide data processing capabilities on the aging IBM 704 computers still used by most Share installations.[24] The new name reflected an increase in its scope to ''Sort, Update, Report Generate, Etcetera.'' Although a 704 version did appear, Surge evolved into an ambitious data processing compiler project for the newer IBM 709 and 7090.[25] The project was designed for nonspecialist programmers, using a rigid check-off format for coding rather than allowing English-like statements along the Cobol model. An initial 709/7090 version of Surge was shipped in 1960 with an improved version planned for 1962.[26]

The other project was 9PAC (709 package), which was a direct evolution of the GE system. According to Russell McGee, his data processing group at GE Hanford aimed to shift its work from an IBM 702 to the new IBM 709. This shift meant reprogramming every application. Thus, ''only the generalized routines made recommendation of the 709 feasible.'' Implementing them on the 709 would mean that ''the same source files and packets could be used on the new machine as on the old'' (p. 59).[13] But reworking the generalized routines for the 709, with its complex tape-control system, was itself a major undertaking. At the second Data Processing Committee meeting in November, McGee gave an update on the project's progress and reported that a rewrite for the 709 was planned.[27] This reimplementation effort soon became a Share project, a transition McGee recalls took place at the tenth Share meeting in Washington DC the next February.[13,28]

The Share development process had a great deal in common with today's open source software projects. A small volunteer committee handled the main development and specification tasks, while a series of Share Secretarial Distributions sent to all IBM 709 sites kept the broader user community informed. These distributions mediated the project's discussion, disseminating a series of documentation drafts, requests for information, comments, and suggestions.

Starting a project was easy. As McGee recalls, ''creation and dissolution of these committees and subcommittees were accomplished by simple word-of-mouth agreements between the chairpersons and the Share officers'' (p. 60).[13] The hard part was persuading Share member companies to devote programmer time to the effort, which McGee accomplished through ''frequent trips to various companies to tell my story to the brass.'' He succeeded in persuading Union Carbide to provide office space to coordinate the Report Generator project in its Long Island City facility. The 9PAC Subcommittee of the Data Processing Committee was formally chartered only after a meeting of interested parties in Los Angeles in May 1959 when the project was largely finished.[29]

Like the earlier Hanford report generator, 9PAC took a set of input parameters and file definitions as its input and produced a program ready for execution. 9PAC consisted of two separate but compatible modules: a report-generator and a generalized file-maintenance system. The latter incorporated capabilities for calculated updates and modifications to the format of existing files.[30]

According to a November 1958 progress report the file maintenance portion of 9PAC was being created by nine people working at GE Hanford and was expected to be finished by the end of the year.[31] Hanford's own 709 was not yet ready for use, and a debugging trip to a Los Angeles installation in January revealed ''many errors,'' which McGee did not expect to be able to correct until the end of that month when Hanford's 709 would be available.[32] Debugging took longer than McGee anticipated (though not as long as we might predict), and 9PAC entered use in May 1959.

Although 9PAC was designed for tape storage, it allowed the creation of hierarchical relationships between records, a crucial advance over the GE system. *Master records* (such as customers) could be associated with the relevant *detail records* (such as orders), which were simply stored on tape immediately following their parent records. This was a translation to tape storage of punched-card practice.[33] The 9PAC file format was surprisingly complex, allowing storage of data dictionary information on a separate dictionary tape or in a header affixed to the beginning of each file. This included field information for records on each level of the hierarchy. It also allowed the storage

of access permission information to restrict the file reading or modification.[34]

Share members hoped to create still more complex file management systems. 9PAC and 709 Surge were seen as short-term solutions, stop gaps until ambitious goals were fulfilled. Surge, for example, was originally announced as an interim 709-7090 commercial data processing system. By August 1958, a subcommittee of the Share data processing committee had already drawn up plans for a full data processing package for the IBM 709. The specification described a memory resident task scheduler loading several processing applications simultaneously and providing communication between them. The system was to provide instructions for I/O and, when given appropriate field descriptions, could extract, store, convert, or move information from ''fields addressed symbolically by the program.''[35] Work on this system does not seem to have progressed further.

Another never-implemented proposal produced during 1959–1960 described the Ingen system, a successor to 9PAC that would integrate its report, file-maintenance, and sort-program generation capabilities into a single system able to generate programs combining these three operations. According to Bachman (who was to lead this effort), its main advance would have been to make it much easier to reorganize record hierarchies. A hierarchical tape system like 9PAC might sort the file for orders by customer number, following each customer record with a series of orders for that customer sorted by date. Producing a report listing all orders by date would create a new file holding the same data but structured to have date rather than customer as the top level of the hierarchy. Ingen would have performed this kind of restructuring automatically, based on a metadata system that understood that a particular record type might be a master record in one context and a detail record in another.[36]

As far as I am aware, none of these plans for next-generation, tape-based file management and report-generation systems came to fruition. They were stymied by the inherent limitations of tape storage and the rapid adoption of disk technology during the early 1960s by the kind of high-end computer centers that might have contributed to their development or benefitted from their use. Still, the key requirements identified for these systems were the same needs that drove the development of DBMSs in the years that followed. Chief among these were a memory resident data management system able to support multiple applications simultaneously and the flexibility to access the same data in different sequences and through different master/detail hierarchies, depending on the job at hand.

9PAC, RPG, and Mark IV, 1960s

File management packages were so useful for data processing work that they quickly developed from user-supported projects into standard elements within the system-software suite supplied by computer vendors. Because Cobol included special support for defining file formats, its general adoption in data processing from the mid-1960s onward also simplified programmer labor.

Although IBM had not contributed to 9PAC's creation, it quickly recognized the system's value and assumed responsibility for maintaining and updating it. Russell McGee dates this transfer to the summer of 1960 (p. 64).[13] By 1961, an updated version for the IBM 7090 (the transistorized successor of the 709) was ''currently being maintained and improved by IBM Applied Programming.''[37] 9PAC became an official part of IBM's sprawling IBSYS OS suite for the 7090/7094 and thus a core part of its system software offerings until the transition to the third-generation System/360 range.

It wasn't just these multimillion dollar computers that needed standard file management and report-generation systems. IBM addressed the challenge directly when launching its 1401 computer, the successor to the humble 650 and the workhorse of second-generation data processing during the first half of the 1960s. As the first computer marketed as a viable alternative to conventional punched-card technology for mainstream punched-card installations, the 1401 had to be easy to use for these simple tasks. IBM supplied a new programming system called Report Program Generator (RPG) for this purpose.

According to Bill McGee, RPG was ''patterned after 9PAC.'' It played a similar role, to help ''users migrate from punched card equipment to electronic data processing,'' but for the mass market of smaller data processing installations rather than the largest and richest computer centers served by Share.[38] Its design and capabilities were also closely modeled on the earlier system. RPG took four decks of cards as its input, each using a simple programming notation to

describe the data file format, the desired records and fields to be included from that file, desired report format, or calculated fields (such as interest accumulated). It generated an efficient program that, when executed, would produce the specified report. RPG's inputs mimicked the kind of configuration work formerly handled by wiring the control boards of punched-card machines, although the calculation facilities were more flexible than those offered with regular punched-card equipment. RPG worked with data on punched cards, in tape files, or (by the mid-1960s) on disk. Its simplest version was usable on a computer with only 4,000 words of memory. Like 9PAC, RPG could process files in which master records were followed by the relevant detail records. When translating this input to a report, RPG could insert headings and subtotals as appropriate.[39] RPG was an enormous success and was soon offered for other machines. RPG II, created for the System/360 machines, replaced 9PAC on larger computers too. Even today, in much-improved versions, RPG is relied on by many thousands of corporate programmers.

File management systems also proved an important niche for the nascent independent software package industry. Mark IV—the most successful product of the early independent software industry—was a file management system descended from report software produced for the Douglas Aircraft Company. Its origins have been discussed by its creator John A. Postley.[40] Like other early software, Mark IV had its roots in user efforts and data processing practices. The successful launch of Mark IV as a standard product in 1968 rested on three years of development work funded by a small group of initial users, who in turn, relied on experience producing similar custom systems under contracts going back to 1960. The direct influence of 9PAC on Mark IV is unclear. Postley mentions no specific systems as influences, but when Russell McGee first saw a paper on Mark IV he ''thought someone had reinvented 9PAC'' (p. 130).[13] Mark IV was developed within Informatics, a company cofounded by Francis V. Wagner, who had chaired Share in 1957 and 1958. Walter F. Bauer, another founder, had taken part in the October 1957 Share Data Processing Committee meeting where GE's report generator was showcased. At the very least, we can conclude that Share had exposed them to the concept and value of generalized reporting software.

The third generation, 1957–1969

The GE Hanford system and its descendents 9PAC, RPG, and Mark IV were utility programs with which ad-hoc reports could be generated and file formats modified by entering the appropriate parameters. This was a fundamentally different approach from traditional application programming, in which calls to I/O macros (or later OS functions) were embedded within custom programs. Generalized utilities worked well for report generation and file maintenance, but they supplemented rather than replaced application programs. (The distinction blurred a little as report-generation systems incorporated more powerful logical processing and even the ability to invoke functions written in traditional languages).

The DBMS was quite different, although it built on the work of these packages. File management systems were designed around tape storage (although they were later widely used with disk drives). They worked efficiently when processing an entire file in sequence. In contrast, DBMSs were designed around disk storage. Instead of treating files in isolation, they worked on a database of multiple files, representing linkages between individual records within those files. Unlike file management systems, they could be used by application programs, remaining resident in memory to process data operations.

Disk drives were first offered as a mainstream option for most major computer systems in 1962, although they had been available in a handful of IBM systems a little earlier.[41] By ordering the right items from the catalog, it was possible to hold up to one billion characters of data on the disk drives connected to a single large IBM computer. A large disk system promised a single repository into which business data could be placed and from which they could be checked, retrieved, and updated by many different application programs. Together with the larger memories of the newer computers, it heralded a gradual shift away from the need to break administrative jobs into dozens of separate runs. Disks were called ''random access'' because information stored on any part of the disk could be rapidly retrieved, making it much easier to create special reports or to build online business systems. This contrasted with the sequential nature of tape storage, where it was often necessary to play through the entire tape to find a specific record. But random access storage demanded a whole new set of

programming techniques, analysis methods, and conventions.

Random access was not entirely new, having arrived with magnetic drums in the early 1950s, but it had previously been confined to niche roles. As Arthur Norberg has shown, this technology was pioneered by Engineering Research Associates in 1948 under a Navy contract.[42] IBM's popular 650 used a magnetic drum for its internal memory as a cost-cutting feature. Drums drove early interactive business systems such as the Univac File Computer and American Airlines' Magnetronic Reservisor, and high-speed magnetic drums were a key feature of real-time systems like the SAGE air defense system.

The 305 Random Access Method of Acounting (Ramac) introduced by IBM in 1957 was the first computer to use disk storage. Early magnetic disks were slower than drums, but they were cheaper and stored much more data. For the first time, it was practical to build an administrative application in which a data file was updated continually and could be queried as needed. Ramac bundled a simple vacuum-tube computer with five million characters of storage spread over 50 discs spun at 1,200 revolutions per minute. It could process a transaction every one and a half seconds, which was about as quickly as its printer could produce the resulting output. All transactions were punched onto cards, and the machine was queried by feeding special trigger cards into it. Square D, of Milwaukee, used a prototype model to store stock levels for 24,000 inventory items. Different decks of cards represented changes in desired stock levels and order quantities as well as bills of materials to alert the system of new orders received. Another Ramac user, American Bosch, had switched to ''continuous flow'' inventory accounting. Transactions were posted as they arrived in the office and operators could request special reports at any time.[43]

Ramac's production run—around a thousand machines—was much larger than the number of special-purpose online systems produced during the same 1957–1962 period. The limits of early-1960s technology made projects of this kind hugely expensive. Unlike batch operation, where different jobs were scheduled and run in succession, real-time operation demanded the commitment of an entire computer to a single application. The night-shift was occupied with backups, the loading and unloading of data, report generation, compression, upgrades, and testing.[44] The best known of these systems, Sabre, supported reservations for American Airlines. The first full version relied on two high-end 7090 mainframes, six of IBM's fastest drum units and 13 of its larger but slower 1301 disk units. The control units for operators were specially built (at a cost of $19,000 each), and programming was complex and expensive. American paid $30 million for Sabre, and even that reportedly failed to come close to covering IBM's costs.[45]

By the mid-1960s, disks were common options on many of the newly announced third-generation systems,[46] which represented a fundamental departure from earlier practice in two key areas: complex OSs intended for use without modification by users and hardware features for interactive, real-time operation. These promised to take the innovations that had required so much carefully crafted assembly language and custom hardware for a system like Sabre and build them into the hardware and software installed in a typical large data processing center.

That proved harder than expected. Random access promised almost instant record retrieval, but although it was easy to order the computer to read a particular part of a disk (such as drive 4, platter 5, side 1, track 3, sector 15), there was no easy way to jump straight to a particular record (e.g., customer account 15274). Combing through the whole disk to find this record destroyed the benefits of random access. Some kind of index was required so that the program could look up where the specific record was stored and then go straight to that location. Programmers experimented with various strategies to arrange and index data on random-access devices. No single technique was suitable for all situations, and most of them were complicated to program. Any method adopted would add substantially to the program's complexity; the storage space needed on the disk; and the work required to insert, delete, or update particular records. New headaches arose when it was necessary to make a backup copy of the disk, alter the format in which records were stored, or change (expand, shrink, or move) portions of the disk array allocated to particular programs.[47]

Having several programs share a single disk array, each using different program code to read and write records, caused another set of problems. Among these were the risk that an errant program might

scramble an area of a disk holding information belonging to another, the overhead imposed by writing several different versions of the code required to handle complex indexing techniques, and the certainty that at some point the physical layout of the disk storage would be changed (for example, to shift a growing file to its own disk drive) and all the programs would have to be modified at once.

The complexity of building online applications and working with random-access storage held back the spread of these technologies into mainstream use during the 1960s. A survey of managers with responsibility for computer purchasing by the *Wall Street Journal* in 1968 shows that most computing was still performed in batch mode.[48] That same year *Business Automation* magazine revealed that disk drives had arrived in just 44% of data processing installations. Even the transition from punched cards to tapes for file storage was incomplete. Twenty-eight percent of installations were still without a single tape drive, although the median installation had four. Punched cards retained their supremacy for data input with 85% of the departments still reliant on the simple keypunch as the main source of data.[49]

Random-access software: 1963–1969

Computer vendors readied new software to help their customers manage the enormous increase in complexity associated with random-access storage. By the end of the 1960s, every major computer manufacturer offered at least one such package.[50] These systems were intended to speed program development, reduce maintenance costs, shield application programs from the consequences of changes in the physical disk layout, and make it easier to selectively retrieve records based on their contents. They were usually based on the expansion of systems originally produced for use within a single organization. Although they would later be called DBMSs, the earliest of them predated that term and were known by a number of different names.[51]

The most innovative, and influential, of these systems was GE's IDS, created by Charles Bachman. IDS began its life about 1963, as part of an effort known internally as Integrated Systems Project II. Its goal was the production of an integrated system for production control, flexible enough to be easily customizable by GE's many departments but powerful enough to give rapid results to queries on production scheduling and inventory levels while automatically placing orders and calculating the optimum order quantities. The resulting system, sometimes (but not always) called the Manufacturing Information and Control System (MIACS), relied on IDS to handle its data storage and retrieval needs.[52] In the early 1960s, many companies launched ambitious efforts to produce integrated systems tying together different business functions.[53] These systems were often called management information systems (MISs). Thanks to Bachman's ingenuity, the MIACS project was a rare success in this unhappy area.[54]

Manufacturing involves assembling multiple components into larger parts, which themselves usually serve as components in a larger assemblage. The ''parts explosion problem'' made it particularly important for IDS to support the creation of links between different kinds of records. Although earlier systems had supported the idea of subrecords, stored sequentially and hierarchically within master records, IDS was more flexible. This generalized concept of linking record types, known later as the *network data model*, was a major influence on later systems.

IDS was designed from the beginning for use with disk drives. It took over an entire GE 225 computer, providing basic OS functions, including an early implementation of paged virtual memory to squeeze out maximum performance from the computer's 8,000-word memory. The task scheduler itself relied on IDS to store data, making it the foundation for the OS. MIACS application programs (written in GE's GECOM language) used simple instructions to navigate through the relationships between records and to store, get, modify, or delete individual records. In the first IDS implementation, a preprocessor replaced these special instructions with the appropriate strings of assembly instructions (as in traditional systems of I/O macros). Efficiency concerns inspired a different approach, where IDS performed this expansion by interpreting the requested operation according to metadata about the record type involved. This part of IDS remained resident in memory, waiting to deal with data requests from the application programs.[54]

Similar to file management programs, IDS stored metadata to describe file formats. This let it isolate application code from some of the details on how data was physically stored on the disk. Using standard routines to

manipulate data made it easier to reconfigure the file-format or hardware-configuration details without significant changes to application source code. File formats were no longer specified within application programs. This separation ultimately resulted in a new role, the database administrator (DBA). However, abstraction and efficiency are sometimes antithetical. IDS was remarkably efficient, but it delivered only limited abstraction compared with modern systems. Most notably, its instructions worked one record at a time, and programmers explicitly navigated through files to locate related records. Its separation of application code from physical storage was incomplete. E.F. Codd, creator of the relational database model, noted that application programs in IDS had to be explicitly modified to make use of new indexes added by file designers.[55]

IDS was used at several sites internally within GE. An optimized version was issued by International General Electric for its 225 system and supplied to customers including Mack Truck and Weyerhauser. With the advent of the third-generation GE 400 and 600 series computers, a new version of IDS was produced, in which application programs were written in the now standard Cobol rather than GECOM. These versions were produced by the systems programming groups for the new machines. These standard IDS implementations lacked transaction processing capabilities—IDS was loaded and unloaded from memory along with the application programs using it.

A special version of IDS was in use at Weyerhauser, in its Weyerhauser Comprehensive Operating System (WEYCOS). The first version of WEYCOS handled inventory control and order processing. It worked in Tacoma, Washington, on a GE 225, which was hooked up to a Datanet 30 computer interfaced with teletype machines across the country to accept input data and requests for reports directly from sales offices, mills, and warehouses. Requests were dumped onto disk and processed by IDS's integral task scheduler, the *problem controller*. Participants recall that this system was fully operational by 1965.

A more ambitious system, which Bachman calls WEYCOS 2, was developed for the GE 600 computer from 1966 to 1968. This, he believes, was the first multiprogramming database system because it let multiple application programs run simultaneously, each executing its own transactions against a shared database. This meant adding capabilities to lock records and detect deadlock conditions. The system was also intended to support multiple processors with shared memory, although this goal was eventually abandoned.[13,54]

IDS was not the only package of its kind. The first version of what eventually became IBM's Information Management System (IMS) was produced in 1965 by IBM in collaboration with North American Rockwell to handle the proliferation of parts involved in the Apollo program.[56] The original version of this application, known as the Generalized Update Access Method, ran on an IBM 7010 computer and used a specialized hierarchical file management system to store its data on disk. IBM and NAA also developed a system called Remote Access Terminal System (RATS) so that interactive application programs could be accessed via terminals.

In 1966, work began on a new version created to run as an application under OS/360 on the new System/360 machines, and it was this version that IBM distributed to other customers from 1968 onward.[57] Like IDS, IMS was used by application programmers, using packaged procedures to embed data-handling capabilities in their code. The OS/360 version allowed one memory resident copy of IMS to simultaneously service the data needs of multiple application tasks.[58] It went on to great commercial success in the 1970s. (See the ''The Commercialization of Database Management Systems, 1969–1983'' article in this issue for more details.)

IMS was not the only IBM data management product of the late 1960s. In the mid-1960s, members of the Share user group devoted far more time in their sessions and workshops to discussing the forthcoming Generalized Information System (GIS). This was expected to support file manipulation, reporting, full text indexing, interactive querying, and batch updates. Its actual capabilities were the subject of much speculation prior to its release.[59] Although GIS was part of IBM's product line in the 1970s, its actual use seems to have been as a query language, and it did not achieve much success as a free-standing product.

GE offered an improved version of IDS to users of its computers, and IBM did the same with IMS. During the 1960s, computer vendors bundled their software with hardware, using it as a free promotional tool to entice users into buying computers. There

was essentially no market for systems of this kind as commercial products in the 1960s. The first successful independent product was Cincom's Total, which was probably released only in 1970—though some claims have put this as early as 1968.[60]

SDC and the database concept, 1960s

To understand the DBMS's evolution, we must now step sideways, from the world of software to the world of ideas. IDS and IMS both predated the idea of the DBMS. They did not, however, predate the ''data base'' concept itself, which is older but was initially separate from file management technology. (One of my earlier articles explored this process, so I will not dwell on the details here.[61]) The most relevant point is that ''data base'' was originally a fashionable but vaguely defined phrase floating around cutting-edge, interactive computing projects. It was only gradually associated with the specific technologies that became known as the DBMS.

In fact, the database concept originated among the well-funded Cold War technologists of the military command and control world. The Oxford English Dictionary records an early use in 1962 by the System Development Corporation, possibly in connection with the famous SAGE antiaircraft command and control network. SAGE was far more complex than any other 1950s computer project and was the first major real-time system—responding immediately to requests from its users and to reports from its sensors.[62] SAGE (and later command and control systems from SDC) had to present an up-to-date, consistent representation of the various bombers, fighters, and bases to all its users through various display systems. The ''database'' held the shared data collection on which all these views were based.

Looking to extend its business beyond this niche, SDC identified ''computer-centered data base systems'' as a key application of time-shared systems—hosting (in collaboration with military agencies) two symposia on the topic in 1964 and 1965.[63] These were crucial in spreading the database concept to high-ranking military officials, business data processing celebrities, and corporate and academic researchers.

The phrase carried some specific associations, based on the particular characteristics of firms like SDC and of military command and control projects. One of these associations was with the idea of real-time operation—the database would be constantly and, if possible, automatically updated with current information gathered from different sources. It was also assumed that, as in SAGE, a database could be ''interrogated'' in real time by its users, answering questions interactively within seconds. In addition, the database would be shared among many different programs, each one using only a subset of the overall information contained within it. SDC used its database symposia to showcase its own online systems developed with military money. These led, later in the decade, to innovative but unsuccessful commercial offerings like Time-Shared Data Management System (TDMS), which were intended to allow nonprogrammers to create database structures, load data into them, and then issue queries and retrieve their results online.[64]

Reporting on the event in *Datamation*, Robert V. Head observed that databases had already unleashed the ''biggest single strike'' of new jargon ''since the great time-sharing gold rush of 1963.'' He thought it ''possible that users, led by the military, will surrender to these data base systems without a shot being fired in anger.''[65]

Until about 1968, the ''data base'' was much discussed but little realized. This concept remained fairly distinct from the practical world of file management systems, report generators, and disk-oriented packages such as IDS and IMS. It was supposed to be used interactively online, could be used by nonspecialists, and was closely associated with the idea of a single huge reservoir of corporate information.[53] In contrast, file-maintenance and report-generation systems and their more complex descendents such as IDS and IMS were used primarily by programmers to reduce development and maintenance costs for routine data processing applications.

DBMSs and DBTG, 1965–1973

Combining the database and the file management system created the DBMS. Its idea was shaped and promoted through the work of a body called the Data Base Task Group (DBTG), an ad-hoc committee of the Codasyl industry group. Codasyl focused on creating data processing standards, and it is best known for its work designing and maintaining the Cobol programming language used for most business application programming from the late 1960s to the early 1990s. Its creation was prompted by the

realization within Codasyl that Cobol, while doing a great deal to standardize data storage on tape systems and to separate record definitions from program logic, was entirely inadequate when faced with the challenge of random-access, disk-based storage. The committee's members were drawn from computer vendors, universities, consulting companies, and a few large companies making heavy use of computers in their own business operations. IDS creator Bachman was an early committee member and promoted the ideas he developed for IDS as the basis for its work.

When it was formed in October 1965, the DBTG had originally been called the List Processing Task Force (its name was changed in 1967).[66] W.G. Simmons of US Steel was the group's founder and initial chair. It moved slowly its first few years; an official history included in a later report noted that "because the membership of the group changed constantly a major amount of the DBTG's efforts was directed toward listening to and studying the views of as many persons as possible," conceding that this "affected the progress rate of the group" (section 1.6).[67]

As its name suggests, the DBMS was intended to be a new kind of product, extending the capabilities of existing file management packages to support the advanced, online, interactive capabilities and huge integrated data stores associated with the database concept. The DBTG's purpose was to define the capabilities of these new systems and develop new standards for them.

In early 1968, the group presented a draft proposal, including a summary of Bachman's IDS, to the Cobol Language Subcommittee. In response, the subcommittee approved the resolution that "COBOL needs the Data Base concept." By January 1969, Appollon Metaxides of Bell Labs had taken over as chair, formalizing the committee's membership and focusing its work on the production of functional and language specifications for Cobol's new capabilities (section 1.7).[67] An initial public report specifying a Data Description Language (DDL) and Data Manipulation Language (DML) was completed in October 1969 and published shortly afterward as a draft for public comment.[68] The group received 179 formal responses.[67] Its final recommendations were published in 1971 and endorsed by its parent group within Codasyl (the Programming Languages Committee).[69]

Another Codasyl group, the Systems Committee, was chaired by RCA's William Olle, who did important work in promoting the DBMS concept.[70] The committee worked on examining the capabilities of existing generalized DBMSs, issuing a hefty interim report in 1969.[71] It surveyed the strengths and weaknesses of existing systems such as GID, IDS, Mark IV, and TDMS and began the attempt to identify a full list of desirable characteristics. It already included the "data structure class" concept (equivalent to the later data model), which it used to characterize IDS as hierarchical and IMS as a network—terms that stuck. Despite lobbying by firms such as GE to get their own systems adopted as the basis for a new standard, the group decided that no single existing system came close to providing the range of features required. A second report, issued in 1971, provided additional material, coverage of new systems, and analysis against the DBTG's proposal.[72] These reports did a great deal to popularize the term "database management system" as a category for products such as IMS, GIS, and IDS, which had not previously had a standard label.

This is a typing error - IMS and IDS should be reversed.

The DBTG's specific proposals were controversial at the time, and several Codasyl members opposed them (including mainframe suppliers IBM, RCA, and Burroughs).[73] The IBM user groups Share and Guide had been conducting a joint database specification project in opposition to Codasyl, beginning around 1969, with the creation of a wish list for future systems.[74] The group was active until at least 1973 and presumably influenced Share's board vote to oppose the adoption of the Codasyl proposals as industry standards.[75] No complete implementation of the DBTG specification was ever produced, and while some successful systems of the 1970s claimed to be heavily influenced by Codasyl, many others did not.

Codasyl's project on DBMS languages were not as successful as its work on Cobol in setting industry standards. In other ways, however, its work proved influential. The DBTG's work provided both a broad conceptual outline for the DBMS and detailed draft specifications for two specific parts of the overall system: the DDL for defining the database structure and the DML for accessing the data from within Cobol. It also outlined a way of giving individual programs access to selective or simplified versions of the full database.

The DBTG provided a new vocabulary for the field. It standardized terms such as

record, set, and database and added some new ones, including schema (which remains ubiquitous today) to describe the logical format of data within the database, and subschema. A subschema (similar to what would be called a view in today's SQL) let different users and applications see only a portion of the overall database, allowing selective access to records and potentially shielding the application from changes in the underlying schema—this was a property referred to as *data independence*.

The DBTG separated the DML that programmers used to add, delete, update, and retrieve particular records from the DDL used by the DBA to define the database's logical structure. That distinction remains fundamental in the database field to this day. By May 1969, Codasyl had realized that the DDL was not Cobol specific and could be used to define data structures for use with other languages such as Fortran or PL/I.[76] Although the DDL was to be a new and universally applicable language, the DML took the form of a programming-language-specific set of additions seamlessly integrated into that language's existing instructions. This realization led Codasyl to split the DBTG's work in two following approval of its 1971 report. The DML Task Group, later renamed the Data Base Language Task Group, was responsible for reworking the proposed additions to Cobol into a form suitable for publication in Codasyl's *Journal of Development* as an official standard in 1973.[77] The second group, the Codasyl DDL Committee, was established as a standing committee to publish and enhance the standard DDL.[67] These standardization efforts failed to set a marketplace standard, in part because of IBM's unwillingness to commit to the network concepts inherent in the Codasyl model while its own flagship IMS product retained a hierarchical approach.[78]

Its final contribution was to insist that a standard DBMS allow more complex linkages to be established between different files (or record types) within the same database. This favored Bachman's idea of allowing networks of relationships between record types over the more restrictive hierarchical approach used by systems such as IMS. The DBMS was intended to make these relationships (or sets) as explicit and enforceable as previous file management systems had made the specification of fields within an individual file. Because most of the logic to maintain these relationships had previously been hidden within individual programs, placing relationships inside the DBMS along with the data ensured that all application programs and user requests would have access to them.

This conceptual framework for the DBMS ultimately proved more influential than the DBTG's detailed proposals. When the Codasyl work on databases is mentioned at all today, it is usually as a synonym for the network data model. Since IDS, a commercial product, used this model prior to the DBTG's establishment, this would seem a rather limited contribution to history. In fact, the DBTG created—or at the very least, widely disseminated for the first time—the very idea of the DBMS.

Most of the DBMS characteristics stipulated by the DBTG had been demonstrated by at least one system. The novelty was its insistence that future systems must combine them all. A DBMS was expected to provide the efficient operational access for application programs and networked record-linking features that existing systems such as IDS specialized in. However, it was also expected to allow nonprogrammers to use a simple, specially tailored interface to query and update the database directly—the province of systems such as Mark IV. Likewise, a DBMS was expected to support batch operation and interactive online usage, previously offered only by specialized systems such as SDC's failed TDMS, with equal felicity.[72] In practice, most commercial systems of the 1970s failed to provide strong coverage across this spectrum of capabilities, but over time (and with the addition of external transaction-processing systems such as CICS, they evolved toward the goal.

Conclusions

In 1973, Bachman was awarded the ACM's Turing Medal, the most prestigious award in computer science. The citation singled out his creation of the IDS system and his work with Codasyl. This award was in itself an important event, representing a new level of acceptance among computer science researchers of database problems as intellectually respectable subjects of inquiry. The event is better remembered, however, for Bachman's speech.[79] Entitled "The Programmer As Navigator," it developed the idea that the shift to DBMS technology represented something akin to the Copernican revolution, in that the work of programmers would now revolve around the database

rather than computer hardware. Although this prophecy took several decades to come true, knowledge of database systems has now become a fundamental requirement for virtually all administrative applications programming, systems analysis, and advanced Web design work.

The DBMS concept advanced by Codasyl proved far more important and longer lasting than the particular methods for its realization put forward by the DBTG. Although the commercialization and adoption of DBMS technology posed its own challenges (something elaborated on in ''The Commercialization of Database Management Systems, 1969–1983,'' a companion article I wrote with Tim Bergin in this issue), it ultimately cut the cost of application development and made the maintenance and adaptation of administrative systems much easier. During the 1970s, the shift of mainstream data processing applications to random-access storage and online operation introduced additional complexity that would have overwhelmed most development teams without the adoption of DBMSs and other new kinds of system software, such as online transaction-processing packages.

The database concept was originally rather vague and associated with ideas such as online access, totally integrated MISs, flexibly structured data, and the interactive definition of data formats by users. These ideas, some of which reflected concepts from the field of information retrieval, played little part in the leading commercial systems of the 1970s. Instead, the acceptance of the DBTG concept of a DBMS implied a more concrete vision of the database—basically a body of electronic data that could be managed by a DBMS.

The DBMS, as realized through software modeled on existing systems such as IDS, was influenced by a stream of practice going back through 9PAC to the tabulating machine era. It embodied the highly structured, administrative transaction-oriented view of information held by data processing staff. It was well suited to the bureaucratic records for things such as payroll administration because each record included the same pieces of data (years of service, Social Security number, hourly rate, overtime status, and so on). Subsequent DBMSs based on the relational model continued to incorporate the same assumptions about information as earlier file management systems. They made it simple and efficient to update information and so are well suited to administrative transactions where records are constantly updated. Only with the rise of data warehousing in the 1990s did attention return to huge, integrated data stores optimized for analysis and managerial querying. Likewise, only with the rise of the Web did mainstream attention turn to the indexing and management of huge amounts of loosely structured data.

Over the past half century, we have been exposed to a great deal of talk about the information age, information revolution, information society, information technology, and so on, but this story shows that each kind of information technology embeds its own definition of information. We do not deal with information in any pure, abstract form. We deal with particular technologies, able to inform us in particular ways. Like all technologies, they make some things easy and other things hard. Software technologies such as the DBMS are generalized from prevalent practices and around the material constraints of available hardware. They themselves become part of the material infrastructure that shapes the development of future applications and the evolution of subsequent practice.

Acknowledgments

This article was written during my time as a fellow of the Center for 21st Century Studies at the University of Wisconsin—Milwaukee and as a participant in the European Science Foundation's Software for Europe project. I thank Mary Ellen Bowden for encouraging me to begin this research, Boyd Rayward for his close attention to my earlier work on this topic, David Gugerli and his group at ETH for inviting me to spend a week there discussing database history, Rick Snodgrass and ACM SIGMOD for funding my oral history interview with Charles W. Bachman, and Burt Grad and the Computer History Museum for supporting my oral history interview with Robert L. Patrick. Some material here is adapted from the 2002 *Proceedings of the Conference on History and Heritage of Scientific and Technological Information Systems* (Information Today, 2004).

References and notes

1. The development of the mainframe DBMS market is explored in M. Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry,* MIT Press, 2003,

pp. 145–149, 184–191. A short history focusing on the role of public funding in the emergence of the relational model is found in National Research Council, *Funding A Revolution: Government Support for Computing Research,* chapt. 6, Nat'l Academy Press, 1999.

2. Many database textbooks include a few pages on the development of database theory along with their introductory definitions (for example, R. Elmasri and S.B. Navathe, *Fundamentals of Database Management Systems,* Benjamin/Cummings, 1989, does this well), but this can mean little when stripped of its historical context. The closest thing to a detailed history is a quarter-century old technical primer by J.P. Fry and E.H. Sibley, ''Evolution of Data-Base Management Systems,'' *ACM Computing Surveys,* vol. 8, no. 1, 1976, pp. 7–42, 19–29. On the technical side, detailed comparisons of early systems are given in C.J. Byrnes and D.B. Steig, ''File Management Systems: A Current Summary,'' *Datamation,* vol. 15, no. 11, 1969, pp. 138–142; Codasyl Systems Committee, ''Feature Analysis of Generalized Data Base Management Systems,'' ACM Press, 1971; L. Welke, ''A Review of File Management Systems,'' *Datamation,* vol. 18, no. 10, 1972, pp. 52–54; and D.B. Steig, ''File Management Systems Revisited,'' *Datamation,* vol. 18, no. 10, 1972, pp. 48–51.
3. T. Pinch and R. Swedberg, eds., *Living in a Material World,* MIT Press, 2008.
4. T. Hughes, *Networks of Power: Electrification in Western Society, 1880–1930,* Johns Hopkins Univ. Press, 1983.
5. Administrative computing during this era is discussed in T. Haigh, ''The Chromium-Plated Tabulator: Institutionalizing an Electronic Revolution, 1954–1958,'' *IEEE Annals of the History of Computing,* vol. 23, no. 4, 2001, pp. 75–104.
6. R.F. Osborn, ''GE and UNIVAC: Harnessing the High-Speed Computer,'' *Harvard Business Rev.,* vol. 32, no. 4, 1954, pp. 99–107.
7. A.D. Meacham and V.B. Thompson, eds., *Total Systems,* American Data Processing, 1962, p. 153.
8. By the 1940s, most punched cards included 80 columns of data, each one of which coded a single number or letter. Information within each card was grouped into fields, each occupying a fixed width within each record card. Consider a factory using punched cards to process its payroll. Some fields needed only one column—for example, sex (M or F). Other fields, such as last name, might be assigned a dozen columns. Each record would be punched onto one, or in some cases several, of the cards in the deck. The complete deck representing all the factory workers was known as a file, by analogy with conventional paper records. Each record card within the file had to follow exactly the same layout of fields, and to process a particular job, the machine operators had to rewire each machine's control panel (such as sorter, collator, or tabulator) to reflect this specific field layout. Many jobs involved ''merging'' information from several files—for example, combining wage information from the master file of personnel cards with the attendance information punched onto a weekly punched card by an IBM time clock. See J.J. McCaffrey, ''From Punched Cards to Personal Computers,'' John J. Mc Caffrey Memoirs, CBI 47, Charles Babbage Inst., Univ. of Minnesota, 10 June 1989.
9. IBM, ''IBM 705 Generalized Sorting Program Sort 51 Bitsavers,'' 1956; www.bitsavers.org/pdf/ibm/705/32-6831_705_Generalized_Sorting_Pgm_1956.pdf. The logic behind a total of 17 runs is 15 sorting runs of half tapes plus final and initial runs of a full tape. The program was designed to store a maximum of four records in memory at once, which explains both its dismal performance and the relatively large maximum record size of up to 2,494 characters. A more complex program presented with a smaller maximum record size would have been able to sort short record sequences within core memory, reducing the number of sorting passes required through the tape. Sorting methods of this era are explored in E.H. Friend, ''Sorting on Electronic Computer Systems,'' *J. ACM,* vol. 3, no. 3, 1956, pp. 134–168.
10. D.D. McCracken, H. Weiss, and T.-h. Lee, *Programming Business Computers,* John Wiley & Sons, 1959, pp. 178–204.
11. IBM, ''IBM Electronic Data-Processing Machines Type 705 Preliminary Manual of Operations (22-6627-4) Bitsavers,'' 1957; http://www.bitsavers.org/pdf/ibm/705/22-6627-4_705_Oper_Jun57.pdf. Other specialized control units hooked up to the same tape drives could transfer data between tapes and cards or print without involving the main computer.
12. For contemporary system documentation, see W.C. McGee, ''Generalization: Key To Successful Electronic Data Processing,'' *J. ACM,* vol. 6, no. 1, 1959, pp. 1–23, and R.C. McGee and H. Tellier, ''A Re-Evaluation of Generalization,'' *Datamation,* vol. 6, no. 4, 1960, pp. 25–29.
13. R.C. McGee, *My Adventures with Dwarfs: A Personal History in Mainframe Computers,* CBI, 2004.
14. ''Share Reference Manual for the IBM 704,'' Share records, CBI 21, 1958.

This is the wrong title! Should be "Verbatim Transcript of the 9th Meeting of Share, October 3, 1957".

15. Russell McGee is not to be confused with W.C. McGee, who headed the scientific computing group at Hanford during the same period. I quote both in this article.
16. ''Verbatim Transcript of the 9th Meeting of Share, October 3, 1957, Morning Session,'' Share records, CBI 21, 1957, p. 49.
17. This group's contributions included a version of the powerful Autocoder assembler. C.J. Bashe, et al., *IBM's Early Computers,* MIT Press, 1986, pp. 345–347, 355–356.
18. Share is discussed, with particular reference to SOS, in A. Akera, ''Voluntarism and the Fruits of Collaboration,'' *Technology and Culture,* vol. 42, no. 4, 2001, pp. 710–736.
19. Progress on implementation of the I/O package is discussed in J. King, ''Progress Report on 709 Input-Output, SSD 017,'' Share records, 1955–86, NMAH 567, Archives Center, Nat'l Museum of Am. History (NMAH), Behring Center, Smithsonian Inst., 19 Aug. 1957. SOS eventually included a macro-based input system known as INTRAN and an output system known as OUTRAN as well as support for buffering and transmission of I/O in its memory resident monitor programs. IBM, ''SOS Reference Manual (incl Distributions 1 to 5) Bitsavers,'' 1961; http://www.bitsavers.org/pdf/ibm/share/SOS_Reference_Manual_Jun61.pdf.
20. C. Mock, ''The MockDonald Multiphase System for the 709, SSD 19,'' Share records, 1955–86, NMAH 567, 9 Sept. 1957.
21. C.W. Bachman, ''Report of the Share Data Processing Committee, October 2, SSD 020,'' Share records, 1955–86, NMAH 567, 1957.
22. This yielded the DP glossary in C.W. Bachman, ''Share Data Processing Committee: Selected Glossary, SSD 022,'' Share records, 1955–86, NMAH 567, 18 Nov. 1957. This glossary included the concept of a key as a unique identifier used to locate a record.
23. A revised version of this material was published in W.C. McGee, ''Generalization: Key To Successful Electronic Data Processing,'' *J. ACM,* vol. 6, no. 1, 1959, pp. 1–23, which remains an excellent introduction to the practices of this era.
24. W. Orchard-Hays, ''Letter to Bill Dobrusky, March 16, SSD 049,'' Share records, 1955–86, NMAH 567, 1959. In the letter, Orchard-Hays says, ''The 704 Data Processing Sort Routine ... is now known as SURGE.'' Surge has rather a low historical profile, and citations in surveys are generally to an undated document, ''SURGE: A Data Processing Compiler for the IBM 704,'' issued by North Am. Aviation. This might be the same document archived as R. Paul and W. Davenport, ''Untitled SURGE Manual for IBM 704,'' Share records, CBI 21, ~1959.
25. Early objectives of and specification for the 709/7090 Surge project are discussed in B. Went, ''Minutes of 709-7090 SURGE Meeting, September 14–16, 1959 - Columbus Ohio, SSD 59,'' Share records, 1955–86, NMAH 567, 1959, and E. Austin, ''Minutes of 709-90 SURGE Subcommittee Meeting, December 7–9,'' Share records, 1955–86, NMAH 567, 1959.
26. L.F. Longo, ''SURGE: A Recoding of the COBOL Merchandise Control Agreement,'' *Comm. ACM,* vol. 5, no. 2, 1962, pp. 98–100.
27. C.W. Bachman, ''Share Data Processing Committee Meeting, November 21–22, 1957, Midland, Michigan, SSD 023.''
28. Share minutes report that Russell McGee made another progress report and also note his chairmanship of a newly formed ''709 Report and File Maintenance Generator Subcommittee.'' C.W. Bachman, ''Report of Share Data Processing Committee,'' Share, ed., *Proc. 10th Meeting of Share,* 1958, pp. 51–55.
29. R.C. McGee, ''Letter to Jerry Koory, SSD 054,'' Share records, 1955–86, NMAH 567, 23 June 1959.
30. Specifications for the file-maintenance system are in R.C. McGee, ''Preliminary Manual for the Generalized File Maintenance System, SSD 046,'' Share records, 1955–86, NMAH 567, 23 Dec. 1958. Their shared file structure is described in R.C. McGee, ''The Structure of a Standard File, SSD 045,'' Share records, 1955–86, NMAH 567, 1958.
31. R.C. McGee, ''Progress Report on Generalize Routines, SSD 040,'' Share records, 1955–86, NMAH 567, 7 Nov. 1958.
32. R.C. McGee, ''Letter to Charles E Thoma, January 21, SSD 046,'' Share records, 1955–86, NMAH 567, 1959.
33. With conventional punched-card machines, as long as the same part of the card was always used to code for a customer number, it was easy to use sort and merge operations to create a card file in which each customer record was followed immediately by cards giving information on each order placed by that customer.
34. R.C. McGee, ''The Structure of a Standard File, SSD 045,'' Share records, 1955–86, NMAH 567, 1958.
35. J.T. Horner, ''709 Data Processing Package, SSD 035,'' Share records, 1955–86, NMAH 567, 1958.
36. C.W. Bachman, ''INGEN Proposal,'' Charles W. Bachman, Papers, CBI 125, May 1959 or 1960. Bachman clarifies INGEN's fate in C.W. Bachman, ''Oral History Interview by Thomas Haigh,'' ACM Oral History Interviews collection, 25–26 Sept. 2004.
37. IBM, ''IBM 7090 Programming Systems Share 7090 9PAC Part 1: Introduction and General

Principles (J28-6166) Bitsavers,'' 1961; http://www.bitsavers.org/pdf/ibm/7090/J28-6166_9PAC_Part1_1961.pdf.

38. W.C. McGee, ''Book Review: Installing Electronic Data Processing Systems,'' *Computing News,* vol. 5, no. 115, 15 Dec. 1957, pp. 12–14.
39. The discussion of RPG's capabilities is based on IBM, ''Report Program Generator: IBM 1401 Card and Tape Systems, J24-0215-2 Bitsavers,'' 1965; http://www.bitsavers.org/pdf/ibm/140x/J24-0215-2_cardTapeRPG.pdf, and IBM, ''Report Program Generator (On Disk) Specifications, IBM 1401, 1440, and 1460, C24-3261-1 Bitsavers,'' 1964; at http://www.bitsavers.org/pdf/ibm/140x/C24-3261-1_1401_diskRPG.pdf. Because of its apparently mundane nature, RPG has received a lot less historical attention than its usage would justify. Discussion in the secondary literature seems limited to M. Campbell-Kelly and W. Aspray, *Computer: A History of the Information Machine,* Basic Books, 1996, p. 133, and Bashe et al.'s *IBM's Early Computers* (MIT Press, 1986, pp. 479–480).
40. On Mark IV, see J.A. Postley and H. Jackobsohn, ''The Third Generation Computer Language: Parameters Do the Programming Job,'' *Data Processing,* vol. 11, Data Processing Management Assoc., 1966, pp. 408–415; R.L. Forman, *Fulfilling the Computer's Promise: The History of Informatics, 1962–1982,* Informatics General, 1984; and J.A. Postley, ''Mark IV: Evolution of the Software Product, a Memoir,'' *IEEE Annals of the History of Computing,* vol. 20, no. 1, 1998, pp. 43–50.
41. E. Webster and N. Statland, ''Instant Data Processing,'' *Business Automation,* vol. 7, no. 6, 1962, pp. 34–36, 38; N. Statland and J. R. Hillegass, ''Random Access Storage Devices,'' *Datamation,* vol. 9, no. 12, 1963, pp. 34–45; Anonymous, ''Disc File Applications: Reports Presented at the Nation's First Disc File Symposium,'' Am. Data Processing, 1964.
42. A.L. Norberg, *Computers and Commerce: A Study of Technology and Management at Eckert-Mauchly Computer Company, Engineering Research Associates, and Remington Rand, 1946–1957,* MIT Press, 2005.
43. W.L. Jerome and L. Hartford, ''RAMAC at Work,'' *Systems and Procedures,* vol. 8, no. 4, 1957, pp. 30–38, and Anonymous, ''New Accounting Concept Based on 'Assembly-line' Processing,'' *Management and Business Automation,* 1961. Use of the RAMAC is also discussed in R.H. Gregory, ''Preparation for Logic—An Orderly Approach to Automation,'' *Management Control Systems,* D.G. Malcolm and A.J. Rowe, eds., John Wiley & Sons, 1960, pp. 169–183, and Anonymous, ''Programming An Information Explosion,'' *Business Automation,* vol. 14, no. 5, 1967, pp. 47–50.
44. On the programming of real-time systems in this period and its relationship to hardware features, see W.L. Frank, W.H. Gardener, and G. L. Stock, ''Programming On-Line Systems. Part Two: A Study of Hardware Features,'' *Datamation,* vol. 9, no. 6, 1963, pp. 28–32.
45. Sabre is discussed in R.W. Parker, ''The SABRE System,'' *Datamation,* vol. 11, no. 9, 1965, pp. 49–52, and D.G. Copeland, R.O. Mason, and J.L. McKenney, ''SABRE: The Development of Information-Based Competence and Execution of Information-Based Competition,'' *IEEE Annals of the History of Computing,* vol. 17, no. 3, 1995, pp. 30–57.
46. E.W. Pugh, L.R. Johnson, and J.H. Palmer, *IBM's 360 and Early 370 Systems,* MIT Press, 1991.
47. A good idea of the techniques available to ordinary programmers faced with early random-access storages systems is provided in D.D. McCracken, H. Weiss, and T.H. Lee's *Programming Business Computers,* John Wiley, 1959, chapter 19. For its evolution, see R.H. Buegler, ''Random Access File System Design,'' *Datamation,* vol. 9, no. 12, 1963, pp. 31–33, and R.G. Canning, ''New Views on Mass Storage,'' *EDP Analyzer,* vol. 4, no. 2, 1966.
48. ''Management and the Computer: A Wall Street Journal Study of the Management Men Responsible for their Companies' Purchases of Computer Equipments and Services,'' *Wall Street J.,* 1969, Data Processing Management Assoc. records, CBI 88.
49. Anonymous, ''EDP Salary Survey–1969,'' *Business Automation,* vol. 16, no. 6, 1969, pp. 48–59.
50. Developments in this area were analyzed in Canning's ''New Views on Mass Storage'' and R.G. Canning, ''Data Management: File Organization,'' *EDP Analyzer,* vol. 5, no. 12, 1967.
51. IDS is often, and with some justification, called the first DBMS, but the initial version of IDS lacked some of the features in the Codasyl definition of a DBMS. Record definitions were punched directly onto cards in a special format rather than being defined and modified via a data-definition language. It did not provide an interface for ad-hoc querying, or support for online access, because it was created purely to support MIACS. It did not provide different views or subsets of the overall database to different users. Neither did it support multiple databases simultaneously.
52. An early description of IDS in the context of the integrated systems project is given in C.W. Bachman, ''GEICS - General Electric

Integrated Computer System,'' Charles W. Bachman, papers, CBI 125, n.d.
53. T. Haigh, ''Inventing Information Systems: The Systems Men and the Computer, 1950–1968,'' *Business History Rev.,* vol. 75, no. 1, 2001, pp. 15–61.
54. C.W. Bachman, ''Oral History Interview by Thomas Haigh,'' ACM Oral History Interviews collection, 25–26 Sept. 2004.
55. E.F. Codd, ''A Relational Model for Large Shared Databanks.'' *Comm. ACM,* vol. 13, no. 6, 1970, pp. 277–390.
56. K.R. Blackman, ''IMS Celebrates Thirty Years as an IBM Product,'' *IBM Technical J.,* vol. 37, no. 4, 1998, pp. 596–603.
57. An excellent contemporary introduction to the first public version of IMS is given in J.W. Adams, ''IMS - Information Management System/360,'' *Proc. Share 31,* vol. 2, Share, 1968, pp. 231–263.
58. R.L. Patrick, ''Oral History Interview with Thomas Haigh,'' Computer History Museum, 2006.
59. Many sessions were devoted to GIS at Share 27 in 1966 as well as 28 and 29 in 1967. By 1969, it had at least some actual users; see J.F. Fry, ''Recent User Applications with GIS,'' *Proc. Share 33,* Share, 1969, pp. 295–296. Its early capabilities are summarized in Codasyl Systems Committee, ''Survey of Generalized Data Base Management Systems,'' sect. 4, 1969.
60. M. Campbell-Kelly's, *From Airline Reservations* (MIT Press, 2003, p. 147) claims a 1968 date for Total's introduction, based on an analyst report. However, the Computer History Museum's online Software Histories Collection suggests that Total was developed during 1969 and formally released in Jan. 1970 with advertisements in *Computerworld.*
61. T. Haigh, '''A Veritable Bucket of Facts': Origins of the Data Base Management System,'' *Proc. 2nd Conf. History and Heritage of Scientific Information Systems,* M.E. Bowden and B. Rayward, eds., Information Today Press, 2004, pp. 73–78.
62. SAGE is discussed in P. Edwards, *The Closed World: Computers and the Politics of Discourse in Cold War America,* MIT Press, 1996, and T.P. Hughes, *Rescuing Prometheus,* Pantheon Books, 1998.
63. Anonymous, ''A Panel Discussion on Time-Sharing,'' *Datamation,* vol. 10, no. 11, 1964, pp. 38–44, and System Development Corp., ''Preprint for Second Symposium on Computer-Centered Data Base Systems, Sponsored by SDC, ARPA, and ESD,'' Burroughs Corp. Records, CBI 90, 1 Sept. 1965.
64. C. Baum, *The System Builders: The Story of SDC,* System Development, 1981, pp. 116–121, and A.H. Vorhaus, ''TDMS: A New Approach to Data Management,'' *Systems & Procedures J.,* vol. 18, no. 4, 1967, pp. 32–35.
65. R.V. Head, ''Data Base Symposium,'' *Datamation,* vol. 11, no. 11, 1965, p. 41.
66. The phrase ''data base management system'' was used at least once before the renaming of the DBTG to describe IBM's forthcoming GIS. J.H. Bryant and P. Semple, ''GIS and File Management,'' *Proc. ACM 21st Nat'l Conf.,* ACM Press, 1966, pp. 97–107. List processing seems in retrospect an odd choice, it was perhaps fashionable from its association with work in artificial intelligence.
67. Codasyl Data Description Language Committee, ''Codasyl Data Description Language: J. Development,'' US Govt. Printing Office, 1974.
68. Anonymous, ''Codasyl Data Management Report in Print,'' *Data Base,* vol. 1, no. 4, 1969, p. 4. The report was widely circulated, and large excerpts were published as Codasyl Data Base Task Group, ''Data Base Task Group Report to the CODASYL Programming Language Committee,'' *Data Base,* vol. 2, no. 2, 1970, pp. 11–18.
69. Codasyl Data Base Task Group, ''Codasyl Data Base Task Group: April 1971 Report,'' ACM, 1971.
70. For example, see T.W. Olle, ''Recent CODASYL Reports on Data Base Management,'' *Data Base Systems,* R. Rustin, ed., Prentice-Hall, 1972, pp. 175–184, and a series of other articles as well as participation in a number of discussion panels, such as V.C. Hare, Jr. ''A Special Report on the SIGBDP Forum: 'The New Data Base Task Group Report,''' *Data Base,* vol. 3, no. 3, 1971, p. 1. Olle tells the story of this era in T.W. Olle, ''Nineteen Sixties History of Data Base Management,'' *Int'l Federation of Information Processing History of Computing and Education 2* (HCE2), vol. 215, J. Impagliazzo, ed., Springer, 2006, pp. 67–75.
71. Codasyl Systems Committee, ''Survey of Generalized Data Base Management Systems,'' 1969. This report was until recently hard to obtain outside the archival holdings of the Charles Babbage Inst. but has now been added along with other Codasyl materials to the ACM Digital Library.
72. Codasyl Systems Committee, ''Feature Analysis of Generalized Data Base Management Systems,'' currently distributed by ACM, 1971.
73. V.C. Hare, Jr., ''A Special Report on the SIGBDP Forum: 'The New Data Base Task Group Report,''' *Data Base,* vol. 3, no. 3, 1971, p. 1.
74. Guide/Share Data Base Requirements Project, ''Resolution 69-001-00, November 4, 1969,'' *Proc. Share 34,* Share, 1970, pp. 697–712.
75. See material in the proceedings of Share 38 and 40 (March 1973). A good summary of the work of this group through 1972 is given in R.G. Canning, ''The Debate on Data Base Management,'' *EDP Analyzer,* vol. 10, no. 3, 1972,

pp. 1–16. I am not sure whether the group eventually produced actual specifications, but its work does not appear to have had much influence on product developments.
76. The Codasyl Data Description Language Committee's ''Codasyl Data Description Language'' dates this decision to Codasyl's 10th Anniversary Meeting. It is reflected in the DBTG report issued later that year.
77. The committee's work on the DML first appeared as Codasyl Database Language Task Group, Codasyl Cobol Database Facility Proposal. Ottawa: Dept. of Supply and Services, Government of Canada, Technical Services Branch, 1973 and was subsequently issued in the Cobol *J. of Development.* Work on these standards continued into the 1980s, first through a new committee set up within Codasyl, and later at ANSI. This included a FORTRAN DML, to complement the Cobol DML in the earlier reports. Codasyl Fortran Data Base Manipulation Language Committee, ''CODASYL Fortran Data Base Facility,'' *J. Development,* 110-GP-2, 1977.
78. Performance Development Corp., ''An Interview With Charles W Bachman (Part II),'' *Data Base Newsletter,* vol. 8, no. 5, 1980, Charles W. Bachman papers, CBI 125.
79. C.W. Bachman, ''The Programmer as Navigator,'' *Comm. ACM,* vol. 16, no. 11, 1973, pp. 653–658.

Thomas Haigh is a historian of computing and an assistant professor in the School of Information Studies at the University of Wisconsin, Milwaukee. He has published on many aspects of the history of computing (see www.tomandmaria.com/tom). Haigh has a PhD in the history and sociology of science from the University of Pennsylvania. He is also Biographies department editor of *Annals* and chairs the SIGCIS interest group for historians of computing. Contact him at thaigh@computer.org.