Date on Database

Writings 2000–2006

C. J. Date

Apress®

Date on Database: Writings 2000–2006

ISBN-13 (pbk): 978-1-59059-746-0

ISBN-10 (pbk): 1-59059-746-X

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Jonathan Gennick
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Keir Thomas, Matt Wade
Project Manager: Tracy Brown Collins
Copy Edit Manager: Nicole LeClerc
Assistant Production Director: Kari Brooks-Copony
Production Editor: Kelly Winquist
Compositor: Susan Glinert
Proofreader: Lori Bring
Indexer: C. J. Date
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

CHAPTER 2

An Interview with Chris Date

The following interview was conducted by Tony Williams and originally appeared on the O'Reilly website (`http://www.oreillynet.com/pub/a/network/2005/07/29/cjdate.html`) on July 29th, 2005. Chris Date adds by way of preamble:

> *Another interview I did recently was published in the Northern California Oracle User Group (NoCOUG) Journal, Vol. 19, No. 2 (May 2005), and some of the questions and answers in that interview are inevitably very similar to some of those in what follows. Also, answers to many of the same questions—along with further discussion, in some cases—can be found in my recent book* Database in Depth: Relational Theory for Practitioners *(O'Reilly Media Inc., 2005).*

How did you get started with relational theory?

As with so many things, this was basically just luck—a matter of being in the right place at the right time. I was working at IBM in England, where I had been hired as a programming instructor. I had been doing that job for a while, but IBM had a very enlightened policy according to which you weren't allowed to spend all your time just teaching—from time to time you had to rotate out and get down into the trenches, as it were. So I rotated out and joined a little research group, where I was given the job of figuring out what the PL/I language should do to support this new thing called database management (this was early 1970). So I played with IBM's database product IMS—that was IBM's major product offering at the time—and I studied the CODASYL DBTG database specifications; IMS was hierarchies and CODASYL was networks. Then Ted Codd published his famous paper "A Relational Model of Data for Large Shared Data Banks" (*Communications of the ACM,* Vol. 13, No. 6, June 1970). So I read that paper, and—speaking here as a mathematician!—to me it was obvious that the relational model was the right way to go. Looking back, if I'd realized how long it was going to take to get the world at large to agree with that position, I don't know if I would have been quite so enthusiastic ... Anyway, I began corresponding with Ted at that time, I met him some little while later, and one thing led to another.

In the introduction to your new book (*Database in Depth*) you write that some things needed to be said again. How would you summarize those?

Goodness! Where to begin? There's so much nonsense out there ... so little true understanding, that it seems to me that just about *everything* needs to be said again. Perhaps I can illustrate by quoting a typical document off the Web. The document I have in mind is called "Weaknesses of the Relational Model," and it comes from an academic institution in Germany (so the lack of understanding I'm talking about certainly isn't limited to the U.S., nor is it limited to the commercial world). Here are the alleged "weaknesses," quoted *verbatim,* except that I've numbered them for purposes of subsequent reference:

1. With regard to data modeling, you can't define attributes which are of complex types (arrays, records, tables). Each relation has to be in first normal form. Or in other words: A "simple" natural structure must be divided into many flat structures (= tables/relations). The result is a complex structure of relations.
2. Result of a query is a flat table. Any complex structure, which has been input of the query has got lost.
3. No way to define recursive programme structure (using SQL).
4. The type system of SQL doesn't match with the type system of the embedding language ("type mismatch").
5. Controlling integrity constraints costs a lot of time (need to control the usage of primary/foreign keys).
6. Lack of mechanisms to control the physical level of the database (only simple clustering).
7. Definition of operations detached from data definition.

Considered as "weaknesses of the relational model," every single one of these is just plain wrong. Number 1 displays a lack of understanding of first normal form; ditto number 2 (and by the way, that phrase "flat table" all by itself demonstrates a huge failure to understand what relations truly are; I could write several pages on this issue alone). Number 3 is not only false as a statement of fact, it also makes the mistake of equating the relational model and SQL—as does number 4 also. Number 5 is 180 degrees wrong; I mean, *not* "controlling integrity constraints" is what costs us, and I don't mean costs only with respect to time—there are other costs too, ones that I regard as much worse. Number 6 ignores the fact that the relational model *deliberately* has nothing to say about "the physical level of the database"; one objective for the relational model was always to give implementers a high degree of freedom at the physical level, in order to make possible a high degree of data independence (I discuss this issue at some length in the O'Reilly book, as a matter of fact). And number 7 is just flat wrong again.

In a later question you mention the fact that the O'Reilly book has an epigraph from Leonardo da Vinci, and I'll get to that question in due course. But I think it's pertinent to mention here that the book has a second epigraph also, from Josh Billings: "The trouble with people is not that they don't know but that they know so much that ain't so." The document just quoted illustrates this point only too clearly.

By the way: If you're not familiar with it already, you might like to take a look at Fabian Pascal's website `http://www.dbdebunk.com`, which contains (among other things) numerous ignorant quotes like the one at hand, together with some analysis and deconstruction of those quotes on Fabian's part. I contribute to that website myself on a fairly regular basis.

Since the first edition of Codd and Date we have seen many trends such as 4GLs through to object-oriented databases. Any that have appealed or do you feel they are a flash in the pan?

First, a small point (or perhaps it's not so small): I don't quite know what you mean by "the first edition of Codd and Date." For some years Ted and I did have a consulting company called Codd and Date; however, your phrasing suggests we might have collaborated on a book. In fact we never did. The history was like this. Ted was a genius; there's no question about that, and he came up with these wonderful ideas. But like so many geniuses, he wasn't too good at communicating his ideas to ordinary mortals. So that was my job: to be that ordinary mortal! Once I'd understood the ideas myself, I was able to explain them to other people ... and that's what I did. In particular, I wrote a book called *An Introduction to Database Systems,* which was a textbook on database management that covered the relational model in particular (though it covered a lot of other things too). The first edition of that book appeared in 1975 (it's in its eighth edition now). But yes, I agree that many things have happened since that first edition, if that's what you meant. So let me turn to your question regarding the "many trends" ... Once again I fear my answer has to be fairly lengthy; please bear with me.

Actually I rather like your use of the word "trends" here. Because the truth is, so many of the things that have arisen over the years have indeed been just that, trends. The relational model is not and never was a mere "trend." The relational model is a solid scientific theory. *Nothing* has appeared on the horizon to replace it, and in my opinion it's extremely unlikely (I don't say impossible, you never know what's around the next corner) that anything ever will. The reason I say this is that the relational model is based on set theory and predicate logic, elements of which go back well over 2000 years. These disciplines in their various forms have been studied, analyzed, used, and extended by huge numbers of very smart people over the past two millenia (I refer here to logicians and mathematicians, of course); thus, it seems to me very unlikely that there's anything seriously wrong with them. By contrast, alleged competitors to the relational model—things like the so-called object "model" or the so-called semistructured "model" (and I set the term "model" in quotes very deliberately here)—simply don't have the same kind of pedigree as the relational model does, nor do they enjoy the same kind of scientific respectability.

So I certainly don't find "appealing" any of the things that (it's been claimed at one time or another) are supposed to replace the relational model. And in this connection I'd like to say explicitly that I reject, as proposed "replacements" for the relational model, both (a) XML and the semistructured "model," which I see as reinventing the old failed hierarchic "model," and (b) objects and the object-oriented "model," which I see as reinventing the old failed network "model."

But the trouble with the relational model is, it's never been implemented!—at least, not in commercial form, not properly, and certainly not fully. So while it's true that there have been a couple of developments in the marketplace over the past few years that I do quite like, I like them primarily because I see them as attempts to implement pieces of the relational model that should have been implemented years ago but weren't. I refer here to (a) "business rules" and (b) "object/relational DBMSs." I'll take them one at a time.

- *Business rules:* Business rule systems are a good idea, but they certainly aren't a new idea. Without going into a lot of detail, business rule systems can be seen as systems that attempt to implement the integrity piece of the relational model (which today's mainstream SQL products still—over 35 years after the model was first described!—so signally fail to do).

- *Object/relational DBMSs:* To a first approximation, "object/relational" just means the domains over which relations are defined can be of arbitrary complexity. As a consequence, we can have attributes of relations—or columns of tables, if you prefer—that contain geometric points, or polygons, or X rays, or XML documents, or fingerprints, or arrays, or lists, or relations, or any other kinds of values you can think of. But this idea too was always part of the relational model! The idea that the relational model could handle only rather simple kinds of data (like numbers and strings and dates and times) is a huge misconception, and always was. In fact, the term object/relational, as such, is just a piece of marketing hype ... As far as I'm concerned, an object/relational system done right would simply be a *relational* system done right, nothing more and nothing less.

That said, I must now add that, as so often, "Between the idea / And the reality / ... Falls the shadow" (T. S. Eliot—one of my favorites). While business rules and object/relational systems are both good ideas in principle, it doesn't follow that today's commercial realizations of those ideas are well done. In particular, today's object/relational systems (and the SQL standard)—in what I regard as a totally misguided attempt to accommodate certain ideas from object systems—have introduced pointers into SQL databases! Ted Codd very deliberately excluded pointers from the relational model, for all kinds of good reasons, and I think the addition of pointers to SQL has to be seen as the final nail in the coffin of any claims SQL might ever have had of being a true relational language.

Enough of this grousing. The next thing I want to say is that, while the relational model is certainly the foundation for "doing databases right," it's *only* the foundation. There are various ways we can build on top of the relational model, and various attempts have been made to do such a thing over the past 25 years or so. Here are a couple of examples:

- *Higher-level interfaces:* It was never the intention that every user should have to wrestle with the complexities of something like SQL (or even the complexities, such as they are, of the relational algebra). I always rather liked the visual interfaces provided by Query-By-Example and the "visual programming" front-ends to Ingres, for instance. And there are many other attractive front-ends that simplify the business of building applications on top of a relational (or at least SQL) DBMS. 4GLs too can be regarded as a higher-level interface—but I was never very impressed by 4GLs as such, in part because they never seemed to be precisely defined anywhere; the idea might have been OK, but without a formal definition of the semantics of the language some very expensive mistakes can be (and were) made. Natural language systems are another example; I still have an open mind about these, but I don't think anyone could claim they've been a huge success as yet.

- *Special-purpose applications:* I think the right way to view things like OLAP and data mining is as special-purpose applications that run on top of the DBMS. I mean, I don't think these things should be part of the core DBMS (I could be wrong). Either way, however, I do want to complain about the CUBE stuff in SQL, which takes one of the worst aspects of SQL—its support for nulls—and "exploits" it to make it even worse. But that's a particular hobbyhorse of mine ... I think I'd better stop right here.

On the dedication page you quote Leonardo da Vinci on theory without practice. Do you feel there is a lack of theory in most practice of database design? Do you feel this is a problem for the industry?

There are several issues here. If by "database design" you really do mean design of databases as such, then (as I explain in the O'Reilly book) there really isn't very much theory available anyway—though it's true that what little theory does exist is ignored far too often (especially in data warehouses, where some truly dreadful designs are not only found in practice but are actually recommended by certain pundits). But even if designers do abide by what design theory we have, there are still far too many decisions that have to be made without any solid theory to help. We need more science!

On the other hand, if by "database design" you really mean "DBMS design"—if not, then forgive me, but use of the term "database" for "DBMS" and/or "DBMS instance" is all too common and is (in my opinion) very confusing and deserves to be thoroughly resisted—then most certainly there are all too many departures from theory to be observed. And yes, it's a huge problem for the industry, and indeed for society at large (since society at large places such reliance on those defective products). This is an issue I do feel very strongly about, but I'm afraid it'll take a few minutes for me to explain why. Again, please bear with me.

First of all, it's a very unfortunate fact that the term "theory" has two quite different meanings. In common parlance it's almost a pejorative term—"oh, that's just your theory." Indeed, in such contexts it's effectively just a synonym for *opinion* (and the adverb *merely*—it's *merely* your opinion—is often implied, too). But the meaning in scientific circles is quite different. To a scientist, a theory is a set of ideas or principles that, taken together, *explain* something: some set of observable phenomena, such as the motion of the planets of the solar system. Of course, when I say I it explains something, I mean it does so *coherently:* It fits the facts, as it were. Moreover (and very importantly), a scientific theory doesn't just explain: It also *makes predictions,* predictions that can be tested and—at least in principle—can be shown to be false. And if any of those predictions do indeed turn out to be false, then scientists move on: Either they modify the theory or they adopt a new one. That's the scientific method, in a nutshell: We observe phenomena; we construct a theory to explain them; we test predictions of that theory; and we iterate. That's how the Copernican system replaced epicycles[1]; how Einstein's cosmology replaced Newton's; how general relativity replaced special relativity; and so on.

As another example, consider the current debate in the U.S. over the theory of evolution versus creationism (also known as "intelligent design"). Evolution is a scientific theory: It makes predictions, predictions that can be tested, and in principle it can be falsified (namely, if those predictions turn out to be incorrect). In particular, evolution predicts that if the environment changes, the inhabitants of that environment will probably change too. And this prediction has been widely confirmed: Bacteria evolve and become resistant to medical treatments, insects evolve and become resistant to pesticides, plants evolve and become resistant to herbicides, animals evolve and become resistant to disease or parasites (in fact, something exactly like this seems to have happened in the U.S. recently with honeybees). There's also the well-documented case of the Peppered Moth (*Biston betularia*) in England, which evolved from a dark form, when the air was full of smoke from the Industrial Revolution, to a light form

1. At least in the scientific world. As is well known, there were those who refused to accept the Copernican system for many, many years—but the naysayers were certainly not scientists.

when the air was cleaned up. By contrast, creationism is not a scientific theory; it makes no testable predictions, so far as I know, and as a consequence it can be neither verified nor falsified.

By the way, Carl Sagan has a nice comment in this regard:

> *In science it often happens that scientists say, "You know, that's a really good argument, my position is mistaken," and then they actually change their minds, and you never hear that old view from them again. They really do it. It doesn't happen as often as it should, because scientists are human and change is sometimes painful. But it happens every day. I cannot recall the last time something like that happened in politics or religion.*

Now let's get back to databases. The point is, the relational model is indeed a theory in the scientific sense—it's categorically *not* just a matter of mere opinion (though I strongly suspect that many of those who criticize the model are confusing, deliberately or otherwise, the two meanings of the term "theory"). In fact, of course, the relational model is a *mathematical* theory. Now, mathematical theories are certainly scientific theories, but they're a little special, in a way. First, the observed phenomena they're supposed to explain tend to be rather abstract—not nearly as concrete as the motion of the planets, for example. Second, the predictions they make are essentially the theorems that can be proved within the theory. Thus, those "predictions" can be falsified only if there's something wrong with the premises, or axioms, on which the theorems are based. But even this does happen from time to time! For example, in euclidean geometry, you can prove that every triangle has angles that sum to 180 degrees. So if we ever found a triangle that didn't have this property, we would have to conclude that the premises—the axioms of euclidean geometry—must be wrong. And in a sense exactly that happened: Triangles on the surface of a sphere (for example, on the surface of the Earth) have angles that sum to more than 180 degrees. And the problem turned out to be the euclidean axiom regarding parallel lines. Riemann replaced that axiom by a different one and thereby defined a different (but equally valid) kind of geometry.

In the same kind of way, the theory that's the relational model *might* be falsified in some way—but I think it's pretty unlikely, because (as I said in my answer to the previous question) the premises on which the relational model is based are essentially those of set theory and predicate logic, and those premises have stood up pretty well for a very long time.

So, to get back (finally!) to your question—do I feel the lack of attention to theory is a problem?—well, of course my answer is *yes*. As I said in another recent interview: This is the kind of question I always want to respond to by standing it on its head. Database management is a field in which, in contrast to some other fields within the computing discipline, there is some solid theory. We know the value of that theory; we know the benefits that accrue if we follow that theory. We also know there are costs associated with not following that theory (we might not know exactly what those costs are—I mean, it might be hard to quantify them—but we do know there are going to be costs).

If you're on an airplane, you'd like to be sure that plane has been constructed according to the principles of physics. If you're in a high-rise building, you'd like to be sure that building has been constructed according to architectural principles. In the same kind of way, if I'm using a DBMS, I'd like to be sure it's been constructed according to database principles. If it hasn't, then I know things will go wrong. I might find it hard to say exactly what will go wrong, and I might find it hard to say whether things will go wrong in a major or minor way, but I *know*—it's guaranteed—that things will go wrong.

So I think it's incumbent on people not to say "Tell me the business value of implementing the relational model." I think they should explain what the business value is of *not* implementing it. Those who ask "What's the value of the relational model?" are basically saying "What's the value of theory?" And I want them to tell me what the value is of *not* abiding by the theory.

The Third Manifesto introduced the relational language Tutorial D, and you use it for the examples in the new book. Do you think it has a future as an implementation or do you never intend it to be implemented?

Again I'd like to clarify a couple of things up front. First, I'd like to explain what *The Third Manifesto* is. *The Third Manifesto* is a formal proposal by Hugh Darwen and myself for the future of data and data management systems. It's a fairly short document (maybe 12 pages); it's also pretty terse and, to be frank, not all that easy to understand. So Hugh and I wrote a book of some 500 pages (!) to explain it. The third edition of that book (title *Databases, Types, and the Relational Model: The Third Manifesto*) is due to be published late this year or early next year.[2] And the first confusion factor is that people often refer to that book, loosely, as *The Third Manifesto*—but it really isn't; in fact, the *Manifesto* proper constitutes just one chapter in the book.

Now, in the *Manifesto* proper we—I mean Hugh Darwen and myself—use the name **D** generically to refer to any database language that conforms to the principles laid down in the *Manifesto*. Note that there could be any number of distinct languages that all qualify as a valid **D**. (Perhaps I should say explicitly in passing that SQL is not one of them!) And in the *Manifesto* book we introduce a particular **D,** which we call **Tutorial D,** that's meant to be used as (we hope) a self-explanatory basis for illustrating relational ideas—self-explanatory in the sense that if you're familiar with any conventional programming language, you should have no difficulty in following it. (I should mention in this connection that I've previously used **Tutorial D** in several other books, as well as in live seminars, and my experience does tend to confirm that it's sufficiently self-explanatory for the purpose.)

However, **Tutorial D** is, in a sense, only a "toy" language; it has no I/O and no exception-handling and is incomplete in various other ways as well. It follows that **Tutorial D** *per se* could never serve as a true industrial-strength language. But that doesn't mean we don't want to see it implemented! We believe it could serve as an extremely useful vehicle for teaching and reinforcing relational concepts before students have to get their heads bent out of shape from having to learn SQL. And as a matter of fact some implementations do exist; you can find details (including downloadable code) at the website `http://www.thethirdmanifesto.com`.

As you might expect, Hugh and I have a long list of follow-on projects to investigate when we get the time—and one of them is to take **Tutorial D** and extend it to become what you might call **Industrial D**. When it's defined, we would certainly like to see **Industrial D** (which by that time will probably be given some fancier name) to be implemented in commercial form. That's definitely one of our goals.

You say some fairly harsh things about SQL. What are the major flaws? Do you think SQL has a future or do you think it will be replaced by something closer to the relational model?

"You say some fairly harsh things about SQL": Well, you're not the first person to make this comment. One correspondent wrote: "As a practitioner, SQL is what I have to work with, and

2. It was published by Addison-Wesley in 2006.

while I don't mind some criticism of its shortcomings, [your] criticism starts to look more like a personal vendetta" (though the same correspondent did subsequently say it "finally made sense at the end of the book as to why [you] did this"). This is how I replied to these comments:

> *Oh dear. I tried hard to tone down my rude remarks; obviously I didn't succeed. I could have been much ruder, though! But the fact is that SQL fails in so many ways to support the relational model adequately—it suffers from so many sins of both omission and commission—that any book that both covers the relational model faithfully and discusses SQL as well cannot avoid being negative about SQL to some extent. It was interesting to me to discover (it wasn't explicitly intended on my part) how, almost always, the SQL formulation of any given problem was lengthier and more convoluted than its* ***Tutorial D*** *counterpart.*

The same correspondent also felt that **Tutorial D** was a "distraction" and didn't "seem to help in illustrating relational concepts"—implying, I suppose, that if I wanted to use any language for the purpose I should have used SQL. Here I replied as follows:

> *This comment I do find surprising. I have to use some syntax to illustrate the concepts (even Ted Codd did this in his very first papers), and SQL manifestly doesn't do the job.* ***Tutorial D*** *not only does do the job but was expressly designed to do so! How would you illustrate the relational EXTEND or SUMMARIZE operators—or even projection—using only SQL, without getting involved in a number of distracting irrelevancies? And how would you illustrate the relational concepts that SQL doesn't support at all (e.g., no left-to-right ordering to attributes, relation-valued attributes, TABLE_DEE, etc.)? I could go on.*

At the same time the correspondent also wanted me not to be so explicit regarding SQL's problems: "I think the book would be better served if [you] limited the discussion to just relational theory and let the reader (one enlightened) make [his or her] own judgments about SQL's inadequacies." Here I replied that I didn't agree, because:

- First, I think [the inclusion of SQL coding examples] is desirable to serve as a bridge between what the reader is supposed to know already (database practice) and what I'm trying to teach (relational theory). A book that covered the theory in a vacuum would be a different book—one that I explicitly suggested at the outset I didn't want to write, and one that would probably turn out to be (or at least look) too theoretical for my intended audience. (Incidentally, that different book would certainly have to use **Tutorial D,** or something equivalent, to illustrate the ideas—so this objection contradicts the previous one, in a sense.)
- Second, it's my experience that readers typically do need to have their hands held, as it were—points made only implicitly have a tendency to escape many people. (I don't mean this remark to be offensive! I include myself among those who often need their hands held in this kind of context.)

So yes, I do think SQL is pretty bad. But you explicitly ask what its major flaws are. Well, here are a few:

- Duplicate rows
- Nulls

- Left-to-right column ordering
- Unnamed columns and duplicate column names
- Failure to support "=" properly
- Pointers
- High redundancy

Do I think SQL has a future? Yes, of course—of a kind. Here I'd like to quote something Hugh Darwen and I say in the *Manifesto* book:

> *We reject SQL as a long-term foundation for the future. However, we are not so naïve as to think that SQL will ever disappear; rather, it is our hope that some* ***D*** *will be sufficiently superior to SQL that it will become the database language of choice, by a process of natural selection, and SQL will become "the database language of last resort." In fact, we see a parallel with the world of programming languages, where COBOL has never disappeared (and never will); but COBOL has become "the programming language of last resort" for developing applications, because preferable alternatives exist. We see SQL as a kind of database COBOL, and we would like some* ***D*** *to become a preferable alternative to it.*

Given that SQL databases and applications are going to be with us for a long time, however, we do have to pay some attention to the question of what to do about today's "SQL legacy" (I like this phrase). To this end, the *Manifesto* suggests that (a) SQL should be implementable in **D**—not because such implementation is desirable in itself, but so that a painless migration route might be available for current SQL users—and (b) existing SQL databases should be convertible to a form that **D** programs can operate on without error. And we've given enough thought to these objectives to believe they can be achieved without compromising on any of the other goals we have for the language **D**.

Do I think SQL will be replaced by something closer to the relational model? Well, I think I've answered this question by now. Indeed, the hope of performing such a replacement is one of the major reasons we wrote the *Manifesto* in the first place.

So where is the future of database development?

Again I assume that by "database" here you really mean "DBMS" ... Some answers to this question are given in the O'Reilly book (pages 172–176); here I'll just briefly mention a few points.

- The first and overriding one is this: We need DBMSs that (unlike the mainstream products of today) truly and fully support the relational model. Again, that's what *The Third Manifesto* is all about.
- Second, there's a promising (and radically different) implementation technology called *The TransRelational™ Model* coming down the road that looks as if it might be very well suited to that task of implementing the relational model properly. This possibility is under active investigation.

- Third, I'd like to see good support for *temporal data*. I think support for the time dimension is important already and due to become much more so. Now, several researchers have proposed approaches to this problem—approaches that are, however, fundamentally flawed for the most part (in my opinion), because they violate fundamental relational principles. By contrast, it's my belief that the relational model already includes what's needed to support the time dimension properly. All that's needed is a set of well chosen and carefully designed shorthands. Along with two coauthors, Hugh Darwen and Nikos Lorentzos, I've written about this topic at some length in another book, *Temporal Data and the Relational Model* (Morgan Kaufmann, 2003).

Perhaps I should say explicitly too that I *don't* think XML and XQuery are going to "take over the world" (despite all of the marketing hype to the contrary). People who think otherwise are simply displaying their lack of understanding of database fundamentals. As I mentioned earlier, I see XML (and the semistructured "model" on which it is allegedly based) as an attempt to reinvent the old failed hierarchic "model." This isn't to say that XML has no role to play—it probably does, but (to repeat) the role in question isn't that of taking over the world. XQuery in particular seems to me to be even worse than SQL! No, let me put a positive spin on that: If you like SQL, you're going to love XQuery. (Coming from me, you can take this remark as damning indeed.)

Finally, I have to ask: When Chris Date wants to build a database, what product does he turn to? Do you have time for consumer level products such as Access? Do you think they have a place?

My lawyer has told me to say "No comment" to this one. Seriously, I do have a policy of never answering this question directly; I'm only too aware of what might happen if I were thought to be endorsing some specific product. (Did you expect anything different?) So all I can do is offer some platitudes: Even though they're all deeply flawed, you do have to use one of the existing DBMSs, because they're all that's out there—they're the only game in town, pretty much (though I'm being a little unfair here to at least one product, Dataphor from Alphora, which does constitute an attempt to implement the ideas of *The Third Manifesto*). Which particular product you choose will naturally depend on your own specific requirements (and yes, it might be one of the "consumer level products such as Access"). But whichever one you do choose, you'll be able to make much better use of it if you're familiar with the underlying theory than you would be able to do otherwise ... which is one of the reasons I wrote the O'Reilly book.