# REAL-WORLD MVD's

Edward Sciore

Dept. of Computer Science
SUNY at Stony Brook
Stony Brook, N.Y.

Abstract

According to database theory, a database designer can specify any set of data dependencies, no matter how complex, to constrain a database scheme. This paper investigates how much complexity is actually needed in real-world situations. It is shown that every "natural" set of mvd's must belong to a class of mvd's called conflict-free. Conflict-free sets of mvd's have the desirable property that they allow a unique 4NF dependency preserving database scheme; moreover, non conflict-free sets have no such normalization. If a set of mvd's is not conflict-free, then the dependencies are inadequately specified; there are semantic concepts that are unrepresented in the scheme. These concepts are isolated, and it is shown that adding these concepts amounts to making the set of mvd's conflict-free.

## 1. Introduction

A critical part of relational database design is the selection of attributes; the set of attributes provides a definite and specific meaning for the value in a relation. For example, a scheme may contain the two attributes SUPP and PART. These attributes were chosen by the database designer to denote a certain relationship. A tuple (s,p) might denote the fact that supplier s is currently supplying part p, or that s is able to supply p, or that p is out of stock for s; but whatever the meaning, it is determined once by the database designer and does not change.

Although the actual meaning of the attribute realtionships is unknown to a database system, the structure of the relationships can be specified, using data dependencies. A data dependency, for example, can assert that one attribute value functionally depends on another, or that a set of values is independent of another set. In specifying a set of data dependencies, the database designer describes the structure of the database. A user interacts with the database through this structure; consequently, the structure should be simple and easy to understand -- otherwise, the database designer cannot be sure that he has correctly represented all of the relevant semantics, and the user will not be able to understand the interrelationships between the data.

In theory, the database designer can specify any possible set of data dependencies in his scheme. This possibility has caused trouble for automatic synthesis and decomposition algorithms; there may be several different candidate schemes, none of which can be considered best [BBG]. In "real-world" situations, this problem does not seem to arise; once a proper set of attributes has been chosen, a natural database scheme always seems to present itself. What happens to the structural complexity that current theory allows? Is it ever really needed?

In this paper, we investigate these questions with respect to the set of multivalued dependencies in the scheme. Our results imply that not only are complex sets of mvd's undesirable and avoidable, they are also unnatural; that is, for every complex set of mvd's, there is a simpler set that represents the application better. Therefore, the existence of a complex set of dependencies indicates a poorly designed scheme, rather than a complex application.

The formal criterion for a set of dependencies

to be simple is provided in Section 3, where we define conflict-free. Conflict-free sets of mvd's have several desirable properties. For any conflict-free set G of mvd's, there is a single join dependency $D=*[X_1,\ldots,X_n]$ such that G is equivalent to D. D belongs to a special class of join dependencies which itself has interesting properties [B+]. Furthermore, the database scheme $\{X_1,\ldots,X_n\}$ is in 4NF and preserves dependencies. Thus, $\{X_1,\ldots,X_n\}$ is the best possible database scheme representing G.

Our contention is that the notion of conflict-free characterizes the simple and natural sets of mvd's. In Section 4, we demonstrate the undesirability of non conflict-free sets of mvd's; they are impossible to normalize adequately, since they cannot have a dependency preserving 4NF database scheme. In Section 5, we show why they are unnatural. If a set of mvd's is not conflict-free, then it either has a transitive anomaly or a split-key anomaly. The presence of one of these anomalies indicates that the scheme is inadequately specified. We investigate where this inadequacy is, and show that by fixing the inadequacies, we make the mvd set conflict-free.

Our results have a strong impact on data dependency theory. The only sets of mvd's that need ever be considered are conflict-free, and conflict-free sets of mvd's are equivalent to one join dependency. Therefore, the database designer can spend his time looking for the one join dependency rather than specifying mvd's. The implications of this point are discussed in the conclusion.

## 2. Preliminaries

The state of relational database theory is such that there is no standard notation currently in use. The reader can find a good introduction to relational theory in [U1]; we shall adopt the notation and terminology of that book. In addition, familiarity with the problems and concepts of data dependency theory [BBG, MMS] is useful.

A universe U is a finite set of attributes. As a convention, we use the symbols A,B,C... for single attributes, and Z,W,X... for sets of attributes. If X is a set of attributes, the $|X|$ is the number of attributes in X. The union of attribute sets X and Y is written XY. Associated with each attribute A is a set of values, called the domain of A. An X-tuple is a mapping from each attribute A in X to a value in the domain of A. If t is an X-tuple and $R \subset X$, then t(R) is the R-tuple which is the restriction of t to R. If the set X is ordered, an X-tuple can be thought of as a row of values. A relation over X is a set of X-tuples. When X is the universe U, then the relation is called a universal relation.

Let r be a relation over R. The attribute set R is called the relation scheme or r. A database scheme is a set of relation schemes. Let $S=\{X_1,\ldots,X_n\}$ be a database scheme. Then $X_i$ is maximal in S if there is no $X_j$ in S such that $X_i \subset X_j$; max(S) is the set of all maximal relation schemes in S.

If R is a relation scheme, a data dependency for R is a constraint on the relations over R that are considered meaningful. In this paper, we shall consider functional dependencies (fd's), multivalued dependencies (mvd's), and join dependencies (jd's). Let r be a relation over R, let X and Y be subsets of R, and let Z=R-X-Y. Then r satisfies the FD X->Y if for all tuples s and t in r, s(X)=t(X) implies that s(Y)=t(Y). Relation r satisfies the mvd X->->Y if for all s and t in r such that s(X)=t(X) there exist tuples u and v in r such that u(XY)=s(XY), u(Z)=t(Z), v(XY)=t(XY), and v(Z)=s(Z). Join dependencies are defined later in this section.

For the remainder of this paper, we shall assume that any set of dependencies are defined over the universe U; also, unless otherwise specified, "relation" means "relation over U". The more general case involves what are known as "embedded dependencies", and is an important research problem.

Let G be a set of data dependencies. Then sat(G) is the set of all relations (over U) that satisfy every data dependency in G. We say that G entails D for a data dependency D if every relation satisfying the data dependencies in G also satisfies D; that is, sat(G)=sat(G∪{D}). For example, the reader can check that {X->Y} entails X->->Y. The set G is trivial if every relation is in sat(G). There is a complete set of inference rules for fd's and mvd's [BBG]; D can be derived from G using these rules if G entails D. The

closure of G, written $G^*$, is the set {D|G entails D}. If G' is another set of data dependencies such that $G'^*=G^*$, then G' is called a cover of G.

Two important operations on relations are projection and join. Let r be a relation over R, and let XcR. Then the X-projection of r, written $\pi_X(r)$, is the relation {t(X)|t is in r} over X. Let $r_i$ be a relation over $R_i$ for i between 1 and n. The join of $r_1,\ldots,r_n$, written $*\{r_1,\ldots,r_n\}$, is the relation {t|t is a tuple over $UR_i$, and for each i there exists a tuple $t_i$ in r, such that $t(R_i)=t_i$} over $UR_i$.

Let $\{X_1,\ldots,X_n\}$ be a database scheme. Then the relation r satisfies the join dependency (jd) $*[\{X_1,\ldots,X_n\}]$ if $r=*\{\pi_{X1}(r),\ldots,\pi_{Xn}(r)\}$. For notational convenience, we drop the set brackets when writing jd's; thus $*[\{X_1,\ldots,X_n\}]$ is written $*[X_1,\ldots,X_n]$. For example, it is well known that if a relation r satisfies the mvd X->->Y, then r also satisfies the jd *[XY,XZ], where Z=U-X-Y. If $D=*[X_1,\ldots,X_n]$ and $D'=*[Y_1,\ldots,Y_m]$ are two jd's, then it is known [BMSU] that D entails D' if for all $X_i$ there exists a $Y_i$ such that $X_i \subseteq Y_j$. If D does not entail D' and D' does not entail D, then D and D' are called incomparable.

One purpose of data dependency theory is to be able to syntactically determine when a database scheme S is a good model of an application. One standard criterion is that S be in fourth normal form (4NF). Let G be a set of data dependencies. Then S is in 4NF if *[S] is in $G^*$, and for all R in S, if XYcR and X->->Y is in $G^*$, then X->Y is in $G^*$. 4NF is desirable, because it ensures that redundancy in relations is minimized [BBG]. Another desirable criterion is dependency preservation.

Definition [S2]. Let S be a database scheme, F a set of fd's, and G a set of jd's. Then S preserves F if there is a cover F' for F such that for all X->Y in F', there exists an R in S such that XYcR. S preserves G if for all $*[X_1,\ldots,X_n]$ in $G^*$ and R in S, $*[X_1 \cap R,\ldots,X_n \cap R]$ is in $G^*$. []

## 3. Conflict-free Mvd's

Definition. Let G be a set of mvd's for U. Then a set of disjoint relation schemes $\{Y_1,\ldots,Y_n\}$ is a dependency basis for a relation scheme X if X->->$Y_i$ is in $G^*$, $Y_i \neq \emptyset$, $X \cap Y_i=\emptyset$ for each i, and for any mvd X->->Z entailed by G, Z-X is the union of some of the $Y_i$'s. We write $DEP(X)=\{Y_1,\ldots,Y_n\}$, and X->->$Y_1|\ldots|Y_n$. []

In other words, a dependency basis for X is the finest partition $Y_1,\ldots,Y_n$ of attributes in U-X such that X->->$Y_i$. It is known [B] that a dependency basis must exist for any X and G. A dependency basis is a useful canonical form in which to express mvd's.

Definition. Given a set G of mvd's, relation scheme X is a key of G if there exists an mvd X->->Y in G. Each Y in DEP(X) is called essential dependent of X if X->->Y cannot be derived from G without using an mvd having left hand side X. A key is essential if it has an essential dependent; an essential key X is minimal if there is no essential key ZcX. []

Example. Let G be the set of mvd's {X->->ABC|D, XA ->->B|C|D}. Then D is not an essential dependent of XA, since XA->->D is entailed by X->->D. However, B and C are essential dependents of XA, so XA is an essential key. []

It is easily seen [L] that every cover of a set G of mvd's has the same essential keys. Since the set consisting of no mvd's is trivial and has no essential keys, it follows that G has no essential keys iff G is trivial.

Definition [L]. Let G be a set of mvd's. Then G is conflict-free (cf) if for any two essential keys X and Y of G, the following condition holds.

$$DEP(X)=\{V_1,,,,,V_k,X_1,\ldots,X_i,(Z_kY_1\ldots Y_j)\}$$

and

$$DEP(Y)=\{V_1,\ldots,V_k,Y_1,\ldots,Y_j,(Z_yX_1\ldots X_i)\}$$

where

$$\{V_1,\ldots,V_k\} \subseteq DEP(X \cap Y) \text{ and } Z_xX=Z_yY. \text{ []}$$

The reader can verify that the above example set G of mvd's is conflict-free.

The definition of conflict-free is specifically designed so that the following property holds.

Proposition 3.1 [L]. A cf set of mvd's has a unique 4NF database scheme. []

Lien does not directly determine this database scheme S; in order for us to do so, we shall need some definitions. Let $\{X_1,\ldots,X_n\}$ be the essential keys of a set G of mvd's (not necessarily cf) over attributes R, and let $DEP(X_k)=\{Y_{k1},\ldots,Y_{km}\}$ for each k. That is, we have

$$X_1 \text{->->} Y_{11} | Y_{12} | \ldots | Y_{1m}$$
$$\vdots$$
$$X_n \text{->->} Y_{n1} | Y_{n2} | \ldots | Y_{nm}$$

Let $j = (j_1, \ldots, j_n)$ be a sequence of n integers; call j legal if $1 \leq j_k \leq m_k$ for all k. A legal sequence j can be thought of as choosing the $j_k$th dependent of each essential key $X_k$. Call $S = \{X_1Y_{1j}, \ldots, X_nY_{nj}\}$ the selection set for the sequence j; the common attributes of S are $W_j = \{Z | Z \text{ is in } S\}$. That is, $W_j$ is the maximum set of attributes such that $W_j \subseteq X_kY_{kj}$ for all k. We now define the database scheme $S_G$ to be the maximal elements in the set of all possible $W_j$'s. Formally, we have

$$S_G = \max(\{W_j | j \text{ is a legal sequence}\}).$$

Example. Let G= {X->->ABC|D, XA->->B|C|D} as in the above example. Then $S_G$={XD, XAB, XAC}. []

The definition of $S_G$ has the following interpretation: If X->->Y|Z holds, then any relationship between Y-values and Z-values must be a derived one; that is, yz is in $\pi_{YZ}(r)$ iff there exists an x such that xy is in $\pi_{XY}(r)$ and xz is in $\pi_{XZ}(r)$. Therefore, a good database scheme should separate Y from Z. The database scheme $S_G$ does exactly that. In the above example, BC cannot appear in any R in $S_G$, since XA->->B|C|D splits B and C; on the other hand XAB is in $S_G$ since none of the mvd's split X,A, and B. $S_G$ can be thought of as the database scheme that group together as many "unsplittable" attributes as possible.

It turns out that if G is conflict-free, then $S_G$ is the database scheme mentioned in Proposition 3.1. The proof of this fact is given in the following series of lemmas.

Definition. For an essential key X, Y in DEP(X) is basic for X if there does not exist an essential key $Z \neq X$ and W in DEP(Z) such that $ZW \subseteq Y$. []

Example. Let G={A->->BDEHI|CFJ,
AB->->DHI|E|CFJ,
ABD->->H|I|E|CFJ,
AC->->F|J|B|DEHI}

Note that G is cf. Dependents H and I are basic for ABD, E is basic for AB, and F and J are basic for AC. There are no other basic dependents. []

Lemma 3.2. For any nontrivial cf set G of mvd's there exists an essential key X and a Y in DEP(X) such that Y is basic for X.

Proof. If G is nontrivial, then there exists an essential key. Choose X and Y such that X is a minimal essential key, Y is in DEP(X), and for all minimal essential keys $Z \neq X$, if W is in dep(Z) then $|XY| \leq |ZW|$. Suppose Y is not basic for X; then there exists an essential key Z with dependent W such that $ZW \subseteq XY$. We can assume without loss of generality that Z is minimal, so XnZ is not an essential key. By the definition of cf, the dependency bases of X and Z must be

$$X \text{->->} Y | Y_1 | \ldots | Y_m$$
$$Z \text{->->} Z_1 | \ldots | Z_k | Q$$

where $Y = V_xZ_1 \ldots Z_k$, $Q = V_zY_1 \ldots Y_m$, $W = Z_k$, and $V_xX = V_zZ$. Since $|XY| \leq |ZZ_i|$ for any i, it must be that k=1 and $V_x = \emptyset$; that is, Y=W. So if $ZW \subseteq XY$, then $Z \subseteq W$ -- which is a contradiction. []

Lemma 3.3. If Y is basic for X, then for all A in Y and all essential keys Z, A is not in Z.

Proof. Suppose WA is an essential key. Since Y is basic for X, WAX is not a subset of XY for any Z in DEP(WA). So we must have the following dependency bases.

$$X \text{->->} Y | Y_1 | \ldots | Y_m$$
$$WA \text{->->} W_1 | \ldots | W_k$$

Since A is in Y, the definition of cf implies that XY=WAZ for some Z. Consequently, since WA is not a subset of XY, there must be a dependent $W_i$ of WA such that $W_i \subseteq Y$. Thus $WAW_i \subseteq XY$, which is a contradiction. []

Lemma 3.4. If Y is basic for X, then XY is in $S_G$.

Proof. From Lemma 3.3, we know that each A in Y is not in any essential key. The definition of cf then implies that each essential key $W \neq X$ must have the attributes XY in the same dependent; that is, there must exist a Z such that XYZ is in DEP(W). Therefore, there exists some R in $S_G$ such that $XY \subseteq R$. But Y in DEP(X) requires that $R \subseteq XY$. Thus, XY is in $S_G$. []

Lemma 3.5. Let G be a cf set of mvd's such that each mvd is essential, and let Y be basic for X. Let G' be G with the attributes of Y removed. Then G entails G'.

Proof. It is well known that removing an attribute from the right hand side of an mvd D leaves an mvd entailed by D. Lemma 3.3 ensures that the

attributes in Y only appear in the right hand side of mvd's of G. []

Lemma 3.6. Let Y be basic for X, and let G and G' be as in Lemma 3.5. Then $S_G=\max(S_G,u\{XY\})$.

Proof.

$\supseteq$ : Suppose R is in $S_{G'}$. Then by the definition of $S_{G'}$, either RcXY or R is in $S_G$. Since XY is in $S_G$ from Lemma 3.4, the containment is proved.

$\subseteq$: Consider any W in $S_{G'}$. If WnY=$\emptyset$, then clearly W is in $S_{G'}$. If there exists an A in WnY, then by Lemma 3.3 and the definition of cf there must exist a $Q_Z$ for each essential key Z such that $Q_ZXY$ is in DEP(Z). Thus W=XY. []

We are now in a position to prove our theorem.

Theorem 3.7. If G is cf, then G entails $*[S_G]$.

Proof. We can assume without loss of generality that every mvd in G is essential. Lemma 3.2 states that if G is nontrivial, then there exist X and Y such that Y is basic for X. Let G' be as in the previous lemmas. Since $S_G=\max(S_G'u\{XY\})$, we need only to show [S1] that G entails $*[S_{G'},XY]$. Using the chase procedure of [MMS], we know that because X->->Y, in order to show that G entails $*[S_{G'},XY]$ it suffices to show that G' entails $*[S_{G'}]$. Now clearly G' is cf, and by Lemma 3.5 is entailed by G. Therefore G' has its own set of essential keys (which will be a subset of those of G), and if nontrivial, has some basic attributes. We continue inductively until G' becomes trivial. Since a trivial set of mvd's over attributes R has $S_G=*[R]$ (the trivial jd), the result follows. []

The technique used in proving this theorem provides a means to construct $S_G$, given G; consider the following example.

Example. Let G be the set of mvd's of the previous example. Since attributes F and J are basic for AC, ACF and ACJ are in $S_G$. So we can remove FJ from G to get

G'={A->->BDEHI|C,
AB->->DHI|E|C,
ABD->->H|I|E|C,
AC->->BDEHI}.

This last mvd is nonessential, so we remove it. H and I are still basic for ABD, and now C is basic for A. Since AC is not maximal, we add only ABDH and ABDI to $S_G$, and remove attributes CHI from G'. The resulting mvd set is

{A->->BDE,
AB->->D|E,
ABD->->E},

of which only AB->->D|E is essential. Here, D and E are basic for AB; ABD is not maximal, so we add ABE to $S_G$ and remove attributes DE from the dependencies. The final set is

{AB->->$\emptyset$},

which is trivial. Since AB is not maximal, it is not added to $S_G$. $S_G$ is therefore {ACF,ACJ,ABDH, ABDI,ABE}. The reader can verify that G actually does entail the jd *[ACF,ACJ,ABDH,ABDI,ABE]. []

Corollary 3.8. $S_G$ is a 4NF database scheme.

Proof. Suppose not. Then there must exist an R in $S_G$ and X->->$Y_1|\ldots|Y_n$ in G such that XcR and there exists i and j such that $Y_i$nR$\neq\emptyset$ and $Y_j$nR$\neq\emptyset$. However, if this were the case, then clearly the definition of $S_G$ would imply that R is not in $S_G$. []

Theorem 3.7 and Corollary 3.8 show that when G is cf, $S_G$ is indeed the 4NF database scheme of Proposition 3.1, and is therefore unique. We now consider $S_G$ when G is arbitrary.

Theorem 3.9. For any set G of mvd's, $*[S_G]$entails G.

Proof. The definition of $S_G$ implies that every attribute appears in $S_G$. It therefore suffices (from the results of [BMSU] ) to show that for all essential keys X and R in $S_G$, there exists a Y in DEP(X) such that RcXY. But this fact follows directly from the definition of $S_G$. [ ]

Not only does $S_G$ entail G, but it is the weakest jd to do so, as the following corollary states.

Corollary 3.10. Let G be any set of mvd's, and let D be a jd. If D entails G, then D entails $S_G$.

Proof. If D entails G, then for every R in D and essential X->->$Y_1|\ldots|Y_n$ in G, there must exist an i such that R$\subseteq$XY$_i$. Since this is true for all essential keys X, the definition of $S_G$ implies that there is a Z in $S_G$ such that XY$_i\subseteq$Z. Thus R$\subseteq$Z, which implies that D entails $S_G$. [] ·

Theorems 3.7 and 3.9 show that if G is a cf set of mvd's, then G is equivalent to the single jd $*[S_G]$. The converse also holds.

Proposition 3.11. [Y] Let G={X->->Y|D entails X->->Y} for some jd D. Then G is cf. []

If we start with a cf set G of mvd's, then Theorems 3.7 and 3.9 imply that the mvd's entailed by $*[S_G]$ will be $G^*$. Now suppose we start with a jd D and find the set G of mvd's entailed by D. Then $*[S_G]$ will certainly be entailed by D (since D entails G which entails $*[S_G]$), but it may not be equivalent to D. For a simple example, let D=*[AB,AC,BC]. Then there are no non-trivial mvd's entailed by D, so $*[S_G]$=*[ABC], the trivial jd.

The results of this section show that a set G of mvd's is equivalent to jd $*[S_G]$ iff G is cf. It is interesting to note that there exist several very different characterizations of the class of jd's that are equivalent to their implied mvd's [B+]. Inasmuch as this class is quite general and has many desirable properties, it is natural to wonder whether there are any real-world schemes that do not belong to this class. Sections 4 and 5 consider this question in detail.

## 4. Normalization Considerations

In the previous section we saw that cf set G of mvd's is equivalent to exactly one jd, namely $*[S_G]$. $S_G$ is a "perfect" 4NF, dependency preserving database scheme representing G. Conversely, if G is not cf, then by Proposition 3.11, there must be two incomparable jd's entailed by G. Suppose, for example, that G={A->->B|C, B->->A|C}. Two incomparable jd's entailed by G are *[AB,BC] and *[AB,AC]. What would be a good database scheme representing G? Consider the database scheme {AB,AC}. Our definition of jd-preservation implies that the two jd's *[ACnAB,ACnBC] and *[ABnAB,ABnBC] hold. The latter jd is *[AB] which trivially holds, but the former jd is the cartesian product *[A,C], which does not hold. Thus, {AB,AC} does not preserve the jd *[AB,BC]. The reader can check that ABC is the only database scheme that preserves both jd's. Since a relation over ABC contains redundant information, however, this scheme also is not entirely satisfactory.

In general, whenever there are two incomparable jd's, there is a problem in finding a 4NF dependency preserving database scheme.

Theorem 4.1. Let $\{R_1,\ldots,R_n\}$ be a 4NF database scheme, and suppose $*[R_1,\ldots,R_n]$ does not entail $*[S_1,\ldots,S_m]$. Then $[R_1,\ldots,R_n]$ does not preserve $*[S_1,\ldots,S_m]$.

Proof. Define the attribute set M={A| there exist i, j such that A is in $S_i \cap S_j$}; that is, M is the set of attributes appearing in more than one $S_k$. Since $\{R_1,\ldots,R_n\}$ is in 4NF, it can be shown that whenever $M \subseteq R_i$ there is an $S_j$ such that $R_i \subseteq S_j$ (otherwise, $*[S_1,\ldots,S_n]$ would split $R_i$). It is known [BMSU] that if $*[R_1,\ldots,R_n]$ does not entail $*[S_1,\ldots,S_m]$, then there exists an $R_k$ such that for all $S_j$, $R_k$ is not a subset of $S_j$. Thus M is not a subset of $R_k$. Now consider the jd $*[R_k \cap S_1,\ldots, R_k \cap S_m]$. Since $R_k$ is not a subset of $S_j$ for all j, the jd $D=*[R_k \cap S_1,\ldots,R_k \cap S_m]$ is non-trivial. Since M is not a subset of $R_k$, it follows that there must exist i, j and some attribute A such that A is in both $S_i - R_k$ and $S_j - R_k$. It is known [S1] that this fact implies that D is not entailed by $*[R_1,\ldots, R_n]$ and $*[S_1,\ldots,S_m]$. The result follows. []

Corollary 4.2. Let G be a set of mvd's. Then G has a dependency preserving 4NF database scheme iff G is conflict-free.

Proof. If G is cf, then $S_G$ is the required database scheme. If G is not cf, then by Proposition 3.11 there are at least two incomparable jd's entailed by G. The result then follows from Theorem 4.1. []

Until this point we have been considering sets of data dependencies that contain only mvd's. When we also consider fd's, complications arise. For example, let G={A->B, B->C}. The set of mvd's entailed by G is {A->->B|C, B->->A|C}, which is not cf. However, the database scheme {AB,BC} is a well-known "optimal" dependency preserving database scheme representing G. Although this scheme does not preserve the jd *[AB,AC] according to our definition, it does not need to, as long as it preserves the fd A->B. That is, as long as A->B holds in relation AB, the jd*[AB,AC] will be preserved.

This extra power of fd's means that our notion of cf should not apply to mvd's entailed by fd's. Recall the definition of an essential dependent in Section 3; Y is an essential dependent of X if X->->Y can be derived only by using some mvd with left hand side X. We can extend the definition to fd's by including the inference rule "X ->Y derives X->->Y." So if X->Y, then Y is not an essential dependent of X. In G={A->B, B->C}, for example, there are no essential keys; G is trivially cf.

Although this paper is concerned only with "real-world" sets of mvd's, we shall need to consider fd's briefly (see Theorem 5.3). The above treatment shall be satisfactory for our purposes, although we do not claim that it is entirely correct. That is, in the next section we shall show that every non-cf set of mvd's is in some way inadequate and "unreasonable"; however, it is an open problem to determine whether the reasonableness of cf mvd's, as developed in these last two sections, extends to fd's as well.

## 5. Improving Schemes That Are Not Conflict-Free

Conflict-free sets of mvd's have been shown easy to normalize, whereas non-cf mvd's pose insurmountable normalization problems. Are these problems avoidable? We contend that they are. A well-formed set of dependencies for an application can only have so much complexity; if the dependencies are not cf, then the scheme is unduly complex, and not well thought out. In this section we will argue that every "real-world" situation that can be modeled using mvd's is modeled by a cf set of mvd's. Thus a database designer need only consider cf sets of dependencies -- if his dependencies are not cf, then he knows something is wrong.

There are several things that might be wrong. An important situation is the case where the database designer simply makes a mistake, as in the following example.

Example. Consider the scheme with attributes EMP, DEPT, and FLOOR. We want to model the fact that each employee works in a set of departments and each department is located on a set of floors. A first attempt at specifying mvd's gives {EMP->->DEPT, DEPT->->FLOOR}, which is not cf. However, the first mvd is incorrect; DEPT and FLOOR belong together in each mvd, and therefore EMP->->{DEPT,FLOOR} is correct. This new scheme is cf. []

Mvd's are sufficiently difficult to understand that such mistakes are not unlikely. Therefore, an immediate advantage of requiring cf mvd's is that these simpler errors are easily caught. It is also possible for a scheme to be correctly specified, and yet the mvd's are not cf. In this case, we claim that the scheme is inadequately specified.

Consider the following scheme.

Example. Suppose we have the scheme (AUTH TITLE LOC), which represents authors, titles of their books, and the locations where the books are published. Futhermore, we have the data dependencies AUTH->->TITLE|LOC and TITLE->->AUTH|LOC. The first dependency states that each book written by an author is published in the same set of locations. The second dependency states that if a book has several authors, then each of the authors uses the same set of locations to publish. This set of dependencies is not conflict-free. The following relation satisfies the mvd's.

| AUTH | TITLE | LOC |
|---|---|---|
| Ullman | Intro. to Automata | Reading |
| Ullman | Intro. to Automata | London |
| Hopcroft | Intro. to Automata | Reading |
| Hopcroft | Intro. to Automata | London |
| Ullman | Principles of Compilers | Reading |
| Ullman | Principles of Compilers | London |
| Aho | Principles of Compilers | Reading |
| Aho | Principles of Compilers | London |
| Jensen | PASCAL User's Manual | Berlin |
| Jensen | PASCAL User's Manual | Heidelberg |
| Wirth | PASCAL User's Manual | Berlin |
| Wirth | PASCAL User's Manual | Heidelberg |

The problem with this scheme centers around the following observation. Since Hopcroft and Ullman wrote "Intro. to Automata" and Aho and Ullman wrote "Principles of Compilers," the mvd's imply that Aho, Hopcroft, and Ullman all must publish in the same locations (here, Reading and London). We can take advantage of this implicit semantic property of the scheme to group together all authors that must have common publishing locations, storing the data in two tables.

| AUTH | TITLE | G |
|---|---|---|
| Ullman | Intro. to Automata | 1 |
| Hopcroft | Intro. to Automata | 1 |
| Ullman | Principles of Compilers | 1 |
| Aho | Principles of Compilers | 1 |
| Jensen | PASCAL User's Manual | 2 |
| Wirth | PASCAL User's Manual | 2 |

| G | LOC |
|---|---|
| 1 | Reading |
| 1 | London |
| 2 | Berlin |
| 2 | Heidelberg |

Since Aho, Hopcroft, and Ullman all must publish in the same locations, they are assigned the same group number, as are Jensen and Wirth.

Is there any meaning to the grouping attribute G? Yes; it denotes the publishers of the books. The meaning of the scheme then becomes: authors have only one publisher, and books can be published by only one publisher; each publisher publishes its books in a certain set of locations. The new data dependencies are {AUTH->PUB, TITLE->PUB, PUB->-> LOC}. By acknowledging the existence of publishers, the scheme now has a cf set of dependencies, which is equivalent to the original dependencies (in a sense to be defined later). []

What makes the original set of mvd's for the above scheme not cf is the existence of essential keys X and Y such that X->->Y and Y->->Z are in $G^*$, XYnZ=$\emptyset$, and XnY->->Z is not in $G^*$; the reader can check the definition to see that in such cases, the mvd's cannot be cf. We made the scheme cf by adding an attribute and some fd's to the scheme. We now show that this technique works in general; the new scheme we get will more accurately model the application than the original one.

Definition. $\pi_{(X=x)Y}(r)=\{y \mid xy \text{ is in } \pi_{XY}(r)\}$. When X is obvious from context, we shall write $\pi_{xY}(r)$.[] Our first result is a generalization of Theorem 2 of [KKY].

Lemma 5.1. Let G be a set of mvd's, and suppose attribute sets X,Y, and Z are such that X->->Y and Y->->Z are in $G^*$, and XYnZ=$\emptyset$. Let r be a relation over R, where XYZ$\subseteq$R. Then

a) If there exist x, x', y such that both xy and x'y are in $\pi_{XY}(r)$, then $\pi_{xZ}(r)=\pi_{x'Z}(r)$;

b) If there exist x, y, y' such that both xy and xy' are in $\pi_{xy}(r)$, then $\pi_{yZ}(r)=\pi_{y'Z}(r)$.

Proof.

(a) Suppose xz is in $\pi_{XZ}(r)$. Since xy is in $\pi_{XY}(r)$ and YnZ=$\emptyset$, X->->Y implies that xyz is in $\pi_{XYZ}(r)$. Now since x'y is in $\pi_{XY}(r)$ and XnZ=$\emptyset$, Y->->Z implies that x'yz is in $\pi_{XYZ}(r)$. Thus x'z is in $\pi_{XZ}(r)$, and $\pi_{xZ}(r)\subseteq\pi_{x'Z}(r)$. The proof of the reverse containment is similar.

(b) This is analogous to (a), and is left to the reader. []

We can combine parts (a) and (b) of Lemma 5.1 to connect several tuples together. For example, if xy, xy', and x'y' are in $\pi_{XY}(r)$, then $\pi_{xZ}(r)=\pi_{x'Z}(r)$ and $\pi_{yZ}(r)=\pi_{y'Z}(r)$. In general, if $x_1y_1,\ldots,x_ny_n$ are in $\pi_{XY}(r)$ such that $x_{2i}=x_{2i+1}$ and $y_{2i-1}=y_{2i}$, then $\pi_{xiZ}(r)=\pi_{xiZ}(r)$ and $\pi_{yiZ}(r)=\pi_{yjZ}(r)$. Consequently, whenever there exist XY-tuples xy and x'y' in $\pi_{XY}(r)$ that can be connected in this fashion, they must have the same set of Z-values. Let us say that the set of such XY-values form a group; all tuples in a group must have the same set of Z-values. In our above example, we grouped together the tuples (Ullman, Intro. to Automata), (Hopcroft, Intro. to Automata), (Ullman, Principles of Compilers), and noted that each of these tuples must have the same set of locations. The tuples (Jensen, PASCAL User's Manual), and (Wirth, PASCAL User's Manual) formed a second group.

In general, a relation may be partitioned into several groups. This partitioning will denote extra semantic information, except in the case when there can only be one group. The next lemma characterizes this case.

Lemma 5.2. Let G be a set of mvd's, and let X,Y, Z be as in Lemma 5.1. Then there exists a relation in sat(G) which will be partitioned into more than one group iff the mvd XnY->->Z is not in $G^*$.

Proof.

(if): Suppose XnY->->Z is in $G^*$. Then given x and x', $\pi_{xZ}(r)=\pi_{x'Z}(r)$ for every r in sat(G), regardless of Y-values. Consequently, every tuple will be in the same group.

(only if): Let W=XnY. If W->->Z is not in $G^*$, then there exists a relation r and two tuples xwyz, x'wy'z' in r such that xwyz' is not in r. Thus, xwy must be in a different group from x'wy'. []

For a set G of mvd's, let us call the dependencies X->->Y, Y->->Z in $G^*$ a transitive anomaly if XYnZ=$\emptyset$ and XnY->->Z is not in $G^*$. Our results say that whenever there is a transitive anomaly, any relation in sat(G) can be partitioned into groups. These groups are semantically important; recall from the above example that they permit a less redundant representation of relations. However, the concept underlying the group (e.g. in the above example the notion of "publisher") is not specified anywhere.

The existence of groups means that there is some aspect of the application that is inadequately represented in the scheme. We rectify this situation by adding an attribute (say, A) to the scheme, describing the groups. Since each X-value and Y-value belong to exactly one group, the fd's X->A and Y->A hold. The following theorem asserts that by adding the mvd A->->Z to the scheme, we remove the transitive anomaly; moreover, the data dependencies not involving A in the new scheme are the same as the data dependencies in the original scheme.

Theorem 5.3. Consider a scheme with attributes R=XYZW, and mvd's G and fd's F. Suppose X->->Y and Y->->Z are in $(GuF)^*$ and form a transitive anomaly. Let R'=XYZWA for some A not in R, F'=Fu{X->A, Y->A}, and G'=(G-{X->->Y, Y->->Z})u A->->Z . Let D be a data dependency not involving attribute A. Then D is entailed by FuG iff D is entailed by F'uG'.

Proof.

(if): The easiest way to show this is to consider the tableau obtained by using F'uG' to prove D in the chase [MMS]. If A->->Z is used to derive a row w, then there must be two rows having the same values for A. These rows can only be obtained by using X->A (or Y->->A) on two other rows. It is easy to see that using X->->Z on these latter two rows will produce the same row w.

(only if): This follows directly from the definition of F' and G' and the fact that X->A and A->->Z entail X->->Z. []

Theorem 5.3 states that the data dependencies GuF were correctly specified, given the attributes that were available; adding the attribute A allows other important data dependencies to be specified as well. Moreover, we can show that the old and new schemes are related in the following manner. Every legal relation for the new scheme embeds a legal relation for the old scheme and conversely, every legal relation for the old scheme is a projection of a legal relation for the new scheme. Because of this close relationship between the two schemes, we say that the new scheme is equivalent to the old one with respect to the original attributes. Therefore, adding the attribute A results in a strictly more accurate specification of the application, and we therefore are able to conclude that transitive anomalies result from inadequate attribute selection.

The fact that our method of adding attributes produces equivalent schemes is stated formally in the following theorem.

Theorem 5.4. Consider the relation schemes R and R' of the previous theorem, having data dependency sets GuF and G'uF' respectively. Then

a) for all r in sat(FuG) there is an r' in sat(F'uG') such that $r=\pi_R(r')$;

b) for all r' in sat(F'uG') there is an r in sat(FuG) such that $r=\pi_R(r')$.

Proof.

a) Consider any r in sat(FuG). Let s be the relation over R'=RA defined as follows. Relation s has the same number of tuples as r, and for each tuple t in r, there is a tuple t' in s such that $\pi_R(t')=t$ and $\pi_A(t')$ is some uniquely appearing value. Relation s is not necessarily in sat(F'uG'), since the fd's X->A and Y->A may be violated. (Note that these are the only dependencies that can possibly be violated.)

Suppose X->A is violated. Then there exist tuples $t_1$ and $t_2$ in s such that $t_1(X)=t_2(X)$ but $t_1(A) \neq t_2(A)$. Change s to a new relation s' by changing $t_1(A)$ to $t_2(A)$ everywhere in s. We claim that this change will not violate any data dependencies in F'uG'. First, since A does not appear in the left hand side of any fd in F', the change cannot violate F'. Second, the only mvd that can be violated is A->->Z. But since $t_1$ and $t_2$ have the same X-values, they belong to the same XY-group; thus, if $t_1$=xyzwa and $t_2$=xy'z'w'a, Lemma 5.1 implies that there exists tuples xyz'w and x'y'zw' in r. Consequently, A->->Z will no be violated in s'.

We can continue changing the relation until X->A and Y->A are satisfied. Since we never change any values other than A-values, in the final relation r', $\pi_R(r')=r$.

b) It is sufficient to show that $\pi_R(r')$ is in sat(FuG). This follows immediately from Theorem 5.3. []

For another example of transitive anomalies, consider the following example from [ZM].

Example. Consider a dictionary database of technical terms in three different languages (say E,F,G). Each word can stand for only one concept, but there may be several words in any language

that stand for one word in any other language. That is, several words in one language may stand for the same concept. The mvd's that hold are {E->->F|G, F->->E|G, G->->E|F}. Considering the first two mvd's, our rule adds a new attribute C (for CONCEPT) such that E->C, and C->->G. The dependencies for the new scheme become H={C->->G|E|F, G->->E|F|C, E->C, F->C}. It might seem appropriate to use the rule again on the mvd's C->->G and G->->E. However, this is incorrect, since G->->E is not essential; the fd G->C is entailed by H, and this, along with C->->E, entails G->->E. In fact, H is cf, since it has the cover H'= C->->E|F|G, E->C, F->C, G->C}. []

So far, we have shown in this section that if a set G of data dependencies is not cf because of a transitive anomaly, it is possible to remove this problem by adding an attribute and some fd's. Given a set G of mvd's attributes can be added to the scheme until no transitive anomalies remain. Will this process transform every non-cf scheme into a cf one? No; consider the mvd's {AB->->C|D, CD->->A|B}. This scheme is not cf, but there are also no transitive anomalies. The following theorem describes the general case.

Theorem 5.5. Let G be a set of mvd's. If G is not cf, then either there exists a transitive anomaly or there are essential keys ZX and ZY such that

$$DEP(ZX) = \{V_1Y_1,\ldots,V_nY_n,P_1,\ldots,P_h\}$$
$$DEP(ZY) = \{W_1X_1,\ldots,W_mX_m,Q_1,\ldots,Q_k\}$$

where $XnY=\emptyset$, $UY_i=Y$, $UX_i=X$, and for all i, if $Z->->Q_i$ is not in $G^*$, then $Q_j \subseteq UV_j$, and if $Z->->P_i$ is not in $G^*$, then $P_i \subseteq UW_j$; in addition, either m or n is greater than one.

Proof. Since G is not cf, there exist two essential keys ZX and ZY, where $XnY=\emptyset$ and such that ZX and ZY do not satisfy the definition of Section 3. We can express the dependency bases of ZX and ZY as follows.

$$DEP(ZX)=\{V_1Y_1,\ldots,V_nY_n,P_1,\ldots,P_h\}$$
$$DEP(ZY)=\{W_1X_1,\ldots,W_mX_m,Q_1,\ldots,Q_k\}$$

where $UY_i=Y$ and $UX_i=X$. Consequently, the mvd's $ZX->->UV_iY_iZ$ and $UV_iY_iZ->->Q_j$ are in $G^*$ for all j. If there is not a transitive anomaly, then either $Z->->Q_j$, or $ZXnQ_j\neq\emptyset$. The latter case must be true iff $Q_i \subseteq UV_j$ for all i. Similarly, we see that if $Z->->P_i$ is not in $G^*$ then $P_i \subseteq UW_j$. Finally, if n=m=1, then ZX and ZY satisfy the definition of conflict-free. Since ZX and ZY were chosen so as not to satisfy the definition, either n>1 or m>1. The theorem follows. []

Suppose we have essential keys ZX and ZY satisfying the conditions of the above theorem. If n>1, then the key ZX splits the key ZY into pieces $V_1Y_1,\ldots,V_nY_n$. So if G={A->->CD|E, CE->->A|D}, for example, then A->->CD|E splits the key CE. Therefore, we call this case a split-key anomaly. Theorem 5.5 states that every non-cf set of mvd's contains either a transitive anomaly or a split-key anomaly.

We have seen that the existence of a transitive anomaly implies that there are concepts still unrepresented in the database. It was shown that by adding appropriate attributes to the scheme, we could explicitly model these concepts and remove the anomalies. This solution is elegant, in that the new scheme does not contain any data dependencies involving the original attributes that were not present in the original scheme (see Theorem 5.3). Split-key anomalies also indicate that the scheme is inadequately specified; sometimes this inadequacy can be resolved by adding more mvd's.

Example. Consider the scheme ESPM, where the tuple espm means that employee e has salary s, and works on project p for manager m. The following two mvd's hold. First, every project has a set of managers, so P->->M|ES. Second, an employee can work on several projects for each manager; thus EM->->P|S. These two mvd's form a split-key anomaly, since P splits EM in P->->M|ES. We see, however, that not all mvd's have been specified; in particular, there has been no mention of the fact that employees have salaries. This new fd, E->S, entails the mvd E->->S|PM, and makes EM no longer an essential key; the resulting mvd set {E->->S|PM, P->->M|ES} is cf. []

The inadequacy demonstrated in the above example resulted simply from not finding every mvd that holds in the scheme. Once the mvd's are found, the anomaly disappears. However, there are cases when a split-key anomaly will remain even after all possible mvd's are found. In this case, a split-key anomaly means that mvd's are inadequate to model the scheme satisfactorily. Consider the following example.

Example. Let the attributes for a scheme be CVAP, where a tuple cvap means that country c buys a product p from a vendor v through agent a. With regard to constraints, we know the following. Agents will not handle any product they cannot immediately sell. Therefore, an agent will handle a product p iff it is already mediating between a country c and a vendor v such that v sells p and c needs p. In this case, the agent knows it can sell p from v to c due to the following semantic constraint: A country c buys from a vendor v iff c needs a product that v sells, and there is an agent that can handle the transaction.

Translating these constraints into mvd's, we get G={CV->->A|P, AP->->C|V}, which form a split-key anomaly. Our contention is that something is wrong with the scheme; what can it be? Note that the constraints, as we given them, contain circular reasoning. That is, in order for an agent to handle a product, CV->->A|P implies that there must already be a relationship between a country and a vendor. But AP->->C|V implies that in order for a country and a vendor to be related, there must be an agent that can supply a needed product. Some important concept, clearly, is missing; the constraints must be incompletely described. Perhaps we need to be able to say that an agent has an agreement with a country or a vendor. The first constraint then translates to "An agent will handle a product p if it has agreements with country c and vendor v, v sells p and c needs p." The second constraint becomes "A country c buys from v if there is a product that v sells and c needs, and there exists an agent that has agreements with both c and v." Note that both conditions are the same, and are equivalent to the jd D=*[CP,VP BC,BV]. The reader can verify that D entails the two original mvd's, but is not equivalent to any set of mvd's. []

In this example, in order to adequately specify the semantics of the scheme, we had to explicitly specify a jd; this jd contained extra semantics not entailed by the given mvd's (or any other set of mvd's). Consequently, mvd's are inadequate for specifying the semantics of some applications.

If a set G of mvd's has a split-key anomaly, then some information is missing. In both of the above examples, this extra information is contained in the jd $*[S_G]$. Indeed, Corollary 3.10 implies that if any single database scheme is to represent the set of mvd's, it must be $*[S_G]$. In the first example, we were able to add more mvd's so as to entail $*[S_G]$; in the second example, we had to specify this jd directly. However, it is not clear why $*[S_G]$ should hold for every instance of a split-key anomaly; resolving this conjecture is an open problem.

## 6. Conclusion

In this paper, we have tried to characterize those sets of mvd's that are important in "real-world" applications. To this end, we defined conflict-free sets of mvd's. Conflict-free sets of mvd's were shown to be desirable, in that they always allow a unique, dependency preserving 4NF database scheme. Moreover, non cf sets of mvd's do not have any dependency preserving database scheme, and therefore should be avoided.

Is it always possible to avoid non cf mvd's? Perhaps there is an application where the best representation involves a non cf set of mvd's. In Section 5 we showed that this is not the case; the best set of dependencies for an application is always cf. If a set of mvd's is not cf, then we showed that it possesses either a transitive anomaly or a split-key anomaly. The existence of these anomalies indicates an inadequate representation of semantic information. This implies that for every non cf set of dependencies, there is a better cf set, whose dependencies entail the dependencies of the original set. For transitive anomalies, this better set resulted from adding extra attributes; for split-key anomalies, it involved recognizing that mvd's might not be sufficient to describe the semantics of the scheme.

Our solution to the transitive anomaly case is elegant and interesting. Adding an extra attribute allows more data dependencies to be specified, and the data dependencies that are needed to remove the anomaly involve only this new attribute. Moreover, the presence of the new attribute allows the database to be represented with less redundancy.

The solution for split-key anomalies is not as nice. Here, the best we can say is that given a set of attributes, there may be no set of mvd's that adequately describes the scheme, and that the jd $*[S_G]$ should describe the necessary additional semantics. Note that we certainly have not proven that $*[S_G]$ should always hold; it only seems to be a reasonable assumption. For example, there may be a way of adding extra attributes to the scheme that will remove the anomaly, as was the case for transitive anomalies. However, we do not see how to improve the CVAP example of Section 5, using such a technique. It is hard to derive insight into this problem, due to the tremendous difficulty in finding examples of split-key anomalies; it also indicates that such anomalies are in some sense not natural.

Since we have concluded that all natural sets of mvd's are ones that are equivalent to a single jd, it is natural to wonder whether it is easier to specify the set of mvd's or the one jd. Traditional database design methods suggest that finding the jd is more natural. Designers tend to think in terms of facts (denoted by "objects" in [S2]) or "primitive predicates" [FMU]. If this is the case, then the jd can be built up by specifying facts without considering mvd's at all. Mvd's are therefore unnecessary in database design.

One important open problem concerns how fd's and embedded dependencies interact with jd's. Our definition at the end of Section 4 implies that a cf set of data dependencies has a cover consisting of one jd and a set of fd's. Is such a set always reasonable? Is there always a good database scheme representing such a set? If not, how do we characterize the reasonable ones? These questions deserve more study.

## 7. Acknowledgements.

The author wishes to thank Scott Parker and David Maier for helpful discussions; Billie Goldstein and a referee provided extensive comments on earlier versions.

## 8. References.

[B] Beeri, C. "On the Membership Problem for Multivalued Dependencies in Relational Databases." ACM TODS (5,3) Sept. 1980.

[BBG] Beeri, C., P. Bernstein, and N. Goodman. "A Sophisticate's Guide to Database Normalization Theory." Proc. VLDB Conference, 1977.

[BMSU] Beeri, C., A. Mendelzon, Y. Sagiv, and J. Ullman. "Equivalence of Relational Database Schemes." Proc. ACM-SIGACT Conference, 1979.

[B+] Beeri, C., R. Fagin, D. Maier, A. Mendelzon, J. Ullman, and M. Yannakakis. "Properties of Acyclic Database Schemes." Proc. ACM-SIGACT Conference, 1981.

[FMU] Fagin, R., A. Mendelzon, and J. Ullman. "A Simplified Universal Relation Assumption and its Properties." IBM TR RJ2900, Nov. 1980.

[KKY] Kambayashi, Y., T. Katsumi, and S. Yajima. "Semantic Aspects of Data Dependencies and Their Application to Relational Database Design." Proc. COMSAC Conference, 1979.

[L] Lien, Y. "On the Equivalence of Database Models." Unpublished Manuscript, Bell Laboratories, 1980.

[MMS] Maier D., A. Mendelzon, and Y. Sagiv. "Testing Implications of Data Dependencies." ACM TODS (4,4) Dec., 1979.

[S1] Sciore, E. "A Complete Axiomatization of Full Join Dependencies." Princeton University, 1979. To appear, JACM.

[S2] Sciore, E. "The Universal Instance and Database Design." TR271, Princeton University, 1980.

[U1] Ullman, J. Principles of Database Systems. Computer Science Press, 1980.

[Y] Yanakakis, M. Personal Communication.

[ZM] Zaniolo, C. and M. Melkanoff. "Relational Schemas for Database Systems." TR UCLA-ENG-7801, UCLA, Jan. 1978, p99.