NORTH-HOLLAND

Informatics and Computer Science

Characterization of the Effects of Schema Change

C. A. EWALD*

School of Computer and Information Science, University of South Australia, The Levels, S.A. 5095, Australia

and

M. E. ORLOWSKA

Department of Computer Science, University of Queensland, St. Lucia, Brisbane, Qld. 4072, Australia

ABSTRACT

Schema evolution occurs frequently in practice. In this paper, we show that the entire schema does not need to be redesigned every time a change is made. We characterize the subschema affected by basic schema operations, which allows us to limit the scope of checking required and have a guarantee of the overall design being correct (safe changes). In addition, we characterize the effects on the actual relations of the addition and removal of constraints. In some cases, relations may have to be added to or removed from the database in order to preserve correctness.

1. INTRODUCTION

Changes to schemata are required frequently in practice. The widely used SQL query language supports an ALTER TABLE command which allows the addition of a column to a relation. Typically, access to this command is restricted to certain privileged people. No corresponding command for the removal of a column is supported, due to the problem of handling deletion once the database is populated.

*This work was completed while the first author was at the Department of Computer Science, University of Queensland, Q.4072, Australia.

0020-0255/96/$15.00
PII S0020-0255(96)00134-X

Restrictions on schema evolution are enforced due to the serious nature of the consequences of evolution. For example, addition of a column may lead to a serious redundancy problem, causing update anomalies. Removal of a column can lead to the inability to identify some "objects," and determination of what elements of the population need to be deleted is a nontrivial issue. Another significant question is: "Does the database have to be completely redesigned once a change has taken place?." Redesign of a large database, due to a single simple change, is computationally expensive. Figure 1 shows an example database, an SQL statement which introduces potential redundancy, and an SQL statement which does not introduce any potential problem. The difference between the two cases is that the first introduces a transitively redundant functional dependency, while the second does not. Other types of redundancy may occur in practice.

The effects of evolution in relational databases can be precisely characterized by examining the changes to constraints required to implement any given change. This is not to say that only changes to constraints are possible. However, any change to the structure of a relational database includes, and is characterized by, some change to functional dependency constraints which hold over the database. In this paper, we characterize the part of a relational database schema affected by the addition or removal of a single functional dependency constraint. We examine the implications for the entire database design. Our results are extended to handle the addition and removal of attribute sets. This enables a database administrator to know before a change is made exactly what the consequences of the change will be. Resulting changes to the population can then be better managed. Thus, schema evolution operators become less dangerous and may be able to be used more widely.

```
Existing relations
 R1 [A B C D] Key A
 R2 [D F] Key D
Addition of Redundant Information
ALTER TABLE R2 ADD B varchar
Safe Addition
ALTER TABLE R2 ADD G varchar
```

Fig. 1. Some instances of schema evolution.

2. NOTATION AND BASIC CONCEPTS

In this section, we review relevant fundamental concepts from functional dependency theory. Let X and Y be nonempty attribute sets. A functional dependency (FD) $X \rightarrow Y$ is a statement that if a pair of tuples has the same X value, it must also have the same Y value. FDs are the most widely used constraint in normalization, and play a vital role in determining the structure of a normalized relational database. A set of inference rules for functional dependencies, known as Armstrong's Axioms, is described in [2]. A functional dependency graph is a directed, possibly cyclic, connected extended graph which represents a set of functional dependencies. Functional dependency graphs and graphical algorithms for dependency manipulation were first presented by Ausiello in [1].

Yang [3] discusses such graphs, and a synthesis algorithm based on the FD graph representation, E_f denotes the set of full arcs, while E_d denotes the set of dotted arcs. Unlike a graph, an FD graph has both simple and compound nodes. A dotted arc is one from a compound node to its component node. FD graphs are constructed as follows [3]: Given a set σ of FDs on U, the FD graph for σ, denoted by $G_{\sigma}=(N,E)$, where N is a nonempty finite set of nodes, is constructed such that:

1. For every attribute A in U, there is a simple node labeled by A in N, and

2. For every FD $X \rightarrow Y$ in σ with $|X| \geqslant 1$, there is a compound node labeled by X in N and E is a nonempty finite set of arcs partitioned into subsets E^f and E^d such that

2a. For every FD $X \rightarrow B_1 \ldots B_p$ in σ (where X is in N), there are p full arcs (X, B_k) in E^f for each k, $1 \leqslant k \leqslant p$ and

2b. For every compound node $X = C_1, \ldots, C_q$ there are q dotted arcs (X, C_k) in E_d for each k, $1 \leqslant k \leqslant q$. The simple nodes C_1 through C_k, forming the compound node X, are referred to as the component nodes of X.

In [1], an FD path is defined as follows: Given an FD graph H and two nodes $i, j \in N$, a directed FD path $\langle i, j \rangle$ for i to j is a minimal subgraph $G' = (N', E')$ of H such that $i, j \in N'$ and either $(i, j) \in E'$ or one of the following possibilities holds:

1. J is a simple node, and there exists a node k such that $(k, j) \in E'$ and there is an FD path $\langle i, k \rangle$ included in G' (transitivity)

2. J is a compound node with component nodes $m_1, \ldots, m_r$ and there are dotted arcs $(j, m_1), \ldots, (j, m_r)$ in G' and r FD paths $\langle i, m_1 \rangle, \ldots, \langle i, m_r \rangle$ included in G' (union).

An FD path from i to j is defined to be a dotted path if all arcs incident from i on the path are dotted, and a full path otherwise. A u path from i to j is a complex path made up of full arcs from i to all component nodes of j and of dotted arcs from j to these components. A derivable u path from i to j is a complex path made up of FD paths from i to all component nodes of j and of dotted arcs from j to these components. Intuitively, there is a full path from one node to another on the graph if an FD between them can be derived using transitivity. Other axioms are also used in constructing the graph closure.

An FD f is said to be derivable from a set F of FDs ($F \models f$) iff it can be derived from FDs in F using only Armstrong's axioms. This concept carries over directly to the graphical representation.

We use the above definition of functional dependency graph throughout the paper. The synthesis algorithm for relational database design is fully explained in [2]. The problem of computing the closure of a set of FDs is computationally expensive, and methods have been developed for determining if a particular dependency is in the closure without computing the entire closure. An algorithm is explained in [2]. The closure F^+ of a set of FDs F is the set of all FDs which can be derived from F using Armstrong's axioms. Since the addition and removal of FDs can affect the derivation paths of existing FDs, we must examine the normalization process. For this reason, we give a brief overview of the concepts used as defined in [2]. The Member algorithm is a polynomial time algorithm which determines, given a set of FDs F and an FD $X \rightarrow Y$, if $F \models X \rightarrow Y$. By definition, a redundant FD $X \rightarrow Y$ must satisfy the condition Member $(F - \{X \rightarrow Y\}, X \rightarrow Y)$. Similarly, an attribute A in R is extraneous in $X \rightarrow Y$ with respect to F if

$$X = AZ,\ A \neq Z$$

$$\{(F - (X \rightarrow Y)) \cup (Z \rightarrow Y)\} \equiv F.$$

$$Y = AW,\ Y \neq W$$

$$\{(F - (X \rightarrow Y)) \cup (X \rightarrow W)\} \equiv F$$

In other words, an attribute is extraneous if it can be removed from the right or left side without changing the closure of the left side. Two sets of attributes X and Y are equivalent under F iff $F \models X \rightarrow Y$ and $F \models Y \rightarrow X$. Similarly, two nodes X and Y are equivalent in the graph H iff there is an arc from X to Y and Y to X in the closure of H. The closure of an FD graph H (called the graph closure and denoted H^-) may be regarded as the FD graph of F^+. The closure of a node X (denoted X^+) is the set of nodes reachable by an FD path starting at X. In FD theory, the closure of an attribute set is the set of all attributes Z such that $X \to Z$ can be derived from the given FD set using Armstrong's axioms.

The graph closure is defined in [1] as follows: The closure of an FD graph $H=(N,E)$ is the graph $H^+=(N,E^+)$, labeled on the nodes and on the arcs, where the set N is the same as in H while the set $E^+=E_0 \cup E_1$ is defined in the following way:

$$(E^+)_1 = \{(i,j) \mid i,j \in N \text{ and there exists a dotted FD path } \langle i,j \rangle\};$$

$$(E^+)_0 = \{(i,j) \mid i,j \in N, (i,j) \notin E_1^+ \text{ and there exists a full FD path } (i,j)\}.$$

This definition follows from the definitions of graph transitivity and graph union.

The concept of equivalence relation is used to partition a set of FDs into equivalence classes, as described in [2]. For a set of FDs F over R and a set $X \subseteq R$, let $E_F(X)$ be the set of FDs in F with left sides equivalent to X. Let E_F be the set

$$E_F(X) \mid X \subseteq R \text{ and } E_F(X) \neq \emptyset.$$

$\overline{E_F(X)}$ is always a partition of F. We denote the set of left sides in $E_F(X)$ as $e_F(X)$. X is said to directly derive $Y (X \rightsquigarrow Y)$ under G iff for every nonredundant cover F for G, $X \to Y$ can be derived using only FDs in $F - E_F(X)$.

This concept carries over without modification to the FD graph environment. We denote the node equivalence class of a node X as $SC(X)$ since equivalence classes on the graph are strongly connected components. It has been proved [2] that every nonredundant set of FDs that is not minimum has some $E_F(X)$ containing distinct FDs $Y \to U$ and $Z \to P$ such that $Y \rightsquigarrow Z$. Algorithms to produce a minimum cover find these two FDs and replace them both by the single FD $Z \to UP$. The synthesis algorithm takes an arbitrary set of FDs as input, removes redundancy and extraneous attributes on left and right sides, and then creates a minimal cover. A compound functional dependency (CFD) [2] has the form $(X_1, X_2, \ldots, X_k) \to Y$, where $X_1, X_2, \ldots, X_k$ are all distinct subsets of a schema $\mathbf{R}$ and Y is also a subset of $\mathbf{R}$. A relation scheme R is the schema of a single-relation. A relation $r(R)$ satisfies the CFD $(X_1, X_2, \ldots, X_k) \to Y$ if it satisfies the FDs $X_i \to X_j$ and $X_i \to Y$, $1 \leqslant i,j \leqslant k$. In this CFD, $(X_1, X_2, \ldots, X_k)$ is the left side, $X_1, X_2, \ldots, X_k$ are the left sets, and Y is the right side. A set F of CFDs is annular if there are no left sets X and Z in different left sides with $X \leftrightarrow Y$ under F. A minimal annular cover may be produced from a minimum set of FDs, but a reduction step is then required to ensure that all CFDs are reduced. The interested reader is referred to Maier [2] where explanations, proofs, and algorithms are presented. From the minimal cover, a reduced minimal annular cover is created.

A graphical synthesis algorithm has been developed by Ausiello [1] and is explained in [3]. The algorithm consists of creating a nonredundant, minimum covering, then an LR-minimal covering. A nonredundant covering is free from redundant nodes, a minimum covering is a graph corresponding to a set of FDs free from directly derivable FDs within an equivalence class, and and LR-minimal covering is minimal and free from redundant arcs (arcs which correspond to FDs derivable using the axioms). This applies to both full and dotted arcs on the graph.

Relations are created from the equivalence classes of the LR-minimal covering. The complexity of [1] computing a minimal covering is $O(max(p^2 + p \cdot \|U\|, n))$ where n is the size of the strings of symbols used to represent the input and p is the number of FDs in the input. U is the set of all attributes which appear on the graph. The complexity of finding an LR-minimal covering is $O(t \cdot n)$ where t is the number of redundant arcs. Relations can then be created as described in [1]. We present illustrations on FD graphs of the concepts discussed. Figure 2 shows two FD graphs with redundancy. The first represents $F = \{A \rightarrow B, A \rightarrow C, C \rightarrow D, D \rightarrow B\}$. The second is for a set $F = \{AB \rightarrow CD, B \rightarrow D, A \rightarrow C\}$. In the first, the arc from D to B is redundant. In the second, the node CD is redundant. Figure 3 shows an FD graph which is not LR-minimal since it contains a redundant dotted arc from ABC to C. Figure 4 contains an FD graph which is not minimal since the node EF is superfluous. A compound node is superfluous if it can be removed without changing the graph closure.

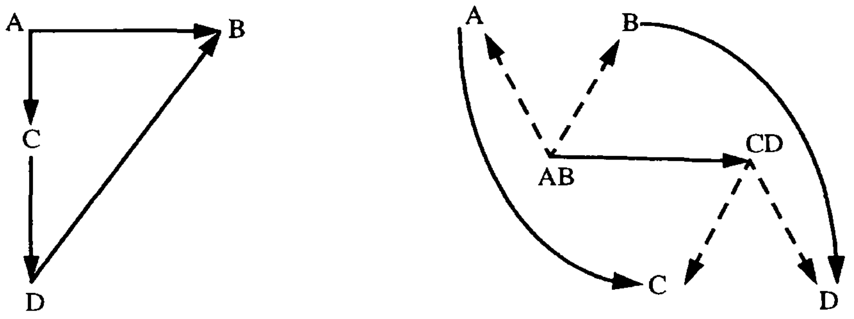

Fig. 2. Two redundant graphs.

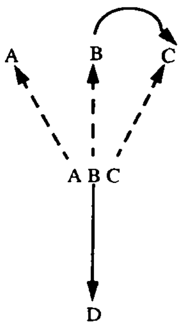

Fig. 3. A subgraph with an extraneous node C in ABC.

Throughout this paper, we use the following inference rules developed by Ausiello [1]. These are a direct translation of Armstrong's axioms to the graphical environment.

Reflexivity: If $X \supseteq Y$ or $X = Y$, then $H \models X \rightarrow Y$.

Augmentation: If there is an arc from X to Y and $Z \in N$, then $H \models ZX \rightarrow ZY$.

Transitivity: If arcs $\{X \rightarrow Y, Y \rightarrow Z\}$ are in H, then $H \models X \rightarrow Z$.

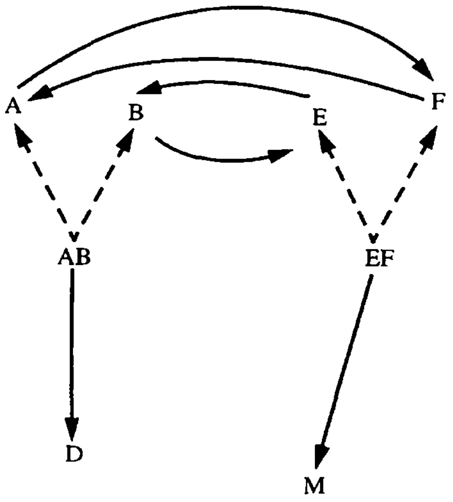

Fig. 4. A subgraph in which node EF is superfluous.

3. SCHEMA EVOLUTION

We now characterize the effects of a change to functional dependency constraints.

THEOREM 1. *Let H be an FD graph, and X and Y nodes on the graph. Let $X \rightarrow Y$ be an FD to be added to H. Changes to the structure of H resulting from the addition occur only in the part of the graph corresponding to X^+.*

Since we are dealing with addition, X^+ is the closure after the change has occurred. Consider $X \rightarrow Y$. Any full or dotted arc already existing between the two nodes must be in the node closure. Therefore, if there is already a path between them (and hence the new arc is redundant), such a path will be in this subgraph. Similar observations hold for redundant and superfluous nodes by definition of node closure. □

THEOREM 2. *Let H be an FD graph and X and Y nodes on the graph. Let $X \rightarrow Y$ be an FD to be removed. Changes to the structure of H resulting from the removal occur only in the part of the graph corresponding to X^+.*

Consider X^+, the closure of X before the change. If an FD $X \rightarrow Y$ is removed, it will no longer be derivable from H unless there exists another FD path from X to Y in X^+. Therefore, the result follows immediately. □

We have assumed that nodes involved in any FD added already exist on the graph. This assumption can be made without any loss of generality since if Y does not appear on the graph before addition, then $Y^+ = Y$. X must appear on the graph; otherwise, it is not connected and is therefore not an FD graph. Clearly, if an FD $X \rightarrow Y$ is to be removed, both X and Y must be on the graph. We also refer to the node closure as the connected subgraph incident from X ($SGR(X)$) since it consists of all full or dotted FD paths from X to any leaf node, where a leaf node is a node with no outgoing arcs. Note that, by using the term leaf, we do not intend to imply that the structure need be a tree.

In the next section, we examine the effects of removal of a single functional dependency f from a set F of FDs on database relations. Intuitively, the best way to attack this problem is to examine the results of performing the synthesis process on F and F'. This work has important implications for incremental approaches to schema evolution using the relational model. We identify certain conditions which, if present, ensure that complete recomputation is not required. If the complexity of checking these conditions is less than that of performing recomputation, then it is clear that cases exist where incremental methods are cheaper than re-design of the entire database. We characterize the special case in which no recomputation is needed. This case occurs when the FD removed is in the minimal cover of the original FD graph, and is not on any FD path used to prove any node or arc redundant or superfluous.

We now present a formal characterization of cases. Some computation is required in all cases, however, in order to create a correct annular cover. We discuss and characterize this computation. Recall that the derivation sequence of a derivable FD f [2] is the set of FDs used to derive F. The derivation path of f on an FD graph H is the minimal subgraph containing all arcs used to derive f. Let H_F be an FD graph for a set of FDs F, and let H' be a copy of H_F with the arc from X to Y removed. F' is the FD set represented by H'.

OBSERVATION 1. *It cannot be guaranteed that F is a cover for F'.*

Given an FD graph H and a pair of nodes X and Y, the nodes X and Y exist in the nonredundant, LR-reduced, minimal cover G' for G iff neither node is redundant or superfluous. There is an arc between them in G' iff there is a nonredundant arc between them in G.

THEOREM 3. *Let H be an FD graph for a set F of FDs. Let G be a nonredundant, LR-reduced minimal FD graph formed from H using graphical synthesis. Let $X-Y$ denote an arc removed from H to create a graph H'. Removing $X-Y$ from G creates a graph representing a minimal LR-reduced cover for the set of FDs represented by H' iff $X-Y$ is in G, and is not part of the derivation path of any arc or node in H which is not in G.*

Proof. G' must be nonredundant if G is nonredundant since nothing has been added. Similarly, no redundant arcs, nodes or superfluous nodes can be introduced. Hence, if G' is not an LR-reduced, minimal cover for H', G' cannot be a cover for H'. The node closure of some node in G' is different from the node closure of the node in H'. By definition of derivation path, this must result from some derivation path having been disconnected. Since the only change made has been the removal of $X-Y$, it follows that $X-Y$ must have been on the derivation path of some redundant arc or node, or superfluous node in H which does not appear in G. □

If the arc to be removed takes part in the derivation path of some redundant arc or node, or superfluous node, then the previously derivable arc or node needs to be restored. Clearly, only outgoing arcs originating from X or Y need be considered. Otherwise, we can proceed directly to the creation of an annular cover since by Theorem 3 we have a minimal cover for our new set of FDs. Similarly, when a dependency is added, we assume that any redundancy created and any superfluous nodes added are then removed. Thus, we can proceed immediately to consider the creation of annular covers and the generation of database relations.

4. REMOVAL OF A SINGLE FUNCTIONAL DEPENDENCY

Changes to the FDs which hold over a database sometimes require changes to the structure of relations if we wish to preserve properties of the database. For the remainder of this paper, we assume that we wish to preserve normalization and syntactic correctness. Having considered changes to the FD graph itself, we now consider the creation of an annular cover and the generation of database relations. We examine this step now. We show that changes to relations and their keys may be required in order to preserve normalization and syntactic correctness.

Let F be a set of FDs and $F' = F - \{f\}$. In other words, we wish to characterize $\mathbf{R}'$, the new relational database design created from F'. We give several examples of the possible effects.

EXAMPLE 1

```
The original FDs
A->B
D->A
*
The original design
R<1>=A B
K<1>={A}
R<2>=D A
K<2>=D
R1 is removed if A->B is removed.
```

On the other hand, if the original dependency set above contains an additional attribute C, such that $A \rightarrow C$, the relation is not removed. Rather, B is deleted from the relation.

In the following example, the removal of an FD "splits" a node equivalence class into two, resulting in the creation of an additional relation. Note that the input set of FDs is really two distinct sets, as they are not connected.

EXAMPLE 2

```
The original input to the design process (not reduced)
 A->B C1 D E J C2
 B C2->C1 D E J
 C1 D->J
 B C1->A
The original design
 R<1>=A C2 B C1 D E
 K<1>={ C1 B, C2 B, A }
 R<2>=C1 D J
 K<2>={ D C1 }
 Design after removal of B C1->A
 R<1>=A C2 B
 K<1>={ A }
 R<2>=B C2 E D C1
 K<2>={ C2 B }
 R<3>=C1 D J
 K<3>={ D C1 }
```

Consider now the case where the only FDs are $A \rightarrow B$ and $B \rightarrow A$. In this case, we have a single relation, with two keys. The removal of one FD results in the corresponding left side becoming a nonkey attribute, but the relation is not split. We assume that no derivation paths of previously derivable arcs or nodes have been destroyed by the removal—if they have, our previous results may be used to isolate and restore them.

We now examine the changes to the relational database design which may occur when a single FD $X \rightarrow Y$ is removed. We also informally state what results these would have if the relations were populated. Intuitively, the following cases are possible.

1. A relation containing only XY may be removed. In this case, the population of the relation is deleted.
2. Y may be removed from a relation of which X is the key if some other functional dependency is represented in the relation, i.e., the dependency $X \rightarrow Z$ is retained. In this case, the population of Y is removed from the table (but not necessarily from the database, as the attribute Y may occur elsewhere, although we assume that dependencies are not repeated).

3. A table containing attribute sets which were formerly candidate keys but which are no longer equivalent may be split into two or more other tables. Some key attributes may then become nonkey attributes in one of these subrelations. In this case, the new tables must be populated by projections of the population of the original table. At most, t new tables can be created, where t is the number of nodes in the node equivalence class split. This worst case scenario would require that no node in the class remained equivalent to any other, and is not likely except for very small classes.

4. X may become a nonkey attribute set in the table in which it was originally placed.

By definition of synthesis, left sides in $SC(X)$ become keys and right sides nonkey attributes. We denote the set of equivalent left sides in $SC(X)$ as $sc(x)$.

We examine the results of using the synthesis algorithm on the FD graph to reconstruct the relational design after removal of the functional dependency $X \rightarrow Y$ from an FD graph H.

Figure 5 shows an example of an equivalence class which is split by the removal of an FD. This happens because, after the removal of $D \rightarrow A$, D and A are no longer equivalent. Therefore, $SC(D)$ in the new graph contains only D, while $SC(A)$ contains only A. These two graphs map to different sets of relations.

In the following, we assume as usual that any previously derivable information which is no longer derivable after removal has been restored. Recall that an FD graph must be a connected graph. We consider only the schema level in this theorem.

THEOREM 4. *Let H be an FD graph. Cases* 1–4 *above are the only possible effects on a relational database design of the removal of a single FD from H.*

Proof. Based on our assumptions as stated above, there are two possible cases.

1. $sc(X) - X = \emptyset$. There are two cases here.

(a) If Y is the only nonkey attribute set in the relation (the only right side in $SC(X)$), then $SC(X)$ is removed, and therefore the relation is removed.

(b) If Y is not the only right side in $SC(X)$, then $SC(X)$ is retained and the arc or path from X to Y is simply removed. The only change to the corresponding relation is to remove Y.

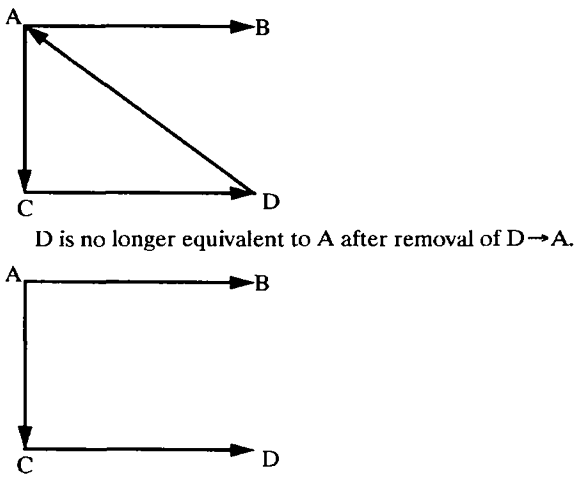

Fig. 5. An equivalence class split by removal of $D \rightarrow A$.

These are the only cases. Assume that some other case is possible here. Then either attributes other than X and Y must have been involved in the dependency removed, since Y must either be the only right set of C or there must exist some other right set or $SC(X)$ must contain some other node (see case 2). This contradicts our assumption that only the FD $X \rightarrow Y$ is removed.

2. $SC(X)$ contains other attribute sets equivalent to but not equal to X. We denote the set of such attribute sets as K. Formally, $K = SC(X) - X$. Once again, there are two possible cases.

(a) All elements of K remain equivalent. More formally $\forall k_i, k_j \in K, SC(k_i) = SC(k_j)$. In other words, the node equivalence class to which X belonged is still strongly connected. Since $X \rightarrow Y$ is no longer in $SC(X)$, Y is removed from the corresponding relation. However, since no other changes have been made, we assume that $\exists k \in K | H \models k \rightarrow X$. Hence, X becomes a nonkey attribute set in the corresponding relation, but no other changes are made.

(b) There are some elements of K which are no longer equivalent. $\exists k_i, k_j \in K | SC(k_i) \neq SC(k_j)$. Note that there may be more than one such pair of elements. Thus, if n such pairs of elements are found, n new node equivalence classes will be created, and these will then map to n new relations. Y will not appear in any of these relations. X will appear in some relation iff $\exists k \in K | H \models k \rightarrow X$. Since X was equivalent to every element of K before the change and no other dependency has been removed, this condition must hold, and X will be a nonkey attribute set in some relation.

Assume that some other case is possible. Since elements of K must satisfy either condition 1 or condition 2, this new case must involve an empty K. However, this case satisfies our first condition. Therefore, some other functional dependency must be involved, which contradicts our initial statement of the problem. Hence, these are the only possible cases. □

However, if our starting point is an arbitrary set of dependencies which is not minimal, there may be cases in which removal of a dependency has no effect. This occurs when a functional dependency which is derivable from the other dependencies in the set is removed. Since we have shown that a design is correct after removal if the FD removed did not either participate in any redundant subgraph or have any extraneous or superfluous node, it is not necessary to check these conditions when the original design is known to be free of such structures. In such cases, it only remains to create equivalence classes and form relations based on this construction. More precisely, we require a test to check if the FD in question is involved in such a structure.

If an attribute set is removed, the factors listed below must be considered:

- It is not possible to remove Y from a relation of which X is the key—all functional dependencies involving X and the corresponding information are to be removed.
- It is not possible to retain X as a nonkey attribute.

All relations in which X is a candidate key must therefore be removed, and X must be removed from all other relations in which it appears as a nonkey attribute set.

This result means that only the equivalence class of the left side of a removed FD can undergo changes as a direct result of removal, allowing us to further restrict the amount of recomputation required.

5. ADDITION OF A FUNCTIONAL DEPENDENCY

In this section, we examine the effects on a relational database design if a single functional dependency is added. Let F be a set of functional dependencies which hold over a set $\mathbf{R}$ of relations. Let $F' = F \cup \{X \rightarrow Y\}$, and $\mathbf{R}'$ be the new relational database design (set of relations). We wish to characterize $(F',\mathbf{R}')$. We assume that F is a reduced minimum cover. We also assume that if F' is not a reduced minimal cover after the addition of $X \rightarrow Y$, it is minimized before the creation of the annular cover. This can be done without recomputing the entire design by calculating $SGR(X)$ where X is the left side of the new FD. This is the only part of the schema which needs to be redesigned. First, we examine the creation of an annular cover from the reduced minimal cover. We consider the CFDs created from a minimal FD graph for F'.

One may add a derivable FD. This has no effect on the relational database since normalization algorithms detect it and do not allow it to influence the design. There are five possible cases.

1. No change is made. This occurs when a derivable FD is added, but not stored. The population of the existing relations are unchanged.
2. A new relation containing XY is created, and X becomes the key. This occurs when X is not a member of any existing node equivalence class. This relation must be populated with new instances.
3. Y is added to the nonkey attributes of an existing relation. This occurs when X is the only left set in some existing node equivalence class. It is up to the designer to determine if null values are allowed in the new attribute, or if values must be entered for all existing tuples.
4. X becomes a key of some existing relation in which XY already exists. This occurs when X becomes equivalent to the left sides in some existing node equivalence class in which X and Y are right sides. By existing relation, we mean that one has had no changes to its attributes. Null values must no longer be allowed in X.
5. A number of relations are merged, and X becomes one of a set of keys. Relations containing previously nonequivalent attribute sets, which are now equivalent to X, are merged into a single relation. This occurs when multiple equivalence classes $SC(X)$ are merged due to some attribute sets becoming equivalent. In other words, the equivalence class $SC(X)$ after the addition of $X \rightarrow Y$ contains $SC(A)$ and $SC(B)$ for some sets of attributes A and B. The populations of these relations are joined.

No population instances should be rendered invalid by the addition of information. New instances may need to be added, and the relation in which some information is stored may change.

We now examine the effects of adding information to the examples presented in the previous section. Consider the FD set of Figure 1, after removal of $A \rightarrow B$. If $A \rightarrow B$ is replaced, $R1$ is restored. The addition of $A \rightarrow B$ to a set of dependencies $A \rightarrow C$, $D \rightarrow A$ would result in the addition of B as a nonkey attribute to the relation of which A is key. If the FD removed in example 2 were restored, the equivalence class would be rejoined, and the original design recreated.

Consider the case where the only dependency is $A \rightarrow B$. The addition of a dependency $B \rightarrow A$ would simply result in B becoming an alternate key for the relation. If the new left side has an equivalence class on the graph, effects are once again confined to this class. As in the section on removal, we consider only the schema level in this theorem.

THEOREM 5. *Let H be an FD graph. Cases* 1–5 *above are the only changes to a relational database design which can occur due to the addition of a single FD.*

Proof. By the synthesis algorithm, all derivable functional dependencies will be removed from the graph. Hence, if an existing stored or derivable functional dependency is added, this change will have no effect on the final design. This may be assumed not to affect the minimum annular cover produced since redundant and superfluous arcs and nodes are removed. We may now proceed to examine this stage. There are two possible cases. Either $sc(X)$ existed on the original graph for F or it did not. Clearly, there can be no other alternative. If $sc(X)$ did not exist before, a new node equivalence class $SC(X)$ is created which had no equivalent in the old design. Hence, either a new relation is created, or X becomes the key of a (previously all-key) relation containing XY. Alternatively, X is equivalent to the left sides of some existing FD. The new $SC(X)$ consists of an existing one with X added as a new left set. Normally, this results in X becoming a key of some relation in which it was previously a nonkey attribute. However, it may also result in the mergining of two or more equivalence classes to form $SC(X)$. If this is the case, the corresponding relations are merged to form a single relation. If $sc(X)$ existed previously, then Y is added to the corresponding node equivalence class as a new right side. Y then becomes a new nonkey set of attributes in the corresponding relation. Since no other dependency has been added or removed, these are the only possible cases. □

We define a new set of attributes as one which does not correspond to any node already existing on the graph. When a new set of attributes X is to be added, it is clearly not possible that X is already the left set of some FD or that XY are nonkey attributes in some relation. X cannot be part of a derivation path (or any FD path) by definition. Hence, no checking is required when a set of attributes known not to appear anywhere on the graph is added. Whenever a new set of attributes is added, at least one dependency involving those attributes must be added. Apart from this, the results presented above apply to the addition of new attributes. To summarize then, we have shown that previously stored FDs cannot be affected by a single change unless they are in $SGR(X)$, where X is the left side of the FD changed.

6. CONCLUSION

In this paper, we characterize the effects of changes to relational database schemata. We show that only a limited subschema needs to be checked when a single attribute or constraint is added or removed, and we precisely characterize this part of the schema. Our work enables the designers and administrators of databases to more easily support ALTER TABLE commands requested by users since the results presented here enable changes which may affect data belonging to other users to be accurately detected and prevented. An understanding of the results of change allows the administrator to compensate for any unwanted side effects.

REFERENCES

1. G. Ausiello, A. D'atri, and D. Sacca, Graph algorithms for functional dependency manipulation, *J. ACM* 30(4):752–766 (Oct. 1983).
2. D. Maier, *Theory of Relational Databases*, Computer Science Press, 1983.
3. C. C. Yang, *Relational Databases*, Prentice-Hall, Englewood Cliffs, NJ, 1986.

Received 1 June 1995; revised 27 April 1996