

Some thoughts about Object Orientation and the Relational Model of Data.

In the following I have talked loosely about objects, not bothering to discriminate between object types, classes, and so on. Each concept should be interpreted in context.

Two points are being emphasized:

- The theory of normalization is still applicable in object oriented systems, although with some modification. One focuses on removing redundancy rather than on the normal forms.
- There is a continuing need for entity modelling in the object-oriented environment.

How are the concepts of normalization carried over to the OO world? (Is there a formal theory of normalization for objects? If you've seen a theory please let me know. There *are* papers on relations not in 1st normal form.)

Objects are often not even in 1st Normal Form. That is, the attributes of an object do not have to be atomic. For instance, a customer object may own many account objects, and this 'many ness' is actually stored as part of the customer object (it could be argued that the customer object actually stores a single reference to a collection, and is therefore still in 1st Normal Form, however, for this discussion lets assume that 1NF is broken).

We need to probe more deeply into the *essence* of 3NF (or 4NF) tables. Because the normalization technique involves such technical steps, it is often not realized that the essential result of normalization is that different types of *business fact* are separated into separate tables. This leads immediately to significant business advantages. The tables are simple and *easy to understand* from a business perspective. The tables provide an *optimally changeable* framework since side effects of the change of any single business fact are minimized. The tables automatically represent the *intrinsic business structure* of the data, rather than being biased to one or another manager's needs.

These days we don't design tables by starting with some ad hoc large table and then splitting it up into 3NF tables. Rather, by applying certain disciplines to our Data Models (Entity Models), the tables derived from them are automatically normalized. We thus get the dual benefit of 'clean' models and normalized tables. It is important to note that the *underlying mechanism* for splitting a large undisciplined table into *normalized tables*, and for drawing a *well-disciplined entity model*, is *the same*. In both cases, the *focus is on the keys* (unique identifiers).

In the case of the normalization of tables, functional dependencies with the same or equivalent left hand side are clustered together (the left hand side is the key) to form the 3NF tables (there are other steps not relevant here). In the case of the entity models, the process is really one of clustering attributes around common keys.

In both cases, *redundancy has to be removed or controlled to prevent corrupt data*.

Now, what about Object Orientation?

One of the models required to describe the objects is known as the *Object Model* (there are others to represent the flow of messages etc). It is *almost identical to the Entity Model*, the main difference being that the Object Model has to show the methods as well as the attributes of each object.

Another difference, which is relevant to our focus, is that some OO standards place no emphasis on keys (they use an OID - an object identifier assigned internally). Composite keys in entity models generally have component attributes whose meaning is derived from other entities, and so it is not surprising that these composite keys actually enforce business rules. It follows that the *object models*, which use only OIDs, *might lose the benefit of these rules* - the rules would have to be explicitly encoded.

As far as relationships are concerned, the entity models and the object models are identical at the conceptual level, except that there may be an extra symbol against each cardinality of the object model relationship indicating whether the object on that side is reachable from the source entity. This translates into an implementation in which entities reachable on the many side, are reached via a collection of references, where the collection is stored in the source object.

Now to the point!

Normalization in entity models is achieved by removing all redundant relationships (with some exceptions) and extraneous attributes. The tables then built from these models are in 3NF. With object models, it is often thought that normalization is inapplicable, since relational tables are not generally created from these models.

However the essential point is that *with the entity models*, whether or not one constructed tables from them, it was evident that *redundant relationships* could *lead to* the retrieval of *corrupt results* simply because there was more than one way to retrieve the desired information and no guarantee that the actual details obtained along the different paths would be consistent.

This observation can be made in much the same way for the object models.

The rules which enable the discovery of these redundancies and their consequent removal or control are therefore just as applicable to object models as to entity models (the 5 derivation rules etc) even if a relational database is not being used for storage of the data.

The alternative to using these rules for objects is a corrupt database of objects! (Of course, just as with relational databases, people are getting it right 90% of the time anyway, without formal rules. The problem is with the other 10%!)

Since the object model is almost identical to the entity model (entity models these days include inheritance structures although there are some interesting differences in the interpretation of these), *the skills of entity modelling are directly transferable to the area of object modelling*. Entity modelling is really about the semantics (the business meaning) of the data. One focuses on the business thing itself and relates it to other business things. So is object modelling! Most organizations that are using object orientation nevertheless continue to use relational databases as their data platform (reaction to this comment?) and so entity modelling (lets say object modelling) will be important for years to come.

Any comments would be welcome.
Email mikeking@metamod.com