DOI:10.1145/3649887

A discussion of the evolution of the database industry over the past half century, and why the relational database concepts introduced by E.F. Codd have proven to be so resilient over several decades.

BY DONALD CHAMBERLIN

50 Years of Queries

E.F. CODD'S "A Relational Model of Data for Large Shared Data Banks"[10] is one of the most influential papers in all of computer science. In it, Codd defined concepts that are still in widespread use today, more than five decades later, including defining the theoretical foundation of the relational database industry.

When Codd's paper appeared in *Communications of the ACM* in June 1970, I was a student member of ACM, but I didn't receive the issue right away. I was driving cross-country from Stanford University to take a summer job at IBM's T.J. Watson Research Center in Yorktown Heights, New York. Before long, my summer job turned into a permanent IBM job, and I joined a group that was looking into the future of data management. My first task was to get up to speed on the current state of the art.

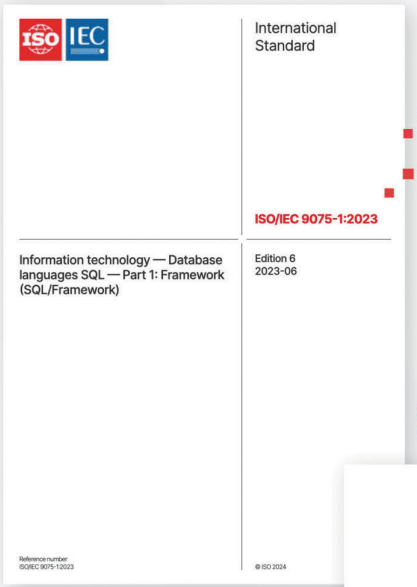
Data has been stored in digital form for a long time. Herman Hollerith invented punched cards to process the 1890 U.S. census. Punched cards had a successful 65-year product life until they were largely replaced by magnetic tapes in the 1950s. In the mid-20th century, data was typically stored on a magnetic tape and dedicated to a specific application. A tape might, for example, be used by an inventory-control application. Periodically, maybe once a week, the inventory-control job would read the tape sequentially, applying updates as it went along and producing a new, updated inventory tape. (As a college student in 1964, I had a summer job as a computer operator, running jobs like this.)

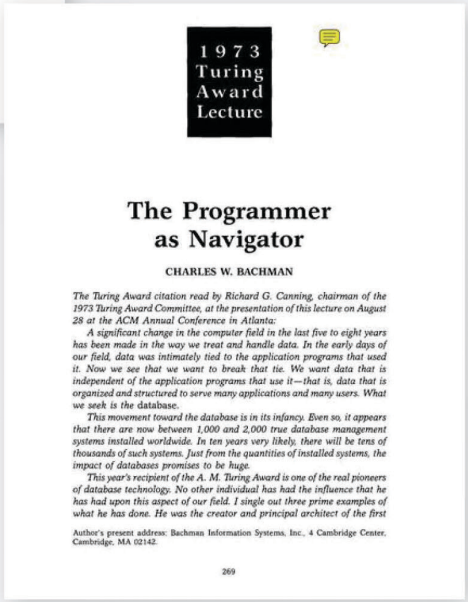
The advent of magnetic disks, introduced with the IBM RAMAC in 1956,[16] had a radical impact on how data was stored and processed. It was no longer necessary for applications to process data sequentially, since data items stored on disks could be accessed directly in any order. This gave rise to a new wave of innovation in how data should be organized on disk.

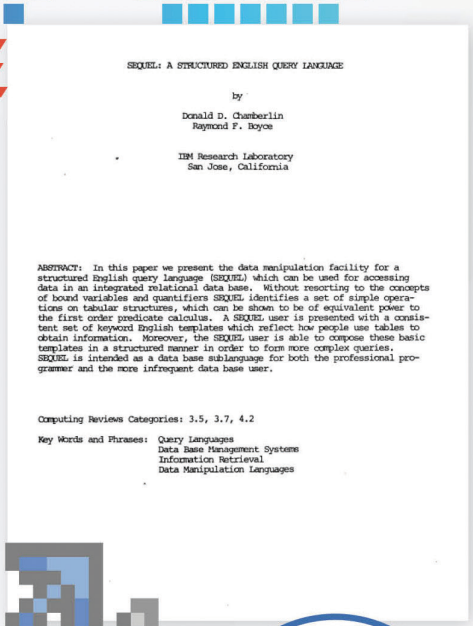
In the 1960s, a team of IBM engineers working on a NASA contract developed a disk-based information storage and retrieval system for use in the Apollo moon landing program.

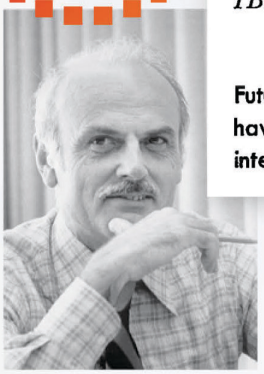
» key insights

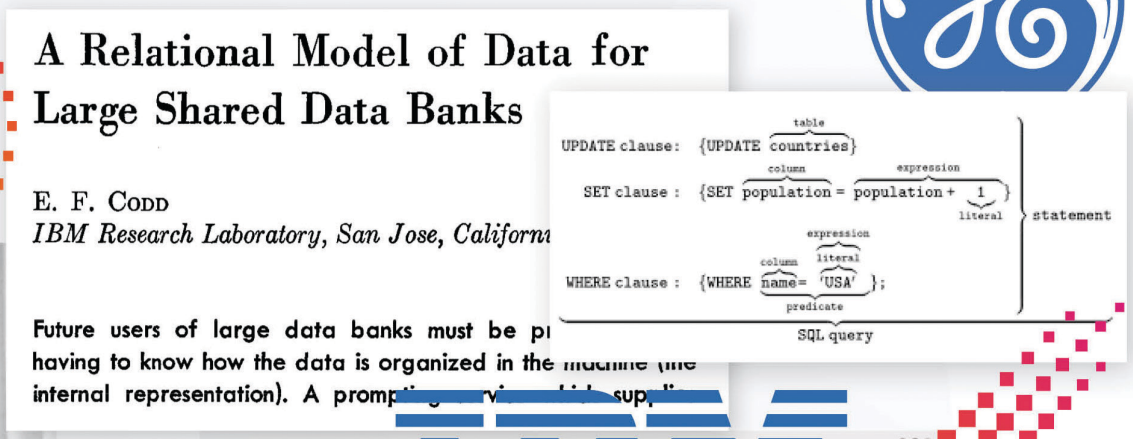
- **The relational data model, proposed by E.F. Codd in 1970, is the most widely used format for business data. Its practical feasibility was demonstrated in the 1970s by experimental prototypes at IBM Research and the University of California. The 1980s saw a proliferation of relational database products.**
- **SEQUEL (later shortened to SQL) was designed in 1974 as a language for untrained users, but it has been used mainly by professional programmers. Acceptance of SQL was aided by its adoption as an ANSI Standard and by the availability of high-quality open-source implementations. Today, SQL remains the most widely used query language.**
- **Current requirements for massive scalability have led to new "NoSQL" system designs that relax some of the constraints of relational systems.**












```
user ~/projects/database
> show GetPost.sql
```

File: GetPost.sql

```sql
SELECT
  post.id AS post_id,
  post.text AS post_text,
  user.username AS user_username,
  user.name AS user_name,
  COUNT(DISTINCT like.user_id) AS num_likes,
  COUNT(DISTINCT comment.user_id) AS num_comments
FROM post
INNER JOIN user ON post.user_id = user.id
LEFT JOIN post_like AS like ON post.id = like.post_id
LEFT JOIN post_comment AS comment ON post.id = comment.post_id
GROUP BY post.id
ORDER BY post.created_at DESC
LIMIT 10;
```

This system, named Information Management System (IMS), was made generally available to IBM customers in 1969. IMS organized data on disk in the form of hierarchies of "parent" and "child" records.

At about the same time, a General Electric employee named Charles Bachman, known to his friends as Charlie, was designing a system—called Integrated Data Store (IDS)—for storing and retrieving data. Like IMS, IDS stored data on disk in the form of records and connections between records. Users retrieved information by explicitly referencing these connections, following paths from one record to another. Unlike IMS, however, IDS did not constrain the records to be connected in a hierarchical pattern but allowed records to be connected in networks of arbitrary complexity.

As he worked on the design of IDS, Bachman had an important insight. If data was to be stored on disk and accessed in arbitrary order, there would no longer be a need for it to be dedicated to a single application. A new abstraction layer could be added above the operating system, managing shared data for multiple applications. This new abstraction layer, called a "database management system," could eliminate redundancy and make data consistent across applications. It could provide control over access to data by different categories of users. The database management system could provide services such as backup and recovery in the event of hardware or software failures. It could also provide transaction semantics to keep multiple concurrent users from interfering with each other.

For his work in developing the concept of an integrated data management system, exemplified by IDS, Charlie Bachman received the ACM A.M. Turing Award in 1973. That year, at the ACM annual conference in Atlanta, Bachman presented a Turing Lecture titled "The Programmer as Navigator,"[2] in which he presented the concept of data as a "space" in which programmers could navigate, following connections between records to find the answer to a question. The topology of a data space might be based on hierarchies, as in IMS, or on more general networks, as in IDS. Systems based on one of these data models came to be known as "navigational" systems.

In the 1960s, a team of IBM engineers working on a NASA contract developed a disk-based information storage and retrieval system for use in the Apollo moon landing program.

DBTG

When I arrived at IBM Yorktown in 1970, Charlie Bachman had not yet received the Turing Award, but his ideas were already quite influential in the database industry. A movement was underway to standardize the interface presented to application programs by a database management system. Standardization of this interface would allow multiple vendors to develop compatible database systems and applications to run on multiple database systems.

COBOL, a popular programming language for business applications, had been designed by an organization called the Conference on Data System Languages (CODASYL). In the late 1960s, CODASYL created a working group called the Data Base Task Group (DBTG) to define a standard sublanguage for database applications, to be embedded in COBOL. Charlie Bachman was a member of DBTG, and its work was strongly influenced by his ideas and experience in designing IDS. DBTG issued a preliminary report in 1969 and a final report in 1971.[9] In the early 1970s, the DBTG report was considered the leading candidate for a standard database language.

In mid-1972, my research group at IBM acquired another new employee, Ray Boyce, who had just completed his Ph.D. at Purdue. Together, Ray and I studied the DBTG report. The report called for data to be organized using a concept called *sets* (I use italics here to distinguish a DBTG *set* from a mathematical set.) A *set* consisted of one owner record and possibly many member records. A member record of one *set* could be an owner record of one or more other *sets*. Each type of record had a "location mode" that controlled how the record could be accessed, and a "*set* occurence selection" rule that controlled how new records would be assigned to *sets*. An application program could navigate, one record at a time, following connections from one *set* to another. A collection of "currency indicators" re-

corded the "current record" of each *set* and of the "run unit." Navigation was done by a FIND command that had seven different formats. The meaning of each command depended implicitly on currency indicators, location modes, and *set* occurrence selection rules. The navigation process was constrained to follow paths that had been anticipated and built into the database design.

Ray and I studied the DBTG programming interface and wrote a few example applications. We hoped to contribute to the technical literature by writing a review of the DBTG report and making some proposals for improving it. We were confident that if we could master something as complex as this, our careers would be off to a good start.

As I was learning the ropes at IBM Yorktown, E.F. (Ted) Codd was continuing his work on the relational data model at the IBM San Jose Research Laboratory, a sister lab to Yorktown. As part of my work in learning the state of the art in database management, I read Codd's 1970 paper. On first reading, I was not too impressed. The paper contained a lot of mathematical jargon. It introduced the concepts of data independence and normalization, defined a relation as a subset of the Cartesian product of a set of domains, proposed that the first-order predicate calculus could serve as a standard for measuring the expressive power of query languages, and introduced a set of operators that became known as the "relational algebra." My impression was that the paper was of some theoretical interest but was not grounded in practical engineering.

In the fall of 1972, I attended a seminar at IBM Yorktown that improved my understanding of Codd's relational data model. I was interested enough in this work that I requested permission to attend an event later that year in Miami Beach called COINS-72, the 4th International Symposium on Computer and Information Sciences. Traveling from New York to Miami Beach in December had a certain appeal; best of all, one of the scheduled speakers was Chris Date, presenting a tutorial on relational database systems. I attended COINS-72 and met Ted Codd in person for the first time, on the beach at the Fontainebleau Hotel.

I attended Chris Date's tutorial, and I can only describe it as a conversion experience. For the first time, I understood the simplicity and power of the relational data model. The model had no currency indicators or set occurrence selection rules, yet it allowed queries to be expressed, in a compact and accessible form, that would require long and complex programs in the DBTG approach. At the symposium, I spent some time with Ted Codd and Chris Date, discussing relational databases and learning about Ted's ongoing work at the IBM research lab in San Jose.

When I returned to New York, I was no longer interested in DBTG queries, and my enthusiasm spread quickly to my friend Ray. We stopped thinking about incremental changes to DBTG and began thinking about relational query languages.

Research Prototypes

The essence of a relational system is that all information is represented by data values, never by explicit connections between records. Queries are framed in a high-level descriptive language based only on data values. An optimizing compiler then translates each query into an efficient plan, using access aids that underlie the data values (B-tree indexes, hash tables, sort-merge join algorithms, and so on). Users do not need to see the access aids—in fact, they can be changed and new ones can be added, without affecting existing applications (except possibly by improving performance). This is basically the same idea found in high-level programming languages, in which mathematical formulas are compiled into procedures for loading registers and performing arithmetic.

Following the publication of Codd's 1970 paper, relational databases were getting a good deal of attention, but it still wasn't clear how practical they were. The whole idea hinged on building an optimizing compiler to translate high-level descriptive queries into efficient access plans. Some people were skeptical that a compiler could do this job as well as an expert human programmer could. The job of optimizing a database query is much more complex than managing registers. The advantages of the relational approach for users were well understood, but the question remained whether a relational system could meet the requirements of large-scale, multi-user database applications.

In the early 1970s, work on relational database systems was underway at multiple IBM locations. The Peterlee Relational Test Vehicle (PRTV) was under construction at the IBM Scientific Center in Peterlee, U.K.[23] Development of a relational product called Business System 12 was taking place at IBM's Bureau Service subsidiary in Uithoorn, Netherlands.[17] In 1973, the IBM Research Division decided to create a new project at their San Jose, CA laboratory, where Ted Codd was working. The project, to be called "System R," would build an industrial-strength relational database prototype, to gain experience and to influence IBM's plans for future database products.

Research Division employees with interest or experience in data management were gathered from several locations and moved to San Jose at the company's expense to work on System R. Already in Codd's group at San Jose were Morton Astrahan, Jim Gray, Jim Mehl, Phyllis Reisner, Don Slutz, and Irv Traiger. Moved to San Jose from Yorktown were myself, Mike Blasgen, Ray Boyce, Frank King, Franco Putzolu, and Vera Watson. From IBM's Cambridge Scientific Center came Raymond Lorie. We were soon joined by some new hires with recent Ph.Ds: Tom Price from Stanford, Bruce Lindsay from UC Berkeley, Pat Selinger from Harvard, and Brad Wade from Purdue. Other participants at various times included Ron Fagin, Bob Yost, Mario Schkolnick, Ray Strong, and Kapali Eswaran. The typical size of the System R staff during the seven-year life of the project was about 14 professionals.

At about the same time, a research project was taking shape at UC Berkeley under the leadership of Professors Michael Stonebraker and Gene Wong. This project was called INGRES, an acronym for Interactive Graphics and Retrieval System (also the name of an 18th century French artist, Jean Auguste Dominique Ingres). Like Sys-

tem R, INGRES intended to explore relational database technology and demonstrate its feasibility for use in a production environment. Funding for INGRES was provided by several federal agencies, including the National Science Foundation. Over its active life from 1973 to 1979, INGRES provided research opportunities and practical experience for about two dozen UC students, many of whom went on to take leading positions at various companies in the rapidly growing database industry.

Both the System R group and the INGRES group had ambitious agendas. They had to develop software techniques for implementing relational data on top of an operating system (VM/CMS in the case of System R; Unix in the case of INGRES). They also had to design a user interface, including a relational query language, and build an optimizing compiler to translate that query language into efficient execution plans. Both System R and INGRES existed in environments that encouraged their members to attend conferences, share experiences with colleagues (including each other), and publish papers in the open technical literature. This open collaborative environment would prove to be crucial to the impact that both projects would have on the software industry. Over the course of their existence, System R and INGRES each published more than 40 technical papers.[6,22] In 1988, System R and INGRES jointly received the ACM Software System Award for their contributions to relational database technology.

SIGFIDET 1974

The principal venue for exchange of research on data management in the early 1970s was the annual meeting of ACM SIGFIDET, the Special Interest Group on File Definition and Translation. The SIGFIDET meeting of 1974 took place in June, in Ann Arbor, Michigan. This meeting was particularly interesting because it was attended by both Bachman and Codd, the principal advocates of the network and relational data models, respectively. In a special session, Bachman and Codd each presented a talk on the advantages of their respective data models. The prepared talks were followed by a face-to-face panel discussion featuring both Bachman and Codd (it was billed as a panel discussion, but everyone knew that it was a debate).

From my perspective, the SIGFIDET meeting of 1974 was a watershed event for the database industry. Before this meeting, the network data model, exemplified by the DBTG report, was considered the "mainstream" in data management, and the relational data model was considered a "challenger," a disruptive and unproven proposal. After the SIGFIDET meeting, the advantages of the relational model, isolating logical information from its physical representation, were well understood. The relational model had become the new mainstream for research in data management, but the question of a practical implementation remained open: Were relational databases a form of science fiction, or were they ready for prime time? The two research groups, INGRES and System R, were focused on finding the answer to this question.

Two more facts about SIGFIDET 1974 may be worth mentioning. The first is that, after this meeting, the participants in the Special Interest Group realized that what they were doing was managing data, and changed the name of the group to SIGMOD, the Special Interest Group on Management of Data. SIGMOD continues to hold annual meetings, which are among the most widely respected conferences in the field of data management. The second fact is that, hidden on page 249 of the Proceedings of SIGFIDET 1974 was a short paper by Don Chamberlin and Ray Boyce, titled "SEQUEL: A Structured English Query Language".[8]

A paradigm shift like the change from network-structured data to relational data happens slowly. But, because it featured a direct confrontation between advocates of the two alternative data models, I consider SIGFIDET 1974 to be the event that "starts the clock" on 50 years of relational databases.

SEQUEL

When Ray Boyce and I arrived at the IBM Research Laboratory in San Jose, at the start of the System R project, our interests were focused on user interfaces and query languages. In his early papers, Codd had described two relational query languages; a "relational algebra" consisting of operators like "projection" and "join"; and a "Relational Calculus," based on first-order logic. Codd's papers proved that these two languages were equivalent in their expressive power.

Ray and I understood the power and elegance of the relational approach, but we thought that Codd's ideas might get more traction if they were couched in terminology that was more familiar to people with no mathematical background. We set out to define a more "user-friendly" relational query language. Our target user was someone whose work required access to large volumes of data, but who had no programming experience and did not want to become a computer programmer. This user might be an urban planner or an insurance analyst. This user might make up questions and want them answered quickly, without turning them over to a technical staff. The questions might vary from day to day and could not be anticipated in advance by a database designer. To serve the needs of this user, Ray and I wanted to express questions in a way that was as close as possible to natural language, while still having a well-defined syntax and semantics. Our specific goals were to design a query language with the following properties:

- The language should be declarative (non-procedural) and based on Codd's relational concepts.
- The language should be framed in familiar English keywords, with no jargon or special symbols, and easy to type on a keyboard.
- In addition to the usual relational operations of selection, projection, and join, the language should provide a way to partition a table into groups and apply aggregating functions such as SUM or AVERAGE to the groups.
- Queries should resemble natural language to the extent that a user with no specialized training could, in simple cases, understand the meaning of a query simply by reading it. We called this the "walk-up-and-read" property.

We called this language SEQUEL, an acronym for "Structured English Query Language." While design-

ing SEQUEL, Ray and I engaged in something that we called the "Query Game." Experimenting with different syntaxes, we made up example queries and challenged each other to express them. Some of these queries were based on a simple table of employees with the following structure:

Table 1. A table of employee records.

Emp

Name	Deptno	Salary	Manager
Harry	A15	65000	Sally
Sally	A01	59000	Megan

Queries 1 and 2 (below) are examples from the Query Game, with their expressions in SEQUEL, as the language (now SQL) currently exists. These queries demonstrate two important features: join (in this case, self-join) and grouping.

Query 1: Find employees who earn more than their managers, and list their names, salaries, and manager's salaries.

```
SELECT e.Name, e.Salary,
  m.Salary AS mgr_salary
FROM Emp AS e, Emp AS m
WHERE e.Manager = m.Name
AND e.Salary > m.Salary
```

Query 2: List the department numbers and average and maximum salaries of departments having 10 or more employees.

```
SELECT Deptno,
  AVG(Salary) AS avgsal,
  MAX(Salary) AS maxsal
FROM Emp
GROUP BY Deptno
HAVING COUNT(*) >= 10
```

The first publication of the proposed SEQUEL language was the paper that appeared in the Proceedings of SIGFIDET 1974.[8] Shortly after this paper appeared, my friend Ray Boyce died, suddenly and unexpectedly, of a brain aneurism.

Ray's death was a great loss, but the SEQUEL work continued. To test our hypothesis that SEQUEL could be understood by non-programmers, a psychologist at IBM Research named Phyllis Reisner conducted an experiment at San Jose State University in which she showed that college students with no experience in programming or data handling could gain a reasonable proficiency with SEQUEL after a few hours of instruction.[21]

The relational model had become the new mainstream for research in data management, but the question of a practical implementation remained open: Were relational databases a form of science fiction, or were they ready for prime time?

As the System R project matured, the SEQUEL language continued to evolve. A 1976 paper titled SEQUEL 2[7] extended the query syntax to cover insert, delete, and update operations; view definitions; integrity assertions; and triggered actions. The language defined in that paper would be immediately recognized by database developers working today. In 1977, the SEQUEL name was shortened to SQL, an acronym for "Structured Query Language."

Commercialization

The System R project produced an experimental prototype that was used at about 20 internal IBM laboratories and, on a "joint study" basis, by three selected IBM customers: Boeing, Upjohn, and Pratt & Whitney. The prototype ran on IBM mainframes under the VM/CMS operating system. The System R research project ended in 1979 and its results were turned over to IBM product development divisions for commercialization.

In 1977, the founders of a small company named Software Development Laboratories (SDL) took an interest in some of the System R papers, including the SQL specifications published in 1974 and 1976. The SDL founders saw an opportunity here. Assuming correctly that IBM would eventually release an SQL product on its mainframe computers, they decided to build a compatible product on a less expensive platform, to be named Oracle, which was developed on a DEC PDP-11. Its source code was written in C, which made it easily portable to other platforms. Oracle Database, the first commercial implementation of the SQL language, was released in 1979. Available on the popular DEC VAX minicomputer, Oracle was an immediate commercial success. In 1983, the SDL company changed its name to Oracle.

The INGRES project at UC Berkeley also produced an experimental prototype and distributed it freely to other universities and research labs. By 1978, INGRES had about 300 installations and had become the de facto standard for use in university classes on database management. In

1980, the leaders of the INGRES project spun off a commercial company, funded by venture capital and initially named Relational Technology Inc. (RTI), which had its own management and technical staff that was independent of the university. This enabled the INGRES project at the university to continue its focus on research issues of academic interest. The first task for RTI was to port the INGRES code from Unix to run on the DEC VAX platform. The resulting commercial product was released in 1981, supporting a query language called QUEL. RDI changed its name to Ingres Corporation in 1989.

IBM was not in a hurry to release a relational database system on its strategic mainframes to compete with its successful IMS database product. But IBM's mid-range platform, a competitor to DEC VAX, needed a database system to compete with Oracle and INGRES. It took IBM about two years to turn the System R prototype into a commercial product running on the VSE and VM operating systems. This product, called SQL/DS, was released in 1981, at about the same time as INGRES but two years behind Oracle.

IBM eventually released a relational database product on MVS, its strategic mainframe platform. This product, named DB2, was released on a limited basis in 1983, followed by general availability the following year.[15] By this time, Oracle had established a commanding lead in the relational database industry.

Standardization

In 1978, as the database industry was growing in importance, the American National Standards Institute (ANSI) formed a committee to develop standards for database languages. The purpose of these standards was to create a market in which database vendors could compete to implement a standard interface, and users could be assured of multiple sources of compatible products. The original name of the database committee was ANSI X3H2, but the committee has undergone several name changes and is currently called INCITS DM32. In this article, I'll refer to the committee as "H2," which has been an enduring part of its official name during most of its history.

The first SQL language standard, about 90 pages long, was adopted as an American National Standard in 1986.

The initial work of the H2 committee was focused on developing a network data language (NDL) based on the CODASYL DBTG Report of April 1971. But in 1982, as relational database systems were beginning to appear in the marketplace, H2's charter was extended to include development of a standard relational database language, initially referred to as RDL. The first task of the committee was to find a starting point for the design of RDL.

Two relational query languages were available in the marketplace in 1982: SQL, marketed by IBM and SDL (later Oracle); and QUEL, marketed by RTI (later Ingres). Either of these languages might have served as the starting point for a relational language standard. The H2 committee decided to base its future work on SQL, possibly because the IBM representative to H2 presented a detailed language specification.

Beginning with the SQL specification from IBM, the H2 committee spent about two years debating various modifications and improvements to the language. By 1984, the committee faced a difficult decision. SQL products from IBM and Oracle were achieving success in the marketplace and were (mostly) compatible with each other. The H2 committee could choose to base its standard on the existing products, in which case it would have a living standard with multiple implementations and would be in a strong position to influence the future evolution of the industry. Alternatively, it could continue its work developing an (arguably better) language that might never be implemented. At a meeting in October 1984, the H2 committee chose the former approach.[12] The first SQL language standard, about 90 pages long, was adopted as an American National Standard in 1986[1] and as an International Standard early in the following year.[18] Since its initial publication, the SQL Standard continues to evolve, with a new version published about every five years. The latest version, informally called SQL:2023, was published in June 2023.

Another significant development came from the National Institute of Standards and Technology (NIST).

Unlike ANSI, which is a voluntary association of private companies, NIST is a branch of the federal government. In 1992, NIST published a Federal Information Processing Standard, called FIPS-127,[19] which specified the requirements for relational database systems to be purchased by the U.S. government. FIPS-127 was essentially identical to the ANSI SQL standard that was current at the time (SQL:1992 Entry Level). Most importantly, NIST created a test suite of several hundred test cases, and offered a service of testing systems for conformance to FIPS-127. About a dozen companies had their SQL products certified under FIPS-127 and became eligible to sell them to the federal government. Naturally, this was a big help in marketing these products.

The H2 committee's strategy of tying standards closely to commercial products proved to be successful. Over several decades, H2 provided a mechanism for the controlled evolution of SQL to meet changing requirements. Under the guidance of H2, the SQL standard has grown to include referential integrity, outer joins, date and time datatypes, OLAP features, window functions, recursive queries, stored procedures, constraints and triggers, and many more features.

During the nearly four-decade life of the SQL Standard, the H2 committee has been chaired by Don Deutsch, and the editor of the Standard, doing most of the writing, has been Jim Melton. I believe that Don and Jim deserve a great deal of credit for maintaining SQL as a well-defined standard while allowing it to evolve to meet changing requirements.

Proliferation

The 1980s saw the introduction of a new generation of minicomputers that were dramatically less expensive than earlier computers. For the first time, computers capable of managing data were within the financial reach of small companies or departments of large companies. The result was an explosion of demand for database systems. Relational databases, with their simple data model and easy-to-learn user interface, were ideally positioned to meet this demand. Since IBM, Oracle, and INGRES had paved the way and the federal government had given its blessing with FIPS 127, relational databases clearly represented the future of data management. A good account of the growth of the relational database industry during the 1980s has been published in a special issue of *IEEE Annals of the History of Computing*.[14]

During the early 1980s, the market leaders in relational database management were Oracle and RDI (later INGRES), both of which ran on the popular DEC VAX platform. When the IBM Personal Computer became available, it became a popular tool for developing Oracle applications, which could then be moved to the VAX for production. The INGRES system continued to feature the QUEL query language until 1984, when it added an SQL interface in order to compete directly with Oracle.

Informix (initially Relational Database Systems Inc.) was formed in 1980 by Roger Sippl to bring relational database technology to the Unix world (interestingly, at about the same time, INGRES, developed on Unix, was being ported to run on VAX/VMS). Informix initially marketed its own query language but, like INGRES, it transitioned to SQL in 1984. Informix was ultimately acquired by IBM.

Other notable SQL implementations that became available during the 1980s include Sybase, founded in 1984 by Bob Epstein, an alumnus of the INGRES project; NonStop SQL, a fault-tolerant system released by Tandem in 1986; and Microsoft SQL Server, released by Microsoft in 1989.

Relational database systems were attracting so much attention during the 1980s that Codd published a list of Twelve Rules[11] (actually 13 rules, numbered 0 to 12) which served as his definition of a genuine relational system. The most important of these rules, called the Information Rule, stated that "all information in a relational database must be represented explicitly at the logical level and in exactly one way: by values in rows and columns of tables."

Open source. The mid-1990s saw some game-changing developments in the database industry. Three open-source SQL implementations became available for free: MySQL, PostgreSQL, and SQLite. For companies developing new Web applications, these systems offered a compelling business model.

MySQL (https://mysql.com), developed by Michael Widenius and David Axmark, was first released in 1995 by the Swedish company MySQL AB. MySQL soon became popular as part of the LAMP stack (Linux, Apache, MySQL, and PHP) for developing Web applications. It now has an active installed base of 5 million users. In 2008, MySQL was acquired by Sun Microsystems, which in turn was acquired by Oracle in 2010.

At the time of the Oracle acquisition, a copy of MySQL was separated from the Oracle version and is being maintained independently by Michael Widenius and some of the other original MySQL developers under the name MariaDB (https://mariadb.org). MariaDB is promised to be open-source forever and is now evolving separately from MySQL. MySQL and MariaDB are named after Widenius' two daughters, My and Maria.

PostgreSQL (https://postgresql.org) is derived from POSTGRES, the successor to the INGRES project at U.C. Berkeley. As a research project, POSTGRES focused on an extensible type system. When the research project ended, the POSTGRES code continued to be maintained by a volunteer organization called the PostgreSQL Global Development Group. Its first SQL-based version was released in 1997. PostgreSQL is the most fully featured and most complex of the open-source SQL implementations. For their work on POSTGRES, Michael Stonebraker and Larry Rowe received the SIGMOD Systems Award in 2015.

SQLite (https://sqlite.org), designed by Richard Hipp, was first released in 2000. Unlike MySQL and PostgreSQL, SQLite is not a client-server system in which the server runs as a separate process. Instead, SQLite is a library of C-language functions that can be called directly from application code and run in the application process. SQLite claims to be the most widely deployed database system in the world. It is embedded (invisibly) in every Apple or Android smartphone; every Mac or Windows computer; every Firefox, Chrome, or Safari brows-

er; and countless well-known applications. For designing SQLite, Richard Hipp received the SIGMOD Systems Award in 2017.

MySQL, MariaDB, PostgreSQL, and SQLite are all reliable, high-performance, standard-compliant database systems. They are all well-documented and supported by large, active user communities, and all are available with open source to everyone at no cost. Each provides an optional means by which users can pay for additional services and functionality. Because of their high quality and low cost, these four systems have become among the most widely used database systems in the world.

Controversy

Over the years, SQL has attracted a good deal of criticism, from intelligent and thoughtful critics. The points they have raised are substantive and deserve a respectful response. Among these criticisms, the following three stand out as being serious and persistent:

1. As a language, SQL lacks orthogonality.
2. Unlike relations, SQL tables (and query results) may contain null values.
3. Unlike relations, SQL tables (and query results) may contain duplicate rows.

Of course, all of these statements are true. What follows is my personal perspective on each of these statements.

Orthogonality. The operators of an orthogonal language return values without side effects, and can be nested with full generality. Orthogonality is a good design principle because it simplifies the rules that govern how the operators of a language can be combined.

SQL is not an orthogonal language. It has some operators, like GROUP BY, that can be used only in a specific context, and that have side effects. This is because, in the early days, Ray Boyce and I did not think we were designing a language for programmers. As described earlier, our target user was someone who had no programming experience and did not want to become (or rely on) a computer expert. We were aiming for a language with the "walk-up-and-read" property, in which an untrained user could often understand a query simply by reading it. For this purpose, we modeled SEQUEL on a small subset of the English language (hence the name). English, of course, is not an orthogonal language.

As it turned out, Ray and I were wrong about the predominant usage of SQL. Typically, SQL is embedded in a host programming language and used by professional programmers. In this environment, orthogonality might be more important than an English-like syntax. It's useless to speculate whether Ray and I would have put more emphasis on orthogonality if we had known that most of our users would be programmers. It's also unknowable whether our attempt to make SQL a "walk-up-and-read" language played some role in its widespread acceptance.

Nulls. In real data, values are sometimes missing. They might be missing because they are unknown, not applicable, not available yet, or for some other reason. Dealing with missing values is one of the biggest challenges of data management. Every way that I know to solve this problem has drawbacks.

The SQL CREATE TABLE statement allows users to declare, on a column-by-column basis, whether null values are permitted. The general philosophy of SQL is to provide a flexible set of tools and to trust users to use these tools to serve their own best interests. One aspect of this flexibility is the choice of whether to use the NOT NULL declaration. An example might be helpful in examining the trade-off involved in this choice.

Table 2. A table that records the daily results of some stores.

Stores_data

Store (P.K.)	Day (P.K.)	Customers	Sales	Returns
Denver	2023-06-10	null	5500	250
Tucson	2023-06-15	150	4500	null

Consider a chain of stores that gathers daily information from each store, including the number of customers who visited the store, the sales volume in dollars, and the returned-item volume in dollars. This information might be represented by the SQL table shown in Table 2. (P.K. indicates that the Store and Day columns, together, form the Primary Key).

On some days, one or more of the data items (Customers, Sales, or Returns) might be missing. One way to represent the missing data items is by null values, as shown in Table 2.

Query 3 is an example that might have been written by an analyst at the store chain.

Query 3: Compute, for each store, the average revenue (sales minus returns) per customer.

```
SELECT Store,
 AVG((Sales - Returns) /
     Customers) AS rpc
FROM Stores_data
WHERE Customers > 0
GROUP BY Store
ORDER BY Store;
```

In this query, the AVG function ignores all rows that contain null values, and returns a result based on rows that are fully populated with non-null values. This is arguably the best available approximation to the answer the analyst is looking for.

If it is desired to avoid null values, the attribute columns (Customers, Sales, and Returns) can all be declared NOT NULL. This table design has a serious flaw: No data can be recorded for a given store on days when any attribute (Customers, Sales, or Returns) is missing. For example, if the Returns data is missing for some day, the table cannot record the Customers and Sales data for that day, even if they are known.

An alternative design might be to replace the Stores_data table by three smaller tables named Customers_data, Sales_data, and Returns_data. Each of these three-column tables would contain the primary key (Store and Day) and one of the non-key attributes. For a store and day on which Customers information is missing, there would simply be no row in the Customers_data table; a similar rule applies to the other two tables. In this

design, every row represents a true statement, and all information that is known can be stored without using null values. However, in this design, the simple Query 3 must be replaced by the three-way join shown in Query 4. Query 4 returns the same result as Query 3, but probably runs more slowly. In addition, the three tables occupy significantly more space than the single table, because each primary key is replicated three times.

Query 4: Same as Query 3, for a different database design.

```
SELECT sd.Store,
  AVG((sd.Sales - rd.Returns)
      / cd.Customers) AS rpc
FROM Sales_data AS sd,
  Customers_data AS cd,
  Returns_data AS rd
WHERE sd.Store = cd.Store
AND sd.Day = cd.Day
AND sd.Store = rd.Store
AND sd.Day = rd.Day
AND cd.Customers > 0
GROUP BY sd.Store
ORDER BY sd.Store;
```

Duplicates. Elimination of duplicate rows is another area in which SQL empowers users to ask for what they want. Consider a query that returns the names and addresses of all University of California students majoring in History. There may be thousands of them. It is not very likely that there are duplicates; that would mean two or more history majors share the same name and the same address. But if there are a few duplicates like that, the user might not care very much. The user might not want to pay the cost of sorting or hashing thousands of records to ensure there are no duplicates. In this case, the user would probably specify SELECT rather than SELECT DISTINCT. SQL allows users to make this choice, both for outer-level queries and nested subqueries.

As another example, consider a point-of-sale application that gathers data as products slide through the scanner at a supermarket. In a straightforward design, each scanned product code might produce a row in an SQL table. If a grocery cart contains three identical cartons of milk, this design would result in three identical rows in the table. SQL queries could organize the data into groups as needed. If it were necessary to avoid duplicate rows, the application could generate a (rather long) primary key for each individual grocery item, but since the data will be processed only in aggregate form, this might seem unnecessary. Alternatively, the application could maintain a counter for each product code, customer, and date, and for each grocery item, it could increment (or create) the appropriate counter. This approach would require a table lookup by a three-part key for each grocery item, a slower and more complex operation than a simple insert. The point here is not that one design is better than another; it is that the database system should be flexible enough to permit the application designer to explore these trade-offs.

The general philosophy of SQL is to provide a flexible set of tools and to trust users to use these tools to serve their own best interests.

In summary, SQL is based on a belief that users have good common sense and will make decisions that serve their own interests if they are empowered to do so. An SQL database can be used with "relational discipline" by designing tables in such a way that each table has at most one column that might potentially have missing data, and by specifying a primary key for each table and NOT NULL on all non-key columns. Query writers would be required to specify SELECT DISTINCT on all query-blocks, and to avoid use of features, such as outer join, that might generate null values. It is the job of a database designer, in the context of a specific application, to weigh the advantages of relational discipline against its cost in terms of time, space, and complexity. Over time, we can expect users to evaluate this trade-off and to "vote with their feet."

Resilience

Fifty-four years after its introduction by Codd in 1970, the relational data model remains pervasive in the database industry. According to the market survey site db-engines.com, the four most popular database systems in the world in December 2023 were all relational systems: Oracle, MySQL, Microsoft SQL Server, and PostgreSQL.[13]

Fifty years after its first publication in 1974, SQL is still the most widely used database query language.

Each year, *IEEE Spectrum* publishes a survey of "Top Programming Languages." The survey gives each language an overall ranking based on its prevalence in popular development sites such as GitHub and Stack Overflow, and a separate "job opportunity" ranking based on job listings in recruiting sites. In the 2023 survey,[4] SQL ranked number 7 overall, but number 1 in job opportunities. Among all the computer languages that were in widespread use 40 years ago, the only ones still in the top 10 of the IEEE 2023 ranking were C and SQL.

It is interesting to consider why, in the rapidly evolving computer industry, the relational data model, and SQL in particular have been able to survive and prosper for five decades. Here is my guess for the main reasons behind this success story:

▸ **Codd got it right**. The relational data model, and especially Codd's Information Rule, established a simple, powerful, flexible, and elegant way to represent information. That's all there is to it. Codd had a fundamentally good idea.

▸ **We answered the performance question**. The System R and INGRES projects proved that a high-level, user-oriented relational language could be implemented with sufficient performance for use in a production environment.

▸ **Research was published early and openly**. IBM allowed Codd to publish his relational data model in the open literature. IBM also openly published the SQL language, and all the System R papers on query optimization and other topics. The INGRES project at Berkeley published all its work also, and made INGRES available with open source. There were no patent or trademark issues to stand in the way of vendors who wanted to exploit this technology.

▸ **Data is sticky**. Relational databases came along at a unique time, when many companies were putting their data online for the first time. Being first counts for a lot. Once your database applications are running, it is expensive to migrate them to a different platform. Fortunately, SQL makes it fairly easy to modify your database by adding new tables or columns, or by defining new views.

To a large extent, the data that we choose to collect, and the ways in which we choose to use it, will determine the kind of world in which our grandchildren will live.

▸ **Standards**. The ANSI Standard provided a formal definition for SQL. It created a market in which new vendors could, and did, compete for business. The H2 committee brought together a group of smart people from multiple vendors to guide the evolution of SQL. And FIPS 127 provided a standard compliance test and a license to sell database systems to the government, which did not hurt a bit.

▸ **High-quality open source implementations**. Web applications have produced an enormous stream of data that needs a place to live. MySQL, MariaDB, PostgreSQL, and SQLite offer robust, standards-compliant SQL implementations, with large, vibrant user communities, all for free. If you are a startup company that needs a database, this is a pretty good place to look.

NoSQL

Currently, many interesting developments in database management are part of a movement broadly known as "NoSQL." As described in a 2010 paper by Rick Cattell,[5] NoSQL systems are characterized by the ability to horizontally scale a high volume of simple transactions across many servers. These abilities are motivated by Web applications, in which thousands or millions of users are performing relatively simple reads and updates on shared data.

NoSQL systems usually achieve their goals of low latency, massive throughput, and high availability by relaxing one or more of the constraints of ordinary relational systems. For example:

▸ **Relational databases have rigid schemas that define their database structure**. NoSQL systems may have relaxed or partial schemas or may have no schemas at all.

▸ **Relational systems usually have transactions that make certain guarantees, including the well-known ACID properties**. NoSQL systems might make some compromises in transaction semantics. For example, an update to some piece of information that is replicated on many nodes might take a little while to propagate to all the nodes. Some applications can afford to be patient about this.

▸ **Relational systems usually implement the full SQL language**. NoSQL systems might support simpler user interfaces that omit some of the more complex and expensive operations, such as joins and grouping. The user interface might look more like an API than like a query language.

▸ **The relational data model consists of homogeneous, flat tables**. NoSQL systems are sometimes based on other data models. If they store tables, they might allow these tables to be nested. Or they might use some document-oriented format, like XML or JSON, to store documents. Or they might even be something very simple, like a key-value store.

A NoSQL system will probably not include all of these features. More likely, it will include one or two. As a result, the term NoSQL encompasses a variety of different systems and represents an active area of research and development.

It is worth noting that not all the NoSQL characteristics listed above are related to query languages. A system that has a relaxed schema and eventual consistency, for example, might still have a high-level query language. That is why NoSQL is sometimes interpreted as "not only SQL." In fact, a compatible extension of SQL called SQL++, designed for handling schema-less JSON data, has been designed at the University of California, San Diego.[20] An open-source implementation of SQL++ is available from the ASTERIX project at U.C. Irvine.[3]

Conclusion

ACM A.M. Turing Award recipients Charles (Charlie) Bachman and E.F. (Ted) Codd laid out the road map that the data management industry has followed for more than five decades. Bachman identified database management as a new level of abstraction, bridging the gap between operating systems and applications. Codd created a simple, powerful, and elegant definition for this new level of abstraction: the relational data model, which now encodes much of the world's business data. Charlie Bachman died on July 13, 2017 at his home in Lexington, MA at the age of 92. Ted Codd died on April 18, 2003 at his home in Aventura, FL, at the age of 79.

Many individuals and groups made important contributions to progress along the road envisioned by Bachman and Codd. The System R project, led by W. Frank King, and the INGRES project, led by Michael Stonebraker, developed industrial-strength relational-database prototypes and validated them with communities of early adopters. Ray Boyce and I, as members of the System R team, published the first SEQUEL language specification. Pat Selinger, also in System R, led the team that developed the first cost-based query optimizer and wrote the classic paper explaining its design. The prototypes created by System R and INGRES led directly to commercial products. Larry Ellison and Bob Miner, founders of Oracle, established the mass market for relational databases with the first widely used relational product. The ANSI H2 committee, chaired by Don Deutsch, maintained the official definition of SQL and controlled its evolution over many years. Jim Melton, editor of the SQL Standard, shepherded the standard document through nine different versions from 1986 to 2023. Leonard Gallagher, Joan Sullivan, and their colleagues at NIST created the SQL Test Suite that validated conformance to FIPS 127. The architects and builders of MySQL, PostgreSQL, and SQLite made professional-quality relational database systems available to everyone for free, ensuring that SQL would become a ubiquitous part of e-commerce infrastructure. Sometimes I wish that my good friend Ray Boyce had lived to see what happened to some of the ideas we were kicking around in 1974.

The database industry has been an exciting place to work for the last half-century. Today, almost any commercial item can be obtained simply by tapping a picture of it on a mobile phone. Within seconds, somewhere in the world, a robot begins moving to find and package that item, and it is delivered to your doorstep on the following day (or sometimes the day after that). Database technology has made this possible, bringing unprecedented convenience to the lives of people with disposable income. It has also affected our culture in many other ways, some of which are arguably less beneficial. Data is a powerful tool. To a large extent, the data that we choose to collect, and the ways in which we choose to use it, will determine the kind of world in which our grandchildren will live. C

References

1. American National Standards Institute. Database language SQL. *Technical Report ANSI X3.135-1986*, (1986).
2. Bachman, C.W. The programmer as navigator. *Commun. ACM 16*, 11 (1973), 635–658; 10.1145/355611.362534
3. Carey, M.J. AsterixDB mid-flight: A case study in building systems in academia. In *Proceedings of the 35th IEEE Intern. Conf. on Data Engineering*, (April 2019), 1–12; 10.1109/ICDE.2019.00008
4. Cass, S. The top programming languages 2023. *IEEE Spectrum* (Aug. 29, 2023); https://bit.ly/3VexjkF
5. Cattell, R. Scalable SQL and NoSQL data stores. *SIGMOD Rec. 39*, 4 (2010), 12–27; 10.1145/1978915.1978919
6. Chamberlin, D.D. et al. A history and evaluation of System R. *Commun. ACM 24*, 10 (1981), 632–646; 10.1145/358769.358784
7. Chamberlin, D.D. et al. SEQUEL 2: A unified approach to data definition, manipulation, and control. *IBM J. Res. Dev. 20*, 6 (1976), 560–575; 10.1147/RD.206.0560.
8. Chamberlin, D.D. and Boyce, R.F. SEQUEL: A structured English query language. In *Proceedings of 1974 ACM-SIGMOD Workshop on Data Description, Access and Control*, ACM, (1974), 249–264; 10.1145/800296.811515
9. CODASYL Data Base Task Group. *April 71 Report ACM SIGMOD Anthology 6*, (1971); https://bit.ly/3KBoRqz.
10. Codd, E.F. A relational model of data for large shared data banks. *Commun. ACM 13*, 6 (1970), 377–387; 10.1145/362384.362685
11. Codd, E.F. Does your DBMS run by the rules? *Computerworld* (Oct. 21, 1985); https://bit.ly/4bTpHLs.
12. Deutsch, D.R. The SQL standard: How it happened. *IEEE Ann. Hist. Comput. 35*, 2 (2013), 72–75; 10.1109/MAHC.2013.30
13. db-engines.com. *DB-Engines Ranking*; https://bit.ly/3yXIxm8
14. Grad, B. Relational database management systems: The business explosion [Guest editor's introduction]. *IEEE Ann. Hist. Comput. 35*, 2 (2013), 8–9; 10.1109/MAHC.2013.24
15. Haderle, D.J. and Saracco, C.M. The history and growth of IBM's DB2. *IEEE Ann. Hist. Comput. 35*, 2 (2013), 54–66; 10.1109/MAHC.2012.55
16. IBM Archives. *IBM 350 Disk Storage Unit*. IBM Corp.; https://www.ibm.com/history/ramac
17. IBM Corporation. *IBM Business System 12*; https://bit.ly/4cizpqF
18. ISO. Database language SQL. *Technical Report ISO 9075-1987*. International Organization for Standardization, (1987).
19. National Institute of Standards and Technology. Federal information processing standards publication FIPS 127. *Technical Report*, (1992).
20. Ong, K.W., Papakonstantinou, Y., and Vernoux, R. *The SQL++ Query Language: Configurable, Unifying, and Semi-Structured*, (2015); https://arxiv.org/abs/1405.3631
21. Reisner, P. Use of psychological experimentation as an aid to development of a query language. In *Proceedings of IEEE Trans. Software Eng. 3*, 3 (1977), 218–229; 10.1109/TSE.1977.231131
22. *The INGRES Papers: Anatomy of a Relational Database System*. M. Stonebraker (Ed.). Addison-Wesley, 1986.
23. Todd, S. The peterlee relational test vehicle - A system overview. *IBM Syst. J. 15*, 4 (1976), 285–308; 10.1147/SJ.154.0285

Donald Chamberlin (chamberlin.don@gmail.com) is a retired IBM Fellow, from the Almaden Research Center in San Jose, CA.