Confusing Physcal and Logical Levels of Abstraction

Correspondence Between David McGoveran and Jim Starkey, April-May, 2014

In April, 2014, Jim Starkey posted a commentary "Is the Relational Data Model Spent?" on the Database Architect's Forum of LinkedIn. I was sent a copy of the post by Fabian Pascal, a member of the forum. Choosing not to join the forum in order to respond, Jim and I communicated via postings on Fabian's blog www.dbdebunk.com with two rounds each. The correspondence follows, in order of its appearance and as it appeared, spelling errors and all. I have taken the liberty of improving formatting for readability. I have assumed that, since Jim's commentary was public and not copyrighted (I have added a notice on his behalf herein), that it is fair use to include it herein. Of course, if Jim ever has any objection I will be happy to remove his portion of the correspondence and expand my commentary to so as to explain what he posted without including it.

Is the Relational Data Model Spent?

by Jim Starkey, Database Architect

Let me start by establishing my relational credentials. My first exposure to relational databases was some mimeographed copies of Codd's early papers while working the ARPAnet Datacomputer project. I was unable to convince the company I worked for that relational was the future ("too academic"). I joined DEC to write a relational database, but got sidelined by a hiring bait and switch. I did start the DEC Rdb project, but had to spin off due other other responsibilities. Later, after I had the MVCC idea, I wrote Rdb/ELN, the world's first MVCC relational database. I left DEC to start my first relational database company, Interbase Software, acquired by Ashton Tate, in turn acquired by Borland. (Interbase lives on as the Firebird RDBMS.) After waiting out a non-compete, I wrote an integrated Web application platform with an embedded relational database system that was acquired by MySQL to become the Falcon storage engine. While at MySQL, I had the idea for a radically new architecture for an elastically scalable database. After Sun acquired MySQL, I left and started what is now NuoDB.

In short, having written four or five commercial relational database systems over 35 years, I think I know at least as much about the relational data model as the average bear.

Along the way, I've come to understand applications often succeed despite the relational data model and not because of it. In other words, database schemas frequently reflect pragmatic implementation concerns rather than any rational model of data. Some common examples of this:

- High end applications tend to use generic schemas and a layer to map logical tables into physical tables.
- If the set of table attributes is unknowable at design time, a logical table must be represented with a base table and separate attribute/value table.
- Partitioned database systems that impose catastrophic performance penalties for cross-partition transactions inspiring schemas that Fabian could never love.
- Many applications include tables with hundreds or even thousands of columns, almost all of which will be null for any given row instance.
- Technologies like Hibernate produce garbage schemas with no face validity.

The common denominator of these cases is that creative people are using relational database systems to work around the deficiencies of the relational data model.

I'm not going to suggest we throw out the relational model or relational database systems. They work well for many applications and adequately for many more. But isn't it time to start looking beyond a methodology that was designed for computing systems with less processing power, memory, and connectivity than modern home thermostats?

We need a new data model more flexible and powerful than the relational data model, one that recognizes that humans are data pack rats, that acknowledges that the Web requires very complex retrievals with a single round-trip latency requirement, that most of human knowledge can't be represented in rows and columns, and that unrestricted context-free search has broadened the scope of human information gathering far beyond what we thought possible a couple of decades ago.

I submit that the relational model drove us forward for 25 years, but is obsolete and retarding progress. It has good things like ACID transactions and link-by-common-value and bad things like bounded types and outer join. Let's ditch schemas and tables and let every record contain whatever it needs to contain. Let's have an access language that can return arbitrarily complex results in a single round trip. Let's index almost everything by default so we can support high performance context-free searches by word or phrase while still supporting attribute-specific searches. Let's treat the size of data as a logical constraint, where necessary, but not require the designer to know the size of everything in advance.

Comments on Jim Starkey's "Is the Relational Data Model Spent?"[1]

by David McGoveran, April 23rd 2014

Jim Starkey's opinions reflect those of many professionals who have used and even developed SQL DBMSs and their predecessors. While the concerns with so-called "commercial relational database systems" expressed by Jim are valid, they have nothing to do with the relational (data) model. They are the result of DBMS implementations by those who borrowed something from the relational model, but never understood it and so did not know how to take advantage of it to solve application problems.

Jim Starkey employed important and useful features in both Rdb/ELN and Interbase, and deserves credit for having done so. I met Jim long ago in the early 1980s. I was an early developer using DEC's Datatrieve while Jim was working on that product, designed and developed one of the first large scale commercial applications that ultimately used the first versions of Interbase (I had designed it run on the Britton-Lee Intelligent Database Machine), was an early user of Rdb (when its primary query language was the subverting CODASYL Datatrieve in 1984-1985!), was the consultant who first trained DEC engineers on the relational model and products in the "relational DBMS" category, and wrote/published detailed critical technical evaluations of both Rdb and Interbase (among many others).

The problem we face is this: Relational terminology was hijacked long ago and used for the wrong purposes. Personally, I've come to believe it is now impossible to overcome the confused thinking, miss-education, and miscommunication this hijacking caused. The 'Relational Data Model' of Jim's title is spent because the referent is not The Relational Data Model. Herein I will use RDM (for "The Relational Data Model") to refer to the logical data model that resulted from the work of Dr. E. F. Codd and his colleagues and adherents. I will use "alleged relational" for whatever it is that guides the design and development of everything else that pretends to be a relational product, including SQL and many other commercial DBMSs and the fuzzy concepts that, for example, NoSQL champions attack and allege as being "relational."

What is "spent" is a set of simplistic, highly-constrained physical data storage and retrieval techniques that have jointly been labeled as "relational". For example, RDM was never intended to constrain physical data storage to sets of contiguous rows as records having columns as fields.[2] It was intended to hide physical storage organization and access methods from programmers, allowing them to be changed transparently. As someone who has designed, developed, analyzed, and optimized hundreds of database applications in a variety of

[1] Posted on LinkedIn's Database Architects Group and sent to me by Fabian Pascal

[2] If this statement puzzles you, I recommend a lot of reading on foundations of the relational model.

programming languages from FORTRAN and COBOL to Java and C++, I'm very familiar with the problems (and perceived problems) that developers face when using commercial DBMSs. I've worked on the bleeding edge of applications throughout my professional career, including real-time machine control, OLTP, workflow, decision support/BI/data warehouse, analytics and OLAP, integration, business process, text, image, video, voice, web, search engine, and cloud applications (and probably more I can't think of off the cuff). As a teacher of both college and industrial courses on these subjects, I also know how hard it is for developers to learn abstraction. The natural inclination is to think physically – from conception through deployment and maintenance. To suggest that RDM (taking liberties with Jim's reference to it as a "methodology") was designed for computing systems with any amount of processing power, memory, or connectivity is a complete misrepresentation of RDM history and Codd's intent. RDM is and has always been about the abstract representation of data, data structure, data relationships and data operations.

So let me drive the point home: The bulleted problems that Jim states are limitations of a simplistic, highly constrained physical data storage and retrieval model. Over the years I (and others) have written and lectured extensively over the difference between conceptual, logical and physical. Codd is largely responsible for starting that effort, being the first to clearly exhibit a logical data model. Those efforts did not take root. Logical concepts like RDM and physical characteristics like performance, allocation, concurrency, locking, and availability are treated as if they belonged to the same conversation.

When you talk about physical tables, physical "attributes", partitioning, performance, nulls (as physical placeholders), latency, or the utility or "validity" of a product like Hibernate, the context is physical and cannot be logical. True, if what you achieve physically with your application is constrained by a logical model that is weaker than the physical implementation platform (a computer with a particular operating system) on which your application runs, you may lack the flexibility to achieve some physical objective. However, that is not and cannot be the case with RDM. RDM is an expression of first order predicate logic (with equivalence) or "FPL". In terms of expressive power, FPL with arithmetic is more powerful than a Turing machine that is limited to expressions over finite sets and modern computers performing real computations are such Turing machines.[3] It follows that RDM cannot limit what you can achieve on a modern computer.[4] At worst, RDM can limit the way in which you express what you want to achieve.

Jim's statement that most of human knowledge can't be represented in terms of rows and columns is just nonsense. Rows, comprised of some number **n** of typed attributes, correspond to first order **n**-place predicates which, when specific values are substituted, result in logical propositions. The types of the attributes (i.e., their domains) and the relationships (i.e., constraints) among the attributes provide the semantics (i.e., the intended meaning or interpretation) of the proposition.

[3] There are many subtleties in this comparison which I will not go into as they are of no consequence to our subject.
[4] Certain computations on finite sets, such as transitive closure, require extensions of the original relational operators, but these types of problems are not in general computable in a decidable language.

Nothing in RDM limits the simplicity or complexity of those semantics. While there are expressions that are not first order, I challenge Jim to identify one statement of human knowledge that cannot be represented as an n-place predicate[5].

None of a specific implementation of data types, transaction model, indexing, data sizes, and so on are limitations of RDM. I do agree that outer join is a bad thing, as is outer union or any other "relational" operation that permits, let alone produces SQL-like nulls in its output[6].

Regarding performance (both response time and throughput), concurrency, and storage efficiency, I have always found these complaints to be the product of the rigid thinking or ill-informed. During the 1980s and 1990s I had a standing challenge to the industry: show me a relational database application that I cannot optimize to yield 10x better performance, 10x higher concurrency, and with 10x less storage and my consulting is free. I had numerous takers and no winners.

With respect to data types, a flexible type system based on a rigorous theory of types is needed for programming languages as badly as for database languages. Date and Darwen have published a proposal on the latter problem[7]. Curiously, the problem of transactions can be handled by RDM's logical data independence: If transactional transformation T acting on an RDM-conformant database D results in database D', there exists a derived relation (e.g., a view) and an update of that derived relation such that D transforms to D'.

In conclusion, I do agree that we need to abandon the onerous limitations and outright errors that have been perpetuated in the name of RDM by implementers of alleged relational DBMSs. But let's not keep falsely accusing RDM as the culprit. The culprit is those who do not understand how to differentiate between conceptual, logical and physical levels of abstraction. Perhaps we relational "bigots" need to invent new terminology, letting our frustrated colleagues have the old terminology to use however they wish. On the other hand, I do object to characterizing something as "logical" or a "model" that has no discipline, no logic, and no underlying theory. Imagine an architect of skyscrapers that took that approach – you won't catch me entering his buildings! And Jim, I apologize, but if he is a database architect, I would never rely on his applications for anything critical – they would be likely to get someone seriously hurt.

[5] Even paradoxes and other non-first order expressions can still be given as n-place predicates.
[6] To evade a foreseen complaint here, let me point out that (a) Codd's marks were not equivalent to SQL nulls and (b) we have made progress since Codd in improving and extending RDM concepts and theory.
[7] My own work on this problem is, sadly, still forthcoming.

Reply to "Comments on Jim Starkey's 'Is the Relational Data Model Spent?'"

by Jim Starkey, Database Architect

The article is well worth reading, a welcome break from the insulting, content-free snears from the RDM camp.

David challenges me to name one aspect of human knowledge that can't be represented in rows and columns. Fair enough. David, your article itself is an excellent example of something that can't be represented -- and found -- with a row and column representation. True (he said patting himself on the back), it can be represented as a BLOB and on some systems even an HTML structured blob. But it can't be searched with first order predicate logic.

Personally, I'm a fan of first order predicate logic. Who isn't? It's the fundamental language of mathematics. I'm sure it wasn't lost of David that the Datatrieve language was, indeed, first order predicate logic extended with sufficient (and optional) syntactic sugar to be English-like. I was very pleased with the degree that the language was accepted by people ranging from mathematicians and researchers to secretaries (who, more than often, found they had, in fact, found new careers as programmers).

The problem with first order predicate logical is that each predicate in a full expression must resolve to either true or false (let's ignore nulls). Word search can't be expressed in first order predicate logical. If you search for the phrase "first order predicate logical", you're going a rather fuzzy search for documents that contain words in that phrase. And, unlike first order predicate logic, the application of the search phrase to a specific document isn't true or false but a "hit score" where a document containing those words in order without intervening words will be scored the highest (and ranked among other such has by the relative position of the phrase in the document). At the bottom are documents that containing at most one of the words. It's logic, David, it just isn't first order predicate logic.

David say, "Nothing in RDM limits the simplicity or complexity of those semantics." I respectively disagree. Restricting a data model to first order predicate logic denies the fact that the most successful computing company in mankind's history, Google, is based on search, not first order predicate logic.

How is this possible? The answer, I'm afraid, is that the database community, especially the academic database community, suffers from a profound case of Head in Sand Syndrome (HISS), which can be paraphrased, "if it wasn't in my CS 101 class, it doesn't exist."

David, you write well and are clearly a decent and thoughtful fellow. Pull your head out of the sand. First order predicate logic is not the be all and end all of human thinking. And, not incidentally, first order predicate logic is not restrict to sets.

[Of course Amorphous uses first order predicate logic, Duh. It also implements weighted hit search semantics and user control over the fuzziness in between.]

Response to Jim Starkey's Comments on Predicate Logic and Data Modeling[8]

by David McGoveran, May 5th 2014

Jim Starkey's reply to my April 23rd 2014 "Comments on …" perpetuates the initial mistake I pointed out: confusing a physical data storage and retrieval techniques with RDM. With respect, Jim makes at least the following errors in his reply:

1. Jim says my article can't be represented in rows and columns – I'll assume he means RDM tuples and attributes, and not the physical records and fields he and so many others used to implemented some "alleged relational" but in fact physical data store. His is a categorical statement, presuming that my article comprises knowledge of some specific sort (smile). And it's wrong. Seriously, part of the "problem" Jim confronts is that he doesn't know what kind or level of knowledge he wants to model about my article or its content. Until he does, there are just too many possibilities. Documents have lots of content, lots of metadata, lots of interpretations, and lots of internal relationships (formatting, semantic, structural or syntactic, and so on). At one level, they are just documents. At another level of analysis, they have subject matter or content that might relate to that of other data – for example – documents. How we represent knowledge, and in how much detail, always partially determines the class of queries we can express.

   At the simplest level, RDM can represent the fact that I wrote the article and on what date with a relation - Writings (Author, Article_Title, Date_Written). If we want to go further, for example, a domain of type "pdf_document" with PDF operators could be created and then the article itself represented in the relation – Writings(Author, Article_Title, Date_Written, Content).[9] This is no more complex than a relation with a text domain type and could implement document "substring" functions similar to text substring functions with which everyone is familiar. Notice that I've adhered to the use of typed domains – no truly untyped BLOBs here thank you![10] If we want to, we could design a data model of the grammatical structure of the document showing the relationships among content such as chapters, sections, paragraphs, sentences, noun phrases, verb phrases, and so on. If we wanted to

[8] Posted on LinkedIn's Database Architects Group and sent to me by Fabian Pascal

[9] Can you, the reader, think of one or more reasonable predicates corresponding to each of these relations? You don't need to give a precise expression, just rough it out. Its easy!

[10] I'm alluding to the fact that RDM is based on *typed* FPL: Every domain has a type with a well-defined, *computable* set membership function (possibly a lookup function referencing some other set).

analyze the content, we could – again for example – assign subject matter keywords to each of these structural elements. Logical models at these levels comprise multiple relations.[11]

The problem with documents (or *any* arbitrary content) is not that the knowledge (useful facts) contained therein cannot be given a representation in RDM, it is that data modelers choose not to analyze them. Documents are not "unstructured" – rather, they are very highly structured and come in many types. Few implementers are willing to take the time to model their content, often because of resource constraints but sometimes out of ignorance about how to use RDM.

All too often the asserted and actual needs of those who complain about RDM are *not* about knowledge representation, but *knowledge discovery*. That is the problem, for example, that Google Search attempts to solve. Likewise, many so-called *analytics* and *data integration* application objectives face this problem. It's an expensive, imprecise, and difficult problem.

2. Jim says my article can't be searched in first order predicate logic (FPL herein). His statement is no more true than if he were to say that a text data type can't be searched in FPL. At least, the assertion is not relevant.[12] All that matters is that the query expression does not contain a predicate variable that ranges over predicate variables. Both tasks are easily accomplished, as is clearly demonstrated by the common use of a substring search function operating on an attribute defined over a text domain.

   When a domain operator is used to evaluate an attribute value or to perform type conversions,[13] RDM does not permit any higher ordered logic of the domain operator to be exposed to the relational (and FPL) query language. This guarantees that RDM need not impose any restrictions on how the domain operator is defined: Its expressions can belong to any logical system (second order PL, third order PL, fuzzy, etc.) as long as they are well-defined and always yield properly typed results.

   In RDM, the declarative, relational operators do not have direct access to and so can never be directly combined with the expression necessary to define domain operators, only the operator's typed results (values!). Understanding this domain-based encapsulation of higher ordered expressions is essential to understanding and using the power of RDM. It is why

[11] I won't give an example here – I just don't have the time to teach what would require an entire course in data modeling.
[12] It is true that, if you tried to "flatten" the query expression so that domain operations were forced to be expressed in FPL, that the resulting expression would not be FPL. However, RDM does not require such flattening and, in fact, forbids it.
[13] Technically, these are subtype to subtype conversions and the operator must belong to a domain that is a supertype of both subtypes. SQL (and most other languages) get this wrong – one reason I consider their informal and implicit type system so bad.

domains and the typed attributes based on them are described as "atomic" in RDM – *not* because the data type must be "simple" or "have no internal structure."[14]

3. Sorry Jim, but Datatrieve was not a first order predicate logic language. Containing procedural structures such as loops and conditionals, it exposed a computationally complete language to users. That requires at least second order predicate logic. Those extensions of yours weren't just syntactic sugar. Indeed, those secretaries who learned Datatrieve did become programmers!

4. Scoring algorithms such as those used in search – including Jim's example of fuzzy pattern matching – have nothing to do with the particular logical system employed. These are computations used as input to a decision procedure and can be used to rank hits. That decision procedure either does or does not return a "hit" – consistent with FPL. Even in "fuzzy" search, so-called fuzzy logic is typically not used. And just so you know, I understand fuzzy logic quite well: I knew Lofti Zadeh back in the day, gave an invited talk to his graduate seminar, published two peer reviewed papers on fuzzy logic, and evaluated it thoroughly as a deviant logic.

5. I am extremely familiar with Google and its systems. I am a great admirer of what Google has accomplished, and continues to accomplish, technically. Google does not implement a logical data model, let alone a general purpose DBMS. It comprises a collection of highly specialized and optimized databases. Public disclosures show that Google implements a physical data store with algorithms for managing physical issues (availability, replication, performance, caching, and so on). Search algorithms (such as those built on the MapReduce model) require implementation by programmers and do not comprise a query language *per se*. Google Search does *not* return the answer to *any* knowledge question except by accident: It merely returns blind hits on search terms. The user must then search through those results to *discover*, *access*, and *interpret* possible knowledge sources. It is far too easy for the naïve user to combine bits from multiple hits to conclude meaningless nonsense… and sadly, then to act on it. Worse, sophisticated programmers and analysts fall prey to the same trap, providing automated delivery of unsupportable results to managers upon which to make decisions.

   As I suggested in my previous response to Jim, such ***applications*** have tremendous value. They are not, however, representative of logical data models. Of great importance, they cannot provide physical data independence – their software implementations are strongly coupled to their physical storage structure.

6. Regarding HISS – Is that a backhanded insult? (grin) No matter. The only sand near my head is the sandstorm of ill-informed statements of those who understand little of why logic

[14] In "Is the Relational Data Model Spent?", Jim advocated against bounded types, presumably because he wants to have support for data which can be typed (he wrote "named") later. This is easily handled with a universal type.

matters. I've studied the properties of hundreds of logical systems over the last 40 years, from Lukasiewicz (many-valued) to L. E. J. Brouwer to von Neumann and Birkhoff (quantum logic). I don't think FPL is the be all and end all. I eagerly await an alternative suitable to database work. But I do know, without a doubt, that FPL is currently the only powerful and safe approach to a data model[15] that can enable logical and physical data independence. Why? Because to achieve that requires a declarative language over a decidable, consistent logic. First order predicate logic gives us the necessary expressive power and, when implemented in a real database, every expression on that database has an isomorphic expression in propositional logic[16]. Thus, we can reason about the database with the power of FPL while knowing (a) that every query is decidable – it returns a repeatable result, (b) that it will not give us inconsistent answers, and (c) any knowledge represented in the database is accessible by that query language.

I'm always been astounded by how often those who would propose using a particular logical system are incapable of evaluating the properties of those systems and, often, incapable of given the proposed system a formal definition. They don't elucidate axioms, rules of inference, or truth valuations, nor do they evaluate the properties of decidability, completeness, or consistency. They often have no understanding of proof theory or model theory, yet they pronounce the "power" of their proposed system.

In conclusion, let me reassert my thesis: Jim, you are complaining about physical data storage and retrieval, not RDM, and you are proposing a physical data storage and retrieval technology that is not based on any logical model. Without that, all you can know is that you can write ***some*** program to compute any computable function: Power, but no control. In other words, you will not be able to control or predict data integrity, semantic coherence of query results (witness Google search!), or any other property relevant to knowledge. And your programs will forever be tightly bound to specific data structures, storage allocation, and distribution.

A review (or maybe even intense study) of logic and the foundations of mathematics might be in order. I suggest Gottfried Wilhelm von Leibniz, Charles Saunders Peirce, Gottlieb Frege, Bertrand Russell, Kurt Godel, Stephen Kleene, Nicholas Rescher, and George Boolos and Richard Jeffrey, just to start. Undergraduate level courses just won't do. Learn not to confuse the truth valuation system (including evaluation operators) of an implementation with either the proof theory or the model theory of a system of logic. And make sure you know what all those terms mean. To apply all this to database theory, you might want study my old series "Nothing from Nothing."[17] It is dense and not even complete, but it will point you in the right direction.

[15] By data model I mean a formal system of data representation, integrity, and manipulation (including query) specific applications of which become particular interpretations of the model.
[16] Take care how you read this: I am ***not*** saying that FPL as used in RDM and propositional logic are identical!
[17] I've been working on an expansion of these articles in the form of a text book. Hopefully, my decease won't prove to be the final delay!

Let's get on the same page. Become meticulous in your analysis. Stop dissing RDM (it is not responsible for your woes), start attacking poor physical data storage and retrieval implementations (whether in DBMS products or applications), attack the *alleged and wrong* DBMS implementations that claim to be relational, and teach people the differences. End users and programmers alike deserve something far better than they've been given to date.

If you have a wonderful approach to physical data storage and retrieval, fine: convince me of its benefits and I'll happily support it – programmers need all the help they can get. But don't even suggest it can achieve any of the goals of RDM (which are *not* physical). You are smarter than that. Live up to your past accomplishments.

P.S. Thanks for the kind words regarding my writing, Jim.

