Reminiscences on Influential Papers

Richard Snodgrass, editor

This column, inaugurated in the March. 1998 issue, celebrates the process of scientific inquiry by examining, in an anecdotal fashion, how ideas spread and evolve. I've asked a few well-known and respected people in the database community to identify a single paper that had a major influence on their research, and to describe what they liked about that paper and the impact it had on them. The contributions in this issue evoke the heady 70's, as the now-accepted foundation of databases was just starting to be established, and serve to emphasize the debt our field owes to other areas of computer science that have provided useful insights.

Hector Garcia-Molina. Stanford University, `hector@db.stanford.edu`

[K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System." *Communications of the ACM*, 19(11):624–633, November 1976]

This paper highly influenced my early career. I was a graduate student at Stanford when I got a copy of the paper as an IBM Technical Report. I had just taken a graduate course on concurrent programming, and had learned that proving almost anything about concurrent programs was really hard. Yet, here was this way of looking at concurrency that was really neat and simple. I was also impressed that there was a formal model for an important database systems problem. From that point on, I took database systems more seriously, and did my thesis on concurrency control for distributed databases, basically starting from what I had read in that paper. Looking back, I see that this paper influenced many others too, who started working in the area, extending, formalizing further, and evaluating the performance of the basic ideas of that paper.

Tomasz Imielinski. Rutgers University, `imielins@cs.rutgers.edu`

[R. Reiter. "On Closed World Databases." in **Logic and Databases**, H. Gallaire and J. Minker (eds), Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, 1977. Advances in Data Base Theory. Plenum Press, New York, 1978, pp. 55–76]

I have chosen the paper which impressed me and influenced both mvself and a large number of researchers who either were in the past (like myself) or currently are working on logical foundations of databases. This paper and several other papers by Rav Reiter have started a new way of thinking about databases—with emphasis on precise logical formulation of the hidden assumptions which are made about database content when answering database queries. It made a simple but fundamental observation that there are two equally reasonable ways of interpreting the database content: closed world assumption (facts not derivable from the database are false) and open world assumption (we cannot really say anything about facts which cannot be logically derived from the databases). Reiter observed that SQL queries interpret the database according to the closed world assumption and provided the "missing" axioms. His way of thinking influenced my research in my own PhD thesis and the work which I did later on deductive databases. Although Reiter's paper have not led to any "products" as it is common today—it was a example of a *fundamental* paper which influenced how people think and it is still, after 20 years a very important reference for anybody who studies databases and their logical foundations.

Reiter's work has been instrumental in subsequent research on nonmonotonic logics and different forms of negation by failure and contributed very significantly not only to databases but also to logic programming and AI.

David Maier, Oregon Graduate Institute, maier@cse.ogi.edu

[M. P. Atkinson, P. J. Bailey, K. J. Chisholm, P. W. Cockshott and R. Morrison, "An Approach to Persistent Programming," *The Computer Journal*, 26(4):360–365, November 1983]

I first encountered the work of the Persistent Programming Group at Edinburgh and St. Andrews around 1984. At that time I was consulting with GemStone Systems (then Servio Logic) on the design of their database machine and system. A fundamental shift was going on at the company, from implementing a nested-set data model on custom hardware to producing an object-oriented database that would run on standard workstations. I was trying to get my head around object-oriented programming in general and Smalltalk-80 in particular, and figure out what the challenges and advantages were for an object-oriented data model. Peter Buneman had visited the group in Scotland, and he pointed me at their work after hearing what I was working on with GemStone.

The cited paper is a short introduction to PS-algol, a persistent version of S-algol. It covers some of the design decisions in converting a programming language into a database language (such as how to indicate persistence and how to provide efficient access to large collections) and gives a brief overview of its implementation. (A pair of companion papers in *Software—Practice & Experience* around the same time go into detail on the implementation.) The paper affected my thinking in several ways. First, it made me feel that trying to build a database system by adding persistence to an existing programming language wasn't such a nutso idea after all. Second, it showed me which aspects of the GemStone approach arose from it being a persistent programming language and which depended on object-orientation. For example, PS-algol had persistence orthogonal to type and persistence by reachability, so those aspects didn't require an object model. On the other hand, logical data independence via methods and extensibility via subtyping did depend intimately on having an object model. (However, the Scotland group showed later, by adding persistent procedures and a run-time compiler to PS-algol, that there are non-object-oriented means to achieve type extensibility.) Third, the paper helped me realize that there are common problems in turning any general-purpose programming language into a database system, such as the need for a common schema and associative query, and that there were implementation options I hadn't thought about, such as swizzling reference of objects in memory.

Pat Selinger, IBM Almaden Research Center, pgs@us.ibm.com

[B. Wegbreit, **Studies in Extensible Programming Languages**, Ph.D. Thesis, Harvard University, May 1970]

I joined the IBM database team in the early stages of building System R, our first foray into proving that a relational system could have a practical implementation while maintaining the data independence that the relational model advertises, using a set-oriented query language. So I suppose

I should cite the famous 1971 Codd paper about the relational data model, but that seemed to be too obvious. And actually for me there was a better choice, which you'll see in a moment. I joined the System R project because it had smart people who were fun to talk to. I had done my PhD thesis on a combination of operating systems and programming languages, and literally knew nothing about database technology before joining IBM, where on day 1, they handed me a copy of Chris Date's book and said "read this". I thought I had little chance of being able to contribute much. Well, it turned out I was wrong. Operating systems and programming language technologies actually have a huge relevance to database systems: the concept of compilation, the concept of examining alternatives for generating code and choosing the optimal one, concurrency, multiprocessing, ... and many others. While making the relational engine take shape, we applied what had been learned in these other areas and adapted many of those concepts for databases in that first generation RDBMS.

One programming language technology stream, however, just didn't have a natural exploitation back then. I had read a PhD thesis by Ben Wegbreit in the early 1970s (yes, I know that was a very long time ago). This was some of the first research done on extensible programming languages, function overloading, user-defined types and functions. That work had a profound influence on my thinking about what you *could* do with programming languages: they could be living, active things, not just static syntax in a manual that you use to get a task done. Having read that paper, and having helped apply so many other programming language concepts to database technology, I was very intrigued with the possibility of exploiting language extensibility in database somehow without destroying the simplicity of the relational model. In the mid-1980s, we had the opportunity. As the R* distributed project finished up, we looked at new project ideas, including the possibility of doing a second generation database system, built from the beginning to be extensible, an active, living database engine. That concept of extensibility caught our interest enough to pursue more deeply. The Starburst project was born, and what we at that time called extensible databases has now formed the foundation for the object-relational database systems that are products today, nearly 30 years after the technology was first applied to programming languages.

Jeffrey Ullman, Stanford University, ullman@db.stanford.edu

[P. A. Bernstein. "Synthesizing Third Normal Form Relations from Functional Dependencies," *ACM Transactions on Database Systems* 1(4):277–298, March, 1976]

In 1975, Catriel Beeri took a teaching position at Princeton, after having spent a fellowship year at Toronto working with Dennis Tsichritzis and his student Phil Bernstein on the theory of databases. Catriel taught a course in relational database systems, which I attended along with a number of my students. That course had tremendous leverage in the database field; e.g., I can think of at least five students (plus Catriel himself) who later chaired major database conferences. One of the principal topics of the course focused on the above cited paper, including Phil's schema-design technique and his observations about how earlier papers on functional dependencies, normal forms, and keys contained fundamental errors that he corrected by careful analysis and proofs. This work convinced me there was something deep in the theory of functional dependencies, and that it was worth devoting effort to understanding its subtleties and implications. Today, while the particular algorithm presented in the paper is not often used, the underlying concepts, presented with the precision that Phil and Catriel pioneered, are a staple of a CS undergraduate education, and so common that they are no longer viewed as "theory."