Updating a Database

David McGoveran, June 12, 2012

How Not to Design a Database

Consider a concrete example of a poorly designed database, courtesy of C. J. Date (personal communication, 5/21/2012):

> Let database D consist of two *base* relvars A and B, each with single attribute ENO (employee number). Let the only constraints in effect be, first, that the two ENO attributes are defined on the same domain; second, that {ENO} is the sole key for each of A and B. Let the *intended interpretations* be as follows: for A, "employee ENO is on vacation"; for B, "employee ENO is awaiting counseling". [emphasis added]

Date further states that "… these interpretations aren't very significant for present purposes."[1] He goes on to create derived relvars that the DBMS cannot determine how to update.

Without going further, please note that this "design" is already an excellent illustration of poor database design: to wit, it violates POOD, POC, POM and the Information Rule. A domain is asserted but undefined (though we can assume the unstated intent). From these violation we will see that several ambiguities, confusions, and logical contradictions naturally arise.

1. Principle of Minimal Design Violation (POM): From the point of view of the DBMS, it is impossible to differentiate between A and B except by name. Therefore, the design is not minimal. Given that the two ENO attributes are defined on the same domain and there are no attribute or other constraints, their extensions are identical.
2. Principle of Complete Design Violation (POC): From the point of view of the DBMS, it is impossible to differentiate between a permissible ENO value and an illegal ENO value because the design is not expressively complete with respect to the intended interpretation. Thus, the DBMS cannot properly apply the CWA – it has no rule by which to compute the complement of relvars A, B, or A AND B within any known UoD (Universe of Discourse) nor can the DBMS compute the complement of the relations A, B, or A AND B at any given time. Furthermore, it cannot differentiate between the complement of a relvar and the complement of the corresponding relation. To put it another way, the DBMS cannot compute the UoD or any complement of any relation.

[1] I disagree. I think this intended interpretation – and it's being hidden from the DBMS – is the crux of the issue.

3. Principle of Orthogonal Design Violation (POOD): From the point of view of the DBMS, A and Bare clearly not orthogonal – even though the user's intended interpretation assumes they are distinct and possibly even mutually exclusive, these are merely mental constructs hidden from the DBMS. That is, there is nothing in the asserted interpretation that would lead auser to conclude any of the following for a permissible value of ENO:
    a. If ENO in A, ENO not in B.
    b. If ENO in B, ENO not in A.
    c. If ENO in A, ENO in B.
    d. If ENO in B, ENO in A.
4. Information Rule Violation (IR): If A and B are distinct, then from the above the user clearly has information that is being hidden from the DBMS. Otherwise, the user cannot apply the "intended interpretation".

From the foregoing little can be expected of the DBMS. It doesn't know what the UoD is, why it has two base relvars and not one, how to differentiate between permissible operations and impermissible ones, no criteria for differentiating A and B except some user's assertion at the level of denotation, etc.

Virtually all semantic information is concealed from the DBMS and maintained in the mind of the user. This is clearly an integrity problem: We are leaving consistency to the whims of Fortune, for it is unlikely that the required mental constructs will be the same among two or more users. In fact, in my experience, such mental constructs are not consistent over time even in the mind of one user. And, as a business evolves, one can expect that the details of an intended interpretation will change in subtle ways so that, for example, the relationship between A and B is not constant. In other words, with such a design, A at time t1 may well have a different meaning than A at time t2 corresponding to different membership functions. Hence, A is not one relvar but two or more. With the membership functions not being made explicit, kept up-to-date, and made available to the DBMS, the DBMS cannot manage such a relvar and maintain its integrity.

Now note that, given a tuple <ENO> to be inserted into D, the DBMS cannot determine whether it should be inserted into A, B, or both unless the user specifies A and/or B. In other words, we are already in trouble without the complexity of using views to the hide names of A and B and their relationship.

Nonetheless, let's see what else we can glean from Date's illuminating example. Having thus defined database D, Date continues:

> Now define two views V1 and V2, with defining expressions as follows: for V1, V1==A; for V2, V2 ==A UNION (A INTERSECT B).

Note that the "intended interpretation" of V1 is just "employee ENO is on vacation" and of V2 is "employee ENO is either on vacation, or is both on vacation and awaiting counseling."

Assume that B is not intended to be forever empty. From this definition and the assumption, we conclude that "on vacation" and "awaiting counseling" are not mutually exclusive. So this further construction confirms the previous analysis that A and B are not independent and the design violates POOD.

Nothing in this further construction serves to provide the DBMS with enough information to differentiate A and B, let alone differentiate updates to them. Everything the DBMS needs to know is in the mind of the user and we've yet to devise a DBMS that can read minds.

Fixing the Problem(s)

Let's see how we can rectify this situation while keeping with the spirit of the proposed design. Note that the specifics of the solution given here are intended to be pedagogical rather than prescriptive.

The primary problem with the design is that it does not provide the DBMS with sufficient information. In particular, the DBMS needs to be able to infer the UoD (Universe of Discourse), the relationship among the base relvars (i.e., between A and B in this case), and the membership functions of A and B. There are three possibilities:

1. A and B are mutually exclusive.
2. A and B are distinct but have non-empty intersection.
3. A and B are identical.

Obviously, these conditions can be expressed via database constraints if the membership functions for A and B are expressible symbolically.

The membership functions corresponding to A and B are formalizations of the intended interpretations as predicates. Since these predicates are not derived they are "primitive" and have no internal structure. Instead, we create symbols to represent them and record them along with the natural language description.

Let P(ENO) be a predicate having the intended interpretation "ENO is an employee AND ENO is on vacation and let Q(ENO) have the intended interpretation "ENO is an employee AND ENO is awaiting counseling". Insert the symbols for these relvars, the corresponding predicates, the predicate in terms of base relvar predicates, and the corresponding intended interpretations in a translation table in D's system catalog. For primitive predicates, the relvar predicate symbol is repeated in the second and third cited columns. From these we (or the DBMS) can construct the defining relvar predicates for V1 and V2 from P(ENO) and Q(ENO) and their definitions. The symbols for these derived relvars and their corresponding derived relvar predicates can likewise be recorded in the translation table.

The DBMS can now look up the symbols P(ENO) and Q(ENO) when it encounters them in a formula and can determine with which relvar or relvars they are associated.

The user can now convey intent by symbolic reference to these predicates. Note that the DBMS need perform only a lookup – no further deduction or computation is necessary (or possible). Now consider the following attempted inserts, in which P(ENO) and Q(ENO) are to be treated as symbolic references and not as placeholders for some expanded expression:

a) INSERT ‘E1’ INTO D WHERE P(ENO);
b) INSERT ‘E1’ INTO D WHERE Q(ENO);
c) INSERT ‘E1’ INTO D WHERE P(ENO) AND Q(ENO);
d) INSERT ‘E1’ INTO D WHERE P(ENO) AND NOT Q(ENO);
e) INSERT ‘E1’ INTO D

Case 1: If the relationship between A and B is given by constraint 1 above, then from the algorithm given in the previous paper:

INSERT a) inserts ‘E1’ into table A

INSERT b) inserts ‘E1’ into table B

INSERT c) inserts ‘E1’ into neither table A nor table B

INSERT d) inserts ‘E1’ into table A

INSERT e) inserts ‘E1’ into neither table A nor table B

Case 2: If the relationship between A and B is given by constraint 2 above, then from the algorithm given in the previous paper:

INSERT a) inserts ‘E1’ into table A (& possibly by side effect in table B)

INSERT b) inserts ‘E1’ into table B (& possibly by side effect in table A)

INSERT c) inserts ‘E1’ into table A and table B (one or two tuples)

INSERT d) inserts ‘E1’ into table A

INSERT e) inserts ‘E1’ into neither table A nor table B

Case 3: If the relationship between A and B is given by constraint 3 above, then from the algorithm given in the previous paper:

INSERT a) inserts ‘E1’ into table A (by side effect in table B)

INSERT b) inserts ‘E1’ into table B (by side effect in table A)

INSERT c) inserts ‘E1’ into table A and table B

INSERT d) inserts ‘E1’ into neither table A nor table B

INSERT e) inserts ‘E1’ into neither table A nor table B

Points Arising:

Consider only the first four INSERTs a) through d). In Case 1, there is no ambiguity as to which relvar is assigned the new tuple. However, in Case 2, an ambiguity is apparent and there are (predictable) side effects due to violation of Principle of Orthogonal Design. In Case 3, there is again an ambiguity and again predictable side effects, but this time due to a violation of the Principle of Minimal Design (the design is not minimal).

Now let’s consider INSERT e). Why is it that INSERT e) does not result in a new tuple in D regardless of which constraint is declared on D? The reason is simple: INSERT e) represents a UoD that is not covered by relvars A and B. In particular, INSERT e) cannot be within the UoD unless D contains a relvar that could be satisfied by such an INSERT. So, if INSERT e) is to be considered a valid expression within the intended interpretation of D, then D is not expressively complete – it violates the Principle of Complete Design because there are expressions consistent with the intended interpretations that cannot be captured by D. What is needed in each case is a new relvar C the membership function of which is satisfied by all <ENO> tuples that are not known to satisfy either P(ENO) or Q(ENO). The type of relvar C is then a supertype of both A and B.

The Information Rule

Of course, we should really capture the satisfaction of P(ENO) and Q(ENO) as values of attributes in columns. As it stands, D violates the Information Rule. Making this fix to A and B by extending them with one or more new columns – depending on which database constraint is satisfied – should be obvious and I leave that as an exercise to the reader.

Conclusion: Updating Derived Relations V1 and V2

By providing the DBMS with adequate information about the membership functions of relvars and their relationship to each other, and then updating the database D instead of specific relations, we can let the DBMS figure which relvar membership functions are satisfied. If a derived relvar is referenced symbolically, that reference is understood not as a name for the relvar but as a “macro” for the membership function of that derived relvar. I leave it as an exercise to the reader to show that derived relvars need not be treated differently than so-called base relvars.

