Reminiscences on Influential Papers

Richard Snodgrass, editor

Nora Ephron's movie "You've Got Mail" is a paean to the world wide web, perhaps computer science's most visible contribution, in particular to the creation via chat rooms of small communities within a dissociated world. In a pivotal scene in the third act, a bristling Kathleen Kelly, the protagonist played by Meg Ryan, insists, "Whatever else anything is, it ought to *begin* by being personal."

This column celebrates the process of scientific inquiry by examining, in an anecdotal and yes, highly personal fashion, how ideas spread and evolve. Past columns have including compelling stories of the impact of particular papers on the mind set and discoveries of a few well-known and respected people in the database community.

For this issue, I've gone back to the source, and have asked the authors of some of those influential papers to reflect upon *their* influences. So the subtitle might be, "Reminiscences on Papers that Influenced Influential Papers." The papers mentioned here span the entire history of databases, from work in the late 50's to a reappearance of the AlphaSort paper.

While I have a list of some 40-odd people of whom I wanted to ask to identify their favorite paper, I thought it was appropriate to let someone else have a crack at picking these people. The next and future columns will be edited by Ken Ross, who has appeared before in this column. I look forward to the contributions that he elicits, and I thank all those who contributed to this column during my tenure as editor.

Rakesh Agrawal, IBM Almaden Research Center, ragrawal@almaden.ibm.com

[A. V. Aho and J. D. Ullman, "The Universality of Data Retrieval Languages," in *Proceedings of the Symposium on Principles of Programming Languages*, pp. 110–117, 1979]

A key question concerning the design of query languages is what power they should have. In this beautiful paper, Aho and Ullman enunciated two principles that they postulated a query language must obey. They then showed that although Codd's relational algebra and calculus satisfy these principles, there are queries involving least fixed points that also satisfy these principles but cannot be expressed in these languages. They further considered extensions to relational algebra to enable such queries and discussed techniques for optimizing expressions in this extended algebra. Finally, they defined a programming language to serve as a model of computation for relational database retrieval operations. By defining four different interpretations of the **for** statement in their language:

for t **in** R **do** <statement>

they showed that we get different classes of computable functions. Under one interpretation, the language is equivalent to relational calculus or algebra; under another more general interpretation it is at least as powerful as relational algebra with the fixed point operator. There is tremendous wealth of ideas packed in this 8-page paper. My dream is to be able to write one such paper.

Philip Bernstein, Microsoft Research, philbe@microsoft.com

[R. H. Thomas, "A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases," *ACM TODS* 4(2), pp. 1800-209, June, 1979]

In 1976, people were just starting to think about how distributed transactions and replication could be added to a database system. I believe the first good paper on the topic—and still one of the best—was this one by Bob Thomas. It was first published in 1976 as a Bolt Beranek and Newman technical report, and was widely distributed in that form. The paper introduced two very important ideas that are much used today. First, it defined majority consensus as a way to determine which processes are available in the face of communications failures. This was later extended by Dave Gifford to be a weighted majority, or quorum. Most practical process failure detection algorithms today use this technique. Second, it introduced the use of timestamps to tag transactions, updates, and data items. The timestamp-based rule that bears his name allows updates to propagate in any order, yet all replicas of a data item will eventually converge to the same value. Today, this technique is used in most of the worlds directory services, and in some multi-master replication algorithms for database systems.

The paper was very influential on early work on distributed transactions and replication, particularly on mine. The timestamping approach motivated our work on the SDD-1 distributed DBMS at Computer Corporation of America (gone, but not forgotten). It was also a challenge when developing serializability theory, as replication and majority consensus were aspects that a complete theory had to account for. It motivated some of Leslie Lamport's early work on reliable distributed computing. Moreover, the many multi-master replication algorithms that use timestamp vectors are direct descendants of this algorithm. Judged by density of new ideas and influence on both research and products, this is one of my favorite papers in the transaction literature.

Don Chamberlin, IBM Almaden Research Center, chamberlin@almaden.ibm.com

[E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, 13(6):377–387, June, 1970]

The paper that changed the course of my career was Ted Codd's famous paper in which he introduced the relational data model, defined the concept of data independence, and described how relational algebra could be used as a query language. In the same paper, Codd laid the foundation for the theory of database normalization, and proposed the first-order predicate calculus as a yardstick of linguistic power. I don't know of any other paper that introduced so many concepts of such lasting significance. When I read Codd's paper, I was struggling to master the navigational database interfaces that were in use at the time. It was eye-opening to see how the relational approach made it possible to reduce a complex program to a simple and elegant logical expression. It was clear to me that the relational model would have the same impact on database management that high-level languages had had on programming in general. The challenges posed by Codd's paper seemed to be to find a syntax that was easily understood by non-mathematicians, and to find techniques for implementing that syntax without sacrificing the performance of lower-level interfaces. To a large extent, these challenges have kept me busy since I read Codd's paper.

Jim Gray, Microsoft Research, gray@microsoft.com

[D. Bitton, D. J. DeWitt, and C. Turbyfill, "Benchmarking Database Systems A Systematic Approach," in *Proceedings of the International Conference on Very Large Data Bases*, M. Schkolnick and C. Thanos, eds, Florence, Italy, Morgan Kaufmann, October, 1983]

This paper had a huge impact on us all. It was the seed that started database performance-evaluation research. The paper, laid out a benchmark methodology, and proposed an open-source benchmark still in use today. At the time, it set the performance agenda for the decision support community. It also caused database vendors to add a *DeWitt clause* to their products so that such a paper could never be written again (the clause prevents users from publishing performance results without permission.)

I loved and hated this paper. I loved the fact that it was good science. It did well-documented and reproducible experiments, with interesting and surprising results. The benchmark became a standard part of my performance toolkit. But, I hated the fact that it implicitly defined databases as being used for scans and joins. That model seemed far from what databases were *really* used for (in my world.) It crystallized the dichotomy between DSS and OLTP. I had been building transaction-processing systems, and I felt left out. So, I organized an informal group of researchers and practitioners (including the authors of the paper) to define an on-line-transaction-processing benchmark (DebitCredit, Copy, Sort). That work appeared in a Datamation article ("A Measure of Transaction Processing Performance," Anon. et al., *Datamation*, 31(7), pp. 112–118, April 1985) and eventually evolved to TPC-A.

Don Haderle, IBM Santa Teresa Lab, haderle@us.ibm.com

[T. Haerder and A. Reuter, "Principles of Transaction-Oriented Database Recovery," in *ACM Computing Surveys*, 15(4), pp. 287–318, December, 1983]

I had built commercial real-time operating systems and data management systems for many years in IBM. In 1976 I read the March 1976 *ACM Computing Surveys* issue on DataBase Management Systems. I got excited. While I had been laboring in the software basement, others had figured out how to deal with data in an easy and disciplined way. To my surprise a lot of the work was being done right around the corner from me by the System/R research crowd. Within a year I joined a nascient development team intent on producing the next generation database manager, which was to be DB2.

For several years I was immersed in engineering the components of DB2. The Haerder/Reuter paper brought the abstraction of database management into focus, laying out its properties cleanly—atomicity, consistency, isolation, and durability. It gave me a framework for contemplating database management from a consumer's view rather than a producer's view. Given the clear definitions of the properties it was easy to visualize improvements of those properties to benefit the solving of customer's problems.

Won Kim, Cyber Database Solutions, Inc., won.kim@cyberdb.com

[T. Atwood, "An Object-Oriented DBMS for Design Support Applications," in *Proceedings of the IEEE CompInt Conference*, Montreal, Canada, September, 1985]

There have been about a dozen papers that significantly helped to shape my own ideas about database research during the past twenty years. However, given my ten-year campaign in the

areas of object-oriented databases and object-relational databases, I would mention what many may regard as a rather obscure paper as one that had the most influence on me, this one by Tom Atwood. (Tom's paper described planned features of the Vbase C++-based object-oriented database system, which later became known as Ontos; the paper was even picked as the Best Paper of the conference.)

Around the summer of 1985, I was working at MCC and was asked to re-direct my CAD Database project to an object-oriented database, since the CAD Program, Human Interface Program, and AI Program at MCC were all developing their technologies using CommonLISP with Flavors on the Symbolics LISP machines, and asked the Database Program, to which I belonged, to develop an object-oriented database, if the Database Program expected its technology to be embedded in other MCC technologies.

Since I had to start an object-oriented database project, I had to first learn what an object-oriented database was supposed to be, in turn what an 'object' was. It was confusing and frustrating for a few months when people at MCC could only tell me "everything under the sun is an object," "an object can be a before method, after method, around method," "an object-oriented database makes objects persistent," "an object-oriented database makes SQL totally useless," "locking is very dangerous," etc. Based on all these gibberishes, it appeared to me that an object-oriented database was something terribly mysterious.

When I listened to Tom Atwood's presentation, and, afterwards, I sat next to him at the conference banquet, and listened to him talking about a type definition language, instances of types, and nested transactions, the proverbial light bulb finally came on for me, and I suddenly realized what I would have to do. The system Tom Atwood described to me that day was indeed a database system, but the first few releases of Vbase did not have various traditional database features and instead focused on C++ programming persistence, leading to confusion in the market. In any case, over the course of one year or so after that day, I, along with my ORION object-oriented database team at MCC, systematically re-examined every architectural element of a database system and tried to address changes that the object-oriented concepts of inheritance, types, references, etc. force on the architecture of a database system.

Bruce Lindsay, IBM Almaden Research Center, `bruce@almaden.ibm.com`

[C. Nyberg, T. Barclay, Z. Cvtanovic, J. Gray, and D. Lomet, "AlphaSort: A RISC Machine Sort," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, R. Snodgrass and M. Winslett, eds, pp. 233–242, June, 1994]

The AlphaSort paper dramatically illustrates the importance of cache conscious coding. Prior to reading this simple paper, I had been able to blithely ignore the unpleasant fact that processor memory systems were inexorably evolving towards hierarchies which must be taken into account in system design and implementation. The implications of the AlphaSort paper forced me to realize that database performance tuning would require new tools (such as feedback directed code restructuring) and a new focus on query processing strategies. For example, cache consciousness calls into question the wisdom of pipelined query plans!

C. Mohan, IBM Almaden Research Center, `mohan@almaden.ibm.com`

[R. A. Crus, "Data Recovery in IBM Database 2," *IBM Systems Journal*, 23(2): 178–188, 1984]

In the mid-80s, within the context of the Starburst project at the IBM Almaden Research Center,

I decided to revisit some of the then remaining open problems from the days of System R in the areas of concurrency control and recovery (CC&R). It was while trying to understand how existing DBMS products handled CC&R that I came across the above paper by Dick Crus. This paper gave me a very good introduction to how various types of recovery were implemented in the very first release of DB2/MVS. Without necessarily using those terms, the paper discussed concepts like write-ahead logging, compensation log records and recovery based on log sequence numbers (LSNs). These descriptions then allowed me to try to deduce how those underlying concepts of DB2/MVS's recovery strategy differed from those of System R. It also made me aware of challenging problems (e.g., partial write of a page to disk) that had to be handled in real systems and that were not addressed until then in any research publications. Since the paper did not explain the rationale behind every design decision, I took it as a challenge to try to figure out the rationale and then extend the recovery algorithm to support finer than DB2/MVS's page granularity locking. This was the start of a series of explorations that ultimately led to the invention of the ARIES family of CC&R algorithms and the authoring of numerous research papers by myself and a number of collaborators at IBM. While few people in the research community have read Dick's paper, personally, it has been a very influential paper for me. At a time when it was not common for DBMS product developers to write detailed technical papers, Dick had authored a paper with an unusual amount of detail about the internals of the mainframe RDBMS product DB2/MVS.

Ron Morrison, University of St. Andrews, ron@dcs.st-and.ac.uk

[L. Cardelli and P. Wegner, "On Understanding Types, Data Abstraction and Polymorphism," *ACM Computing Surveys* 17(4), pp. 471–523, December, 1985]

This paper appeared in the grey literature as a technical report about a year before it was published. At the time I was trying to figure out the relationship between complex type systems and database schemas. In our persistent programming systems we already stored code in the database, in the form of functions, along with complex data. We had found that we required one dynamic type to implement persistence but wished in general to retain the safety of static typing without having to endure its restrictiveness.

The paper is a survey yet introduces a new way of thinking about types. We knew, of course, about polymorphism (universal quantification) from the work of Strachey but through this paper learned of its relationship with parametric types and abstract data types (existential quantification). This gave use the confidence to find new and efficient ways of implementing polymorphism in the persistent language Napier88, and of using existential quantification for implementing views in databases systems. I know now that all of the concepts in this paper are expressed in others. However it is in this one that the idea are brought together and expressed in a manner that any computer scientist can understand. I still recommend it to all my graduate students.

Raymond Reiter, Department of Computer Science, University of Toronto, reiter@ai.toronto.edu

[J. McCarthy, "Programs with Common Sense," in *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, December, 1958. Available at http://www-formal.stanford.edu/jmc/mcc59.html]

I should begin with a mild disclaimer: Although I have published a few things in the "mainstream" database literature, I don't consider this work to be peculiar to databases. Neither do I see myself as a database researcher. Rather, my interests have always been with problems of how best to represent knowledge about a world, and how to make use of that representation. From this per-

spective, the representational aspects of database theory, artificial intelligence, high level robotics, natural language, and even certain programming languages become one and the same enterprise. Within computer science, the historical separation of these communities has been unfortunate, and harmful.

All of which brings me to McCarthy's 1958 paper, which was perhaps the very first to propose mathematical logic as a knowledge representation language for artificial intelligence. In it, McCarthy states clearly the advantages of declarative sentences for representing knowledge, and proposes his "advice taker," a system that is "instructed" with sentences of the predicate calculus, and that uses deduction to determine how it should subsequently act in its world. Virtually all of my own research, and that of many subcommunities in artificial intelligence and databases, have been an elaboration of this very early insight of John McCarthy.

Carlo Zaniolo, Computer Science Department, UCLA, zaniolo@cs.ucla.edu

[J. M. Smith and D. C. P. Smith, "Database Abstractions: Aggregation and Generalization," *TODS* 2(2), pp. 105-133, June, 1977]

As the relational approach to databases began gaining wide acceptance, serious concerns emerged on whether its 'spartan simplicity' could capture the richer semantics of database relationships. Then, in 1977 John and Diane Smith published a seminal paper in which database relationships are modeled on the orthogonal dimensions of aggregation and generalization. It was the simplicity and elegance of their model that convinced me that such rich semantics could be captured via minimal extensions of the relational data model and its query languages. Indeed, GEM achieved said goal by combining simple notions, already well-known to to relational database researchers, such as tuple IDs, null values, and path expressions (also influenced by functional query languages).

Many of these ideas have recently been revisited and applied in the design of object-relational databases and their SQL extensions: this represents and enduring tribute to the lasting influence of the 1975 seminal paper by John and Diane Smith.

C. Zaniolo, "The Database Language GEM," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, D. J. DeWitt and G. Gardarin, eds, pp. 207–218, 1983.

S. Tsur and C. Zaniolo, "An Implementation of GEM-Supporting a Semantic Data Model on a Relational Back-End,", in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, B. Yormark, edd. pp. 286–295, 1984.

Erratum

The information on the paper discussed by Patrick Valduriez in the March, 1999 issue was incorrect. Here is the correct reference.

M. M. Zloof, "Query-By-Example: Operations on the Transitive Closure," IBM Research Report RC 5526, Yorktown Heights, New York, October, 1976.