

## Keys in Relational Databases: Part 9.

Dr. Tom Johnston  
MindfulData.com

### Subsetting and its Obstacles.

When two or more tables are related as supertype and subtype, it means that we have distinguished between a general description of something, and one or more specific descriptions of that same thing. But often, an enterprise comes to discover such relationships only after systems have been put into place based on a narrower perspective.

Consider, for example, a bank that from its earliest days distinguished between two kinds of customers--individuals and businesses. So fundamental has this distinction been to the bank that separate collections of databases evolved to deal with these distinct types of customers--databases managed by separate divisions of the bank.

Now, it would only have been by good fortune (or amazing foresight) that the divisions of this bank would have agreed on a common identifier for both kinds of customers. And so, in fact, what happened was that business customers were identified with a nine-character alphanumeric key, and individual customers with a twelve-digit numeric key.

Figures 44 and 45 show a few customers of each type. It shows them with natural keys, *i.e.* primary keys based on one or more business attributes. Another term for such natural keys is "intelligent keys", because the values they contain provide intelligence to the user.

<b>Business Customer Identifier</b>	<b>Customer Name</b>	<b>Other Attributes</b>
FINNE3324	Wachovia, Atl branch 332	.....
OLGSW4301	Pitrelli Shell	.....
OLGSW8903	Smith & Sons Amoco	.....
TLCSE0031	British Telecom	.....
TLCSW8302	Nortel	.....
TXTMA1044	Canon Mills NC #45	.....
TXTMA6503	Foxglove	.....
.....		.....
.....		.....
.....		.....
.....		.....

*Figure 44. The Bank's Business Customer Table, with Natural Key Identifiers.*

Individual Customer Identifier	Customer Name	Other Attributes
10793-561-68	Sean Patrick	.....
10892-356-41	Michael Sullivan	.....
11325-293-08	Mary Smithburger	.....
18439-935-29	Susan Brownmiller	.....
23445-091-77	Sylvia Bruton	.....
44165-002-26	Cynthia Osgood	.....
84301-294-04	John Alexander	.....
.....		.....
.....		.....
.....		.....
.....		.....

**Figure 45. The Bank's Individual Customer Table, with Natural Key Identifiers.**

Of course, there always was a need to produce reports for management which were based on combining both kinds of customers. And so there are plenty of programs around that read in both of these Customer tables, and appropriately process the data.

But what's wrong with this is that there is no common *data structure*-- nothing that we can "build" once only, and gain the benefit of over and over again. Instead, each time the bank wants some new information which is based on looking at both sets of customers, it must write new code. Or, if these tables are truly tables in a relational database, and not still VSAM master files, it still must write SQL queries to access two very different tables. A simple count of total customers, for example, requires querying each table for its count, and then adding the two numbers together.

What we need to do is to move knowledge from *repeated private* forms of expression, to a *single public* form of expression. The knowledge I am referring to, in this case, is the knowledge that both these tables are "about",

at an appropriate level of generality, the same “thing”--in this case, customers.

The private, repeated form of expression of this knowledge is its expression in program code or queries. The public, single form of expression of this knowledge is its expression in a data structure, accessible to all who use the database. In this particular case, what we need to do is to generalize and create a supertype table, and then make the appropriate modifications to the two Customer tables. Let’s see how to do this.

The process of generalizing to create a supertype table, and then modifying the original tables so they become subtypes of this new table, involves the following steps:

1. Create one row in the Customer supertype table for each row in the Business Customer table, and also for each row in the Individual Customer table.
2. Move all attributes which the two tables have in common, into the supertype Customer table.
3. Add a “type code” to the supertype table, indicating for each of its rows, whether it is for a business customer or an individual customer.
4. In the Business Customer table, use as the key for each of its rows the key of the corresponding row in the Customer table. Do the same for the Individual Customer table.
5. Leave those attributes which are specific to business customers in the Business Customer table. Do the same for the Individual Customer table.

**Figure 46. Consolidating Independent Tables Into Type Hierarchies.**

The hard part is Step 4. The Customer table can't have both a ten-digit numeric key, and also a nine-character alphanumeric key! So to complete Step 4, we must change the key of either the Business Customer table or the Individual Customer table, or else change both to some common third key.

Now we have already seen, in our earlier discussion in this chapter, that we should not have intelligence in our keys, because of how difficult it then is to change them. They have to be changed in on-line operational versions of the databases, replicated copies, warehouse versions, archived versions. Moreover, *all* foreign key instances must be changed, and done so as an atomic transaction. This may well require updates to a set of databases distributed around the country or around the world, and very likely lacking distributed DBMS capabilities which could use two-phase commit protocols to manage the propagation of key changes to all foreign key instances. Indeed, some of the more extravagant claims of distributed DBMS vendors to the contrary, in the real world today, major corporations are still a long way from such “under the wraps” functionality.

So key changes across all foreign key instances must be atomic transactions. If they are not, *i.e.* if we make any of the data being changed visible before the transaction completes, we will be making an inconsistent database state visible. In other words, we will be providing incorrect--simply false--information.

But on the other hand, transactions like these often cannot be completed in a short-enough period of time to keep the relevant database table instances locked until the transaction is complete. For suppose we change customer number 123 to ABC. We have to change not only all denormalized copies of customer number 123 to ABC, but also all instances of 123 to ABC in all tables in which it is a foreign key. Being a customer number, at a bank, that's likely to be a lot of tables! And we are doing this across the entire set of the bank's mainframe and LAN server databases, in all branches, around the country and around the world!

In reality, it just won't happen. There's no way a Data Administrator can convince upper management that the cost would be worth the benefit. In fact, the experienced Data Administrator won't even suggest it. To do so

would be to lose credibility; to do so as often as such opportunities are recognized would be to make oneself appear to be an impractical “data nerd”, not worthy of any significant business responsibilities.

So, given the very different key structures of the Business Customer and Individual Customer tables, the Data Administrator who, through enterprise modeling, realizes that these are subtypes of a Customer supertype, can't do anything with this insight. The cost of changing one or both of the keys is just too high. And so the cost of recognizing the commonality between these two tables must continue to be borne many times, by each query or piece of code, instead of one time, in a supertype data structure.

However, it's important to understand that using identical key structures (data type and length) in two tables doesn't ensure that it will later be possible to represent them as subtypes of a supertype. It is a necessary condition, but not a sufficient one. To see why, let's suppose that the Business Customer table and the Individual Customer table were both set up with a sixteen-byte numeric key, and that in both systems, keys were assigned sequentially.

It's easy to see that there will be a great deal of duplication of key values across the two tables. If  $n$  is the count of customers in the smaller of the two tables, then there will be  $n$  pairs of duplicate keys between the two tables. The first ten Business Customers will have identifiers 1 ... 10, and so too for the first ten Individual Customers, and so on for the pairs of customers up to  $n$ . Therefore, when we go to create a supertype Customer table, and assign identifier #1 to the first business customer, what number will we then assign to the first individual customer? Obviously, it can't be 1. Nor can it be any number assigned to any other individual or business customer.

Now we're right back at our original dilemma--that we can't create a supertype table because we would have to change key values, in all instances in all databases, in order to do it. And this just won't be done, because of the very high immediate cost, and the extremely long and difficult-to-quantify payback.

So what's the solution? How *could* we have set up these two tables initially so we could now superset them? If we can answer this question, then we can

begin to implement the answer in our new databases, or in our re-engineered databases. For our new databases, we can avoid the problem outright. For our re-engineered databases, we can bundle the cost of changing key structures and/or values into the overall cost of re-engineering one or more databases--an effort which, by the fact that it is being undertaken, shows that it has been able to provide adequate cost-recovery justification for itself. Within such a project, the incremental cost of key changes will be minimal, but the benefits will be every bit as great as if the key changes were undertaken as a stand-alone effort.

**Subsetting and Its Implementation.**

To see how subsetting can work, given tables with a common key structure and unique key values, let’s begin by creating versions of the Business Customer and Individual Customer tables with just such keys. These are shown as Figures 47 and 48, below.

<i>business-customer-eid</i>	<i>customer-name</i>	<i>other-business-customer-attributes</i>
1045	Nortel	.....
3492	Foxglove	.....
5966	British Telecom	.....
8770	Smith & Sons Amoco	.....
9456	Canon Mills NC #45	.....
9707	Pitrelli Shell	.....
0590	Wachovia, Atl branch 332	.....
.....		.....
.....		.....
.....		.....
.....		.....

**Figure 47. The Bank’s Business Customer Table, With EIDs Instead of Natural Keys.**

<i>individual-customer- eid</i>	<i>customer-name</i>	<i>other-individual- customer-attributes</i>
7668	Sean Patrick	.....
5173	Susan Brownmiller	.....
7474	Cynthia Osgood	.....
0845	Sylvia Bruton	.....
3215	John Alexander	.....
7490	Mary Smithburger	.....
1502	Michael Sullivan	.....
.....		.....
.....		.....
.....		.....
.....		.....

**Figure 48. The Bank's Individual Customer Table, With EIDs Instead of Natural Keys.**

Note, first of all, that the key structure in both tables is identical. Note, secondly, that the key values are unique across both tables, *i.e.* that the two tables have no key values in common. This is because, by hypothesis, we assigned these keys as enterprise identifiers (EIDs).

This is all the infrastructure we need to have in place. Let's now suppose that the Data Administrator has determined that these two tables are subtypes of a common Customer supertype table. She can now proceed to create that supertype table, as described in Steps 1-3 above. Assuming that Customer Name is the only common attribute, the results of applying Steps 1-3 is to create a Customer table like this:

<i>cust-<b>eid</b></i>	<i>cust-<b>name</b></i>	<i>cust-<b>type</b></i>
7668	Sean Patrick	Individual
1045	Nortel	Business
5173	Susan Brownmiller	Individual
7474	Cynthia Osgood	Individual
3492	Foxglove	Business
0845	Sylvia Bruton	Individual
5966	British Telecom	Business
3215	John Alexander	Business
7490	Mary Smithburger	Individual
1502	Michael Sullivan	Individual
8770	Smith & Sons Amoco	Business
9456	Canon Mills NC #45	Business
9707	Pitrelli Shell	Business
0590	Wachovia, Atl branch 332	Business
.....		
.....		
.....		
.....		

**Figure 49. The Bank’s Customer (Supertype) Table, With EIDs Instead of Natural Keys.**

And assuming that “Other Attributes”, in each original table, are the attributes specific to those subtypes, the results of applying Steps 4-5 is to create Business Customer and Individual Customer tables like these:

<i>business-cust-eid</i>	<i>other-business-customer-attributes</i>
1045	.....
3492	.....
5966	.....
8770	.....
9456	.....
9707	.....
0590	.....
.....	.....
.....	.....
.....	.....
.....	.....

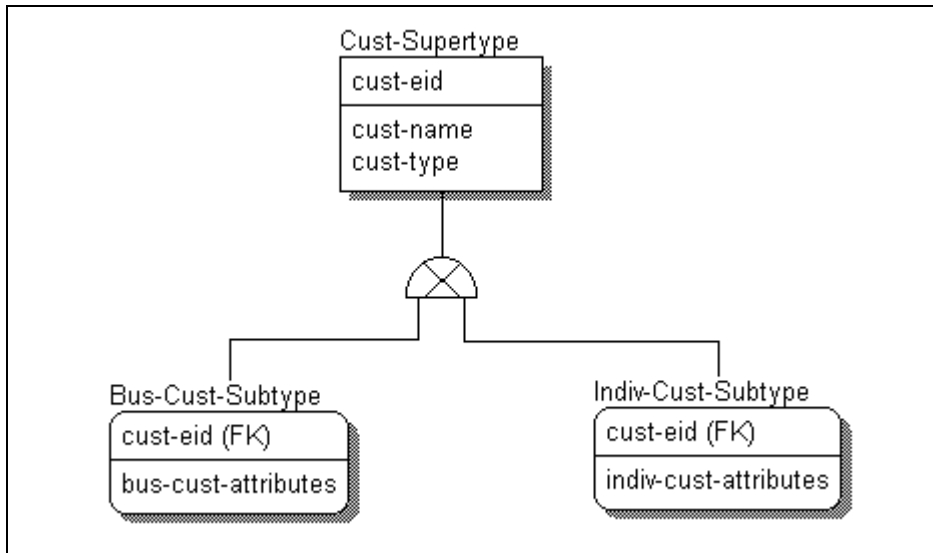
***Figure 50. The Bank's Business Customer Table, With EIDs Instead of Natural Keys.***

<i>individual-cust-eid</i>	<i>other-individual-customer-attributes</i>
7668	.....
5173	.....
7474	.....
0845	.....
3215	.....
7490	.....
1502	.....
.....	.....
.....	.....
.....	.....
.....	.....

**Figure 51. The Bank’s Individual Customer Table, With EIDs Instead of Natural Keys.**

These tables now constitute a superset/subset group. Each row in the supertype table has a corresponding (same EID) row in one of the two subtype tables. And each row in each of the subtype tables has a corresponding (same EID) row in the supertype table. Because the EIDs of rows which keep information about the same customer are identical across the supertype and subtype tables, there are no synonyms among the keys.

The ERD representation of this set of three tables is shown in Figure 52.



**Figure 52. Entity-Relationship Diagram Representation of the Customer Supertype and Two Subtype Tables.**

Relational technology never provided much support for creating and maintaining supertype and subtype hierarchies, relational data modelers use typing very little. But it is clear that deeply embedded into the structure of reality, or at least the structure of how we think about it (an ancient and deep philosophical question), are extensive type hierarchies. Biological classification into kingdom, phylum, class, order, family, genus and species is one explicit and clear-cut example. For another: some persons are employees of a company, and some of those employees are salaried; of the salaried employees, some are managers, and so on. So person-→employee-→salaried→manager is another type hierarchy. (These hierarchies are often called “IS A” hierarchies, because a manager *is a* salaried employee, a salaried employee *is an* employee, and an employee *is a* person.)

Therefore, as systems are re-engineered, either all at once or, more commonly, piece by piece, type hierarchies should be added to them. I have seen type hierarchies 8-10 layers deep, with some layers dozens of types wide, in databases where the typing has clear-cut bottom-line value. As we have seen, creating type hierarchies is a simple process if the tables to be organized into a hierarchy all use globally unique surrogate keys, and a very difficult process otherwise.

We should note one more advantage of using globally unique surrogate keys in our databases. Programs require less code if they can assume that all pointers are unique regardless of the type of object they point to, and that all pointers have the same data type and length. OO systems “swizzle” traditional relational identifiers to achieve this uniformity of pointer structure and behavior. They will not have to swizzle EIDs.

{11/28/2007. Revisions stopped here.}