Proc. 1977 Very Large Data Bases
Conference (Tokyo)

THE DECOMPOSITION VERSUS THE SYNTHETIC APPROACH TO RELATIONAL DATABASE DESIGN

Ronald Fagin

IBM Research Laboratory
San Jose, California 95193

ABSTRACT: Two of the competing approaches to the logical design of relational databases are the third normal form decomposition approach of Codd and the synthetic approach of Bernstein and others. The synthetic approach seems on the surface to be the more powerful; unfortunately, to avoid serious problems, a nonintuitive constraint (the "uniqueness" of functional dependencies) must be assumed. We demonstrate the fourth normal form approach, which not only can deal with this difficulty, but which is also more powerful than either of the earlier approaches. The input of the new method includes attributes (potential column names), along with semantic information in the form of functional and multivalued dependencies; the output is a "good" (fourth normal form) logical design. The new method is semi-automatic, which is especially helpful in the case of a very large database with many attributes that interrelate in complex ways.

INTRODUCTION

An important problem in the management of large databases is the logical design of the database. In the case of relational databases[5], the problem is the selection of an appropriate set of relation schemas, that is, table skeletons. Two of the competing approaches to the logical design of relational databases are the third normal form decomposition approach of Codd[5] and the synthetic approach of Bernstein and others (Bernstein[2]; Bernstein, Swenson, and Tsichritzis[3]). In this paper, we demonstrate the fourth normal form decomposition approach, which is more powerful than either of the earlier approaches, and which lessens some of their difficulties.

Codd's third normal form decomposition approach to database design proceeds as follows. At the start of the design process, one is given (as input) an initial set of relation schemas, along with a set of functional dependencies. By using the information contained in the dependencies, one converts the initial set of relation schemas into a new, more desirable set of relation schemas. That is, this approach converts a good design into a better design. See Figure 1.

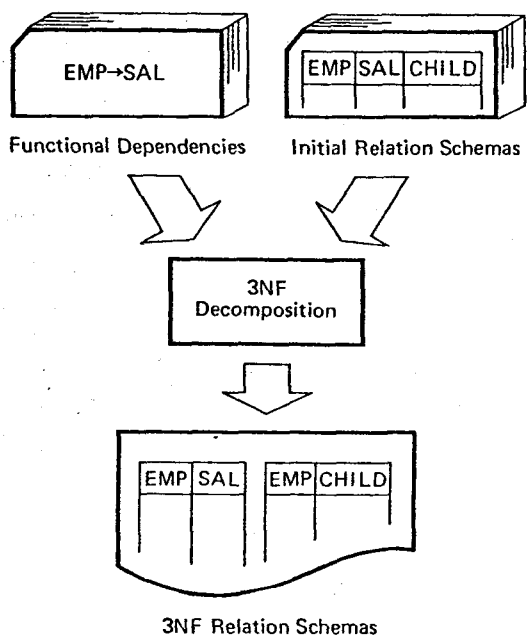

Figure 1. 3NF Decomposition

The synthetic approach of Bernstein et al. is more ambitious. Here the design process begins with only attributes (potential column names) and dependency information, but not with an initial design. The goal is the same - to find a good design. See Figure 2.

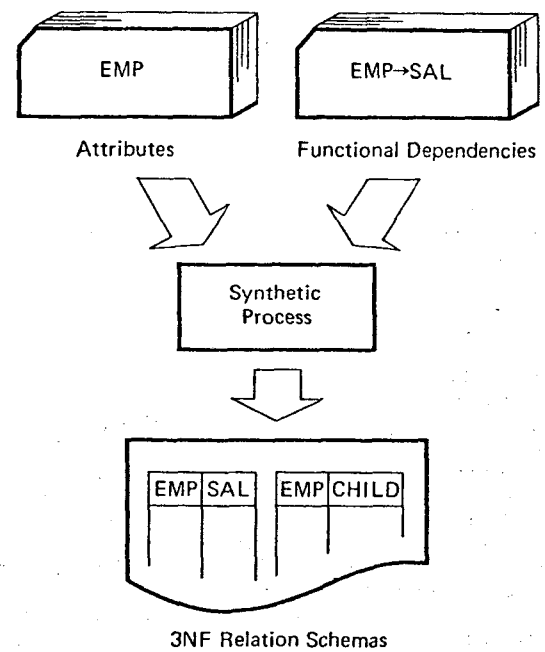

Figure 2. Synthetic Process

Unfortunately, to prevent serious problems it is necessary to make an assumption (the "uniqueness" of functional dependencies) that is nonintuitive and that seems very hard to verify in a large database. We elaborate on this point later. The situation is simpler in the decomposition approach, because in this case, the uniqueness of functional dependencies automatically holds, as we will see.

In our approach, the design process has as input a set of attributes along with a set of functional dependencies and a set of the more general "multivalued dependencies".[7] The final result is a set of relation schemas in fourth normal form[7]. See Figure 3.

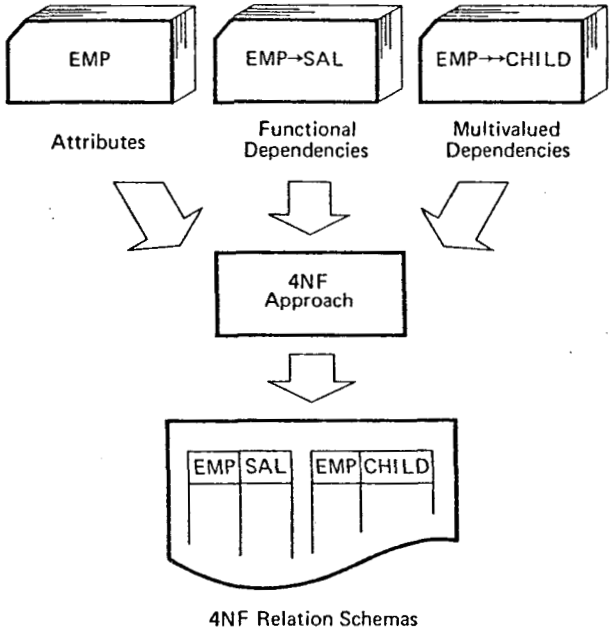

Figure 3. 4NF Approach

Our approach has several advantages over third normal form decomposition and over the synthetic approach. First, our approach is in the spirit of the synthetic approach, since the input consists only of attributes and of dependency information. This is especially helpful in the case of a very large database with many attributes that interrelate in complex ways. For, then an automatic procedure can be invoked that takes this dependency information as input and mechanically generates a design (cf. Bernstein's efficient third normal form algorithm[2] and our fourth normal form algorithm). In practice, it is probably desirable to have a human in the loop (we discuss this point later); thus, the procedure is really semi-automatic rather than automatic. Second, although our approach is in the spirit of the synthetic approach, it "works" mathematically in a similar manner to third normal form decomposition. In particular, the uniqueness problem does not cause us grave difficulties. Third, our approach accepts as input not only functional dependencies, but also multivalued dependencies, which are useful since functional dependencies are inadequate in general (for example, as Schmid and Swenson[10] observed, the information that an employee has a salary is given by a functional dependency, while the information that he has children is not.) Fourth, the output is a set of relation schemas that are not just in third normal form but in the more desirable fourth normal form.

In the second section, we sketch third normal form decomposition and give Codd's definition of a functional dependency.

In the third section, we sketch the synthetic approach and give Bernstein's quite different definition of functional dependency. Bernstein's definition is the source of the difficulties associated with the synthetic approach.

In the next two sections, we introduce multivalued dependencies and demonstrate fourth normal form decomposition.

This paper is informal in nature and proceeds by example. Precise definitions of concepts that appear in this paper are in Fagin[7].

THIRD NORMAL FORM DECOMPOSITION

The basic semantic object associated with third normal form decomposition is the functional dependency. As an example, let S(EMPLOYEE,SALARY,CHILD) be a relation with column names EMPLOYEE, SALARY, and CHILD. Thus, the relation S could be as in Table 1.

EMPLOYEE	SALARY	CHILD
Hilbert	$40K	Hubert
Gauss	$50K	Gwendolyn
Gauss	$50K	Greta
Pythagoras	$20K	Peter

Table 1

Following Codd[5], we say that this relation S obeys the functional dependency EMPLOYEE→SALARY. Intuitively, this means that each employee has exactly one salary. The precise meaning is that if two tuples (that is, rows) of S agree in the EMPLOYEE column, then they agree in the SALARY column. That is, the functional dependency "EMPLOYEE→SALARY" is simply shorthand for the statement "Whenever two tuples of the relation agree in the EMPLOYEE column, they also agree in the SALARY column." When we say that a functional dependency holds for a relation schema,[4] we mean that each relation that is an instance of the schema (that is, each "snapshot") is constrained to obey the dependency. So, a functional dependency is then an "integrity constraint," which constrains the "shape" of the relation.

For the sake of informality, we will be somewhat careless about distinguishing relations from relation schemas. A more careful treatment appears in Fagin[7].

We briefly sketch the basic ideas behind Codd's definition of third normal form (3NF). For a relation schema to be in 3NF it must satisfy three rules. The rules are roughly, but inexactly, as follows (for an exact definition, see Codd[5]): (1) the entries must be atomic; that is, the entries may not be sets or relations. We assume throughout this paper that rule (1) is never violated in any relation discussed. (2) There must be no "partial dependence" on a key. For example, if AB is a key of R(A,B,C,D), then the functional dependency A→C must not hold (of course, the functional dependency AB→C does hold for R since AB is a key). (3) There must be no "transitive dependence." For example, the functional dependencies A→B and B→C should not both hold for R(A,B,C,D).

Codd suggests a number of reasons why relation schemas in 3NF are more desirable than those that are not. Hence, the goal of the 3NF decomposition process is to convert a set of relation schemas that are not in 3NF into a set of relation schemas that carry the same information but that are in 3NF.

3NF decomposition proceeds as follows. The input is a set of relation schemas and a set of functional dependencies. Let, say, R(A,B,C,D) be a relation schema in the set that is not in 3NF. Since we are assuming that rule (1) for 3NF is not violated, we know that either rule (2) or rule (3) is violated.

Case 1: Rule (2) is violated, via, say, AB being a key and A→C holding. Because the functional dependency A→C holds for R(A,B,C,D), it turns out to be possible to replace R(A,B,C,D) by its projections R_1(A,C) and R_2(A,B,D), without loss of information. At this step of the 3NF decomposition process, we indeed do replace R(A,B,C,D) by R_1(A,C) and R_2(A,B,D).

Case 2: Rule (3), and not rule (2) is violated, and rule (3) is violated via, say, the functional dependencies A→B and B→C holding. Because the functional dependency B→C holds for R(A,B,C,D), we can (and do) replace R(A,B,C,D) in this step by R_1(B,C) and R_2(A,B,D), without loss of information.

Case 3: Both rules (2) and (3) are violated. Then by some mechanism, a choice is made as to which decomposition of R(A,B,C,D) should take place.

The decomposition process continues, step by step, until all remaining relation schemas are in 3NF.

There are various objections to 3NF decomposition as a technique for database design. A first objection is that the designer must first find an initial design which is then improved by 3NF decomposition. It would be nice if instead, the designer could begin with just dependency information, rather than with an initial design.

A second objection is that 3NF decomposition only goes "down", not "up" or "sideways". That is, 3NF decomposition can break an existing relation schema into two relation schemas, but it cannot, for example, combine two relation schemas into a single larger schema. So, the output of 3NF decomposition is severely restricted by the input.

THE SYNTHETIC APPROACH

A goal of the synthetic approach is to begin at an earlier stage of the design process than does the 3NF decomposition approach. The input is simply a set of attributes, along with a set of functional dependencies; no initial design is necessary. The output is a set of 3NF relation schemas that in some sense "embody" the functional dependencies.

A definition of functional dependency that is different from the definition in the previous section is called for, since the previous definition required a functional dependency to be true in the context of a given relation. However, in the synthetic approach, functional dependencies are supposed to exist independently of relations that embody them. That is, functional dependencies and not relation schemas are the primitive objects.

Bernstein[2] defines a functional dependency to be a "time-varying function." As an example (due to Bernstein) of the problems associated with this approach, there could be a functional dependency EMP#→MGR# (which maps the number of an employee into the number of his manager) and a functional dependency MGR#→EMP# (which maps a manager's MGR# into his corresponding EMP#). The assumption that the two mappings are inverses would be what Bernstein calls an "invalid syntactic inference" that might require a "semantic analyzer" to reject.

As a second example, there could be a functional dependency EMP#→MGR# (that maps the number of an employee into the number of his manager) and a functional dependency MGR#→STATUS (where the status is a number between 1 and 10). The composition of these functional dependencies is a functional dependency EMP#→STATUS (which gives the status of the manager of an employee). This could be confused with another functional dependency that gives the status of an employee himself.

To prevent such problems, Bernstein suggests the strong assumption that there is at most one functional dependency from a given set of attributes to a given attribute. In the first example we gave, we could change the functional dependency MGR#→EMP# into a functional dependency MGR#→EMP#_OF_MGR. Otherwise there are two functional dependencies EMP#→EMP#: one the identity (which maps an employee number into itself), and the other the composition of the functional dependencies EMP#→MGR# and MGR#→EMP# (this second functional dependency EMP#→EMP# maps the employee number of an employee into the employee number of his manager). In the second example of a problem that we gave, we can obtain uniqueness by having the functional dependencies EMP#→STATUS_OF_EMP and EMP#→STATUS_OF_MGR. As Bernstein notes, "Specifying a set of functional dependencies that lead to no invalid syntactic inferences is clearly a difficult problem." It is very unclear as to how to verify Bernstein's uniqueness assumption in practice.

A further problem with the synthetic approach is that the synthesis algorithm handles only functional dependencies, and hence does not directly handle what Bernstein calls "nonfunctional relationships." For example, an EMPLOYEE may not only have a SALARY (where the relationship is described by a functional dependency EMPLOYEE→SALARY), but the EMPLOYEE may also have a set of CHILDren (where the relationship is not given by a functional dependency, since an EMPLOYEE may have more than one CHILD). Or, there may be a relationship between SUPPLIERs and PARTs where each SUPPLIER supplies several PARTs, and each PART is supplied by several SUPPLIERs. Bernstein deals with these problems by creating dummy attributes θ_1 and creating new functional dependencies {EMPLOYEE,CHILD}→θ_1 and {PART,SUPPLIER}→θ_2. This is a trick that "fools" the synthesis algorithm into creating essentially one relation schema per nonfunctional relationship.

MULTIVALUED DEPENDENCIES

The new semantic object that we consider is the multivalued dependency, which is a generalization of (Codd's definition of) the functional dependency. We present a few examples, then give the definition. As a first example, the relation S(EMPLOYEE,SALARY,CHILD) of Table 1 obeys the multivalued dependency EMPLOYEE↠SALARY (which can be read "EMPLOYEE multidetermines SALARY"), since functional dependencies turn out to be special cases of multivalued dependencies. Furthermore, the multivalued dependency EMPLOYEE↠CHILD holds for S(EMPLOYEE,SALARY,CHILD), because intuitively, an employee's set of children is completely determined by the employee, and is "orthogonal" to the salary. In this case, multivalued dependencies remedy the objection noted earlier to functional dependencies by Schmid and Swenson, that an employee "has" a set of children just as he "has" a salary, and that there should be no arbitrary distinction. Thus, both of the multivalued dependencies EMPLOYEE↠SALARY and EMPLOYEE↠CHILD hold for this schema.

As another example, let T(EMPLOYEE,CHILD,SALARY,YEAR) be a relation that specifies the children and salary history of each employee. See Table 2. A tuple, such as (Pythagoras,Peter,$20K,1976), appears in T iff (1) Pythagoras is an employee, (2) one of his children is named Peter, and (3) during at least part of 1976, his salary was $20K. Although T has no functional dependencies, it does have the multivalued dependencies EMPLOYEE↠CHILD (as in the previous example), and also EMPLOYEE↠{SALARY,YEAR}, because intuitively, an employee's set of children is completely determined by the employee, and is orthogonal to the salary history. Caution: It does not follow from EMPLOYEE↠{SALARY,YEAR} that either EMPLOYEE↠SALARY or EMPLOYEE↠YEAR (See Fagin[7]). The pair {SALARY,YEAR} is in some sense a "cluster".

EMPLOYEE	CHILD	SALARY	YEAR
Hilbert	Hubert	$35K	1976
Hilbert	Hubert	$40K	1976
Gauss	Gwendolyn	$40K	1975
Gauss	Gwendolyn	$50K	1976
Gauss	Greta	$40K	1975
Gauss	Greta	$50K	1976
Pythagoras	Peter	$15K	1975
Pythagoras	Peter	$20K	1976

Table 2

We now present the formal definition of multivalued dependencies. (We give the definition in Beeri, Fagin, and Howard[1], which is slightly more general than the definition in Fagin[7], in that the left- and right-hand sides of the multivalued dependency need not be disjoint.) Let R be a relation. If X is a subset of the column names of R, and u is a tuple of R, then by u[X] we mean the projection of u onto X. When we say that x is an X-value of the relation R, we mean that x=u[X] for some tuple u of R. Let X and Y be subsets of the column names of R. Define $Y_R(x)=\{y:$ For some tuple u in R, both u[X]=x and u[Y]=y$\}$. That is, Y_R is the set of Y-values that appear in R with the X-value x. Let Z be the set of column names of R that are not in either X or Y (thus, Z is the complement of the union of X and Y). The relation R obeys the multivalued dependency X↠Y if for each XZ value xz that appears in R, we have $Y_R(xz)=Y_R(x)$. That is, the multivalued dependency X↠Y holds for R if the set of Y values that appear with a given x also appear with each combination of x and z in R. So, this set is a function of x alone and does not depend on the z-values that appear with x.

As an example, consider the relation T(EMPLOYEE,CHILD,SALARY,YEAR) in Table 2. The multivalued dependency EMPLOYEE↠CHILD holds for T, since, for example, $CHILD_T$(Gauss) equals both $CHILD_T$(Gauss,$40K,1975) and $CHILD_T$(Gauss,$50K,1976), which all equal {Gwendolyn,Greta}.

Many properties of multivalued dependencies are explored in Fagin[7] and in Beeri, Fagin, and Howard[1]. For example, as we already noted, functional dependencies are special cases of multivalued dependencies. Another important property is that multivalued dependencies provide a necessary and sufficient condition for a relation to be decomposable into two of its projections without loss of information (in the usual sense that the original relation is guaranteed to be in the natural join of the two projections). Thus, since the multivalued dependency EMPLOYEE↠CHILD holds for the relation T(EMPLOYEE,CHILD,SALARY,YEAR) in Table 2, it follows that this relation can be decomposed into the two relations T_1(EMPLOYEE,CHILD) and T_2(EMPLOYEE,SALARY,YEAR) without loss of information (see Table 3). We note that T(EMPLOYEE,CHILD,SALARY,YEAR) cannot be decomposed on the basis of any functional dependencies, because there are none (except trivial functional dependencies, such as A→A). In fact, T is in third normal form, and even in the stronger "improved third normal form," or as it has come to be known, "Boyce-Codd normal form"[6], since it is "all key" (that is, no proper subset of the four column names form a key for T). However, as we will see, it is not in fourth normal form. To obtain fourth normal form, it is necessary to decompose T as above into T_1 and T_2. The example is due to Schmid and Swenson,[10] who recommend decomposition on semantic grounds.

EMPLOYEE	CHILD
Hilbert	Hubert
Gauss	Gwendolyn
Gauss	Greta
Pythagoras	Peter

EMPLOYEE	SALARY	YEAR
Hilbert	$35K	1976
Hilbert	$40K	1976
Gauss	$40K	1975
Gauss	$50K	1976
Pythagoras	$15K	1975
Pythagoras	$20K	1976

Table 3

FOURTH NORMAL FORM DECOMPOSITION

Fourth normal form decomposition is a generalization of third normal form decomposition. However, in some ways fourth normal form decomposition "looks like" a sound, powerful version of the synthetic approach. The input is a set of attributes, along with semantic information in the form of functional dependencies and multivalued dependencies. The output is a set of relation schemas in fourth normal form.

Before we can define fourth normal form, we need the simple concept of a "trivial multivalued dependency." Assume that the sets X,Z partition the set of column names of R(X,Z). It is easy to verify that the multivalued dependency X↠Y always holds for R when Y is either a subset of X or a superset of Z. For example, the multivalued dependency {A,B}↠C holds for every relation S(A,B,C) with exactly three column names, A,B,C. We call these "trivial multivalued dependencies." A relation schema R is in fourth normal form (4NF) if whenever a nontrivial multivalued dependency X↠Y holds for R, where X and Y are subsets of the column names of R, then the functional dependency X→A holds for every column name A of R. That is, a relation schema is in 4NF if all dependencies are the result of keys. In particular, a 4NF relation schema can have no nontrivial multivalued dependencies that are not functional dependencies.

We now present an example of the 4NF normalization process. The attributes are PROJECT, PART, SUPPLIER, LOCATION, COST, EMPLOYEE, SALARY, and HIREDATE. Intuitively, a PROJECT (such as Project 17) uses PARTs (such as nails), which are supplied by SUPPLIERs (such as Acme), each of which can have a number of locations (such as Oklahoma City). In this example, we assume that if a SUPPLIER supplies a PART to a PROJECT, then all LOCATIONs of the SUPPLIER supply that PART to the PROJECT and all at the same COST. The COST depends on SUPPLIER and the PART. Finally, each PROJECT has one MANAGER (such as Jones), and a set of EMPLOYEEs (such as Smith), each of whom have a SALARY (such as $30K) and a HIREDATE (such as 1973).

We begin the normalization process by forming a single relation schema

W(PROJECT,PART,SUPPLIER,LOCATION,COST, EMPLOYEE,MANAGER,SALARY,HIREDATE) . (1)

What are the dependencies?

We have the following functional dependencies (note that for simplicity, we do not distinguish between a singleton set {A} and its only member A; e.g., we write COST for {COST}):

{SUPPLIER,PART}→COST (2)

PROJECT→MANAGER (3)

EMPLOYEE→{SALARY,HIREDATE} . (4)

Based on our assumptions, the following multivalued dependencies hold:

SUPPLIER↠LOCATION (5)

PROJECT↠{EMPLOYEE,SALARY,HIREDATE} . (6)

As for (6), the multivalued dependency PROJECT↠EMPLOYEE does not hold. Instead, the SALARY and HIREDATE (properties of the EMPLOYEE) must be "clustered" together with EMPLOYEE on the right-hand side for the multivalued dependency to hold (as the reader can verify).

We now begin 4NF normalization process. The basic rule is that if a functional dependency X→Y or a multivalued dependency X↠Y holds for a relation R(X,Y,Z), where Z is the set of column names not in X or Y, then R can be decomposed into R_1(X,Y) and R_2(X,Z) without loss of information (the original relation R(X,Y,Z) is then the natural join of R_1(X,Y) and R_2(X,Z)). On the basis of the functional dependency (2), we decompose W (as given by (1)) into

W_1(SUPPLIER,PART,COST)

and

W_2(SUPPLIER,PART,PROJECT,LOCATION,EMPLOYEE, MANAGER,SALARY,HIREDATE) .

Although W_1 is now in 4NF, W_2 is not, since, for example, the functional dependency EMPLOYEE→MANAGER holds for W_2, whereas EMPLOYEE is not a key. We now decompose W_2 further. It can be shown[7] that if a multivalued dependency X↠Y holds for a relation, then it holds for every projection that contains at least all of the column names in X and Y. Hence, since multivalued dependency (6) holds for W, it also holds for its projection W_2. On the basis of (6), we decompose W_2 into its projections

W_{21}(PROJECT,EMPLOYEE,SALARY,HIREDATE)

and

W_{22}(PROJECT,SUPPLIER,PART,LOCATION,MANAGER) .

Using functional dependency (4), we decompose W_{21} into

W_{211}(EMPLOYEE,SALARY,HIREDATE)

and

W_{212}(EMPLOYEE,PROJECT) ,

each of which are in 4NF.

Using functional dependency (3), we decompose W_{22} into

W_{221}(PROJECT,MANAGER)

and

W_{222}(PROJECT,SUPPLIER,PART,LOCATION) .

Finally, using multivalued dependency (5), we decompose W_{222} into

W_{2221}(SUPPLIER,LOCATION)

and

W_{2222}(SUPPLIER,PROJECT,PART) .

We are left with the 4NF family

W_1(SUPPLIER,PART,COST)

W_{211}(EMPLOYEE,SALARY,HIREDATE)

W_{212}(EMPLOYEE,PROJECT)

W_{221}(PROJECT,MANAGER)

W_{2221}(SUPPLIER,LOCATION)

W_{2222}(SUPPLIER,PROJECT,PART) .

The final result could have been different if we had decomposed differently. In Fagin,[7] there are mentioned several heuristics (suggested by Zaniolo[11] and Rissanen[9]) as to the order in which to use the dependencies for decomposition. In fact, a 4NF normalization "box" can generate new dependencies (that can then be exploited). That is, if the user inputs a set of functional and multivalued dependencies, then it is possible to obtain new dependencies, that are consequences of the input dependencies, but that were not inputs themselves. As a simple example, the functional dependency A→C is a consequence of the functional dependencies A→B and B→C. As a slightly more complicated example, if the multivalued dependency X↠Y holds in R(X,Y,Z), where the set Z contains all column names of R not in X or Y, then[7] the multivalued dependency X↠Z also holds for R. In Beeri, Fagin, and Howard[1], a complete axiomatization for dependencies is exhibited, from which all dependencies that are consequences of an input set of dependencies can be derived. Using these axioms, one can in principle obtain all possible 4NF designs.

By initially forming a single large relation schema (as in (1)), a 4NF normalization algorithm has at least as many options as if it begins with many small relation schemas that are then decomposed further. That is, there are at least as many possible final results in the former case. Hence, an algorithm has more of a chance to optimize.

If there is a human in the normalization loop, then the problem of determining all functional and multivalued dependencies that hold in a given situation seems less formidable. This is because the human can "notice" a previously neglected dependency at a late stage of the 4NF normalization process, and then either apply it at that stage or incorporate it in the list of "known" dependencies and start over. The human can make a decision to modify the design (and perhaps stop short of 4NF) for performance or even esthetic reasons. Indeed, it has been conjectured[8] that people are just not going to accept a fully computer-generated design in which a human was not actively involved!

FOURTH NORMAL FORM DECOMPOSITION AND BERNSTEIN'S UNIQUENESS ASSUMPTION

The 4NF approach provides a discipline for handling problems related to Bernstein's uniqueness assumption for functional dependencies. A key observation is that the uniqueness always holds within a given relation. For example, if A and B are column names of a relation R, then the functional dependency (under Codd's definition) A→B either holds for R or it does not. There is no possibility (or meaning attached to) two functional dependencies A→B (with the same left- and right-hand sides) both holding for R.

The 4NF approach begins with the designer forming (conceptually) a single large relation schema. He can then write (in a language like English) a description of a typical tuple. We did essentially this at the beginning of our earlier extended example W(PROJECT,PART,SUPPLIER,LOCATION,COST,EMPLOYEE, MANAGER,SALARY,HIREDATE).
Thus, if (pr,pa,su,lo,co,em,ma,sa,hi) is a typical tuple of W, then lo is the location of the supplier su; part pa is supplied by supplier su to the project pr; and so on.

Let us now consider the examples given earlier that violated Bernstein's uniqueness assumption. Assume that the attributes include (among others) EMP#,MGR#, and STATUS. It should be clear from the description of a typical tuple whether the status is that of the employee or of the manager. As is characteristic of the decomposition approach, it cannot be both simultaneously (since we are working within a single relation). If two different types of status are required, such as STATUS_OF_EMP and STATUS_OF_MGR, it will be clear that new attributes should be introduced. Incidentally, at the conclusion of the 4NF decomposition, it is possible to rename the attributes. For example, if we ended with (among others) the relation schemas R_1(EMP#,STATUS_OF_EMP) and R_2(MGR#,STATUS_OF_MGR), we could then rename to obtain R_1(EMP#,STATUS) and R_2(MGR#,STATUS), if we felt that this were more desirable.

In our other example that violated the uniqueness assumption, there were functional dependencies (under Bernstein's definition) EMP#→MGR# and MGR#→EMP# that were not inverses of each other. But if there are two functional dependencies (under Codd's definition) A→B and B→A within a single relation, then they are automatically inverses of each other. Once again, we can tell what it means for an EMP# em and a MGR# ma to appear together in a single tuple by our description of a representative tuple. Thus, we can see whether manager number ma is the manager of employee number

em, or whether em is the employee number of manager number ma. Since this is a single relation, both cannot be true simultaneously. If desired, we can introduce new attributes, so that both relationships can be represented.

Under the synthetic approach, we are in the helpless position of needing an assumption of uniqueness and having no way to verify the assumption. Under the decomposition approach, uniqueness holds automatically, and we have a mechanism for verifying whether new attributes are needed (to encode more information).

Note that because we have multivalued dependencies to work with in the 4NF approach, we have the ability to split our single large relation. However, using only Codd's functional dependencies, we cannot, for example, split the relation T(EMPLOYEE,CHILD,SALARY,YEAR) of Table 2.

SUMMARY

A major disadvantage of using Codd's 3NF decomposition as a tool for the logical design of relational databases is that the output is severely limited by the input. In fact, the input must include an initial design that is usually not very different from the final design (the output). This is because 3NF decomposition only goes "down", not "up" or "sideways". However, under the 4NF approach, it is possible to obtain all possible 4NF designs. This is because, by starting with a single relation schema with all the attributes, there is nowhere to go but "down".

The synthetic approach of Bernstein and others is intended to remedy the deficiencies of 3NF decomposition by supplying as input only semantic information in the form of functional dependencies. Unfortunately, in this approach, in which functional dependencies, rather than relations, are the primitive objects, one is required to make the nonintuitive, hard-to-verify assumption of uniqueness of functional dependencies in order to avoid serious difficulties. A further problem with the synthetic approach is that functional dependencies are inadequate by themselves.

Our 4NF approach is in the synthetic "spirit" in that the input consists of semantic information in the form of dependencies. However, since we work in the context of a single relation, the uniqueness assumption holds automatically. And, there is a mechanism for determining if new attributes are called for. Furthermore, our approach is more powerful than either of the other approaches in that we allow as input not only functional dependencies but also the more general multivalued dependencies.

The 4NF "box" can employ heuristics to decide among the whole spectrum of all "good" (4NF) designs. It can generate new dependencies that are consequences of the input dependencies. It is possible to have a human in the loop who can add in dependencies that were accidentally omitted (the human can also decide to stop short of 4NF for performance or other reasons).

ACKNOWLEDGEMENTS

The author is grateful to David Hsiao and Ted Codd for encouraging him to put to paper the ideas expressed in this report. He is also grateful to Chris Date and Arne Solvberg for reading an early draft and making helpful suggestions.

REFERENCES

[1] Beeri, C., Fagin, R., and Howard, J. H. "A complete axiomatization for functional and multivalued dependencies in database relations." Proc. 1977 ACM SIGMOD.

[2] Bernstein, P. A. "Synthesizing third normal form relations from functional dependencies." Trans. on Database Systems 1, 4 (Dec. 1976), 277-298.

[3] Bernstein, P. A., Swenson, J. R., and Tsichritzis, D. C. "A unified approach to functional dependencies and relations." Proc. ACM SIGMOD, W. F. King (ed.), San Jose, California (May 1975), 237-245.

[4] Cadiou, J-M. "On semantic issues in the relational model of data." Proc. International Symposium on Math. Foundations of Computer Science, Gdańsk, Poland (September 1975), Springer-Verlag Lecture Notes in Computer Science.

[5] Codd, E. F. "Further normalization of the data base relational model." Courant Computer Science Symposium 6, Data Base Systems, Prentice-Hall, N.Y. (May 1971), 65-98.

[6] Codd, E. F. "Recent investigations in relational data base systems." IFIP Conf. Proc., North-Holland Publishing Company (1974), 1017-1021.

[7] Fagin, R. "Multivalued dependencies and a new normal form for relational databases," Trans. on Database Systems (Sept. 1977).

[8] Merten, A. and Taylor, R. W. Personal communication.

[9] Rissanen, J. J. "Independent components of relations." Trans. on Database Systems, to appear.

[10] Schmid, H. A. and Swenson, J. R. "On the semantics of the relational data model." Proc. ACM SIGMOD, W. F. King (ed.), San Jose, California (May 1975), 211-223.

[11] Zaniolo, C. "Analysis and design of relational schemata for database systems." Ph.D. Dissertation, UCLA, 1976 (UCLA technical report UCLA-ENG-7669, July 1976).