Technology | DOI:10.1145/1735223.1735231 Gary Anthes

Happy Birthday, RDBMS!

The relational model of data management, which dates to 1970, still dominates today and influences new paradigms as the field evolves.

FORTY YEARS AGO this June, an article appeared in these pages that would shape the long-term direction of information technology like few other ideas in computer science. The opening sentence of the article, "A Relational Model of Data for Large Shared Data Banks," summed it up in a way as simple and elegant as the model itself: "Future users of large data banks must be protected from having to know how the data is organized in the machine," wrote Edgar F. Codd, a researcher at IBM.

And protect them it did. Programmers and users at the time dealt mostly with crude homegrown database systems or commercial products like IBM's Information Management System (IMS), which was based on a low-level, hierarchical data model. "These databases were very rigid, and they were hard to understand," recalls Ronald Fagin, a Codd protégé and now a computer scientist at IBM Almaden Research Center. The hierarchical "trees" in IMS were brittle. Adding a single data element, a common occurrence, or even tuning changes, could involve major reprogramming. In addition, the programming language used with IMS was a low-level language akin to an assembler.

But Codd's relational model stored data by rows and columns in simple tables, which were accessed via a high-level data manipulation language (DML). The model raised the level of abstraction so that users specified *what* they wanted, but not *how* to get it. And when their needs changed, reprogramming was usually unnecessary. It was similar to the transition 10 years earlier from assembler languages to Fortran and COBOL, which also raised the level of abstraction so that programmers no longer had to know and keep track of details like memory addresses.

"People were stunned to learn that complex, page-long [IMS] queries could be done in a few lines of a relational language," says Raghu Ramakrishnan, chief scientist for audience and cloud computing at Yahoo!

Codd's model came to dominate a multibillion-dollar database market, but it was hardly an overnight success. The model was just too simple to work, some said. And even if it did work, it would never run as efficiently as a finely tuned IMS program, others said. And although Codd's relational concepts were simple and elegant, his mathematically rigorous languages, relational calculus and relational algebra, could be intimidating.

In 1969, an ad hoc consortium called CODASYL proposed a hierarchical database model built on the concepts behind IMS. CODASYL claimed that its approach was more flexible than IMS, but it still required programmers to keep track of far more details than the relational model did. It became the basis for a number of commercial products, including the Integrated Database Management System (IDMS) from the company that would become Cullinet.

Contentious debates raged over the models in the CS community through much of the 1970s, with relational enthusiasts arrayed against CODASYL advocates while IMS users coasted along on waves of legacy software.

As brilliant and elegant as the relational model was, it might have remained confined to computer science curricula if it wasn't for three projects aimed at real-world implementation of the relational database management system (RDBMS). In the mid-1970s, IBM's System R project and the University of California at Berkeley's Ingres project set out to translate the relational concepts into workable, maintainable, and efficient computer code. Support for multiple users, locking, logging, error-recovery, and more were developed.

System R went after the lucrative mainframe market with what would become DB2. In particular, System R produced the Structured Query Language (SQL), which became the de facto standard language for relational databases. Meanwhile Ingres was aimed at UNIX machines and Digital Equipment Corp. (DEC) minicomputers.

Then, in 1979, another watershed paper appeared. "Access Path Selection in a Relational Database Management System," by IBM System R researcher Patricia Selinger and coauthors, described an algorithm by which a relational system, presented with a user query, could pick the best path to a solution from multiple alternatives. It did that by considering the total cost of the various approaches in terms of CPU time, required disk space, and more.

"Selinger's paper was really the piece of work that made relational database systems possible," says David DeWitt, director of Microsoft's Jim Gray Systems

PHOTOGRAPH BY CHAD NACKERS

Laboratory at the University of Wisconsin-Madison. "It was a complete home run." The paper led to her election to the National Academy of Engineering in 1999, won her a slew of awards (including the SIGMOD Edgar F. Codd Innovations Award in 2002), and remains the seminal work on query optimization in relational systems.

Propelled by Selinger's new ideas, System R, Ingres, and their commercial progeny proved that relational systems could provide excellent performance. IBM's DB2 edged out IMS and IDMS on mainframes, while Ingres and its derivatives had the rapidly growing DEC Vax market to themselves. Soon, the database wars were largely over.

Faster Queries

During the 1980s, DeWitt found another way to speed up queries against relational databases. His Gamma Database Machine Project showed it was possible to achieve nearly linear speed-ups by using the multiple CPUs and disks in a cluster of commodity minicomputers. His pioneering ideas about data partitioning and parallel query execution found their way into nearly all commercial parallel database systems.

"If the database community had not switched from CODASYL to relational, the whole parallel database industry would not have been possible," DeWitt says. The declarative, not imperative, programming model of SQL greatly facilitated his work, he says.

The simplicity of the relational model held obvious advantages for users, but it had a more subtle benefit as well, IBM's Fagin says. "For theorists like me, it was much easier to develop theory for it. And we could find ways to make the model perform better, and ways to build functions into the model. The relational model made collaboration between theorists and practitioners much easier."

Indeed, theorists and practitioners worked together to a remarkable degree, and operational techniques and applications flowed from their work. Their collaboration resulted in, for example, the concept of locking, by which simultaneous users were prevented from updating a record simultaneously.

The hegemony of the relational model has not gone without challenge. For example, a rival appeared in the late 1980s in the form of object-oriented databases (OODBs), but they never caught on. There weren't that many applications for which an OODB was the best choice, and it turned out to be easier to add the key features of OODBs to the relational model than to start from scratch with a new paradigm.

More recently, some have suggested that the MapReduce software framework, patented by Google this year, will supplant the relational model for very large distributed data sets. [See "More Debate, Please!" by Moshe Y. Vardi on p. 5 of the January 2010 issue of *Communications*.] Clearly, each approach has its advantages, and the jury is still out.

As RDBMSs continues to evolve, scientists are exploring new roads of inquiry. Fagin's key research right now is the integration of heterogeneous data. "A special case that is still really hard is schema mapping—converting data from one format to another," he says. "It sounds straightforward, but it's very subtle." DeWitt is interested in how researchers will approach the "unsolved problem" of querying geographically distributed databases, especially when the databases are created by different organizations and are almost but not quite alike. And Ramakrishnan of Yahoo! is investigating how to maintain databases in the cloud, where service vendors could host the databases of many clients. "So 'scale' now becomes not just data volume, it's the number of clients, the variety of applications, the number of locations and so on," he says. "Manageability is one of the key challenges in this space."

Further Reading

Codd, E.F.
A relational model of data for large shared data banks. *Comm. of the ACM 13*, 6, June 1970.

Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A., Price, T.G.
Access path selection in a relational database management system. *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, 1979.

Ullman, J.D.
Principles of Database and Knowledge-Base Systems: Volume II: The New Technologies, W.H. Freeman & Co., New York, NY, 1990.

Hellerstein, J. M. and Stonebraker, M.
***Readings in Database Systems (4th ed.)*, MIT Press, Cambridge, MA, 2005.**

Agrawal, R., et al
***The Claremont Report on Database Research*, University of California at Berkeley, Sept. 2008. http://db.cs.berkeley.edu/claremont/claremontreport08.pdf**

Gary Anthes is a technology writer and editor based in Arlington, VA.

Looking Ahead With Michael Stonebraker

An adjunct professor at Massachusetts Institute of Technology, Michael Stonebraker is renowned as a database architect and a pioneer in several database technologies, such as Ingres, PostgreSQL, and Mariposa (which he has commercial interests in). As for the database industry's future direction, Stonebraker says one-third of the market will consist of relational database management systems used in large data warehouses, such as corporate repositories of sales information. But the mainstream products in use today, which store table data row by row, will face competition from new, better-performing software that stores it column by column. "You can go wildly faster by rotating your thinking 90 degrees," he says.

Another third of the market he believes will be in online transaction processing, where databases tend to be smaller and transactions simpler. That means databases can be kept in memory and locking can be avoided by processing transactions one at a time. These "lightweight, main memory" systems, Stonebraker says, can run 50 times faster than most online transaction processing systems in use today.

In the final third of the market, there are "a bunch of ideas," depending on the type of application, he says. One is streaming, where large streams of data flow through queries without going to disk. Another type of nonrelational technology will store semistructured data, such as XML and RDF. And a third approach, based on arrays rather than tables, will be best for doing data clustering and complex analyses of very large data sets.

Finally, "if you don't care about performance," says Stonebraker, "there are a bunch of mature, open-source, one-size-fits-all DBMSs."