

Keys in Relational Databases: Part 10.

Dr. Tom Johnston
MindfulData.com

Arguments Against Non-Intelligent Keys.

I have found no arguments against this quite radical proposal of using enterprise identifiers in relational data models. Perhaps that is because the proposal has not been made, at least in this explicit a form, until now! {11/2007. “Now”, when originally written, meant “in 2001”.} But I have frequently found arguments against using keys with no intelligence in them. And since EIDs must have no intelligence in them, *i.e.* which must contain values which do not represent anything about the entities they identify, these arguments against non-intelligent keys are, *ipso facto*, arguments against EIDs. To these arguments, which are indirectly arguments against EIDs, I now turn.

In the discussion below, recall that the “natural key” for a table is the attribute, or combination of them, by which business users pick out a unique row of a table, used as the primary key. Those same columns, if not used as the primary key, we call the “business key” of the table.

Argument 1: EIDs Consume Extra Space and Time.

One argument against surrogate keys in general (and thus against EIDs in particular), is based on the fact that such keys unnecessarily consume either space or time. Whichever it is, it is unnecessary, so the argument goes, because we have natural keys which can serve as primary and foreign keys.

Take space--the more straightforward case. A surrogate key will be some number of bytes long. Tables with a large number of rows will require a correspondingly large key, in order to contain unique values. Although DASD space is perceived as inexpensive, it isn't free; and in very large databases, the incremental amount of space needed to support a surrogate key in every table can be considerable.

Take time--the more interesting case. In queries or code (where we can think of "code" as implementing queries which are too complex for users to create by themselves), individual rows of tables will not be specified by means of surrogate keys. Instead, these directly or indirectly user-driven accesses to the database will be driven by business keys. Now, these accesses can be quite frequent. And every time they occur, some kind of lookup will be required to map from the business key to a surrogate key.

Update transactions illustrate the point. Every update to an individual row of a table is originally specified in terms of a natural key. The only alternative would be to require the user to know the surrogate key, and use it. But this alternative is a bad one, because it pushes the semantic costs of surrogate keys onto the user herself. Much better to keep that cost contained within the systems which manage the databases.

Since updates, therefore, are originally specified in terms of natural keys, at some point a translation will be required. The natural key index to the table will have to be accessed, and a translation made to the surrogate key. Only then can the row be accessed and updated. The extra, and unnecessary, cost is apparent.

Argument 2: EIDs Require Extra Joins.

A second argument is that natural keys often make joins unnecessary. They do so when the information from the related table that is needed is contained in the natural key of that table itself. Since the natural key is the primary key, each natural key value is also contained in those tables which are foreign-key related to it. This makes a physical join unnecessary when those business key values are the whole reason for doing a join in the first place.

For example, codes are often abbreviations. Those abbreviations themselves are the natural keys of code tables. Since those natural keys are used as primary keys, the code abbreviations themselves will appear in every foreign-key related table. For example, if the code abbreviation "GA" is a primary key of a table of state codes, then the value "GA" will appear as a foreign key in every table which references it. If, on the other hand, a surrogate key is used as the primary key of code tables--for example, the

value "9932" for state code "GA"--then a physical join will be required to obtain the abbreviation "GA" itself.

Argument 3: EIDs Interfere With Intelligent Clustering.

A third argument is this. If we cluster on a surrogate primary key, we are clustering into a sequence which doesn't correspond to anything meaningful to the user. Therefore, we won't be able to gain the advantage of clustering rows close to one another which correspond to sets of rows the user is likely to specify in queries. Instead of all invoices for a customer being clustered, or all invoices for an accounting period, or sales region, or whatever you will, those invoices are guaranteed to be clustered into no meaningful sequence at all--because the surrogate key on which their clustering is based has no business meaning.

Argument 4: Migration Costs to EIDs are Prohibitive.

A fourth argument against surrogate keys is this. If surrogate keys are introduced into an enterprise, they will be introduced one physical database at a time. It simply isn't practical to coordinate the simultaneous switch-over to a set of multiple physical databases, that set being the set of all the databases in which the table or tables being converted to surrogate keys occur.

So given that unintelligent keys for one or more tables are introduced gradually, there will be a period of time during which communicating databases will have to match natural keys to surrogate keys. This matching, whether done as a single on-line transaction, or as massively multiple occurrences of batch update processes, is extra overhead. If the processes involve, let us say, millions of updates each month, the overhead can be significant. And it will exist until the surrogate key has propagated across all the physical databases in the enterprise.

In response, let me deal with each argument in turn.

Response 1: EIDs Usually Save Space.

To begin with, it is true that surrogate keys take space, and that we could, in most cases, use a natural key as the primary key instead. But the comparison here is not between (a) the surrogate key option which carries the indicated cost, and (b) a natural key option which is cost-free. For using a natural key may *not* save space because, while a natural key clearly does eliminate the extra space which a surrogate primary key would otherwise consume, that savings may be eaten up by its foreign key occurrences. If the natural key is much larger than a surrogate key would be, then the amount of space consumed by its foreign key occurrences, compared to the amount of space needed for the foreign keys of a smaller surrogate key, might be considerable.

Response 2: EIDs Eliminate the Cost of Changing Natural Keys.

But the most important reason for using EIDs is that it eliminates the risk taken when natural keys are used. That is the risk that the business community will want, at some point in time, to change that natural key. For example, if the business has allocated one character in a natural key to indicate a generic category of equipment, and the following character to indicate a subtype within each generic category, how much do you want to bet that they won't eventually run out of space for subcategories of one or more generic categories, and demand that we Data Administrators change the entire key structure to give *two* characters to the task of indicating subtype?

And “how much do you want to bet” is no mere colloquialism. For when users demand such changes, IT will have to point out that these key values have been propagated throughout any number of current databases, and also back in time across any number of historical copies of those databases--not just as primary keys, but as often far more numerous foreign keys. The cost of tracking down all those values and changing them is usually immense.

The cost of changing only current databases, and living with the fact that rows of that table now have one identifier for their historical copies, and a different one from that point in time on, is the alternative. On this alternative, either all queries which might range *across* the point in time that

the key values changed will have to be changed to read “where value = X or Y”, instead of “where value = X”, or else the problem will be swept under the rug. {11/2007. I’ve got some interesting examples of sweeping semantics under the rug. I’ll try to dig them out, because they illustrate why and how IT can simply not fix things, and yet business processes continue to function. Simply put, when IT systems fail to track semantics, the task falls on the business user.} None of these alternatives, I suggest, is very attractive.

Response 3: Cluster on the Business Key.

Next, let’s consider the clustering objection. The answer, which may well be obvious right off the bat, is to agree that we shouldn’t cluster on a surrogate key, and for just the reasons given. Instead, we should cluster on either the business key or on whatever else will optimize the most costly queries.

Response 4: Avoiding Extra Joins.

Next, let’s consider the objection that joins can be avoided when the foreign key values contain information that we would otherwise have to use the join to obtain. The response is that, if it is important to avoid these extra joins, then do what we always do to avoid joins: denormalize. As we argued above, foreign keys are a DBMS-controlled form of denormalization. If a natural key value changes, the DBMS will cascade update that change to all foreign key occurrences. The only difference here is that we have to cascade update to all denormalized occurrences with our own code. Implemented as a database trigger – thus implemented once, no matter how many potential update sources there are – the risk of error is minimal.

Response 5: Migration Costs are Not Unique to Surrogate Keys.

Finally, let us consider the last objection--that until a surrogate key has replaced its natural key predecessor in all affected databases, logic to match surrogate and natural keys will be required.

This, of course, is true. But converting from one scheme to another always involves a cost, whether or not surrogate/EIDs are involved. What we need

to keep in mind is that the benefit of eternally stable keys--especially when those keys have the added "goodness" of EIDs--carries business value far beyond any realistic cost of machine resources or even the human resources required to design, implement and manage the conversion processes. The answer, then, is to just get on with it!

Conclusion.

Our understanding of our enterprise's data changes over time. We gain additional perspectives, see things in a new light, and deepen our understanding of the information which that data attempts to faithfully record. And in all but the most trivial cases, a deeper understanding of the information our enterprise needs results in changes in how we think our data should be structured into entities and relationships among entities. Entities grow or shrink in scope, as we deepen our understanding of what they really represent. They merge, or partially merge. They dissolve, and their instances are subsumed under other entities. They are gathered together under the shelter of a supertype, or leave to see if they can stand on their own. These evolutionary processes are recorded in changes to the enterprise's information model--provided such a model is judged worth maintaining, even though, if it is not based on EIDs, the enterprise's physical databases will diverge more and more from it.

It is neither accident nor whim that this language is similar to the language of Darwinian evolution. The evolution of data structures may not often have been described in terms traditionally applicable to the evolution of biological species, but I claim that the language, applied to data structures, is not just a metaphor. This simply *is* what happens to a logical data model of an enterprise--to data structures which are not prevented from evolving by management fiat or by the early binding of semantics to syntax.

But our earlier perspectives also led us to create *physical* databases with specific structures--structures that reflected those earlier perspectives. And while it is comparatively easy to incorporate into our *logical* models the deeper understanding of the information of which we are supposedly the stewards, it is far more difficult to restructure those *physical* databases. They remain recalcitrant to change; and existing for years or even decades, they

impose their outdated parochial perspectives on new generations of users, propagating fundamental misunderstandings, hiding critical information in misshapen data structures from which only a handful of experienced users can extract meaningful information, and even requiring new physical databases to avoid data structures which would implement a clearer understanding of the data--all for the sake of compatibility with these ancient semantic species which should long ago have been allowed to die a natural death.

I identified the root cause of this recalcitrance to change on the part of our physical databases as an issue with identifiers and with their use in relational models to implement inter-entity relationships, *i.e.* as an issue with primary keys and their foreign key doppelgangers. On the basis of that analysis, I urged that all keys, in a relational model, be subject to two constraints--first that they be system-generated so as to contain no “intelligence”, and secondly that they be based on a common data structure and domain, across all entities in the enterprise, and that each row of each table have a unique key value across all entities in the enterprise. These two constraints can be conveniently labeled the “no intelligence” constraint and the “common structure and unique value” constraint, respectively. Keys which satisfy these two constraints, I have called EIDs--enterprise identifiers.

I claim that the arguments for EIDs presented here, and in Part 2, demonstrate that the deployment of EIDs in an enterprise’s physical databases is both necessary and sufficient to permit the evolution of those databases at an acceptable cost. And the expression “acceptable cost” is not a fudge. I mean it as a shorthand for “the cost of modifying a physical database’s primary keys, and all their foreign key occurrences, everywhere they occur--which includes other tables in the same database, tables in other databases, in logs, archives, data warehouses, data marts, and so on”. It is quite clear, without quantification, that the cost of this kind of key-based modification is *orders of magnitude* higher than the cost of any changes to non-key data. And these orders of magnitude are what make the distinction between “acceptable cost” and “unacceptable cost” as clear as needs be, and no fudge at all.

To repeat: EIDs are both necessary and sufficient to enable database evolution at an acceptable cost. They are necessary: nothing else will do, and nothing short of them will do. For example, just removing intelligence from identifiers will not do. Using traditional surrogate keys, whose values are unique only within a single table, will not do.

And they are sufficient: by making it possible to move rows from one table to another--to say that they are not instances of *this* kind of thing, but rather of *that* kind of thing--and by making it possible to do so without requiring a change of key, they go right to the root of the problem, and eliminate it at one stroke. They make database evolution, no matter how radical, something that can occur at an acceptable, indeed often a negligible, cost.

We have now seen why primary keys (and hence foreign keys as well) should never be based on business data. We have also seen why the system-generated values that should be used instead should make a row unique across an entire set of tables.

In the examples used, our EIDs have all been single-column primary keys. But often, primary keys include two or more columns, one or more of which is a foreign key. But as long as we have any tables in our namespace that contain no foreign keys, their primary keys will be single-column keys. This violates the requirement that the syntax of all primary keys within a namespace be identical. But since the use of foreign keys in primary keys is widespread, I want to examine this particular modeling practice in greater detail. Perhaps it is so useful that we should drop our requirement of a single syntax for all primary keys.