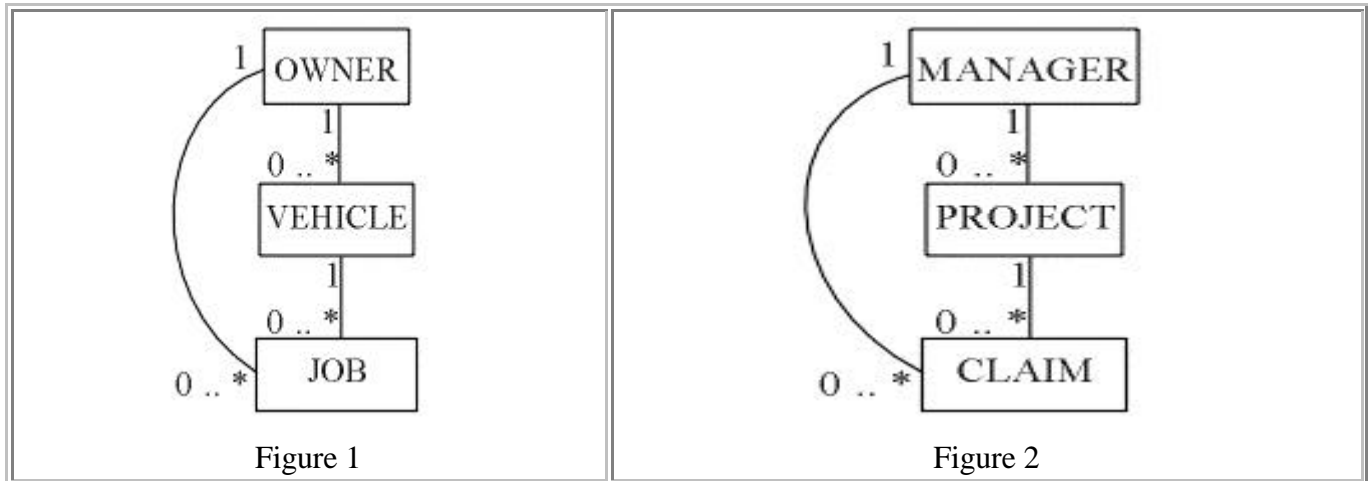


The Impact of Relationships on the Integrity of Databases.

For the purposes of this discussion, Entity Models and Object Models have the same implications. Stated differently, the constraints needed to prevent corruption in the database, are equally applicable to Object Oriented systems and to Relational systems. Object Oriented terms will be used here (the UML notation), but the discussion could be rephrased in Entity Modelling terms.

Some of the potential corruption in databases has its origin in the existence of more than one way to retrieve a set of objects of the same class that are related to a particular 'source' object.



For instance, there are two ways to retrieve OWNER objects (Figure 1). The owner of the vehicle on which the job is being done, or the owner for whom the job is being done.

Consider also Figure 2.

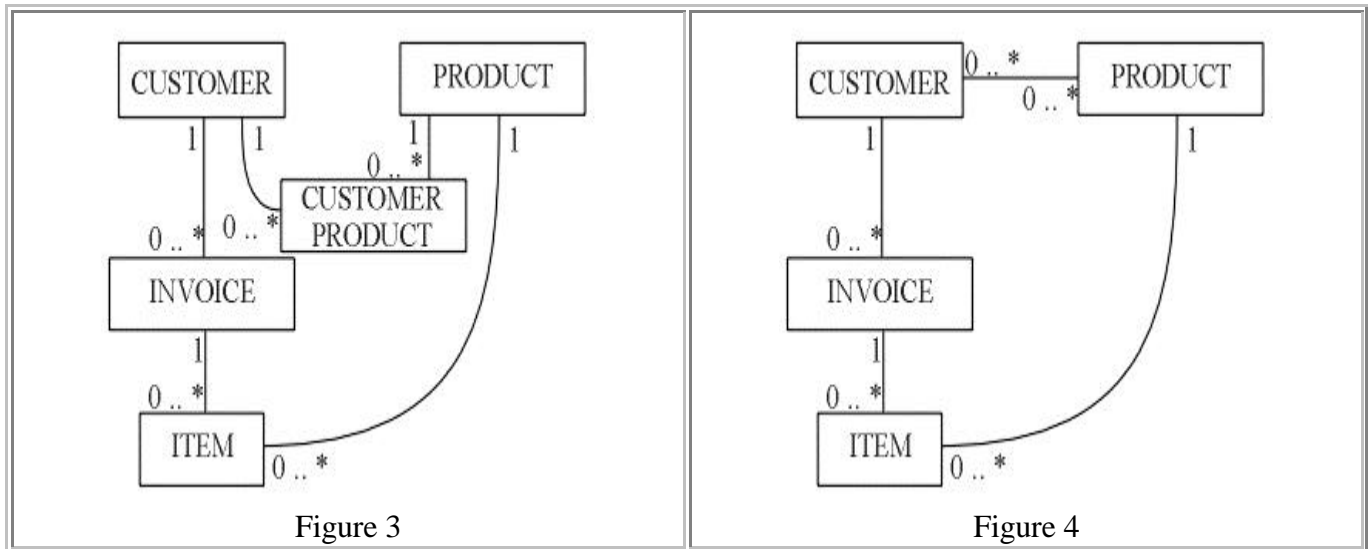
Again there are two paths. The manager of the project for which the claim is being made, or the manager responsible for the type of claim being made (travel, entertainment, ...).

In the first case, if the two paths yield different answers, the database is corrupt. It is therefore essential to detect these situations and prevent the corruption. Clearly there are two ways to prevent the corruption. Either remove the redundant relationship from job direct to owner, or enforce a constraint that ensures that the two paths yield the same owner.

The theory of functional dependencies as applied to object models, is a practical way to detect these cases. (The theory makes use of 5 derivation rules, which describe the patterns leading to corruption. It actually goes further than the simple case shown above.) This theory had its origin in the framework of normalization of relational tables. It can also be used in the object-oriented framework, but without going as far as normalization.

In figure 2 above, it is actually required that the manager supervising a particular type of claim should in general be a different person from the manager supervising the project for which the claim is being made. In this case, neither should an association be removed, nor is a constraint required. Since functional dependency theory would wrongly lead to the removal of an association, we cannot rely on a fully automated algorithm for removing redundancy. A person who understands the business semantics must make the decision, in each case.

In the next model (figure 3) there is a 'many to many' association represented by CUSTOMER_PRODUCT. The same information should be obtainable by retrieving products for each customer via items they have purchased, or by retrieving products for each customer via CUSTOMER_PRODUCT.



That is, according to the business semantics of this situation, the products of a customer should be exactly the same whether they are retrieved via CUSTOMER-PRODUCT or via INVOICE. The same general situation holds for figure 4, which is permissible in an object oriented model. This is also true when navigating in the opposite direction. That is, for the customers who purchase a product, the problem is that, unless explicit controls are implemented, the CUSTOMER-PRODUCT information can be recorded independently of the INVOICE and ITEM information thus leading to a corrupt database.

So these situations need to be detected. However the functional dependency theory is not applicable because the cardinality on the destination side of the association is 0..*, not 1.

A more general way of characterizing the potentially corrupt situations is needed. This characterization is described by the following three conditions:

- (Condition 1) There is a path (path A) of associations (ass A1, ass A2, ass A3, ... , ass Am) starting at class S and ending at class D. ($m \geq 1$)
- (Condition 2) There is another path (path B) of associations (ass B1, ass B2, ... , ass Bn) also starting at class S and ending at class D. ($n \geq 1$)
- (Condition 3) The **business semantics** are such that, the objects of class D retrieved along path B starting at a particular single object of class S, **must always be exactly the same objects (or a subset of them)** as those retrieved along path A starting at the **same** particular single object of class S.

The remedy for a situation with the above conditions is either to remove one or more associations (In searching for associations which are candidates for removal, **only one** association need be examined **at a time** and so to avoid chaos this should be done.), or to enforce the required business semantics via program checks (the appropriate methods in the classes).

It is clear that the use of functional dependency theory only resolves situations where the associations of the paths of conditions 1 and 2 can be interpreted as functional dependencies. That is, the cardinality on the destination side of each association in the path, is 1..1. Thus there will be situations which definitely have the potential for corruption, but for which the functional dependence theory is inapplicable. These generally amount to 'many to many' associations between the objects of class S and the objects of class D. (Note that in the cases where apparent redundancies highlighted by functional dependency theory will actually be intended to have a different business meaning, the functional dependency theory is applicable. It spotlights the apparent redundancy, reminding that we must clearly document the different meaning.)

The application of functional dependency theory to object models should therefore be done with the awareness that it is applicable only to a subset of the redundancy problems. Other techniques must still be applied to remove all corruption. **When used intelligently, functional dependence theory is a very useful way to quickly find some of the problems in object models.**

It is suggested that some algorithm be written to **find the situations of conflicting 'many to many' paths** between the same pair of classes. This is not related to the concept of 4th normal form in relational tables.