

Keys in Relational Databases: Theory and Practice. Part 1.

Dr. Tom Johnston
MindfulData.com

Key Problem #1. Primary Keys Which Also Describe.

The first problem I will describe is what goes wrong when *identifying* and *describing* are implemented in the same syntactic structure, i.e. when one “piece” of the database both identifies and describes. This piece, this syntactic structure, is the primary key. In both identifying and describing, primary keys play two semantic roles. In doing so, they are homonyms, just as much as the one word “bank” is a homonym representing either of two meanings – “the sloping sides of a river” and “a place to deposit and withdraw money”.

Business data can indeed be used not just to describe, but also to both identify and relate. This means that relational DBMSs don’t need the cumbersome internal pointers of earlier database management systems like IMS, IDMS, TOTAL and other pre-relational DBMSs. Neither is the linking mechanism for relational databases part of the DBMS “machinery”. It too provided by the business data itself. That linking mechanism is the foreign key.

Business data is used to distinguish every row in a relational table from every other row. The set of one or more columns, or one of several uniquely identifying sets of columns, is chosen as the primary key of the table. Being a unique identifier of a row, its appearance in any other row of the same table, as a primary key, is forbidden.

However, its appearance in a separate table, whether or not as part of the primary key of that table, does relate the row for which it is a primary key to the row in which it appears as a foreign key. Also, its appearance in the *same* table, but not as all or part of the primary key, relates those two rows of that single table. Finally, its appearance in a third table whose primary key was the pair of keys for the related rows, also relates those two rows.

This appearance of a unique identifier elsewhere in the database is called a “foreign key”. By these means (“means”, in the plural; for note that, as we

have just seen, the term “foreign key” refers to several different implementation mechanisms), the functions of uniquely identifying a row in a table, and of relating that row to any number of other rows, can be implemented without resort to complicated DBMS pointer mechanisms.¹ By these means, related data can be gathered together using set-at-a-time operations.

Since (non-surrogate) primary keys are made up of business data, it follows that, when what that data describes changes, the primary keys must change also. But the values in these primary keys may exist in any number of other tables (or even in other rows and columns of the same table), as foreign keys. So the change in one value must be propagated to any number of rows, in any number of tables. As we shall see, this can become an extremely expensive process.

This is a basic observation. In brief:

Business data describes.
When what it describes changes, the data must change.
But changing primary keys is expensive because it requires changing foreign keys.
It is also unnecessary.

Figure 3. The Problem with using Business Data in Primary Keys.

When business data is used in primary keys, we have what practitioners call an “intelligent key”. So the way to avoid key changes is to avoid intelligent keys. More precisely, the solution is to keep intelligent keys (which I will

¹ “Function” is being used here in the non-mathematical sense of “role” or “purpose”. It is not to be confused with mathematical functions, the ones alluded to in the phrase “functional dependency”.

call “semantic keys”), but not to use them as primary keys (which I will call “syntactic keys”). Practitioners have been doing this for at least a decade. We call it using “surrogate keys”.

I will make three recommendations for improving on this “best practice”:

1. Take it seriously. If it’s good to use surrogate keys for some tables, it’s better to use them for all tables. If it’s good to use them for an Invoice Header table, it’s also good to use them for the related “child” table – the Invoice Line Item table. If it’s good to use them for a Salesperson and a Customer table, it’s also good to use them for the associative table between them.
2. Entity integrity is a good idea. But if having keys which are unique within one table is a good idea, having keys which are unique across all tables is a better idea. For reasons I will explain later, I call this “object integrity”. And having keys which are unique across an entire “namespace” of multiple databases is the best idea of all.

Some practitioners may suspect that I am talking about GUIDs – globally unique identifiers. And I am. I’ll describe why they are important for relational databases, not just for object-oriented systems. And I’ll describe a special kind of GUID which I recommend for use with relational databases.
3. Specific rules must be followed to create well-formed semantic keys, and to avoid any cross-over from semantic onto syntactic keys. I will discuss these rules later on.

Figure 4. Three Recommendations for Improving Surrogate Keys.

{11/28/07. Key Problem #1 discussed in this essay. The remaining key problems will be discussed in Part 2.}