Early History of SQL

Donald D. Chamberlin

Editor: Craig Partridge

Ray Boyce and I first met E.F. (Ted) Codd at a symposium he organized at the IBM T.J. Watson Research Center in Yorktown Heights, New York, in 1972. Ray and I were both recent hires at the Watson Center. I had recently completed my PhD at Stanford University, and Ray had completed his at Purdue University. We were members of a recently reorganized IBM group that was looking for a mission. At that time, Ted Codd was a computer scientist at IBM's San Jose Research Laboratory and was proposing a new way of organizing data that he called the “relational data model.”

One of the most important research areas in computer science in the early 1970s was the development of systems and languages for handling what computer scientists call *persistent data*. This term denotes data that remains in a computer system indefinitely, until it is explicitly deleted. Systems for managing persistent data were spreading quickly in the business world. A database management language proposed by the Codasyl Data Base Task Group (DBTG)[1] was receiving a lot of attention. Ray and I spent some time studying this language, learning concepts such as “currency indicators” and “set occurrence selection.” With a little practice, we learned how to represent a database query in the form of a program that navigated through a network of pointers to find the desired information.

Designing a Relational Language

For Ray and me, our exposure to the relational data model at Codd's research symposium was a revelation. For the first time, we could see how a query that would require a complex program in the DBTG language could be reduced to a few simple lines using one of Codd's relational languages. It became a game for the two of us to invent queries and challenge each other to express them in various query languages.

One of the queries that came out of this game was as follows: “Find names of employees who earn more than their managers.” The query was based on a three-column employee table. Each row of the table represented an employee and contained a name, a salary, and the name of the employee's manager. (This is a simple example. In a real application, employees would be identified by some unique identifier such as an employee number.) Table 1 shows the structure of the table with four example rows.

Table 1. Employee.

Name	Salary	Manager
Smith	45,000	Harker
Jones	40,000	Smith
Baker	50,000	Smith
Nelson	55,000	Baker

The third row of the table indicates that Baker's salary is $50,000 and Baker's manager is Smith. The first row indicates that Smith's salary is $45,000, so Baker earns more than his manager. Similarly, Nelson's salary is $55,000, but Nelson's manager is Baker, who earns $50,000, so Nelson also earns more than his manager. The result of the query, based on these four sample rows, is Baker and Nelson.

In his research papers, Codd introduced two relational query languages, called Relational Algebra[2] and Relational Calculus (also known as the Data Sublanguage Alpha[3]). Relational Algebra consists of several operators, usually represented by symbols such as those in Figure 1. Using these operators, the query about well-paid employees could be represented as in Figure 2a.

Codd's Relational Calculus was based on a notation used in formal logic, using an existential quantifier $\exists$ (meaning “for each”) and a universal quantifier $\forall$ (meaning “for all”). Similar to Relational Algebra, Relational Calculus could represent the well-paid employee query compactly (see Figure 2b).

Ray and I were impressed by how compactly Codd's languages could represent complex queries. However, at the same time, we believed that it should be possible to design a relational language that would be more accessible to users without formal training in mathematics or computer programming. We believed that barriers to widespread acceptance of Codd's languages existed on two levels. The first barrier came from the mathematical notation, which was hard to enter at a keyboard. This barrier was superficial and could be easily dealt with by replacing symbols with keywords—for example, replacing π with “project” and $\forall$ with “for all.” The more difficult barrier was at the semantic level. The basic concepts of Codd's languages were adapted from set theory and symbolic logic. This was natural given Codd's background as a mathematician, but Ray and I hoped to design a relational language based on concepts that would be familiar to a wider population of users. We also hoped to extend the language to encompass database updates and administrative tasks such as the creation of new tables and views, which had traditionally been outside the scope of a query language.

After attending Codd's symposium, Ray and I spent the next year experimenting with language designs.

Published by the IEEE Computer Society 1058-6180/12/$31.00 © 2012 IEEE

Our first attempt, called Square,[4] was based on the notion of mapping and used a subscript notation that was difficult to type. When we moved to the San Jose Research Laboratory in 1973 to join the System R project, we began work on another new language that we called Sequel. Sequel allowed the well-paid-employee query to be represented in a readable form free from mathematical concepts and symbols, as in Figure 2c.

Ray and I hoped that, with a little practice, users could learn to read queries like this almost as though they were English prose. This example query might be read as follows: ''Find an employee (let's call him 'e') and another employee (let's call him 'm') where e's manager matches m's name (in other words, e's manager is m) and e's salary is greater than m's salary (in other words, e earns more than his manager); then print e's name (for every such employee).''

It is important to note that the Sequel version of this query describes the information it is looking for but does not provide a detailed plan for how to find this information. This is why Sequel is called a declarative (rather than a procedural) language. Translating the declarative query statement into a detailed plan for processing the query is the job of an optimizing compiler.

From the beginning, Sequel was intended to be used both for data manipulation (querying and updating data) and for data definition (creation of tables, views, and assertions). To emphasize this duality, Ray and I wrote two papers: ''SEQUEL: A Structured English Query Language,''[5] and ''Using a Structured English Query Language as a Data Definition Facility.''[6] As luck would have it, the first of these papers became quite well known, while the second was never published outside IBM.

$\bowtie$	Join
π	Project
σ	Select
ρ	Rename

Figure 1. Examples of Relational Algebra operators.

In 1974, about one month after presenting a paper on Sequel at a technical conference in Ann Arbor, Michigan, Ray Boyce died suddenly of a ruptured brain aneurysm at the age of 26, leaving behind a wife of five years and a 10-month-old daughter. I often remember how much I enjoyed working with Ray. We had a seamless partnership. In his brief career, Ray collaborated with Ted Codd on the Boyce-Codd Normal Form[7] and with me on Sequel. I think that Ray would have been pleased to see the impact that his ideas have had on the world.

After Ray's untimely death, the Sequel language continued to evolve as a part of the System R project at San Jose Research Laboratory. System R was installed on an experimental basis in three IBM customer sites, and a more complete Sequel language design was published in 1976,[8] based in part on the experience collected by early users. In 1977,

$$\pi_{e.name} (\sigma_{e.salary > m.salary} (\rho_{e}(employee) \bowtie_{e.manager = m.name} \rho_{m}(employee)))$$

(a) Relational Algebra version

RANGE employee e;
RANGE employee m;
GET w (e.name): $\exists m((e.manager = m.name) \wedge (e.salary > m.salary))$

(b) Relational Calculus version

```
select e.name
from employee e, employee m
where e.manager = m.name and e.salary > m.salary
```

(c) Sequel (SQL) version

Figure 2. Three versions of the query, ''Find names of employees who earn more than their managers.''

because of a trademark issue, the name Sequel was shortened to SQL.

The SQL standard has been helpful in providing a mechanism for the controlled evolution of the language.

The SQL Standard

Commercial implementations of SQL, such as Oracle and DB2, began to appear in the late 1970s and early 1980s. By 1986, a standard language definition called "Database Language SQL" had been formally adopted by the ANSI and ISO standards groups.[9] A conformance test suite for the SQL standard was developed by the National Institute of Standards and Technology, and many SQL-based products were validated by this test suite between 1988 and 1996. New versions of the SQL standard were published in 1996, 1999, 2003, 2006, and 2008.

The SQL standard has been helpful in providing a mechanism for the controlled evolution of the language, providing a forum in which both users and implementers have a voice. Over the years, the evolving standard has corrected many of the initial deficiencies of SQL and has added many new features, including outer joins, table expressions, recursion, triggered actions, user-defined types and functions, and online analytic processing (OLAP) functions. In the hands of the ANSI X3H2 Committee, the definition of SQL has evolved from a 12-page research paper to an international standard comprising hundreds of pages.

SQL has been a more successful query language than Ray Boyce and I had any reason to expect in 1974. I think that the most important ingredient in this success was Ted Codd's breakthrough work in defining the relational data model and raising the level of abstraction with which users could interact with stored data. While SQL has been criticized as departing from some of Codd's original principles, experience has shown the language to be simple enough to learn easily and expressive enough to do useful work. Other important contributions to the success of SQL include the following:

- The language benefited greatly from having robust early implementations on multiple platforms. System R provided a multiuser implementation with transactional semantics and a sophisticated optimizing compiler. At roughly the same time, Oracle implemented the language on the widely used Unix platform.
- By combining query, update, and administrative tasks into a single language, SQL makes it easy for authorized users to modify database schemas and install new applications while the system is running. These tasks had traditionally been performed by specialized database administrators during system maintenance intervals. Making every user his own database administrator removed an important bottleneck from application development.
- The SQL standard provided a common ecosystem in which vendors could develop competing implementations, tools, and training materials. The standard also reassured users that they would not become dependent on a single software vendor, although vendors had a regrettable tendency to implement different subsets of the standard and to include proprietary features.

Criticisms of SQL

Like most successful languages, SQL has attracted its share of criticism, which has tended to focus on the following issues.

Orthogonality and Completeness. The earliest versions of SQL lacked support for some aspects of the relational data model, including primary keys and referential integrity. The early language also lacked orthogonality because it did not allow subqueries to be used in place of named tables and it failed to provide a way to name the columns of a query result. All these serious deficiencies were corrected by the 1992 version of the SQL standard, illustrating the helpful influence of the standards process.

Nulls. SQL supports a "null value" that represents a data item that is missing or inapplicable. The null value is not comparable to any other value, and for this reason, SQL implements a three-valued logic in which a search condition might be neither true nor false. Nulls and three-valued logic were both introduced by Ted Codd in his early papers, and Rule 3 of Codd's famous ''Twelve Rules''[10] requires a relational database system to support nulls. Various writers have complained that nulls and three-valued logic make queries more confusing and optimization more difficult. Researchers have proposed other approaches to the problem of missing data, but none of these approaches is without disadvantages. SQL lets users specify, on a column-by-column basis, where nulls are permitted and where they are prohibited. Over the years, nulls have proven useful in the design of various features, such as outer join, that have been added during the evolution of the language.

Duplicates. Unlike Codd's original definition of the relational data model, SQL permits duplicate rows to exist, both in a database table and in a query result. SQL also allows users to selectively prohibit duplicate rows in a table or in a query result. The intent of this approach is to give users control over the potentially expensive process of duplicate elimination. In some applications, duplicate rows might be meaningful—for example, in a point-of-sale system, a customer might purchase several identical items in the same transaction. In other applications, such as printing an address list, duplicate values could be unexpected but users might prefer not to pay the cost of detecting and eliminating them. As in the case of nulls, the SQL approach provides users with tools to control duplicate rows according to the needs of specific applications. The cost of sorting a million records to detect duplicates seemed more significant in 1974 than it does today, when an ordinary laptop has thousands of times more memory and processing power than a mainframe computer of the 1970s.

Impedence Mismatch. SQL was designed to be used both as a stand-alone language for interactive queries and as an application development language for online transaction processing (OLTP). This is a helpful unification of concepts, but in OLTP applications, SQL is usually embedded in (or called from) a host programming language such as C or Java. Often the data types of the host language are not the same as those of SQL, and the host language is usually more procedural, whereas SQL is more declarative. The resulting ''impedance mismatch'' tends to increase application complexity and interfere with global optimization. One approach to this problem has been the development of computationally complete SQL-based scripting languages such as Persistent Stored Modules (PSM).[11]

SQL was designed to be used both as a stand-alone language for interactive queries and as an application development language for OLTP.

Legacy

When Ray and I were designing Sequel in 1974, we thought that the predominant use of the language would be for ad-hoc queries by planners and other professionals whose domain of expertise was not primarily database management. We wanted the language to be simple enough that ordinary people could ''walk up and use it'' with a minimum of training. Over the years, I have been surprised to see that SQL is more frequently used by trained database specialists to implement repetitive transactions such as bank deposits, credit card purchases, and online auctions. I am pleased to see the language used in a variety of environments, even though it has not proved to be as accessible to untrained users as Ray and I originally hoped.

Looking back at my experience at IBM Research in the 1970s, I feel fortunate to have been working at that place and time. Ted Codd, Ray Boyce, and the System R team were wonderful people to work with, and the impact of our work has been gratifying. I am grateful for having had the opportunity to participate in this work.

References

1. ''CODASYL Data Base Task Group,'' April 71 Report, ACM, 1971.
2. E.F. Codd introduced the operators of relational algebra in ''Relational Completeness of Data

Base Sublanguages,'' IBM Research Report RJ 987, Mar. 1972. Several versions of the relational algebra exist, all of which include some version of the operators used in this paper. There is no recognized standard notation for these operators. The notation used in this article is taken from H. Garcia-Molina, J. Ullman, and J. Widom, *Database Systems: the Complete Book,* Prentice-Hall, 2002, pp. 189–237.

3. E.F. Codd, ''A Data Base Sublanguage Founded on the Relational Calculus,'' *Proc. ACM SIGFIDET Workshop on Data Description, Access, and Control,* ACM Press, 1971, pp. 35–68.
4. R. Boyce et al., ''Specifying Queries as Relational Expressions: the SQUARE Sublanguage,'' *Comm. ACM,* vol. 18, no. 11, 1975, pp. 621–628.
5. D. Chamberlin and R. Boyce, ''SEQUEL: A Structured English Query Language,'' *Proc. ACM SIGFIDET Workshop on Data Description, Access, and Control,* ACM Press, 1974, pp. 249–264. See also http://www.almaden.ibm.com/cs/people/chamberlin/sequel-1974.pdf.
6. R. Boyce and D. Chamberlin, ''Using a Structured English Query Language as a Data Definition Facility,'' IBM Research Report RJ1318, Dec. 1973.
7. The Boyce-Codd Normal Form is a database design discipline taught in most advanced textbooks on database management. For example, see R. Ramakrishnan and J. Gehrke, *Database Management Systems,* 3rd ed., McGraw Hill, 2003, pp. 615–617.
8. D. Chamberlin et al., ''SEQUEL 2: A Unified Approach to Data Definition, Data Manipulation, and Control.'' *IBM J. Research and Development,* vol. 20, Nov. 1976, p. 560.
9. See the ANSI/ISO/IEC 9075-1, Information technology - Database languages - SQL - Part 1: Framework (SQL/Framework); ANSI/ISO/IEC 9075-2, Information technology - Database languages - SQL - Part 2: Foundation (SQL/Foundation); and so on at http://www.ansi.org or http://www.iso.ch.
10. E.F. Codd., ''Does Your DBMS Run by the Rules?'' *Computer World,* vol. 21, Oct. 1985. See also http://en.wikipedia.org/wiki/Codd's_12_rules.
11. ANSI/ISO/IEC 9075-4, Database Language SQL, Part 4: Persistent Stored Modules (SQL/PSM); http://www.ansi.org.

Donald D. Chamberlin *is an adjunct professor at the University of California, Santa Cruz. His work on SQL and System R at IBM has been recognized by the ACM SIGMOD Innovation Award and by the Computer History Museum. Contact him at chamberlin.don@gmail.com.*