

## DAMA2008-Johnston. Speaker Notes.

Dr. Tom Johnston.

MindfulData.com

### *Slide 1.*

- an independent consultant specializing in data modeling, ontologies and data architecture.
- contact information is on this first slide.
- business cards here at the podium.
- topics: surrogate keys.
- Why this matters to the business: total cost of ownership.
- articles on the topic: in the archives at DMReview.com.
- will post my lecture notes for this presentation on my website.

### *Slide 2.*

- standard practice: surrogate keys ok for *some* tables.
- my proposal: surrogate keys advisable for *all* tables.
- a basic argument from practice: total cost of ownership.
- a basic argument from theory: BK PKs are homonyms.
- the implications for Codd's Information Principle: RDBMSs can enforce entity and referential integrity if but only if PKs are BKs.
  - Too costly.
  - Theoretically wrong (semantics).

- The relational model was a great step forward. But there is still a long way to go.

### *Slide 3.*

- Note: identifying a row vs. identifying the object the row represents.
- The row is the data. Its only value is in providing information about the object it represents.

### *Slide 4.*

- five types of data. Data that (1) identifies the row, (2) describes the row, (3) identifies the object, (4) describes the object, and (5) relates the row to other rows.
- Note again: row vs. the object, what the RDBMS manipulates vs. what that row represents, DBMS "mechanics" vs. the semantics that makes data meaningful.
- Aside: "semantics". A definition and commentary at [MindfulData.com](http://MindfulData.com).

### *Slide 5.*

- yellow tables: data models are built out from these important concepts by adding on dependent (green) and associative (blue) tables.
- Standard practice: if surrogate keys are used at all, use them for the yellow tables.
- My claim: all three types of tables should use surrogate keys.
- EID: a special kind of surrogate key. Not argued for here. See my "Primary Key Reengineering Projects" articles.

### *Slide 6.*

- BKs will change. Reason: they describe.
- BKs that are PKs: values must change in many tables because of:
  - Direct FKs.
  - Indirect FKs (via identifying relationships).
  - Pointers (Same values as BK, but not managed as FKs.)
    - In other databases in your enterprise.
    - In other databases – customers, regulators, partners.
- Usually can't be done as an atomic transaction. So: need for a "key pair" file/table.
- What are the chances that all queries will be changed to use it? Slim to none.

### *Slide 7.*

- Even more difficult: BK PK data type or length change.
- So can't split work into one atomic transaction per each value changed.
- Instead, one atomic transaction for changing the data type in all affected tables.
- Thought experiment. High level project plan. If it's less than several man-years of effort, spread over a calendar year or more of time, be glad that you are dealing with so simple a situation.

### *Slide 8.*

Point and talk.

## *Slide 9.*

Point and talk.

## *Slide 10.*

- What trouble? The DBMS can and obviously will insure that SK PKs are unique across the table they occur in. But here again the distinction between rows and what they represent crops up.
- Suppose it's a table of customers. Guaranteeing entity integrity when SK PKs are used is just guaranteeing that there are no duplicate *rows*. It doesn't guarantee that there are no duplicate *customers*.
- But a unique index on the BK *will* guarantee that there are no duplicate customers. And if our customer data is not clean enough to allow us to enforce BK uniqueness, we can always create a *non-unique* index on the BK with the intent that, some day when we have cleaned up the data, we can convert it to being a unique index.
- This isn't really trouble. It's more like a benefit. It permits us to manage dirty data where the dirt has gotten into the BKs themselves, and to do so with a table schema that does not have to be changed when we eventually clean up that data enough that we can require those BKs to be unique.

## *Slide 11.*

- But we don't have the luxury of starting over, from scratch. So:
- get the SKs on all the tables they should be on.
- then to alter those table schemas to stop using the BKs as PKs, and to start using the SKs instead. [POINT]
- Easy to say, of course!

- Indeed, this is still so difficult that only a very large savings in total cost of ownership could justify it.
- The savings is there. But it's not recognized.
- But we can begin to "sneak it in", as part of more easily funded database changes.

### ***Slide 12.***

- Denormalized data is problematic because it creates synonyms.
- BK PKs are problematic because they create homonyms.
  - BK PKs, and BK FKs, each play two roles.

### ***Slide 13.***

- It is well-known that the relational model does not support bags.
- It is also well-known, at least among data management professionals, that a lot of data that must be managed is dirty data.
- When it is *keys* that are dirty, what are we supposed to do? Tell the business that relational theory doesn't permit such tables?
- But what now of relational entity integrity? What of referential integrity? Now they have nothing to do with semantics. Now they are nothing to write home about. They are just a mechanical thing that RDBMSs need to find and relate data.

### ***Slide 14.***

- Lack of support for inheritance is a symptom of a larger problem, the lack of support for the full semantics of relationships.
- And this in turn reflects the focus of all databases (not just relational ones) on instance data.

- Some of us have heard about ontologies. Their introduction into business IT is happening because they promise to help us manage unstructured data. But most fundamentally, ontologies help us build software that can reason about types of things.
- Relational databases don't do that very well. But help is on the way.

### *Slide 15.*

- Mathematical operations transform mathematical objects. Codd's great insight: how to map our data onto those objects, and our manipulations of that data onto those transformations.
- So how will we progress beyond the relational model? By defining new mathematical objects and new mathematical operations, and showing us how to map our data onto those new objects, and how to manipulate that data with those new operations.
- Instances and types. In relational databases, our catalog defines the types, and our DML populates those types with instances, and answers queries about those instances. It is instances that are queried, instances that are uniquely identified, instances that are related.
- But aren't types important, too? Shouldn't we be able to ask things like "what types of customer receive invoices, and what types don't?"

### *Slide 16.*

- The most important of those new objects: propositions and predicates of formal logic.
- Rules for making those objects are the rules which define "well-formed formulas" of logic.
- Rules for constructing "proofs" in logic are the rules for transforming them. Proofs in logic are what correspond to result sets in SQL.

### *Slide 17.*

- So why do we need a controlled vocabulary?
- For people. First, definitions based on a controlled vocabulary are more precise, freer of the inherent ambiguities of natural language. We know more clearly what we each mean when we talk in these terms.
- For inference engines. Definitions based on a controlled vocabulary can be translated into a formal system of logic, e.g. a definition of "customer".
- The controlled vocabulary is the set of predicates.
- The data dictionary and other definitions are statements in FOPL using those predicates.
- The set of predicates and definitions are an ontology of the key concepts that run your business.

### *Slide 18.*

Point and talk.

### *Slide 19.*

- Two rows for the same object. That's a pair of synonyms, at the row level. It's also a table waiting to be de-dupped.
- One row for two objects. That's a homonym, at the row level. That's a "file" waiting to be split.
- But using business data to avoid row-level synonyms and homonyms is what leads to the orders of magnitude total cost of ownership increases I described earlier. Any primary key that is not a surrogate key is one of these PKRP disasters waiting to happen.

### ***Slide 20.***

- Advocates of the relational model remind us of how semantically expressive that model is, compared to pre-relational ways of structuring and maintaining data. Entity integrity avoids row-level synonyms. Referential integrity insures that what we refer to, in one row, really exists in another one.
- But these semantic benefits really are provided by the relational model, provided that we use BK PKs!
  - But now we are caught in a conundrum. We can achieve the semantic enforcement benefits of entity integrity and RI only if we violate semantic rules by using PKs which are functional homonyms, which identify but which also describe.

### ***Slide 21.***

### ***Slide 22.***

### ***Slide 23.***

### ***Slide 24.***

### ***Slide 25.***

- There are two kinds of databases: those that have undergone at least one primary key reengineering project (PKRP), and those that haven't yet done so. (8>)
- Each PKRP will cost your company more than it cost to build the database in the first place. Some of that cost will be hidden – the cost

of semantic disintegrity that will occur despite best efforts otherwise. But IT dollar costs, and calendar time to completion of the PKRP, will not be hidden.

- Databases are the second most important asset managed by a CIO. But until the total cost of ownership of this asset is taken seriously, which means until it is measured, no one in IT will be recognized, no one in IT will be rewarded, for minimizing that total cost.
- What is lacking is financial accounting that would attribute those costs to the database, and thus an inability to measure the total cost of ownership of that asset. Until management can see the immense cost of the BK PK mistake, it will continue to be made. It's up to us to open their eyes.