# The Birth of the Relational Model - Part 1 (go to Part 2)

Note - Applied Information Science has copied article from its source at http://www.intelligententerprise.com/9811/frm_online2.shtml in fear that the original posting may be removed. Certain sections have been emphasized and/or commented by the editor.

**A look back at Codd's original papers and how the relational model has evolved over time in today's leading database systems**

It was thirty years ago today / Dr. Edgar showed the world the way ...
-- *with apologies to Lennon and McCartney*

**Excuse the poetic license**, but it was 30 years ago, near enough, that Dr. Edgar F. Codd began work on what would become The Relational Model of Data. In 1969, he published the first in a brilliant series of highly original papers describing that work -- papers that changed the world as we know it. Since that time, of course, many people have made contributions (some of them quite major) to database research in general and relational database research in particular; however, none of those later contributions has been as significant or as fundamental as Codd's original work. A hundred years from now, I'm quite sure, database systems will still be based on Codd's relational foundation.

## The First Two Papers

As I already mentioned, Codd's first relational paper, "Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks," was published in 1969. Unfortunately, that paper was an IBM Research Report; as such, it carried a Limited Distribution Notice and therefore wasn't seen by as many people as it might have been. (Indeed, it's since become something of a collector's item.) But the following year a revised version of that first paper was published in Communications of the ACM, and that version was much more widely disseminated and received much more attention (at least in the academic community). Indeed, that 1970 version, "A Relational Model of Data for Large Shared Data Banks," is usually credited with being the seminal paper in the field, though that characterization is perhaps a little unfair to its predecessor. Those first two papers of Codd's are certainly unusual in one respect: They stand up very well to being read -- and indeed, repeatedly reread -- nearly 30 years later! How many papers can you say that of? At the same time, it has to be said that they're not particularly easy to read. The writing is terse and a little dry, the style theoretical and academic, the notation and examples rather mathematical in tone. As a consequence, I'm sure I'm right in saying that to this day, only a tiny percentage of database professionals have actually read them. So I thought it would be interesting and useful to devote a short series of articles to a careful, unbiased, retrospective review and assessment of Codd's first two papers.

As I began to get involved in writing that review, however, I came to realize that it would be better not to limit myself to just the first two papers, but rather to take a look at all of Codd's early relational publications. Over the next few months, therefore, I plan to consider the following important papers of Codd's in addition to the two already mentioned: "Relational Completeness of Data Base Sublanguages;" "A Data Base Sublanguage Founded on the Relational Calculus;" "Further Normalization of the Data Base Relational Model;" "Interactive Support for Nonprogrammers: The Relational and Network Approaches;" and "Extending the Relational Database Model to Capture More Meaning."

One last preliminary remark: I don't mean to suggest that Codd's early papers got every last detail exactly right, or that Codd himself foresaw every last implication of his ideas. Indeed, it would be quite surprising if this were the case! Minor mistakes and some degree of confusion are normal and natural when a major invention first sees the light of day; think of the telephone, the automobile, or television (or even computers themselves; do you remember the prediction that three computers would be sufficient to serve all of the computing needs of the United States?). Be that as it may, I will, of course, be liberally applying the "20/20 hindsight" principle in what follows. Indeed, I think it's interesting to see how certain aspects of the relational model have evolved over time.

## Codd's Fundamental Contributions

For reference purposes, let me briefly summarize Codd's major contributions here. (I limit myself to relational contributions only! It's not as widely known as it ought to be, but the fact is that Codd deserves recognition for original work in at least two other areas as well -- namely, multiprogramming and natural language processing. Details of those other contributions are beyond the scope of this article, however.)

Probably Codd's biggest overall achievement was to make database management into a science; he put the field on solid scientific footing by providing a theoretical framework (the relational model) within which a variety of important problems could be attacked in a scientific manner. In other words, the relational model really serves as the basis for a theory of data. Indeed, the term "relational theory" is preferable in some ways to the term "relational model," and it might have been nice if Codd had used it. But he didn't.

Codd thus introduced a welcome and sorely needed note of clarity and rigor into the database field. To be more specific, he introduced not only the relational model in particular, but the whole idea of a data model in general. He stressed the importance of the distinction (regrettably still widely misunderstood) between model and implementation. He saw the potential of using the ideas of predicate logic as a foundation for database management and defined both a relational algebra and a relational calculus as a basis for dealing with data in relational form. In addition, he defined (informally) what was probably the first relational language, "Data Sublanguage ALPHA;" introduced the concept of functional dependence and defined the first three normal forms (1NF, 2NF, 3NF); and defined the key notion of essentiality.

## The 1969 Paper

Now I want to focus on the 1969 paper (although I will also mention points where the thinking in the 1970 paper seems to augment or supersede that of the 1969 version). The 1969 paper --which was, to remind you, entitled "Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks" -- contains an introduction and the following six sections:

1. A Relational View of Data

2. Some Linguistic Aspects

3. Operations on Relations

4. Expressible, Named, and Stored Relations

5. Derivability, Redundancy, and Consistency

6. Data Bank Control.

The paper's focus is worthy of note. As both the title and abstract suggest, the focus is not so much on the relational model per se as it is on the provision of a means of investigating, in a precise and scientific manner, certain notions of data redundancy and consistency. Indeed, the term "the relational model" doesn't appear in the paper at all, although the introduction does speak of "a relational view ... (or model) of data." The introduction also points out that the relational "view" enjoys several advantages over "the graph (or network) model presently in vogue: It provides a means of describing data in terms of its natural structure only (that is, all details having to do with machine representation are excluded); it also provides a basis for constructing a high-level retrieval language with Œmaximal [sic] data independence'" (that is, independence between application programs and machine data representation -- what we would now call, more specifically, physical data independence). Note the term "retrieval language," by the way; the 1970 paper replaced it with the term "data language," but the emphasis throughout the first two papers was always heavily on query rather than update. In addition, the relational view permits a clear evaluation of the scope and limitations of existing database systems as well as the relative merits of "competing data representations within a single system." (In other words, it provides a basis for an attack on the logical database design problem.) Note the numerous hints here of interesting developments to come.

## A Relational View of Data

Essentially, this section of Codd's paper is concerned with what later came to be called the structural part of the relational model; that is, it discusses relations per se (and briefly mentions keys), but it doesn't get into the relational operations at all (what later came to be called the manipulative part of the model).

The paper's definition of relation is worth examining briefly. That definition runs more or less as follows: "Given sets S1, S2, ..., Sn (not necessarily distinct), R is a relation on those n sets if it is a set of n-tuples each of which has its first element from S1, its second element from S2, and so on. We shall refer to Sj as the jth domain of R ... R is said to have degree n." (And the 1970 paper adds: "More concisely, R is a subset of the Cartesian product of its domains.")

Although mathematically respectable, this definition can be criticized from a database standpoint -- here comes the 20/20 hindsight! -- on a couple of counts. First, it doesn't clearly distinguish between domains on the one hand and attributes, or columns, on the other. It's true that the paper does introduce the term attribute later, but it doesn't define it formally and or use it consistently. (The 1970 paper does introduce the term active domain to mean the set of values from a given domain actually appearing in the database at any given time, but this concept isn't the same as attribute, either.) As a result, there has been much confusion in the industry over the distinction between domains and attributes, and such confusions persist to the present day. (In fairness, I should add that the first edition of my book An Introduction to Database Systems (Addison-Wesley, 1975) was also not very clear on the domain-vs.-attribute distinction.)

The 1969 paper later gives an example that -- at least from an intuitive standpoint -- stumbles over this very confusion. The example involves a relation called PART with (among others) two columns called QUANTITY_ON_HAND and QUANTITY_ON_ORDER. It seems likely that these two columns would be defined on the same domain in practice, but the example clearly says they're not. (It refers to them as distinct domains and then says those domains "correspond to what are commonly called ... attributes.")

Note, too, that the definition of relation specifies that the domains (and therefore attributes) of a relation have an ordering, left to right. The 1970 paper does say that users shouldn't have to deal with "domain-ordered relations" as such but rather with "their domain-unordered counterparts" (which it calls relationships), but that refinement seems to have escaped the attention of certain members of the database community -- including, very unfortunately, the designers of SQL, in which the columns of a table definitely do have a left-to-right ordering.

Codd then goes on to define a "data bank" (which we would now more usually call a database, of course) to be "a collection of time-varying relations ... of assorted degrees," and states that "each [such] relation may be subject to insertion of additional n-tuples, deletion of existing ones, and alteration of components of any of its existing n-tuples." Here, unfortunately, we run smack into the historical confusion between relation values and relation variables. In mathematics (and indeed in Codd's own definition), a relation is simply a value, and there's just no way it can vary over time; there's no such thing as a "time-varying relation." But we can certainly have variables -- relation variables, that is -- whose values are relations (different values at different times), and that's really what Codd's "time-varying relations" are.

A failure to distinguish adequately between these two distinct concepts has been another rich source of subsequent confusion. For this reason, I would have preferred to couch the discussions in the remainder of this series of columns in terms of relation values and variables explicitly, rather than in terms of just relations -- time-varying or otherwise. Unfortunately, however, this type of approach turned out to involve too much rewriting and (worse) restructuring of the material I needed to quote and examine from Codd's own papers, so I reluctantly decided to drop the idea. I seriously hope no further confusions arise from that decision!

Back to the 1969 paper. The next concept introduced is the crucial one --very familiar now, of course -- of a key (meaning a unique identifier). A key is said to be nonredundant if every attribute it contains is necessary for the purpose of unique identification; that is, if any attribute were removed, what would be left wouldn't be a unique identifier any more. ("Key" here thus means what we now call a superkey, and a "nonredundant" key is what we now call a candidate key -- candidate keys being "nonredundant," or irreducible, by definition.)

Incidentally, the 1970 paper uses the term primary key in place of just key. Observe, therefore, that "primary key" in the 1970 paper does not mean what the term is usually taken to mean nowadays, because: a) it doesn't have to be nonredundant, and b) a given relation can have any number of them. However, the paper does go on to say that if a given relation "has two or more nonredundant primary keys, one of them is arbitrarily selected and called the primary key."

The 1970 paper also introduces the term foreign key. (Actually, the 1969 paper briefly mentions the concept too, but it doesn't use the term.) However, the definition is unnecessarily restrictive, in that -- for some reason -- it doesn't permit a primary key (or candidate key? or superkey?) to be a foreign key. The relational model as now understood includes no such restriction.

Well, that's all I have room for this month. At least I've laid some groundwork for what's to come, but Codd's contributions are so many and so varied that there's no way I can deal with them adequately in just one or two articles. It's going to be a fairly lengthy journey.

---

**The Birth of the Relational Model - Part 2**

---

# The Birth of the Relational Model - Part 2 (back to Part 1)



**Last month I began my retrospective review of Codd's first two relational papers** -- "Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks" (*IBM Research Report RJ599,* August 19, 1969) and "A Relational Model of Data for Large Shared Data Banks" (*CACM 13,* June 1970). In particular, I took a detailed look at the first section of the first paper. Just to remind you, that paper had six sections overall:

1. A Relational View of Data
2. Some Linguistic Aspects
3. Operations on Relations
4. Expressible, Named, and Stored Relations
5. Derivability, Redundancy, and Consistency
6. Data Bank Control.

## Some Linguistic Aspects

Codd opened this section with the following crucial observation: "The adoption of a relational view of data ... permits the development of a universal retrieval sublanguage based on the second-order predicate calculus." (Note the phrase "second-order;" the 1969 paper explicitly permitted relations to be defined on domains having relations as elements. I'll come back to this point when I discuss the 1970 paper in detail.)

It was Codd's very great insight that a database could be thought of as a set of relations, that a relation in turn could be thought of as a set of propositions (assumed by convention to be *true*), and hence, that the language and concepts of logic could be directly applied to the problem of data access and related problems. In this section of the paper, he sketched the salient features of an access language based on such concepts. These features include the following: The language would be set level, and the emphasis would be on data retrieval (though update operations would also be included). Also, the language would not be computationally complete; it was meant to be a "sublanguage," to be "[embedded] in a variety of host languages.... Any [computational] functions needed can be defined in [the host language] and invoked [from the sublanguage]." Personally, I've never been entirely convinced that factoring out data access into a separate "sublanguage" was a good idea, but it's certainly been with us (in the shape of embedded SQL) for a good while now. In this connection, incidentally, it's interesting to note that with the addition in 1996 of the PSM feature (Persistent Stored Modules) to the SQL standard, SQL has now become a computationally complete language in its own right, meaning that a host language is no longer logically necessary (with SQL, that is).

Codd also wrote, "Some deletions may be triggered by others if deletion dependencies ... are declared." In other words, Codd already had in mind in 1969 the possibility of triggered "referential actions" such as CASCADE DELETE (and in the 1970 paper, he extended this notion to include UPDATE referential actions as well). Also, the language would provide *symmetric exploitation.* That is, the user would be able to access a given relation using any combination of its attributes as *knowns* and the remaining ones as *unknowns.* "This is a system feature missing from many current information systems." Quite right. But we take it as a *sine*

*qua non* now, at least in the relational world. (The object world doesn't seem to think it's so important, for some reason.)

## Operations on Relations

This section of the paper provides definitions of certain relational operations; in other words, it describes what later came to be called the manipulative part of the relational model. Before getting into the definitions, however, Codd states: "Most users would not be directly concerned with these operations. *Information systems designers and people concerned with data bank control should, however, be thoroughly familiar with [them]."* (Italics added.) How true! In my experience, regrettably, people who *should* be thoroughly familiar with these operations are all too often not so.

The operations Codd defines are *permutation, projection, join, tie,* and *composition* (the 1970 paper added *restriction,* which I'll cover here for convenience). It's interesting to note that the definitions for *restriction* and *join* are rather different from those usually given today and that two of the operations, *tie* and *composition,* are now rarely considered.

Throughout what follows, the symbols *X, Y,* ... (and so on) denote either individual attributes or attribute combinations, as necessary. Also, I'll treat the definition of *join* at the end, for reasons that will become clear in a moment.

**Permutation.** Reorder the attributes of a relation, left to right. (As I noted last month, relations in the 1969 paper had a left-to-right ordering to their attributes. By contrast, the 1970 paper states that permutation is intended purely for internal use because the left-to-right ordering of attributes is -- or should be -- irrelevant so far as the user is concerned.)

**Projection.** More or less as understood today (although the syntax is different; in what follows, I'll use the syntax R{X} to denote the projection of R over X). Note: The name "projection" derives from the fact that a relation of degree n can be regarded as representing points in n-dimensional space, and projecting that relation over m of its attributes *($m \leq n$)* can be seen as projecting those points on to the corresponding m axes.

**Tie.** Given a relation *A{X1,X2,...,Xn},* the *tie* of *A* is the restriction of A to just those rows in which *A.Xn = A.X1* (using "restriction" in its modern sense, not in the special sense defined below).

**Composition.** Given relations *A{X,Y}* and *B{Y,Z},* the composition of *A* with *B* is the projection on *X* and *Z* of a join of *A* with *B* (the reason I say "a" join, not "the" join, is explained below). *Note:* The *natural* composition is the projection on *X* and *Z* of the *natural* join.

**Restriction.** Given relations *A{X,Y}* and *B{Y},* the restriction of *A* by *B* is defined to be the maximal subset of *A* such that *A{Y}* is a subset -- not necessarily a proper subset -- of *B.*

Codd also says "all of the usual set operations are [also] applicable ... [but] the result may not be a relation." In other words, definitions of the specifically relational versions of Cartesian product, union, intersection, and difference still lay in the future when Codd was writing his 1969 paper.

Now let's get to the definition of *join.* Given relations *A{X,Y}* and *B{Y,Z},* the paper defines a join of *A* with *B* to be any relation *C{X,Y,Z}* such that *C{X,Y} = A* and *C{Y,Z} = B.* Note, therefore, that *A* and *B* can be joined (or "are joinable") only if their projections on *Y* are identical -- that is, only if *A{Y} = B{Y},* a condition one might have thought unlikely to be satisfied in practice. Also note that if *A* and *B* are joinable,

then many different joins can exist (in general). The well known natural join -- called the linear natural join in the paper, in order to distinguish it from another kind called a cyclic join -- is an important special case, but it's not the only possibility.

Oddly, however, the definition given in the paper for the natural join operation doesn't require *A* and *B* to be joinable in the foregoing special sense! In fact, that definition is more or less the same as the one we use today.

Let me try to explain where that rather restrictive "joinability" notion comes from. Codd begins his discussion of joins by asking the important question: Under what circumstances does the join of two relations preserve all the information in those two relations? And he shows that the property of "joinability" is sufficient to ensure that all information is thus preserved (because no row of either operand is lost in the join). Further, he also shows that if *A* and *B* are "joinable" and either *A.X* is functionally dependent on *A.Y* or *B.Z* is functionally dependent on *B.Y,* then the natural join is the only join possible (though he doesn't actually use the functional dependence terminology -- that also lay in the future). In other words, what Codd is doing here is laying some groundwork for the all-important theory of *nonloss decomposition* (which, of course, he elaborated on in subsequent papers).

Remarkably, Codd also gives an example that shows he was aware back in 1969 of the fact that some relations can't be nonloss-decomposed into two projections but can be nonloss-decomposed into three! This example was apparently overlooked by most of the paper's original readers; at any rate, it seemed to come as a surprise to the research community when that same fact was rediscovered several years later. Indeed, it was that rediscovery that led to Ronald Fagin's invention of the "ultimate" normal form, 5NF, also known as projection-join normal form (PJNF).

## Expressible, Named, and Stored Relations

According to Codd, three collections of relations are associated with a data bank: *expressible, named,* and *stored* sets. An *expressible* relation can be designated by means of an expression of the data access language (which is assumed to support the operations described in the previous section); a *named* relation has a user-known name; and a *stored* relation is directly represented in physical storage somehow.

I do have a small complaint here (with 20/20 hindsight, once again): It seems a little unfortunate that Codd used the term *stored relation* the way he did. Personally, I would have divided the expressible relations into two kinds, *base* relations and *derivable* ones; I would have defined a derivable relation to be an expressible one the value of which, at all times, is derived according to some relational expression from other expressible relations, and a base relation to be an expressible relation that's not derivable in this sense. In other words, the base relations are the "given" ones; the derivable ones are the rest. And then I would have made it very clear that base and stored relations are not necessarily the same thing. (See Figure 1.) As it is, the paper effectively suggests that base and stored relations *are* the same thing (basically because it doesn't even bother to mention base relations as a separate category at all).

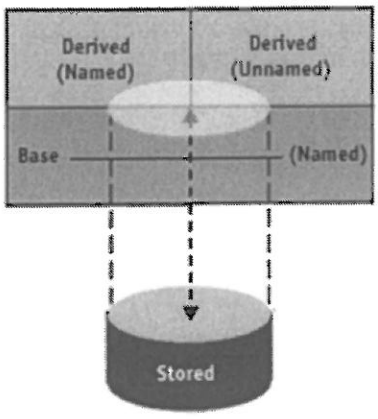


**Figure 1.** Kinds of relations.

It's true that base relations are essentially the same as stored relations in most SQL products today. In other words, most people think of base relations as mapping very directly to physical storage in those products. But there's no logical requirement for that mapping to be so simple; indeed, the distinction between model and implementation dictates that we say nothing about physical storage at all in the model. To be more specific, the degree of variation allowed between base and stored relations should be at least as great as that allowed between views and base relations; the only logical requirement is that it must be possible to obtain the base relations somehow from those that are physically stored (and then the derivable ones can be obtained, too).

As I already indicated, however, most products today provide very little support for this idea; that is, most products today provide much less data independence than relational technology is theoretically capable of. And this fact is precisely why we run into the notorious denormalization issue. Of course, denormalization is sometimes necessary (for performance reasons), but *it should be done at the physical storage level, not at the logical or base relation level.* Because most systems today essentially equate stored and base relations, however, there is much confusion over this simple point; furthermore, denormalization usually has the side effect of corrupting an otherwise clean logical design, with well-known undesirable consequences.

Enough of this griping. Codd goes on to say, "If the traffic on some unnamed but expressible relation grows to significant proportions, then it should be given a name and thereby included in the named set." In other words, make it a view! So Codd was already talking about the idea of views as "canned queries" back in 1969.

"Decisions regarding which relations belong in the named set are based ... on the logical needs of the community of users, and particularly on the ever-increasing investment in programs using relations by name as a result of past membership ... in the named set." Here Codd is saying that views are the mechanism for providing *logical data independence* -- in particular, the mechanism for ensuring that old programs continue to run as the database evolves. And he continues, "On the other hand, decisions regarding which relations belong in the stored set are based ... on ... performance requirements ... and changes that take place in these [requirements]." Here Codd is drawing a very *sharp* distinction between the logical and physical levels.

## Derivability, Redundancy, and Consistency

In this section, Codd begins to address some of the issues that later came to be included in the integrity part of the relational model. A relation is said to be *derivable* if and only if it's *expressible* in the sense of the previous section. (Note that this definition of derivability is not quite the same as the one I was advocating above because -- at least tacitly -- it includes the base relations.) A set of relations is then said to be *strongly redundant* if it includes at least one relation that's derivable (in Codd's sense) from other relations in the set.

The 1970 paper refines this definition slightly, as follows: A set of relations is *strongly redundant* if it includes at least one relation that has a projection -- possibly the *identity* projection, meaning the projection over all attributes -- that's derivable from other relations in the set. (I've taken the liberty of simplifying Codd's definition somewhat, although, of course, I've tried to preserve his original intent.)

Codd then observes that the *named* relations probably will be strongly redundant in this sense, because they'll probably include both base relations and views derived from those base relations. (What the paper actually says is that "[such redundancy] may be employed to improve accessibility of certain kinds of information which happen to be in great demand;" this is one way of saying that views are a useful shorthand.) However, the *stored* relations will usually not be strongly redundant. Codd elaborates on this point in the 1970 paper: "If ... strong redundancies in the named set are directly reflected ... in the stored set ... then, generally speaking, extra storage space and update time are consumed [though there might also be] a drop in query time for some queries and in load on the central processing units."

Personally, I would have said the *base* relations should definitely not be strongly redundant, but the *stored* ones might be (depending -- as always at the storage level -- on performance considerations).

Codd says that, given a set of relations, the system should be informed of any redundancies that apply to that set, so that it can enforce *consistency;* the set will be consistent if it conforms to the stated redundancies. I should point out, however, that this definition of consistency certainly doesn't capture all aspects of integrity, nor does the concept of strong redundancy capture all possible kinds of redundancy. As a simple counterexample, consider a database containing just one relation, for example `EMP { EMP#, DEMP#, BUDGET }`, in which the following functional dependencies are satisfied:

```
EMP# -> DEPT#
DEPT# -> BUDGET
```

This database certainly involves some redundancy, but it isn't "strongly" redundant according to the definition.

I should explain why Codd uses the term *strong* redundancy. He does so to distinguish it from another kind, also defined in the paper, which he calls *weak* redundancy. I omit the details here, however, because unlike just about every other concept introduced in the first two papers, this particular notion doesn't seem to have led to anything very significant. (In any case, the example given in the paper doesn't even conform to Codd's own definition.) The interested reader is referred to the original paper for the specifics.

## Data Bank Control

This, the final section of the 1969 paper, offers a few remarks on protocols for what to do if inconsistencies are discovered. "The generation of an inconsistency ... could be logged internally, so that if it were not remedied within some reasonable time ... the system could notify the security officer [sic]. Alternatively, the system could [inform the user] that such and such relations now need to be changed to restore consistency ... Ideally, [different system actions] should be possible ... for different subcollections of

relations."

This concludes my discussion of the 1969 paper. Next time, I'll turn my attention to that paper's more famous successor, the 1970 paper.

---