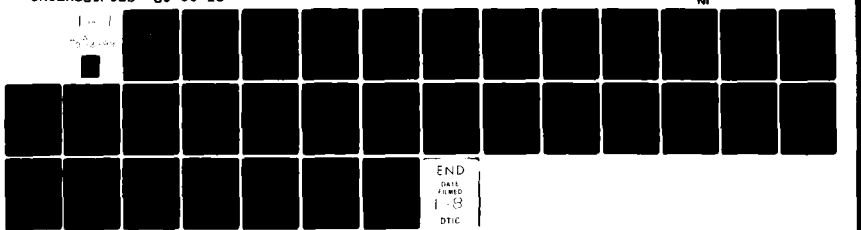
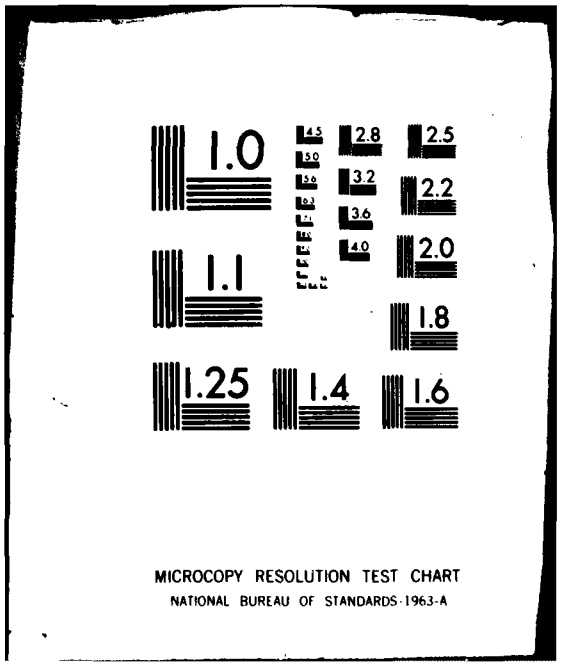
AD-A093 399 WHARTON SCHOOL PHILADELPHIA PA DEPT OF DECISION SCIENCES F/G 9/2
THE CONSEQUENCES OF THE UNIQUENESS ASSUMPTION FOR RELATIONAL DA--ETC(U)
APR 80 A BELLER
UNCLASSIFIED 80-03-13 N00014-75-C-0462
NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: 80-03-13	2. GOVT ACCESSION NO.: AD A093399
3. RECIPIENT'S CATALOG NUMBER: 9	
4. TITLE (and Subtitle): THE CONSEQUENCES OF THE UNIQUENESS ASSUMPTION FOR RELATIONAL DATABASES.	5. TYPE OF REPORT & PERIOD COVERED: Technical rept. Apr 80-Mar 8
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Aaron Beller	8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0462
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Department of Decision Sciences, The Wharton School, University of Pennsylvania, Phila., PA 19104	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Task NR049-272
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research	12. REPORT DATE: Apr 80
	13. NUMBER OF PAGES: 31
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Distribution unlimited

DTIC ELECTE JAN 5 1981 D

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

relational databases, uniqueness or universal relation/assumption. database constraints; Regular FDs, Injective FDs, computable FDs; splitting attributes; FD derivation, schema design algorithms, interrelational concepts.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

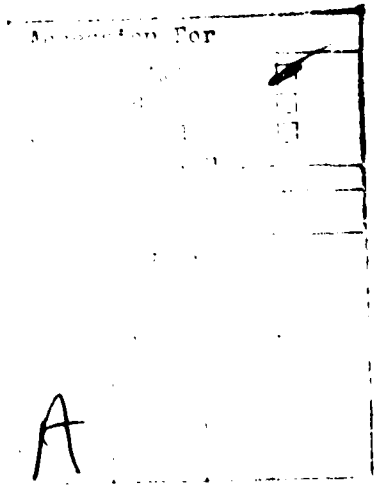
Much of the work on relational databases that deals with data dependencies makes a uniquess (or universal relation) assumption. It has been recognized that this assumption is problematic; nevertheless it is necessary for the axiomatic approach taken in many papers on the theory of relational databases. We will describe the problem, investigate some of the solutions put forward and suggest a new solution. Many of the problems remain intractable within the realm of "classical" relational databases and restrictions must be placed on the use of FDs. An automated method is presented that

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601 | SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

THE CONSEQUENCES OF THE UNIQUENESS ASSUMPTION FOR RELATIONAL DATABASES

Aaron Beller

80-03-13



Department of Decision Sciences
The Wharton School
University of Pennsylvania
Philadelphia, PA 19104

This research was supported in part by the Office of Naval Research under Contract N00014-75-C-0452.

ABSTRACT

Much of the work on relational databases that deals with data dependencies makes a uniqueness (or universal relation) assumption [BBG] . It has been recognized that this assumption is problematic [BBG]; nevertheless it is neccessary for the axiomatic approach taken in many papers on the theory of relational databases. We will describe the problem, investigate some of the solutions put forward and suggest a new solution. Many of the problems remain intractable within the realm of "classical" relational databases and restrictions must be placed on the use of FDs. An automated method is presented that searches for violations of the uniqueness assumption.

1. INTRODUCTION

Much of the work in database theory makes use of a universal relation assumption. This is particularly true for schema design [BERN], [Bb], [BBM] (it is also essential to the concept of the lossless join and other interrelationship concepts). It is assumed that a universal relation exists that contains all the attributes of a database and that all the relations of a database are projections of the universal relation. The assumption is natural for schema design since it means that attributes have an invariant meaning over the database and any joins can be taken without ruining the meaning of the database. The assumption has two consequences; the uniqueness assumption, i.e. that there can be at most one dependency from X to Y, where X and Y are sets of

attributes and the joinability assumption, i.e. that any two relations can be joined on a common attribute [BP'].

We will be concerned with functional dependencies, FDs, though the uniqueness assumption applies to other dependencies as well. Armstrong [ARM] presents an axiomatic approach to FDs and uses a set of axiom schema to derive all the FDs that follow from a given set of FDs. If G is a given set of FDs, G^+ denotes the set of derived FDs. The uniqueness assumption must extend to the derivations of FDs and it can be stated as follows:

Uniqueness Assumption: Let X be a set of attributes and A an attribute. If $X \dashrightarrow A \in G^+$ then any derivation of $X \dashrightarrow A$ represents the same "user intent".

The uniqueness assumption means that syntactically identical FDs are semantically equivalent. The uniqueness assumption causes a fundamental problem in that it prevents the relational databases from modelling real world situations. We shall see that it is sometimes possible, and even neccessary, to have more than one FD between two attributes (or entities). This has been noticed ([BP], [SS]), and the solutions put forth either violate the atomic nature of attributes (1NF) or put artificial restrictions on the derivation of FDs.

There is also the problem of verifying the uniqueness assumption. Let us examine how FDs are originated. If the fundamental construct in defining a relational database is the relation, then the users supply keys with the schema (which is

just specifying some FDs). The users would then be asked to "find" additional intended FDs. In fact, Bernstein [BERN] takes the view that FDs are the fundamental construct and that the users should be asked to supply all "intended" FDs and synthesize the relations from them.

Beeri and Bernstein [BB] have developed a fast algorithm for synthesizing relational database schema in the 3rd normal form from a given set of FDs such that the resulting schema embodies the original FD's. There algorithm uses the uniqueness assumption.

When Beeri and Bernstein's algorithm is used to synthesize relational database schema an attempt should be make to verify the uniqueness assumption. This would be done in two steps; first by having the users go over all the initial FDs making sure that all syntactically identical FDs have the same semantic (user) intent. The first step may cause the database administrator to rename and add attributes and only after he "finalizes" the attributes and FDs could the relations be synthesized. Then, since the first step does not guarantee that all derived FDs will satisfy the uniqueness assumption, the users would have to "decide" whether all derivations (using Armstrong's axioms) of an FD X ---> Y have the same user intent. Though Beeri and Bernstein's algorithm is linear any attempt to verify the uniqueness assumption is doomed to exponential time (counting each human decision as one unit).

An automatic checking for violations of the uniqueness assumption is preferable to interactively "showing" each derivation to the user. Such a semantic analyzer is difficult to find since it is not known how to formalize the "user intent" of an FD. As a partial solution we classify FDs into three types, regular, injective and computable. Armstrong's [ARM] axioms can be applied to these types so that every derivation of an FD will result in classifying the FD as one of the three types (given the types of the initial FDs). When two derivations of an FD result in two different classifications then we have a violation. If two derivations both result in a computable FD it is sometimes possible to decide that the computations are different and there is a violation. In other cases it would not be known if there was a violation.

The usual solution to a violation of uniqueness is to rename some attributes. This can cause a multiplicity of attribute names and in addition may lead to "difficulties" so that sometimes certain derivations must be "outlawed."

2. Definition and Preliminaries

There is much diversity in notation for relational databases and we will primarily follow Fagin [FAG] not going into too much detail. Our view of relational databases is somewhat similar to that of Cadiou [CAD] and Nicolas [NIC]. Let X be a finite set of attributes; an X-tuple is a function with domain X (associating

with each attribute a value. If $Y \subseteq X$ and t is an X-tuple then t[Y] denotes the Y-tuple obtained by restricting the mapping to Y. There are two notions of a relation; intension and extension. The extension of a relation over the attributes X, or simply an X-relation is a finite set of X- tuples. If R is an X- relation and Y X, then R[Y], the projection of R onto Y, is defined by: $R[Y]=\{t[Y] : t \in R\}$.

The intention of a relation includes a set of attributes and as much of the "user intent", in the form of constraints, as possible. The intention of a relation consists of:

1. A relational form made up of a relation name, R, and a set of attributes X, usually written $R(A_1,\ldots,A_n)$, where $X=\{A_1,\ldots,A_n\}$
2. A set of keys (which is a partial listing of the FDs)
3. Functional dependencies and Other types of dependencies
4. Domain definitions of the attributes
5. Other integrity constraints (See Eswarian [ES] and Hammer Mcleod [HM] for a taxonomy).

1. and 2. are usually called a relation scheme and in Beeri and Bernstein [BB], while in [FAG] the entire intension is called the relation scheme.

For each intension $\underline{R}$ there are many extensions. Each extension (or instance) of $\underline{R}$ is a finite set, R, of X-tuples satisfying the constraints of the intention. Thus we differentiate between intension and extension by underscoring the intension.

The intension of a database is a finite collection of relational intensions with additional integrity constraints (that include more than one relation). By the uniqueness assumption attributes and their domain definitions are invariant over the database so the FDs (and other dependencies) can be considered to reside in the database as a whole.

The constraints on a database can be stated in any appropriate language such as: first order predicate logic, SEQUEL, QUERY BY EXAMPLE. It is possible to discuss the set of all extensions of a database (which is infinite in general) but many questions (consistency, derivability) may be undecidable. For details consult Gallaire and Minker [GM] and Nicolas [NIC].

The constraints for which the above questions are important are those that affect the structure of the database. FDs and other dependencies affect the relational database schema since the normal forms are stated in terms of the FDs. Fortunately, under the uniqueness assumption, questions of consistency and derivability about FDs are decidable.

An FD is denoted by X-->Y, where X and Y are sets of attributes. The only information that the above notation imparts is that for any relation $\underline{R}$ whose attributes include $X \cup Y$ and for any instance R of $\underline{R}$, if two tuples coincide on X they must also coincide on Y. In other words For any instance, R, of a relation containing the attributes X Y, $\{<t[X],t[Y]> : t \in R\}$ is a finite partial function from dom(X) to dom(Y).

Formally, $\forall t \in R \forall s \in R((t[X]=s[X] \Rightarrow (t[Y]=s[Y]))$. Sometimes f:X-->Y is written, where f denotes a canonical name for the partial function from dom(X) to dom(Y) which is dependent on the extension (and changes as the extension does). If f:X-->Y and R is an instance of $\underline{R}$ then f R, the realization of f in R, is the finite artial function above. Note that the uniqueness assumption means that function from dom(X) to dom(Y) is invariant over any relation containig those attributes in any given instance of the database.

Armstrong [ARM] gave a set of axiom schema for deriving FDs from a given set of FDs and shows that the system is sound and complete. Beeri and Bernstein [BB] use the following equivalent axioms.

A_1: (Reflexivity) X-->X

A_2: (Augmentation) If X-->Z then $X \cup Y$-->Z

A_3: (Pseudotransivity) If X-->Y and $Y \cup Z$-->W then $X \cup Z$-->W.

5. Checking and Correcting the Uniqueness Assumption

If G is a set of FDs, then G^+ is the closure of G under the above axioms. An important part of Beeri and Bernstein's algorithm is to decide whether a gived FD lies in G^+. If X-->Y can be derived from G it can be derived by an infinite number of derivations. By the uniqueness assumption Beeri and Bernstein can assume any derivation of X-->Y represents a unique "user intent." Hence they need only search for one such derivation. They only have to search derivation trees of height at most the number of attributes among the G since a derivation with a loop (i.e. one that goes through an attribute twice) is the same without the loop since X-->X must be the identity mapping by uniqueness.

The scenario that Beeri envisages [BP] is that first the database administrator checks that the uniqueness assumption is not violated by consulting with the users and then after the attributes and FDs are finally set the linear algorithm for creating the relational database schema in 3NF can be used. The correctness of the uniqueness assumption must be a matter of belief since any method for verifying the uniqueness assumption will involve comparing different derivations of a single FD. By the above method, if X-->Y is not unique there are two derivations of X-->Y by trees of at most height twice the number of attributes (since at most one loop is needed). Since we would have to search all derivations the solution is at best exponential. When a violation of uniqueness is discovered the usual remedy is to change attribute names so that two different FDs are produced.

This would change the final relational database schema synthesized by the Beeri and Bernstein algorithm.

Primitively the "user" could be used as an oracle to decide whether derivations are unique. If a violation is found attributes can be named but it would produce more derivations and possibly violations. It is not clear that such a process terminates by some given bound (an example of this will be discussed later). Beeri and Bernstein suggest an alternative solution--simply reject some inferences. We shall see that this may be necessary.

4. Classification and Discussion of Functional Dependencies

In this section we will classify FDs and discuss the problems they cause visi vie the uniqueness assumption. We will work within the universal relation assumption.

4.1 Classification

Regular FDs:Regular FDs are those that have no additional semantic meaning besides what is demanded in the definition of an FD, i.e. the realized function of X-->Y can be any finite partial function from dom(X) to dom(Y).
Examples: EMP-->DEPT, DEPT-->MGR.

Injective FDs: Injective FDs have the extra restriction that the realized functions must be one to one, i.e.

If X-->Y is injective then

$\forall t \in R \forall s \in R((t[X]=s[X]) <--> (t[Y]=s[Y]))$ We denote such an FD by X <--> Y.

Examples: SS# <-->EMP.

Computable FDs:

An FD f:X-->Y is computable if the t[Y] can be computed from t[X] (In the language that the database constraints are stated), the database instance and the database constraints. We shall denote by F:X-->Y, where F is upper case and represents the "real" function involved (stated in the proper language). We reserve capitals for computable functions.

Examples:

A. F:SALARY, NUMBER.OF.DEP-->WITHOLDING.TAX

This is an example of direct computation from t[SALARY,NUMBER.OFDEP].

B. If A and B are attributes we may have A+B=K a constant and G:A-->B where G=K-A

Here the computable FD is one to one, but the computability is the more essential property.

C. H: DEPT --->NUMBER.OF.EMP This FD is dependent on a column in the particular instance of the database.

The algorithm for H is to count the number of employees in the department for each instance.

Later we will show an example where FDs are computable from other FDs, the database instance and t[X].

4.2 Splitting Attributes

In many cases the uniqueness assumption is salvaged by "splitting" an attribute into two distinct ones. As an example, [DF] consider f:EMP---> DEPT, g:EMP--> FLOOR and h:DEPT-->FLOOR, with the semantic meanings, g(EMP) is where the employee works and h(DEPT) is where the main department office is located. Using transitivity, we get h f:EMP. --> FLOOR which has a different semantic meaning than g, violating the uniqueness assumption. The database administrator splits FLOOR into two attributes, EMP.FLOOR and DEPT.FLOOR. For some purposes it will still remain natural to treat FLOOR as a single attribute, as in FLOOR-->VOLUME.OF.FLOOR, FLOOR-->NUMBER.OF.WINDOWS.ON.FLOOR, etc. Because of splitting the extra FDs EMP.FLOOR--->VOLUME.OF.FLOOR and DEPT.FLOOR--->VOLUMEOF.FLOOR must be added. In addition the attribute VOLUME.OF.FLOOR must be split into VOLUME.OF.EMP.FLOOR and VOLUME.OF.DEPT.FLOOR for the same reasons as above. To show where this can lead take the FDs, SS#-->HOME.ADDRESS and SS#--->BUSINESS.ADDRESS and the chain of natural FDs ADDRESS-->ZIPCODE-->STATE-->GOVERNOR-->PARTY. This could lead to an attribute, HOME.ZIP.STATE.GOVERNOR.PARTY (though HOME.PARTY would do but its meaning would be obscure). This train can indeed be very long causing an enormous proliferation of attributes. Note that the split attribute names actually impart the path taken to them and perhaps this should be a hint as to the direction research should follow.

Smith and Smith solve this [illegible] attribute for address,but in doing so they leave the realm of "flat databases" since address is no longer atomic and the relation would not be in 1NF. The vast majority of researchers in relational databases assume the 1NF and without it both the hierarchical and network databases can be formulation in the relational form [JA].

We have seen two causes for splitting. In the case of EMP--->FLOOR and DEPT--->FLOOR the attribute FLOOR plays two roles and formally must be treated as two distinct attributes. The functional relationship, SS#--->ADDRESS is no a multivalued relationship, but rather should be represented as two different FDs. Since both FDs have different "semantical" ranges which are subsets of the original meaning of the domain of ADDRESS, the uniqueness assumption requires that we split ADDRESS into HOME.ADDRESS and BUSINESS.ADDRESS. Thus we see that the definition of an FD f:X-->Y contains an implicit assumption that the "semantical" range of f and the dom(Y) are the same. The Smith and Smith solution (of retainig the generic attribute ADDRESS) has a drawback (for relational database formalists), in that we must be able to go from ADDRESS to HOME.ADDRESS and BUSINESS.ADDRESS and back. Even though the connection between ADDRESS and HOME.ADDRESS is the identity and hence injective it clearly violates the uniqueness assumption. We will take up the question when we discuss injective FDs.

4.3 Injective FDs

Injective FDs present a more serious problem to the formalism. Consider the classical example of f:EMP--->MGR and g:MGR<--->EMP, where g(MGR) means the managers employee number. This gives us two distinct FDS from EMP-->EMP (one the identity the other gf:EMP-->EMP). This can be temporarily solved by changing g to g:MGR-->MGR.EMP. Unfortunately this leads to additional problems. Assume we have a hierarchy of managers (manager of managers, etc.); how would the manager of a manager be determined. If the manager is treated as a regular employee in EMP-->MGR an FD EMP.OF.MGR<-->EMP is needed which again causes a violation. Otherwise a EMP.OF.MGR--->MGR.OF.MGR is needed, and so on until the highest manager. This is analogous to a geneology database with a Son, Father relation. In such a case attributes for Grandfather, Greatgrandfather, etc. until Adam. With such a procedure it is possible to create an infinite sequence of attributes. The only feasible solution is to outlaw problematic derivations and consider MGR-of-MGR etc. as computable.

If f:X<-->Y is an isomorphism then attributes X and Y represent different aspects of the same "entity". If f:X<-->Y is injective and the "semantic" range of f is a subset of dom(Y) then, if we wish to conform with the above implicit assumption, we must split the attribute Y so that the resulting FD is onto (surjective). But we have seen that splitting does not work for f:MGR<--->EMP, since it would require and infinite sequence of splitting. The semantic meaning that f:MGR<--->EMP is meant to

import is that each manager is also an employee of that dom(MGR)$\subseteq$dom(EMP). This is true even if f is not the identity since dom(MGR) is isomorphic to a subset of dom(EMP) via f in any case. In order to salvage the formalism such FDs must treated in a special way.

4.4 Computable FDs

One may ask if computable attributes should be in a database altogether. The answer is that in general they should not. If the computation is cheap it can be recalculated every time there is a query. Even if not the attribute should be virtual in the following sense:

1. The attribute should be attached to an appropriate relation- but not be considered in the relational schema and should not take part in derivations of FDs or decisions about the various normal forms.

2. Every time an update is made that affects the value of the virtual attribute in a tuple, it should be recalculated.

3. It should be included among the attributes for queries.

Thus computable attributes would not cause any anomalies and need not be considered for the construction of normal forms. Of course if F:A--->B was computable and a user defined an FD B--->C, the database manager would have to include A--->C in the set of

FDs (I have not found an example of a .--->C which was not in itself computable). The only problem this would present is for injective computations F:A-->B and F^{-1}:B-->A (as is the case for A+B=K). Then we would have to decide which was more "basic" and this may not be known by the user. Hence in such cases it is better to leave them in and consider them for the normal forms. It seems that in practice computable FDs are included among the attributes of databases even if they are not one-to-one. This is the case for some of the examples in Beeri and Bernstein [2].

There are computable FDs that depend on other FDs, the database instance and t[X]. take the FDs H:DEPT--->NUMBER.OF.EMP and g:DEPT--->MGR, where one manager may manage more than one department. We can define a computable G:MGR--->NUMBER.OF.EMP that depends on H and g, by adding up the H(dept) such that g(dept)=mgr for a given mgr dom(MGR). Later we develop a syntax for handling such computations. Given the FD f:PERSON-->FATHER, the grandfather, greatgrandfather etc. of a person becomes computable from f by f(f(p)), f(f(f(p))) etc. This assumes that dom(FATHER)$\subseteq$dom(PERSON) which causes an intrinsic problem to the FD formalism.

5. A relation to subset constraints

Beeri [BP] suggested that "FD"s that represent a subset constraint, as MGR--->EMP, should not be treated as an FD. Instead we allow this functional relationship to be expressed by a subset constraint, i.e. MGR$\subseteq$EMP. Note that A B does not necessarily mean that dom(A)$\subseteq$dom(B), because the MGR ID does not have to be equal to the managers EMP ID (the EMPs may six digit numbers and the MGR s may be five digit, for instance). A$\subseteq$B means that the functional relationship between dom(A) and dom(B) is an injection (which is often the identity) and any attribute that B has A also has, i.e. A and B represent the same type of entity and they both have common properties (though A may specific attributes not related to B).

It is also possible to have two attributes that represent the same type of entity but neither is a subset of the other. Beeri calls these compatible attributes and we denote this constraint by COMP(A,B). As an example take a parole officer, PO, that determines a social worker, SW. If we treated PO-->SW as an FD we would obviously have the same problems as before since both are subsets of Person and both would have salaries deriving to FDs from PO--->SALARY. Instead we describe the relationship by PO---}SW and COMP(PO,SW). We need the PO---}SW since COMP(PO,SW) merely states that the attributes are compatible and does not give any functional direction. We may think of ---} as a "dead end" FD, since it will not play a role in the derivations. Note that A$\subseteq$B implies that COMP(A,B) and A---}B. In most cases we will

have comp(A,B) when there is a third attribute C such that A⊆C and B⊆C.

The difficulties that develop with injective and compatible FDS between attributes belonging to the same type of entity can be solved by using the "dead end" FDs, A---}B. Functional relation of the form A---}B must not be used in pseudo-transitivity (which analogous to the natural join). If A---}B, B may end up in a relation where A is the key, but will not cause any anomalies since nothing in the relation will be dependent on B. So, for the relation containing A in a key, nothing is transitively dependent on B and B cannot be an essential part of any key. If A---}B is the identity then B could be left out completely, as in the case of computable FDs, and only be used for queries. Later we will discuss the join on compatible attributes for queries.

The proliferation of attribute names, caused by splitting, can be alleviated by the above method. We can have EMP-->EMP.FLOOR, DEPT--->DEPT.FLOOR and DEPT.FLOOR, EMP.FLOOR⊆ FLOOR. Since this implies DEPT.FLOOR---}FLOOR, there would be no need for continued splitting as in the case in the section on splitting. Though the classification of a subset constraint depends on the "judgment" of the database administrator, in any conceivable case where B must be split into two attributes, B1 and B2, because there are two distinct FDs from A--->B, we will have B1, B2⊆B.

If A ⊆ B, then all the properties of A that are also properties of B (i.e. those that A inherits from B by virtue of being the same type of entity) will not appear in any relation with A in a key and will only be accessed via queries (that will have extra joining ability with conventions to distinguish attributes reached by different paths). On the other hand if A has properties specific for A (not for B) then we may have FDs of the form A--->C for those attributes.

Let us examine a small database as an example. Let us assume that we are dealing with a company that has employees, managers and only managers have assistants and company cars and the FDs concerning FLOORs that we discussed earlier. The following FDs are evident: EMP--->MGR, EMP--->DEPT, EMP--->SALARY. As mentioned above the relationship between EMP and the FLOOR he works on must be of the form, EMP--->EMP.FLOOR, since we also have DEPT--->DEPT.FLOOR. We thus get EMP.FLOOR, DEPT.FLOOR ⊆ FLOOR and the FD FLOOR--->VOLUME. We also have MGR EMP and ASST EMP, hence COMP(MGR,ASST) and we must write MGR--}ASST. On the other hand we can use the FD MGR--->CAR as a regular FD, since they are not compatible and only managers have cars. For the purposes of insert,delete and update you could not have SALARY in relation that contained MGR in its key. Only a query could construct the necessary joins.

The practical outcome of such an approach would be to shorten the derivation paths used for FDs, since paths are cut off by --}s. But this is realistic since long derivations are probably

beyond the user intuitive comprehension. The turning of different paths are now put onto the query language and there must be some uniform system to name attributes on joins over compatible types.

Let us examine how the above solution would behave with a formal treatment of relational databases. Let us assume that the schema are synthesized using Bernstein's method. With the help of the users the database administrator has to decide which attributes are subsets of other attributes or compatible attributes. In particular he will have to decide which functional relationships are of the form A---}B. Then under the "belief" that the remaining FDs (i.e. without the ---} type) are cleansed from inconsistencies, the synthesizing algorithm can be preformed on the FDs. Next each A---}B should be examined; If the relationship is the identity, then it need not appear in the schema and must only be known by the query language (since it is computable). On the other hand if A---}B is not the identity, B must appear in a relation where A is the key. Note that we are assuming that A is a single attribute, since we do not consider a pair of attributes forming a compatible entity. If A already is a key in the synthesized schema, B may be added to the appropriate relation (this must be "remembered" for the query language). Otherwise a special relation of the form R(A,B) is formed. B cannot appear in a relation where A is part of a key since B would not be fully dependent on the key causing the common anomalies.

For the sake of efficiency it may be useful to give some priority to attributes A, where A---}B, when merging equivalent keys in the synthesis algorithm. This saves unecessarily adding relations (in [Bb] the number of relations are minimal).

Beeri [BP] suggests that the query language be allowed to join on compatible attributes. In such joins, new attributes are created and they must be named in a convention that somehow specifies the query path. If we have A---}B, R(A,B,...) and S(B,C,...) the query language may preform a join on B. But the attributes appearing in S must be renamed to reflect that they came from A, i.e. the joined relation should look something like RS(A,B,A.C,...). Of course the B in R must somehow be singled out so that the query language knows how to preform the joins and renaming. This procedure creates a potential infinite sequence of attributes (as in father, grandfather, greatgrandfather etc.).

6. Diferentiating Between FD Derivations

The above solution terminates many derivation paths (i.e. join paths) for FDs by changing many to the form of A---}B. The resulting system still could (and probably does if it is large) contain violations of the uniqueness assumption. Derivation paths come into full view in the query language. Under any circumstance detection of different functional relations between the same same entities (if not attributes) is of importance. It would also be of interest to see which derivations can be equated (see [BUN]).

Classifying FDs into regular, injective and computable can be used, as a first step, in detecting different FDs between the same attributes. The program would be as follows:
The database administrator classifies the FDs and compatible attributes to the best of his knowledge. Regular FDs are written A--->B, injective FDs A<--->B and computable F:A--->B, where F is a pointer to the computation. Armstrong's axioms are adapted to include the FD classifications. Using the axioms FDs can be derived and when more than one derivation for a FD from A to B is found they can be compared. If they are of different types then there is a definite violation (which can be amended). Otherwise if they are both computable then in most cases it will be possible to see if the computations are equivalent (in general the question of equivalence of two computations is undecided for a sophisticated enough language). If both are either regular of injective then the question of uniqueness remains unanswered. In the future it may be possible to make finer classifications combined with equivalence classes of derivations, improving the detection process.

The above process is unfortunately exponential (though we have mentioned that a derivation tree of at most height two times the number of attributes would be needed to check for violations), but it is at least an automated process to search for violations. The adapted Armstrong axioms can be written as follows:

A_1 a. X<-->X

Hence the identity is not considered computable.

b. if $X<-->Y$ then $Y<-->X$

A_2 a. if $X-->Z$ (or $X<-->Z$) then $X \cup Y-->Z$

b. if $F:X-->Z$ then $F^*:X \cup Y-->Z$ where $F^*(X,Y)=F(X)$

A_3 a. if [($X-->Y$ and $Y \cup Z-->W$) or

($X<-->Y$ and $Y \cup Z-->W$) or

($X-->Y$ and $Y \cup Z<-->W$)] then

$X \cup Z-->W$

b. if $X<-->Y$ and $Y \cup Z<-->W$ then $X \cup Z<-->W$

c. if $F:X-->Y$ and ($Y \cup Z-->W$ or $Y \cup Z<-->W$) then $F^*:X \cup Z-->W$ where if f is the canonical name for $Y \cup Z-->W$ ($Y \cup Z<-->W$) then $F^*(X,Z)=f(F(X),Z)$.

d. if ($X-->Y$ or $X<-->Y$) and $F:Y \cup Z-->W$ then $F^*:X \cup Z-->W$ where if f is the canonical name for $X-->Y$ ($X<-->Y$) then $F^*(X,Z)=F(f(X),Z)$

e. if $F_1:X-->Y$ and $F_2:Y\ Z-->W$ then $F^*:X \cup Z-->W$ where $F^*(X,Z)=F_2(F_1(X),\ Z)$,unless $Z=\emptyset$ and F^* is the identity function in which case we have $X<--->X$.

Notes:

1. In A_3 c. and d. F^* is computable in f, which is regular (or injective). F^* is not fully computable, since it depends on f

which changes in the database instance does. Our goal, in regard to computable FDs, is to equate or differentiate between them and to differentiate them from non- computable FDs. This goal can sometimes be accomplished by defining F^* as computable as we shall see later.

2. The identity function is not considered computable, since otherwise any FD would be computable by A_3 c.

3. In A_3 we did not want F^* to be declared computable if it was the identity function. We shall assume that it is possible to decide that F^* is the identity, though theoretrically there are pathological cases where this could not be decided. The detection of the identity would normally occur where F:A--->B is an injection and we also have F^{-1} (and the computable attributes were retained as mentioned above).

We shall illustrate how the above classification of FDs can be used to automatically detect violations (that may be missed by the database administrator) by using the some examples given in Beeri and Bernstein [BB]. In order to define computable functions we need a function manipulation language (we cannot use relations since we are only given FDs). We will develop only sufficient tools to illustrate our examples intuitively.

Non-computable FDs (denoted by lower case letters) are treated as atoms. We allow composition of functions (this was already used in the adapted axioms). Let $\alpha : A \rightarrow B$ (Greek letters represent both computable and non-computable FDs). Let $\overline{dom(A)}$ represent the finite subset realized in a database extension. Let $\alpha(A)$ represent the set $\{\alpha(a) : a \in \overline{dom(A)}\}$. For $b \in \overline{dom(B)}$, $\chi_{\alpha(A)=b}$ is the characteristic function of the predicate $\alpha(A)=b$ defined over dom(A), i.e., $\chi_{\alpha(A)=b}: dom(A) \rightarrow \{0,1\}$ defined by:

$$\chi_{\alpha(A)=b}(a) = \begin{cases} 1 & \text{if } \alpha(a)=b \\ 0 & \text{otherwise} \end{cases}$$

We allow taking the sum of a set hence if f:EMP-->DEPT then we can define a computable function F, F:DEPT-->NUMBER.OF.EMPS by:

$$F(d) = \sum_{\overline{dom(EMP)}} \left(\chi_{f(EMP)=d} \right) \text{ for } d \in \overline{dom(DEPT)}.$$

Thus F would count the number of employees in a department.

Let g:DEPT-->MGR,we can define a computable G:MGR-->NUMBER.OF.EMPS by:

$$\ldots = \sum_{\overline{dom(DEPT)}} [(\chi_{g(DEPT)=m})*(f(DEPT))],$$

which computes the number of employees for a particular manager.

The above is just the informal embryo of a language but it is enough for our own purposes.

Example 1. We are given f_1:DEPT-->MGR, f_2:EMP-->DEPT, FLOOR, F_3:DEPT, FLOOR-->NUMBER.OF.EMPS, and F_4:MGR, FLOOR-->NUMBER.OF.EMPS.

F_3 is computable by:

$$F_3(d,f) = \sum_{\overline{dom(EMP)}} (\chi_{f_2(EMP)=d,f}).$$

F_4 is computable by:

$$F_4(m,f) = \sum_{\overline{dom(DEPT)}} [(\chi_{f_1(DEPT)=m})*F_3(DEPT,f)].$$

Using $A_3(d)$ on f_1:DEPT-->MGR and F_4:MGR, FLOOR-->NUMBER.OF.EMPS we derive

G:DEPT,FLOOR-->NUMBER.OF.EMPS where

$$G(d,f)=F_4(f_1(d),f)=\sum_{\overline{dom(DEPT)}} [(\chi_{f_1(DEPT)=f_1(d)})*F_3(DEPT,f)].$$

Clearly an algorithm could be written which could decide whether G is equivalent to F_3. Hence there are two derivations of DEPT, FLOOR-->NUMBER.OF.EMPS with different user intents. Beeri and Bernstein's solution to this violation is to change F_4 to:

F_4:MGR,FLOOR-->NUMBER.OF.EMPS.OF.MGR

Remark. If we did not have f_2 (as is the case in the Beeri and Bernstein [BB] example, F_3 would revert to a non-computable f_3 but F_4 would remain computable. G would be computable also and a violation would be detected because there were two derivations of DEPT, FLOOR-->NUMBER.OF.EMPS one regular and the other computable.

The above discussion contains only enough detail to see that such a program is feasible and that some of the database administrators work can be done automatically. We shall continue with some more examples.

Example 2. We shall see how the violation caused by MGR EMP could be discovered. Let f_5:EMP-->MGR and f_6:MGR<-->EMP. By A_1(b) applied to f_6 we derive g_1:EMP<-->MGR and this gives two derivations of EMP-->MGR, one regular and the other injective. Here there is a violation. (We could have used transitivity on f_5, f_6 to get g_2:EMP-->EMP as opposed to the derivation of EMP<-->EMP by A_1(a).)

A more likely example would be EMP<-->SS# and EMP-->ASST<-->SS#, giving two derivations from EMP to SS#, one regular and one injective.

Example 5. Let f_7:STOCK#-->STORE and f_8:STOCK#, STORE -->QTY. The "user intent" of f_7 is to map the STOCK# onto STORE of the store that is in charge of ordering that item and f_8 maps STOCK# and STORE of the store in which it is being sold into the quantity on hand. Using $A_3(a)$ we derive g_3:STOCK#-->QTY . Then $A_2(a)$ gives g_4:STOCK#, STORE -->QTY. f_8 and g_4 represent two different intents of STOCK#, STORE -->QTY both classified regular. Hence they could not be distinguished by the classification method. The above violation is corrected by splitting STORE and with the present state of the art must be done by the "users".

7. Conclusion

After having investigated the difficulties arising from the uniqueness assumption, we conclude that the assumption hampers the ability of a relational database to represent real world situations. In particular it natural to allow more than one dependency between the same entities. On the other hand the universal relation assumption is needed schema design algorithms and other interrelational concepts. We presented a solution to the problem which splits attributes when neccessary and restricts the use of FDs when violations may occur. One drawback of the above solution is that it severely shortens the derivation paths for FDs, increases the number of attribute names. As a result the number of synthesized relations will, in general, be greater and the database will be more cumbersome.

In addition the problem of different paths between attributes is thrust up to the query language. An infinite amount of possible attributes may make it difficult for the designer and user. The problem of differentiating or equating between different functional paths seems central (with any database model). Lastly verification of the assumptions is very time consuming at best and a database administrator embarking on design of the relational schema will have to accept them as a matter of (possibly unjustified) belief. In our classification of FDs we have taken a step in the direction of automatically detecting violations of the uniqueness assumption (i.e. differentiating between paths).

Acknowledgement

Through private communication with Catriel Beeri, I have discovered that he has been considering the problems caused by the universal relation assumption. In particular the concepts of compatible and potential attributes originated with Beeri.

Bibliography

[ARM] Armstrong, "Dependency Structures of database Relationships", Proc. IFIP 74, North Holland, 1974, pp. 580-583.

[BB] Beeri and Bernstein, "An Algorithmic Approach to Normalization of Relation Database Schemas", Technical Report CSRG-73, Computer Science Research Group, University of Toronto, Sept., 1976.

[BB2] Beeri and Bernstein,"Problems in Design of Normal Form Relational Schemas", ACM Transactions on Database Systems, March 1979, Volume 4 number 1, pg. 30-59.

[BP] C. Beeri, Private communication.

[BERN] P.A. Bernstein, Synthesizing Third Normal Form Relations From Functional Dependencies, ACM Trans. on Database Systems, Vol. 1, No. 4 (Dec 1976), pp. 277-298.

[BF] Buneman and Frankel, "FQL- A Functional Query Language" SIGMOD 1979- Boston.

[BUN] P. Buneman, The Problem of Multiple Paths in a Database Scema, VLDB 1979, pp 368- 372.

[CAD] Cadiou, "On Semantic Issues in the Relational Model of Data", Mathematical Foundations of Computer Science (A. Mazurkiewiz, Ed.), Vol. 45, Springer- Verlag, 1976, pp. 23-38.

[ES] Eswarain and Chamberlain, "Functional Specifications of a Subsystem for database Integrity", Proc. of the VLBD, Farmington, Mass. 1975, pp. 48-66.

[FAG] R. Fagin, A New Normal Form For Relational Databases That is Based On Domains and Keys. IBM Research Report RJ2520 (May 1979), San Jose California.

[GM] Gallaire and Minker, "Logic and databases",Plenum Press, 1978.

[HM] Hammer and Mcleod, "Semantic Integrity in a Relational database System", Proc. of the VLBD, Farmington Mass. 1975, pp. 25-47.

[JA] B.E. Jacobs, On Database Logic, Technical Report 737, Dec. 1978, MCS 73-03433002, Univ. of Maryland.

[NIC] Nicolas, "First Order Logic Formilazation for Functional, Multivalued and Mutual Dependencies", Sigmod 1978, Austin, Texas, pp. 40-46.

[SS] Smith, J.M. and Smith, D.C.P. Database Abstractions: Aggragation and Generalization. Acm Trans. On Database Syst. 2, 2 (June 1977) pp. 105-133.