

# Keys in Relational Databases: Theory and Practice.

Dr. Tom Johnston  
MindfulData.com

{11/2007. Most of this material written between 2001 and 2005, and is for the most part adapted from articles published in DM Direct and DataWarehouse.com over that period of time.

On reviewing this material, I notice that several themes are discussed three or four times, in separate places. This reflects the state of this document – cobbled together, some work on integrating the pieces into a well-structured essay, but still a lot of that sort of work to do.

On the other hand, expressing the same ideas in different words may actually be helpful. So I'm publishing this on my website, in accordance with my guideline that if something looks like it might be of value to readers, then I'll include it.}

## Introduction.

Two main features of relational theory are:

1. The application of the mathematics of sets, functions and relations to data structures, and of predicate logic to manipulations of those structures; and
2. The implementation of identifying and relating by means of primary keys and foreign keys, and the use of business data (data that describes something of interest to the business) in those keys.

**Figure 1. Two Main Features of Relational Theory.**

The first feature was Dr. E. F. Codd's enduring contribution to the science of managing data. The second was an enduring *mistake* which has cost IT departments literally billions of dollars in unnecessary database maintenance costs.

We can give this mistake various names, based on different perspectives we have on it. But for me, the most fundamental way to describe this mistake is with the language of binding. So, even if you are familiar with concepts like compile-time binding and run-time binding, I recommend that you read Appendix 1 before proceeding. *{11/2007. Somewhere on the website as a separate document on binding.}*

The root cause of the mistake was enshrined as Dr. Codd's Information Principle, which basically says that business data itself is all we need to identify, to describe and to relate: <sup>1</sup>

1. *to identify* by providing unique identifiers for rows within tables;
2. *to describe* features of the things our business is interested in (the basic job that all business data does); and
3. *to relate* by putting copies of unique identifiers in columns of rows we wish to relate to the rows whose identifiers they are.

**Figure 2. Three Semantic Functions in Databases: Identifying, Describing, and Relating.**

So if we follow the Information Principle, we do away with DBMS-specific pointers and other system-created and managed links among business data. Which is exactly what has happened with relational DBMS products.

---

<sup>1</sup> Insert reference.

At the time, this seemed like a good idea. As designers and architects, we are always looking for simplicity. It is a surer sign of goodness in our designs than anything else – simplicity, elegance, indeed beauty. So once Dr. Codd realized that the internal mechanisms of DBMS pointers were not needed, he proceeded to work out the mechanisms by which business data could take over the roles which those pointers had been used for. These were the dual roles of identifying and relating.

But no matter what else it does, all business data *describes*. That's what business data is, data which describes something of interest to the business. If this data is also used to identify, it is performing two functions. If it is also used to relate, it is performing a third function. In any case, data which performs more than one function is a homonym, in the strictest semantic sense.<sup>2</sup> Let's stop for a moment and see how.

Functions (in the informal sense used here, but also in the formal, mathematical sense) provide information just as much as descriptions do. For example, relating an invoice line item row back to its corresponding invoice header row provides information because, without that relationship, we obviously don't know something. Without that relationship, when we look at the invoice header, we don't know that that line item belongs to it. Without that relationship, when we look at the line item, we don't know which invoice it belongs to.

So: if all three of these functions provide information, then using the same syntactic structure to implement two or more of them means that the structure is a homonym. Specifically: just as using one word (one physical, syntactic, object) to represent two or more meanings (logical, semantic objects) creates the semantic anomaly we call a homonym, so too does using a primary key (a physical, syntactic object) to perform two or more functions (identifying, describing and relating) create a homonym.

---

<sup>2</sup> {11/2007. I'll later add material introducing the basic concepts of semantics like synonyms, polysemes, homonyms, antonyms, etc.}

This is not merely an analogy. To identify, to describe and to relate are functions, as we have just seen, that convey information. So they are therefore functions with semantic content. Implementing two or more of those functions in a single syntactic element is therefore to create a homonym.

As we shall see, homonyms are bad. As are synonyms. As is ambiguity and vagueness. These are, one and all, *semantic* anomalies, and we will continue to develop less than optimal data models for as long as we don't know how to recognize and avoid these anomalies.

This is what this series of articles is concerned with, to point out and describe the problems created by these functional homonyms, as they are manifested in primary and foreign keys, and to suggest ways around them.

This series of articles is a partitioning of the following outline. I estimate the series to be about 15 installments in length. Next time, we begin Part 1.

## 1. Introduction.

- 1.1.Key Problem #1. Primary Keys Which Also Describe.
- 1.2.Key Problem #2. Foreign Keys Which Also Identify.
- 1.3.Key Problem #3. Embedded and Non-Embedded Foreign Keys.

## 2. Key Problem #1: Primary Keys Which Also Describe.

- 2.1.Natural Keys, Intelligent Keys and Surrogate Keys.
- 2.2.Propagation of Primary Key Changes to Foreign Keys.
- 2.3.Intelligent Keys.
- 2.4.Unintelligent Keys.
- 2.5.An Easy Way Out?
- 2.6.Three Ways to Guarantee Object Integrity in Enterprise Keys.
- 2.7.Five Drawbacks to Traditional Primary Keys.
- 2.8.Arguments Against Non-Intelligent Keys.
- 2.9.Conclusion.

## 3. Key Problem #2: Foreign Keys Which Also Identify.

- 3.1.Item Catalogs, Standard Items and Identifying Relationships.
- 3.2.A Business Decision is Changed.

- 3.3. Adding Non-Standard Items to the Item Catalog.
  - 3.4. The Foreign Key Ripple Effect.
  - 3.5. The Same Change with Non-Identifying Relationships.
  - 3.6. Business Requirements and What Entities Mean.
- 4. Key Problem #3: Embedded and Non-Embedded Foreign Keys.
    - 4.1. How Relationship Semantics Are Early-Bound to Relationship Syntax.
    - 4.2. Three Ways Around the Problem.
    - 4.3. Use an Associative Table for Every Relationship.
    - 4.4. Use One Associative Table for All Relationships.
    - 4.5. Implement Relationships Without Using Foreign Keys.
    - 4.6. Using Foreign Keys: The Standard Method.
    - 4.7. The Basic Problem and the Basic Solution.
    - 4.8. A Case Study: Organizational Units, Projects and Employees.
    - 4.9. Three Business Tables.
      - 4.9.1. The Table of Business Relationships.
      - 4.9.2. Three New Properties of Relationships.
      - 4.9.3. One Associative Table.
    - 4.10. Relational Databases Without Foreign Keys.
    - 4.11. In Order to Form a More Perfect SQL.
      - 4.11.1. Native SQL with the Nonembedded Method.
    - 4.12. Synopsis.
      - 4.12.1. QBEs and Report Writers?
      - 4.12.2. A Mirror Database?
    - 4.13. Data Modeling Tools and the Nonembedded Approach.
    - 4.14. Conclusion.
- 5. Modeling With Syntactic Keys and Semantic Keys.
    - 5.1. Surrogate Keys and Semantic Keys.
    - 5.2. Surrogate Keys.
    - 5.3. Surrogate Keys and Entity Integrity.
    - 5.4. Semantic Keys.
    - 5.5. Constraints On Semantic Keys.
      - 5.5.1. Constraint #1: All Semantic Key Attributes Must Have Business Meaning.
      - 5.5.2. Constraint #2: The Semantic Key Can't be Part of the Primary Key.

- 5.5.3. Constraint #3: All Semantic Key Attributes Must be Necessary to Create a Unique Identifier.
  - 5.5.4. Constraint #4: The Semantic Key Can't be Part of the Business Key.
  - 5.5.5. Constraint #5: No Semantic Key Attribute Can be Nullable.
  - 5.6. Can Primary Keys Enforce Both Syntactic and Semantic Entity Integrity?
  - 5.7. Summary.
  - 5.8. Conclusion.
6. Appendices.
- 6.1. Appendix 1. Binding.
    - 6.1.1. Binding Values to Variables.
    - 6.1.2. Binding Semantics to Syntax.
  - 6.2. Appendix 2. A Refinement of the Rule That all Tables Must Contain Business Keys.