Designing Globally Consistent Network Schemas

Sharon M. Kuck[1]
University of Illinois, Urbana, Illinois

Yehoshua Sagiv[2]
The Hebrew University of Jerusalem, Jerusalem, Israel

ABSTRACT

In this paper we address several problems relating to functional dependencies and network schemas. We investigate properties of the functional dependencies that are implicitly defined in a network schema. A definition for the satisfaction of functional dependencies by a network database is proposed, and then we give a sufficient condition for the global consistency of a network schema. A network schema is globally consistent if all its databases satisfy the functional dependencies that are implicitly defined in the schema. Finally, we describe a design methodology for producing a network schema whose set of implicitly defined functional dependencies implies a set of functional dependencies specified by the user.

1. Introduction

The design theory for relational databases started with the concepts of relations and functional dependencies [Codd]. These concepts lead to the definition of normal forms, and the goal of the design theory became to develop methods for designing relations in normal form (e.g. [Bern]), since normal forms remove certain update anomalies. Only recently it became evident that even if the relations of the database are in normal form, the data stored in the database may be globally inconsistent with respect to the specified functional dependencies [Hone]. Recent works (e.g., [GrYa,Sa83]) have investigated the question of when a relational database schema is globally consistent. These research results have improved our understanding of how a relational schema should be designed.

The vast knowledge that has been accumulating about functional dependencies and relational design does not have much influence on the way network schemas [CODA] are designed. Few papers [WoKa,Zani] dealt with the issue of converting network schemas to relational schemas (or vice-versa), but did not rely on the recent results of relational design theory, and had little formal justification. Lien [Lien] investigated the equivalence of the relational and the network models on a formal basis, but he considered only multivalued dependencies while claiming that functional dependencies are no more than the multivalued dependencies they imply. Consequently, the current methods for designing network schemas (e.g., [Chen,Lien,Ullm,WoKa]) do not pay functional dependencies the attention they deserve, and produce schemas in which it is rather hard to enforce functional dependencies.

In this paper we show how to design a network schema that preserves a set of functional dependencies F, i.e., a network schema such that every database created for this schema satisfies the functional dependencies of F. We essentially do this by implementing all the many-one

[1]The work of this author was supported in part by NSF grant MCS-80-03308.

[2]The work of this author was supported in part by a grant of the Charles H. Revson Foundation.

relationships implied by F as links (i.e., sets) or paths of several links, and defining calc-keys and search keys that enforce these many-one relationships, that is, the calc-keys and search keys prohibit storing data that do not satisfy the functional dependencies of F. The definition we propose for determining whether a network database satisfies a given functional dependency is based on translating a network database to a universal relation (with marked nulls), in a way similar to that of Lien [Lien], and determining whether the universal relation satisfies the given dependency. We also consider the question of what functional dependencies are implicitly defined in a given network schema. The answer we give is the same as Zaniolo's [Zani][3], but our approach is different in the following two aspects.

(1) We advocate enforcing the functional dependencies of a network schema globally and not locally.

(2) We formally discuss the role of the functional dependencies that follow from database keys, and show when they can be ignored. (Essentially, in [Zani] these functional dependencies are always deleted from the relational schema obtained for a given network schema, but no formal justification for that action is given.)

2. Preliminaries

2.1. The Network Model

A *network schema* consists of *record types*, each having zero or more *data items*, and *sets* that serve as many-one *links* between record types. We assume that all record types are fixed length, data items may contain only simple values (i.e., like the entries of a relation), null values cannot be stored for the data items, multimember sets are not allowed, and there are neither record types of DIRECT location mode nor data items of type DATABASE KEY. In order to be consistent with relational terminology, we refer to data items as *attributes*. We assume that all sets are declared AUTOMATIC MANDATORY, that is, for all sets S, each record occurrence of the member record type of S is always owned by some record occurrence of the owner record type. For each record type, we may declare a combination of some attributes to be a *calc-key*. In each set S, any combination of attributes (of the member record type) may be declared a *search key*. We consider only calc-keys or search keys that are declared DUPLICATES NOT ALLOWED (DNA). This establishes that for all record types, no two record occurrences have identical calc-key values; and for all sets, no set occurrence has two record occurrences with identical search key values. Keys for which duplicates are allowed do not enforce any functional constraint, and therefore are excluded from our model. In practice, each set can have many search keys, but only one calc-key per record type is allowed. However, a search key in a singular set simulates a calc-key and, therefore, we treat search keys of singular sets as if they were calc-keys.

A network schema is drawn as a directed graph, where nodes corresponds to record types and arcs to sets. An arc (or link) is drawn from the member record type to the owner record type. Record types $R_1, \ldots, R_n$ form a (directed) *path* in the schema if for all $1 \leq i < n$, R_{i+1} is an owner of R_i in some set of the schema (i.e., there is a link from R_i to R_{i+1}). We assume that a network schema has no cycles, i.e., it is a directed acyclic graph. If there is a path from R_i to R_j then R_j is an *ancestor* of R_i, and R_i is a *descendant* of R_j. Likewise, we use the terms owner occurrence, ancestor occurrence, and descendant occurrence. Record occurrence r_j is an *owner occurrence* of record occurrence r_i, if r_j owns a set occurrence in which r_i is a member. Suppose that $r_1, \ldots, r_n$ are record occurrences such that for all $1 \leq i < n$, r_{i+1} is an owner occurrence of r_i. Then r_n is an *ancestor occurrence* of r_1, and r_1 is a *descendant occurrence* of r_n.

2.1.1. Traversals

In the network model, the most basic way of correctly joining records is defined in terms of traversals.[4] Suppose that $R_1, \ldots, R_n$ is a path of the schema, and each r_i is a record occurrence of type R_i. Then $r_1, \ldots, r_n$ form a *path traversal* if for all $1 \leq i < n$, r_{i+1} is an owner occurrence of r_i. A *full path traversal* is a path traversal $r_1, \ldots, r_n$ such that R_n has no owners

[3] Similar ideas were later published in [Nava].

(other than SYSTEM). Let r be a record occurrence of type R. The *R-traversal* of r is obtained by concatenating r and all its ancestor occurrences, i.e., by concatenating all full path traversals that start at r.

2.2. The Relational Model

In this paper we consider only *relational schemas* that consist of a single *universal relation scheme*. A universal relation scheme is essentially a universal set of *attributes* U. A *universal relation* is a finite set of tuples defined over U. If t is a tuple and $X \subseteq U$, then $t[X]$ denotes the values of t for the attributes of X, and is called an *X-value*. A universal relation scheme has a set of *functional dependencies* (abbr. FDs) F associated with it, and a universal relation satisfies the FDs of F. The entries of a universal relation are allowed to have *marked nulls* (i.e., null values with subscripts - in order to distinguish between distinct occurrences of nulls). A universal relation with marked nulls satisfies an FD $X \rightarrow Y$ if the following is true. For all tuples t_1 and t_2, if $t_1[X]=t_2[X]$ then $t_1[Y]=t_2[Y]$. However, before checking whether the universal relation satisfies the FDs, we should apply the *chase* process to it. A description of the chase process can be found in many papers (e.g., [Hone,MMS,MUV,Sa83]), and we will not define it here.

2.3. Common Notation for the Two Models

We denote attributes by $A,B,C,...$ and sets of attributes by $...,X,Y,Z$. A set of attributes is written as a string of attributes, e.g., ABC instead of $\{A,B,C\}$. The union of sets X and Y is written XY. By a slight abuse of notation, the set of attributes of a record type R_i is denoted as R_i. The set of all the attributes that appear in the record type R_i and all its ancestors is denoted by $\overline{R_i}$.

2.4. Schemas and Databases

It is important to distinguish between a schema and a particular database for that schema. In the network model a schema N consists of record types and sets, and a *network database* for the schema N is a finite collection of record and set occurrences that conforms to the structure of the schema. In the relational model the schema is given as a universal relation scheme U and a set F of FDs, and a *relational database* is a universal relation over U satisfying F. In this paper we will show how to translate a network schema N to a universal relation scheme U and an associated set of FDs F, and then how to transform a network database for the schema N to a universal relation over U.

In order to represent a network database as a universal relation, we must know what are all the correct access paths in the network schema, since the traversal of each correct access path produces some data that have to be represented in the universal relation. We consider only many-one relationships, since these are easily expressed in the network model by links (i.e., sets) between record types. The traversals that were defined earlier are all the data obtained by traversing correct access paths when only many-one relationships are considered [Kuck]. Actually, all path traversals are correct data, but every set of the network schema is declared AUTOMATIC MANDATORY and, therefore, each path traversal is a part of some traversal (and need not be considered separately).

3. The Universal Relation Scheme Assumption

Our approach for transforming the data of a network database into a relational form is based on the *universal relation scheme assumption* [FMU]. This assumption postulates that the contents of the database can always be represented as a single universal relation over all the attributes appearing in the database schema. A universal relation can unambiguously represent the data only if each attribute has a unique "role". We illustrate this fact in the following example. Suppose that we use the attributes EMP_NO, EMP_NAME, and DEP_NAME to represent information about employees, their departments, and the departments' managers. At first sight it might seem that the universal set of attributes we have chosen is adequate, since each manager is also an employee. However, given a tuple over EMP_NO, EMP_NAME, and DEP_NAME, we cannot tell whether it stores information about an employee and his department or about a department and its manager. Therefore, the attribute EMP_NO must refer only to employees, and

[4] Lien [Lien] has also defined traversals, but his definition is more general, since he assumes that every connected undirected path is a correct access path.

another attribute, say MAN_NO, must be introduced to represent information about managers (although the domain of values for MAN_NO is still the same as for EMP_NO, i.e., if Smith is a manager, then Smith's MAN_NO is the same as his EMP_NO). The most constricted translation of the universal relation scheme assumption to the network model results in assuming that each attribute in a network schema has a unique "role". This constricted translation is the one used in [Lien], and we will use it in this paper.

The fact that each attribute has a unique "role" restricts the structure of the many-one relationships that can be expressed in a network schema. Suppose that R_j is an ancestor of R_i. Then we have a many-one relationship from record occurrences of type R_i to record occurrences of type R_j, or more precisely, to R_j-traversals. As long as there is a single path from R_i to R_j, it is clear that each record occurrence r_i of type R_i has a unique record occurrence r_j of type R_j corresponding to it. The record occurrence r_j is found by starting at r_i and traversing the path from R_i to R_j. If there are several paths from R_i to R_j, however, it is quite possible that distinct paths lead to distinct record occurrences of type R_j. Although two distinct record occurrences can be identical, having two identical R_j-traversals is redundant. Therefore, if two distinct paths from r_i lead to distinct record occurrences of type R_j, we must conclude that each path represents a distinct many-one relationship from R_i to $\overline{R_j}$ (assuming, of course, that the data do not violate the many-one relationships represented in the schema by links). But the existence of two distinct functional (i.e., many-one) relationships, say F_1 and F_2, from R_i to $\overline{R_j}$ can be ruled out by the following argument. Suppose that we have a tuple t with values for the attributes of both R_i and $\overline{R_j}$; then there is no way of deciding whether the $\overline{R_j}$-value of t is the one related to the R_i-value by the F_1 relationship or by the F_2 relationship. In essence, some attribute of R_j violates the unique "role" assumption.

By the preceding discussion, the translation of the unique "role" assumption to the network model implies that traversals must be *consistent* in the following sense. First, a traversal cannot have two distinct record occurrences of the same record type. In other words, if record type R_j is an ancestor of record type R_i, and r_i is a record occurrence of type R_i, then all path traversals that start at r_i (and contain a record occurrence of type R_j) must lead to the same record occurrence of type R_j. Secondly, if a traversal contains two distinct record occurrences r_i and r_j of types R_i and R_j, respectively, and A is an attribute of both R_i and R_j, then r_i and r_j must have the same A-value.

4. Translating Networks to Universal Relations

In this section we consider a network schema N with record types $R_1, \ldots, R_n$ and a set of attributes U. In principle, the data stored in a network database can also be represented as a collection of some relations. Since we have assumed that each attribute of N has a unique "role", it is possible to translate a network database for N to a universal relation over U. Therefore, U is the universal relation scheme that corresponds to the network schema N. A network database for N is translated to a universal relation over U as follows. Each R_i-traversal t can be viewed as a tuple over $\overline{R_i}$, since traversals are consistent. We extend t to a tuple over U by padding t with distinct marked nulls in the columns for $U-\overline{R_i}$. The universal relation is the set of all the tuples obtained in this way from the traversals of the network database, since traversals are all the data that can be retrieved by traversing correct access paths.

4.1. The Functional Dependencies of a Network Schema

We also have to explicitly express all the functional dependencies that are implicitly defined in the network schema N. This task presents a technical problem, since in some cases a record occurrence of a network database is uniquely identified only by its database key (and not by the values for its attributes). Therefore, we augment each record type R_i with a new attribute K_i. For all record occurrences r_i of type R_i, the K_i-value of r_i is the database key of r_i. We let K be the union of all the K_i, and $T_i=R_iK_i$. However, we still refer to the record types of the schema as $R_1, \ldots, R_n$. The set of all the

attributes (including the K_i) that appear in R_i and all its ancestors is denoted by $\overline{T}_i$. R_i-traversals are now tuples over the $\overline{T}_i$, and the universal relation scheme corresponding to N is UK (this is only temporary - until we show that database keys need not be considered). Note that each K_i appears in only one record type and cannot be part of any calc-key or search key.

Clearly, for each record type R_i of the network schema we have the FD $K_i \rightarrow \overline{T}_i$, which expresses the fact that a database key of a record occurrence r_i uniquely determines an R_i-traversal. Let $H(N)$ be the set of all FDs of the form $K_i \rightarrow \overline{T}_i$.

Proposition 1: A universal relation I that corresponds to a network database for N always satisfies $H(N)$.

The FDs of $H(N)$ provide very little information about the semantics of the data, since they only state the fact that no two record occurrences can be stored in the same location, and by Proposition 1 can never be violated in a network database. More meaningful FDs can be expressed only by calc-keys and search keys. Suppose that $R_{j_1}, \ldots, R_{j_m}$ is a path in the schema N, and $X_1, \ldots, X_m$ are nonempty sets of attributes such that X_m is a calc-key of R_{j_m} and for all $1 \leq i < m$, X_i is a search key of R_{j_i} in the set owned by $R_{j_{i+1}}$. Then $X = X_1 \cdots X_m$ is called a ***virtual key*** of R_{j_1}. In particular, if X is a calc-key of R_{j_1} (i.e., m=1), then X is a virtual key of R_{j_1}.

Example 1: Consider the network schema of Figure 4. Note that all calc-keys are underlined. All search keys contain $\sim$ over each attribute in the key. The set, to which the search key belongs, is indicated by the link whose tail emanates from the search key indicator (i.e., $\sim$). R_7 has a virtual key consisting of the calc-key B. R_1 has a virtual key that is composed of the calc-key B of R_7 and the search key D in the set connecting R_1 to R_7. R_3 has a virtual key consisting of the search key K in the set connecting R_3 to R_1, the search key D in the set connecting R_1 to R_7 and the calc-key B of R_7.

Lemma 1: Let X be a virtual key of R_i. Then for all X-values x, there is at most one R_i-traversal t such that $t[X]=x$.

Lemma 1 states the fact that the FD $X \rightarrow \overline{T}_i$ is satisfied in some local sense, i.e., when only R_i-traversals are considered. Therefore, it is reasonable to argue that the schema is intended to preserve this FD.[5] Let $G(N)$ be the set of all the FDs of the form $X \rightarrow \overline{T}_i$ where X is a virtual key of R_i. We view $G(N) \cup H(N)$ as the set of FDs of the network schema N.

4.2. Global Consistency and FD Preservation

A network database for N ***satisfies*** an FD $X \rightarrow Y$ if the corresponding universal relation satisfies $X \rightarrow Y$. The network schema N is ***globally consistent*** if all its databases satisfy the FDs of $G(N)$ (by Proposition 1, they satisfy $H(N)$); and it ***preserves*** a set of FDs D if all its databases satisfy D.

Intuitively, the database keys do not propagate any information about the semantics of the data, and we would like to consider only the following set of FDs.

$$F(N) = \{X \rightarrow \overline{T}_i \mid X \text{ is a virtual key of } R_i\}$$

Note that $F(N)$ is $G(N)$ with the K_i deleted.

Theorem 1: The set $F(N)$ is a cover of all the FDs over U that are implied by $G(N) \cup H(N)$.

Theorem 1 says that if N is globally consistent, then as far as implications of FDs over U are considered, the FDs involving the database keys do not buy us any new FDs over U (besides those that are already implied by $F(N)$). A network schema that preserves $F(N)$ might not be globally consistent, although this seems to happen only in pathological cases. A sufficient condition for the equivalence of global consistency and the preservation of $F(N)$ is stated in the following theorem.

Theorem 2: If every record type of N has a virtual key, then N is globally consistent if and only if it preserves $F(N)$.

4.3. Closed Network Schemas

The network schema N is ***closed*** if the following is true. For all distinct record types R_i and R_j such that

[5] This is also the view of [Zani].

$\overline{R_i} \rightarrow \overline{R_j}$ is implied by $F(N)$, record type R_j is an ancestor of R_i. The concept of a closed network schema is a natural one, since we expect that all the semantically correct many-one relationships be expressed in the network schema as links (or paths consisting of several links). The following theorem provides a sufficient condition for global consistency (even if some record types do not have any virtual key).

Theorem 3: If N is a closed network schema, then N is globally consistent.

4.4. Summary

The set $F(N)$ is always a cover of the FDs over U that are implied by the FDs of the schema N. If we are interested in global consistency, then if N is closed, it is also globally consistent; and whether N is closed can be determined from $F(N)$ alone. Even if N is not closed, but each record type has a virtual key, then obviously the database keys are redundant and global consistency depends only on $F(N)$. The only case where database keys might be important is when N is not closed and some record types do not have virtual keys, since it might be possible that some network databases for N satisfy $F(N)$ but do not satisfy $G(N)$.

5. The Design of a Globally Consistent Network Schema

Let F_0 be a set of FDs over U. In this section we show how to design a network schema N such that N is closed, every record type has a virtual key, and F_0 is preserved (i.e., $F(N)$ is a cover of F_0^+). At first we apply the synthesis algorithm of [Bern,BDB] to F_0. The result is a collection of relation schemes $P_1, \ldots, P_n$ with a set of keys for each P_i, such that the FDs implied by the keys form a cover of F_0^+. We create a network schema from $P_1, \ldots, P_n$ as follows. For each P_i we create a node (i.e., record type) R_i whose attributes are the same as the attributes of P_i. Each key of P_i becomes a calc-key of R_i.[6] The links of the network schema N are as follows. We introduced a link from R_i to R_j if $R_i \rightarrow R_j$ is implied by F_0 and there is no R_k ($k \neq i$ and $k \neq j$) such that both $R_i \rightarrow R_k$ and $R_k \rightarrow R_j$ are implied by F_0. Essentially, we introduce a link from R_i to R_j if the only way to create a path from R_i to R_j is by introducing that link. Clearly, the links of N correspond to many-one relationships between the record types that are implied by the FDs of F_0; and the network schema N is closed. Since no two relation schemes have equivalent keys [Bern], the network schema N has no cycles.

After creating the initial network schema, we apply the following transformations. The purpose of these transformations is to improve the design of the schema by storing each attribute in as few record types as possible. The transformations change the network N, but keep the closure of the preserved FDs invariant. The transformations can be applied in several ways, each producing a different result. Some rationale for applying the transformations is given in the examples that follow.

Transformation 1 (Delete an attribute). If an attribute A appears in R_i, but does not belong to any calc-key or search key of R_i, then A can be deleted from R_i provided that A appears also in some ancestor of R_i.

Transformation 2 (Add or delete a link). We can add a link from R_i to R_j if $\overline{R_i} \rightarrow \overline{R_j}$ is implied by $F(N)$, i.e., if there is already a path from R_i to R_j. Conversely, the link from R_i to R_j can be deleted if there is a path from R_i to R_j that does not use this link.

Transformation 3 (Add a new node). Suppose that X is a set of attributes without extraneous attributes, i.e., there is no attribute A in X such that $X\text{-}A \rightarrow A$ is implied by $F(N)$. If the network does not have a node with a virtual key equivalent to X, then we can add a new node R with a set of attributes X and a calc-key X. After adding the node R, links are created as follows. For every node R_j such that

(1) $R \rightarrow \overline{R_j}$ is implied by $F(N)$, and

(2) there is no descendant R_k of R_j such that $R \rightarrow \overline{R_k}$,

create a link from R to R_j. For every node R_j such that

(1) $\overline{R_j} \rightarrow R$ is implied by $F(N)$, and

(2) there is no ancestor R_k of R_j such that $\overline{R_k} \rightarrow R$,

[6] If a P_i has more than one key, then only one key can become a calc-key. In this case R_i is made a member of a singular set and the other keys of P_i are declared search keys in this set. We treat these search keys as if they were calc-keys.

create a link from R_j to R.

Transformation 4A (Replace a calc-key by a search key). Suppose that node R_i has a calc-key X, and an owner R_j with a virtual key Y such that $Y \subset X$. Then the calc-key X of R_i can be replaced with the search key $X-Y$ in the set owned by R_j.

The next transformation is similar to the previous one, but it also changes the set $F(N)$ of the preserved FDs. Clearly, it does not change the closure of $F(N)$.

Transformation 4B (Replace a calc-key by a search key). Suppose that R_i has a calc-key X and an owner R_j with a virtual key Y (and Y is not a subset of X). We can replace the calc-key X of R_i with the search key $X-Z$ (where $Z \subset X$) if the new $F(N)$ is equivalent to the old $F(N)$. Note that the new $F(N)$ is obtained from the old $F(N)$ by replacing $X \rightarrow \overline{R_i}$ with $(X-Z)Y \rightarrow \overline{R_i}$.

The application of the following transformation should be the last step of the design process.

Transformation 5 (Create singular sets). For every record type R_i without any owner, make R_i a member of a singular set.

5.1. Examples

Example 2: Suppose that the following relation schemes are synthesized from the FDs by the algorithm of [Bern,BDB].

$P_1(\underline{DBG})$, $P_2(\underline{HBGJ})$, $P_3(\underline{DBK}AH)$, $P_4(\underline{HJ}C)$ and $P_5(\underline{JM})$.

The initial network schema is given in Figure 1. Note that each key has become a calc-key in the network schema.

By applying Transformation 1, we delete attribute J from R_2, and attribute H from R_3. Transformation 4A is used to replace the calc-key HJ of R_4 with the search

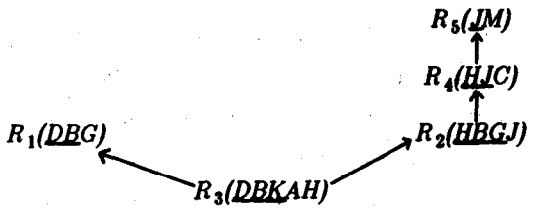

Figure 1

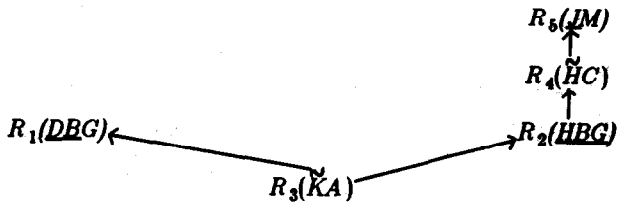

Figure 2

key H, and then an application of Transformation 1 deletes J from R_4. Similarly, the calc-key DBK of R_3 is replaced with the search key K in the set owned by R_1, and then D and B are deleted from R_3. The resulting network is shown in Figure 2.

In order to store B and G in only one place, we now apply the transformations as follows. First, we apply Transformation 3 to create a node R_6 consisting of the attributes BG, and then we apply it again to create a node R_7 consisting of B alone. By Transformation 1, we can now delete G from R_1. The result is shown in Figure 3. By repeated applications of Transformations 4A and 1, we do the following. First, the calc-key BG of R_6 is replaced with the search key G and B is deleted from R_6. Similarly, the calc-key HBG of R_2 is replaced with the search key H, and B and G are deleted from R_2. Finally, we apply Transformation 2 to add a link from R_1 to R_7, and then the calc-key DB of R_1 is replaced with the search key D and B is deleted from R_1. The final network is shown in Figure 4. We can apply Transformation 5 in order to make R_5 and R_7 members of singular sets and, hence, be able to find all record occurrences of these types one by one.

Example 3: Suppose that the relation schemes produced by the synthesis algorithm are

$P_1(\underline{ABCD})$, $P_2(\underline{BGH})$, $P_3(\underline{AEF})$, $P_4(\underline{BE})$ and $P_5(\underline{AFG})$.

The initial network is shown in Figure 5. By Transformations 1 and 4A, we delete F from R_3, replace the calc-key of R_2 by the search key G and delete B. By Transformation 3, we create a node R_6 consisting of E and make it an owner of both R_4 and R_3. Now we delete E from R_4; replace the calc-key of R_3 with the search key A in the set owned by R_6, and delete E from

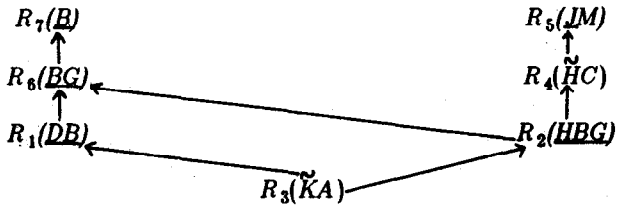

Figure 3

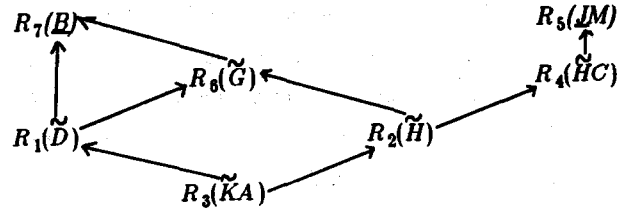

Figure 4

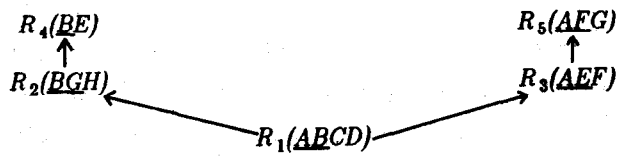

Figure 5

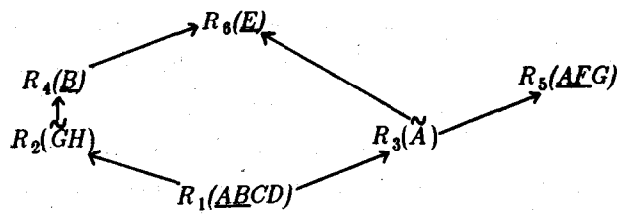

Figure 6

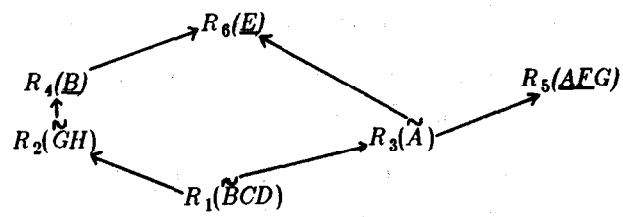

Figure 7

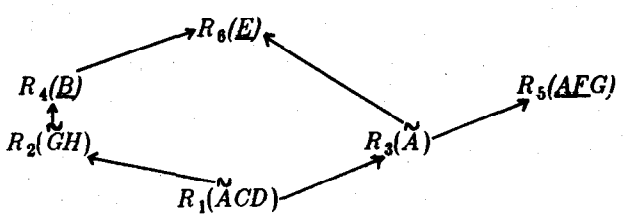

Figure 8

R_3. The result is shown in Figure 6. As for R_1, we can apply Transformation 4B either to replace the calc-key AB with the search key B in the set owned by R_3 and delete A as shown in Figure 7, or replace the calc-key AB with the search key A in the set owned by R_2 and delete B as shown in Figure 8. System owned sets can be created for R_6 and R_5.

Example 4: This example is intended to show that our design methodology produces schemas that are better than those produced by more traditional methods. Suppose that the attributes are C(ourse), T(eacher), and S(tudent), and the FDs are $CS \rightarrow T$ and $T \rightarrow C$. The relation schemes produced by the synthesis algorithm are

$P_1(\underline{TC})$ and $P_2(\underline{CST})$.

The initial schema has two record types and one set as shown in Figure 9. By applying Transformation 1, we delete T from R_2. By Transformation 3, we create a new node, $R_3(C)$, and make it an owner of both R_1 and R_2. Now we delete C from R_1, replace the calc-key of R_2 with the search key S in the set owned by R_3, and delete C from R_2. The result is shown in Figure 10. By Transformation 5, we can create a singular set for R_3. In practice, we may also choose to create a singular set for R_2 and make S a search key DA (i.e., DUPLICATES ALLOWED) in order to be able to find all record occurrences of type R_2 with a specified value.

It is interesting to compare this design with two other designs. The Entity-Relationship diagram [Chen] for this case produces the schema shown in Figure 11 (or a slight variation of this schema). Lien's method [Lien] that is based on multivalued dependencies (or, alternatively, loop-free Bachman diagrams [Bach]) produces the schema shown in Figure 12. Note that in the schema of Figure 12, R_2 and R_3 cannot have calc-keys. In practice, C should be made a calc-key DA (i.e, DUPLICATES ALLOWED) or a search key DA in a singular set for R_3, and similarly for S and R_2. S can also be made a search key DNA in the set owned by R_1. A similar action for C and R_3 is redundant, since each set occurrence of the link from R_3 to R_1 can have at most one member occurrence (because of the FD $T \rightarrow C$).

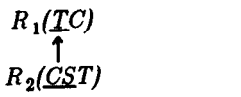

Figure 9

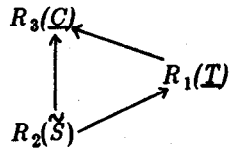

Figure 10

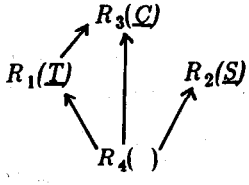

Figure 11

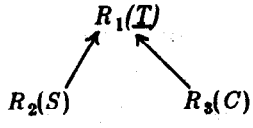

Figure 12

Suppose that we are given a CST-value to be inserted into the database. For each one of the three schemas, we now compute the average time needed to check that this new information does not violate the FDs. Let e be the average number of courses in which a student is enrolled, c be the average size of a class, and t be the average number of teachers per course. For each one of the three schemas, we first have to check whether the given T-value is already in the database. If it is, we have to find the corresponding C-value to make sure that the new CST-value does not violate $T \rightarrow C$; otherwise, we have to insert the new CT-value. For the schemas of Figures 10 and 11, this step takes $O(t)$ time if there are no owner pointers, and constant time if there are owner pointers.[3] For the schema of Figure 12 it takes constant time. Next we have to verify that the given S-value is not already related to the given C-value (i.e., the student is not already in the course), because if it is, then the new CST-value either is redundant or violates $CS \rightarrow T$. If there are owner pointers, it takes $O(e)$ time for the schemas of both Figure 11 and 12; if there are no owner pointers, it takes $O(ec)$ time for the schema of Figure 11, and $O(ec/t)$ time for the schema of Figure 12 (note that c/t is the average size of a set occurrence for the link from R_2 to R_1). In the case of the schema of Figure 10, all we have to do is try to insert the new S-value as a member of the set occurrence owned by the given C-value, and since S is a search key, the insertion will fail if the student is already in the course. The search for an S-value in a set occurrence takes $O(log\ c)$ time. Thus, testing whether the FDs are violated after an insertion takes less time in the case of the schema of Figure 10 (unless $e << c$ and there are owner pointers). It is also conceptually easier, since essentially what we have to do is try to insert the new S-value in the set occurrences for the specified C-value and T-value, and then check that the new S-traversal is consistent.

5.2. Remarks

The transformations can be applied in different ways to produce distinct schemas for the same set of FDs. Therefore, they have to be applied interactively, that is, after reviewing an intermediate schema, the user can decide which transformation should be applied next and how. In practice, the user might occasionally want to undo some of the applications. This can be accomplished by simply going back to a previous schema, or having new transformations that are the "inverse" of the above transformations. For simplicity's sake, we did not describe the "inverse" transformations (except for Transformation 2), but this can be done.

In Transformation 3, the new node that is created is required to be a set of attributes without extraneous attributes. The purpose of this requirement is to ensure that the new node consists of a set of attributes in third normal form. In most cases, we want to create a new node consisting of a set of attributes that is a subset of a calc-key and, hence, has no extraneous attributes.

Transformation 3 can be used to create a new node not only for the purpose of storing a set of attributes X in only one record type, but also in order to be able to store an X-value without having to store values for the other attributes in the record type where X currently is. For example, compare Figures 2 and 3 of Example 2. In Figure 3, we can store a B-value without storing either a DG-value or an HG-value, but this is not possible in Figure 2.

In order to always store an attribute A in only one location, we can create (using Transformation 3) a record type, say R, consisting of A, and substitute the database

[3] Owner pointers are pointers from each member occurrence to its owner occurrence.

key of R for A in all other record types containing A. For the advantages and exact details of this approach see [Kuck].

6. Conclusions

The main contribution of this paper is the design methodology for producing a network schema that preserves the functional dependencies specified by the user. This methodology is based on simple ideas, since the network model has suitable data structures for enforcing FDs. Furthermore, the proposed methodology produces, in many cases, schemas that are similar to those obtained by other techniques, but with the extra benefit of preserving the FDs.

The work reported here is an extension of the ideas presented in [KuSa]. In that paper we also started with a collection of synthesized relation schemes in third normal form and showed how to implement them as a network schema. We also showed how to translate queries posed on the universal relation scheme to network application programs. (These ideas have been implemented in the AURICAL system [CKPS]). However, since FDs were not preserved in the network schemas produced by [KuSa], testing whether an insertion satisfied all the FDs was rather complicated. Now, with the new design methodology, we can still translate relational queries to network application programs as described in [KuSa], and updates can be performed directly on the network schema. For any record type R, a new record occurrence r can be stored in the following way. For every record type R_i that owns R, we have to specify a value of a virtual key of R_i (to locate a set occurrence into which r should be inserted), try to store r, and if this does not fail because of a calc-key or search key violation, we only have to check that the new R-traversal (that starts at r) is consistent. If there are owner pointers, this process takes (on the average) $O(log\ m)$ time; otherwise, it takes $O(m)$ time, where m is the size of an average set occurrence in the database. If FDs are not preserved, then in the worst case (when the network schema is not closed) we might have to perform the chase process on the whole database. In many other cases, testing whether FDs are satisfied after storing a new record occurrence is at least $O(m^2)$. Note that deletions never violate the FDs, since all sets are AUTOMATIC MANDATORY.

The network schema we obtain is, in a strong sense, better than the synthesized relation schemes we start with. The network schema is globally consistent (i.e.,its databases satisfy the FDs in the sense of [Hone]) even if the relation schemes do not have this property (cf. [GrYa,Sa83]). The network schema is also less restricting in terms of updates, since we can create a new node with a set of attributes X and, thus, be able to store new X-values and still maintain the global consistency of the data. Thus, it is better to implement relation schemes as a network database rather than as pure relations.

If we start our design process with a set of relation schemes that are independent (i.e., globally consistent) [GrYa,Sa83], then not every legal update on the relation schemes can be translated to a legal update on the network schema, since all sets are AUTOMATIC MANDATORY. A design process that produces a globally consistent network schema for a set of independent relation schemes, such that all legal updates can be performed on the network schema, is given in [Kuck].

REFERENCES

[Bach] Bachman, C. W., "Data Structure Diagrams," *Data Base*, Vol. 1, No. 2 (1969), pp. 4-10.

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

[BDB] Biskup, J., U. Dayal, and P. A. Bernstein, "Synthesizing Independent Database Schemas," *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, 1979, pp. 143-151.

[Chen] Chen, P. P-S., "The Entity-Relationship Model - Toward a Unified View of Data," *ACM Trans. on Database Syst.*, Vol. 1, No. 1 (March 1976), pp. 9-36.

[CKPS] Chen, H.-H., S. Kuck, J. Peterson and Y. Sagiv, "A User's Manual for AURICAL: A Universal Relation Implementation via Codasyl," UIUCDCS-R-82-1114, Dept. of Computer Science, Univ. of IL at Urbana-Champaign, Urbana, IL, Dec. 1982.

[CODA] "CODASYL Data Description Language Journal of Development," Government Printing Office, Wash., D.C., June, 1973.

[Codd] Codd, E. F., "A Relational Model for Large Shared Data Banks," *Comm. ACM*, Vol. 13, No. 6 (June 1970), pp. 377-387.

[FMU] Fagin, R., A. O. Mendelzon, and J. D. Ullman, "A Simplified Universal Relation Assumption and Its Properties," *ACM Trans. on Database Systems*, Vol. 7, No. 3 (Sept. 1982), pp. 343-360.

[GrYa] Graham, M. H., and M. Yannakakis, "Independent Database Schemes," *Proc. Symp. on Principles of Database Systems*, 1982, pp. 199-204.

[Hone] Honeyman, P., "Testing Satisfaction of Functional Dependencies," to appear in JACM.

[Kuck] Kuck, S. M., "A Design Methodology for a Universal Relation Scheme Implementation via CODASYL," UIUCDCS-R-82-1106, Dept. of Computer Science, Univ. of IL at Urbana-Champaign, Urbana, IL, Sept. 1982.

[KuSa] Kuck, S. M. and Y. Sagiv, "A Universal Relation Database System Implemented Via the Network Model," *Proc. Symp. on Principles of Database Systems*, 1982, pp. 147-157.

[Lien] Lien, Y. Edmund, "On the Equivalence of Database Models," *Journal of the ACM*, Vol. 29, No. 2 (April 1982), pp. 333-372.

[MMS] Maier D., A. O. Mendelzon, and Y. Sagiv, "Testing Implications of Data Dependencies," *ACM Trans. on Database Systems*, Vol. 4, No. 4 (Dec. 1979), pp. 455-469.

[MUV] Maier, D., J. D. Ullman and M. Y. Vardi, "The Equivalence of Universal Relation Definitions," unpublished manuscript.

[Nava] Navathe, S. B., "An Intuitive Approach to Normalize Network Structured Data," *Proc. Int. Conf. on Very Large Data Bases, 1980, pp.350-358.*

[Sa81] Sagiv, Y.,"Can We Use the Universal Instance Assumption Without Using Nulls?" *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, 1981, pp. 108-120.

[Sa83] Sagiv, Y., "A Characterization of Globally Consistent Databases and their Correct Access Paths," to appear *ACM Trans. on Database Systems.*

[Ull] Ullman, J. D., *Principles of Database Systems*, Computer Science Press, Potomac, Maryland, 2nd edition, 1982.

[WoKa] Wong, E., and R. H. Katz, "Logical Design and Schema Conversion for Relational and DBTG Databases," *Proc. Int. Conf. Entity-Relationship Approach to Systems Analysis and Design*, UCLA, Dec. 1979.

[Zani] Zaniolo, C., "Design of Relational Views Over Network Schemas," *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, 1979, pp.179-190.