Logical Database Design with Inclusion Dependencies

Tok Wang LING and Cheng Hian GOH
Department of Information Systems & Computer Science
National University of Singapore

Abstract

Contrary to popular belief, relation schemes in good classical normal forms are not necessarily devoid of redundancies. This arises from the fact that classical data dependencies are oblivious to important constraints which may exist between sets of attributes occuring in different relation schemes. In this paper, we study how *inclusion dependencies (INDs)* can be used to model these constraints leading to the design of better database schemes. A new normal form, called *Inclusion Normal Form (IN-NF)*, is proposed. Unlike classical normal forms, the IN-NF characterizes a database scheme as a whole rather than the individual relation schemes. We show that a database scheme in IN-NF is always in Improved 3NF, while the converse is not true. Finally, we demonstrate how classical relational design framework may be extended to faciliate the design of database schemes in IN-NF.

1. Introduction

Within the last two decades, a number of *data dependencies* [14] have been introduced to facilitate the design of "good" relation schemes. The ones which have been studied extensively include *functional dependencies (FDs), multivalued dependencies (MVDs),* and *join dependencies (JDs).* These *classical* data dependencies gave rise to the definition of a number of *normal forms.* A relation scheme is said to be "good" if it is in Q normal form, where Q can be "third", "Boyce-Codd", "fourth", or "fifth". By designing relation schemes in good normal forms, we eliminate redundancies which may be present in a relation and circumvent the problem of updating anomalies [24]. Ling et al. [16] pointed out that classical normal forms failed to identify redundancies which may exist on a global scale. To address this problem, an *Improved Third Normal Form (Improved 3NF)* was introduced. It has been shown that if a database scheme D is in Improved 3NF, then each of the relation schemes in D is also in Codd 3NF. Furthermore, the database scheme as a whole is devoid of redundancies (with respect to the set of FDs which hold in the database).

It has been suggested that of all the data dependencies which have been proposed, FDs and *inclusion dependencies (INDs)* are probably the two most common kinds of constraints in a relational database [2]. Many researchers however held the notion that while INDs are important constraints in a database, they are irrelevant as far as logical database design is concern [18]. Thus it appears that with the introduction of the Improved 3NF, the logical database design problem is well-solved, at least for most practical purposes.

In this paper, we suggest that the earlier conclusion is fallacious. This is because INDs have an important impact on the classical design goal of minimality. Since classical normal forms (including the Improved 3NF) have failed to consider the effects of INDs on the structure of a database, they are inadequate in characterizing a database scheme which is truly devoid of redundancies. In consideration of the above, we propose a new normal form, called *Inclusion Normal Form (IN-NF),* which accounts for both the FDs and INDs which hold in a database scheme. We illustrate that a database scheme in IN-NF is always in Improved 3NF, while the converse is not true; in other words, the IN-NF is a stronger normal form which eliminates certain redundancies which remain undetected in the Improved 3NF. The design of a database scheme in IN-NF can be accomplished by extending the Deletion Normalization Algorithm proposed in [16]. This design step is shown to be equivalence preserving, conforming to the principles of the classical relational design framework.

The rest of this paper will be organized as follows. Section 2 of this paper gives a formal treatment of inclusion dependencies and some related results. Section 3 illustrates how INDs can be used to model important constraints which hold in a database. In section 4, we give a brief survey of logical database design within the classical database design framework and introduce the Improved 3NF proposed by Ling et al. [16]. Section 5 defines the *Inclusion Normal Form (IN-NF)* and suggests how the classical design framework can be extended to accommodate this new normal form. The last section gives the conclusion and highlights some future work.

2. Inclusion Dependencies

In this section, we provide the preliminary definitions followed by some important results for INDs. Our definitions here follow those in [6] for most part.

A *relation scheme* is an object R(U), where R is the name of the relation scheme and U is a finite sequence $<A_1, \ldots, A_m>$ of *attributes*. For simplicity, we sometimes write $A_1, \ldots, A_m$ for $<A_1, \ldots, A_m>$. To avoid ambiguous references, we also write R.A to refer to an attribute A in a relation scheme R. A *tuple* t over $U=<A_1, \ldots, A_m>$ is a sequence $<a_1, \ldots, a_m>$ where a_i is an element from the *domain* of A_i. A *relation* (over R(U), or simply over R) is a set of tuples over U. If $t = <a_1, \ldots, a_m>$ is a tuple over $U = <A_1, \ldots, A_m>$, and $X = <A_{i_1}, \ldots, A_{i_k}>$ where $i_1, \ldots, i_k$ are distinct members of $\{1, \ldots, m\}$, then t[X] is $<a_{i_1}, \ldots, a_{i_k}>$. If r is a relation over R, then $r[X] = \{t[X] \mid t \in r\}$. A *database scheme* $D = \{R_1(U_1), \ldots, R_n(U_n)\}$ (or simply $\{R_1, \ldots, R_n\}$) is a set of relation schemes. A *database* over D is a mapping which associates each relation scheme $R_i(U_i)$ with a relation r_i over R_i. In the rest of this paper, we will adopt the following notation: if R represents a relation scheme, then r denotes a relation over R; similarly, d represents a database over the database scheme D. We will also denote singleton attributes with letters from the front of the alphabet $(A, B, \cdots)$, and sets of attributes with letters from the back $(X, Y, \cdots)$.

Definition 1. Suppose $R_i(A_1, \ldots, A_m)$ and $R_j(B_1, \ldots, B_p)$ are two relation schemes (not necessarily distinct) in a database scheme $D = \{R_1, \ldots, R_n\}$. If X is a sequence of k distinct members of $A_1, \ldots, A_m$, and if Y is a sequence of k distinct members of $B_1, \ldots, B_p$, then we say that the *Inclusion Dependency (IND)* $R_i[X] \subseteq R_j[Y]$ holds in a database d if whenever r_i and r_j are relations in d, it must be the case that $r_i[X] \subseteq r_j[Y]$.

Casonova et al. [6] showed that the following rules form a complete axiomatization for INDs.

IND1 (*reflexivity*): $R[X] \subseteq R[X]$, if X is a sequence of distinct attributes of R.

IND2 (*projection & permutation*): if $R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]$, then $R[A_{i_1}, \ldots, A_{i_k}] \subseteq S[B_{i_1}, \ldots, B_{i_k}]$, for each sequence $i_1, \ldots, i_k$ of distinct integers from $\{1, \ldots, m\}$.

IND3 (*transitivity*): if $R[X] \subseteq S[Y]$ and $S[Y] \subseteq T[Z]$, then $R[X] \subseteq T[Z]$.

INDs of the form $R[X] \subseteq R[X]$ are said to be *trivial*. An IND is *nontrivial* if it is not trivial. Clearly we are interested only in INDs that are nontrivial. We say that the IND $R[X] \subseteq S[Y]$ is *unary* if X is singleton; otherwise, the IND is *n-ary*.

Inference rules for functional and inclusion dependencies taken together have also been studied [6, 7, 19]. Although it is still not clear whether a complete axiomatization exists for *finite* databases, the following lemma has been shown to be sound.

Lemma 1. (*Pullback Rule* [19]) If $R[XY] \subseteq S[WZ]$ (where $|X| = |W|$) and $W \to Z$, then $X \to Y$.

Proof. See [7].

3. Data Modeling using Inclusion Dependencies

It has been long realized that the relational data model offers the database designer little facilities for modeling semantics of the real world. This gave rise to *semantic data models* [15, 20] which invariably introduce some notions of "entities" and "relationships" for modeling real world objects and associations between these objects. (We will not attempt a formal definition of "entities" and "relationships" but instead suggest that they correspond roughly to the same discussed in the entity-relationship model [8].) Codd [11], in an attempt to reconcile the relational model with these, suggested that we may classify relations in a relational database into those corresponding to entities, entity properties, and relationships. We will adopt this view of a relational database since it allow us to refer to relations in our database more meaningfully.

In this section, we demonstrate that INDs can be used to model important real world constraints on data. We suggest that these constraints fall neatly into two categories, corresponding to those acting on real world entities and those acting on relationships. These correspond to the class of unary INDs and the class of n-ary INDs respectively.

3.1. Data Modeling using Unary INDs

Unary INDs in a relational database arise from a number of different sources. Before we can discuss these in greater detail, we need to define the notion of *entity keys*. Given an entity type E, we can find an attribute K_E called the *entity key*, such that every distinct entity of this type has a unique K_E-value corresponding to it.[†]. In a sense, the entity key K_E may be seen as a lexical surrogate for the real world entity. While it may be possible to find more than one such attribute (i.e., one which is capable of identifying the entity uniquely), we assume that one of these will be designated as the entity key at the database designer's discretion.

† For the sake of simplifying our discussion, we shall ignore the possibility of having composite keys for entity types. We suggest that such cases can be circumvented by renaming the composite key with a new attribute name, or by introducing an artificial attribute which serves as the key.

We assert that for each entity type in our universe of discourse, there exists a *base relation* corresponding to that entity type. The main purpose of the base relation is to list all the entity key values of entities of that type which are currently recorded in the database. (Codd [11] refer to this as the *E-relation*, although his definition differs from ours since a system-generated surrogate is being used in place of a user-specified entity key.) We will denote the base relation corresponding to the entity type E by R_E. It follows from this definition that for every other occurrence of K_E (in say a relation scheme R), the IND $R[K_E] \subseteq R_E[K_E]$ holds. Furthermore, if F is a *subtype* [23] of E, then $R_F[K_F] \subseteq R_E[K_E]$ also holds. Hence, we see that *referential integrity* and *subtype integrity* defined in [11] can be easily formulated in terms of unary INDs.

A third set of constraints which has an important impact on the structure of a database is the set of *participation* constraints. Suppose an entity type E participates in a relationship represented by the relation scheme R. Then there exists an attribute A (which may be distinct from K_E as a result of renaming, in which case it is called a *role name*) such that $R[A] \subseteq R_E[K_E]$ holds in compliance with referential integrity. If however we know that the entity's participation in this relationship is *total* (i.e., every element of this entity type must participate in this relationship), then the converse $R_E[K_E] \subseteq R[A]$ also holds.

Finally, the constraint "whenever an entity of type E participates in a relationship of type R_1, then it must also participate in relationship of type R_2" can be translated to the IND $R_1[A] \subseteq R_2[B]$ where A and B are role names of K_E (not necessarily distinct) in relation schemes R_1 and R_2.

3.2. Data Modeling using N-ary INDs

While every entity in the real world is represented in the database, this is not the case for real world "relationships". On the contrary, meaningful associations between entities are often broken down into basic relationships (during logical design) so that information will not be represented redundantly. Two attributes which do not appear in the same relation scheme may be related in some meaningful way. Furthermore, important constraints may exist between these meaningful associations.

We first examine the notion of *attribute compatibility* which determines how meaningful association between attributes can be identified. Loosely speaking, two attributes are compatible if it makes sense to perform an equi-join using them. Beeri and Korth [4] suggested that the attributes A and B (not necessarily distinct) in relation schemes R and S respectively are compatible if either $R[A] \subseteq S[B]$ or $S[B] \subseteq R[A]$. We find this definition too restrictive since it is possible that both of the INDs fail to hold and yet $r[A] \cap s[B] \neq \varnothing$. Instead we suggest the following definition. The attributes R.A and S.B are *compatible* if there exists a base relation T with key K such that $R[A] \subseteq T[K]$ and $S[B] \subseteq T[K]$.

Definition 2. Let D′ be a set of relation schemes $\{R_1, \ldots, R_m\}$ which is a subset of some database scheme D. We say that a relation scheme R is *pj-derived* from D_1 if every relation of R is obtained from relations in a database of D_1 using only the repeated application of

(i) the projection operator; and

(ii) the equi-join operator, such that two relations R and S are joined on attributes R.A and S.B only if R.A and S.B are compatible.

Intuitively, every pj-derived relation scheme embodies a relationship among attributes present in it.

Given a database scheme $D = \{R_1, \ldots, R_n\}$. In analogy to containment constraints acting on entities, we can also identify such constraints acting on relationships (or associations of attributes). These constraints take the form "every relationship of type R_1 is also an instance of relationship type R_2".

Example 1. Suppose we are given the attributes E(mployee#), EN(ame), D(epartment), O(ffice), P(hone), M(anager#), MN(ame), M(gr)P(hone), and the following FDs which hold in the database: $E \rightarrow \{EN, O\}$, $O \rightarrow \{P, D\}$, $P \rightarrow \{O\}$, $M \rightarrow \{MN, MP, D\}$, $D \rightarrow \{M\}$. We may model these information in the relation schemes as follows:

R1 ($\underline{E}$, EN, O)
R2 ($\underline{O}$, $\underline{P}$, D)
R3 ($\underline{M}$, MN, MP, $\underline{D}$)

Since every manager is an employee (a subtype relationship), clearly the IND $R3[M, MN] \subseteq R1[E, EN]$ holds. The constraint "if an employee is a manager of a department, then he/she must belong to that department" can be modeled as an IND $R3[M, D] \subseteq (R1 \bowtie R2)[E, D]$. Similary, the IND $R3[M, MP] \subseteq (R1 \bowtie R2)[E, P]$ holds in the database. □

Since we are concerned with containment relationships between groups of attributes, the INDs corresponding to these constraints are invariably n-ary. Furthermore, if the n-ary IND $R[A_1, \ldots, A_n] \subseteq S[B_1, \ldots, B_n]$ holds, then it must be the case that the unary INDs $R[A_i] \subseteq S[B_i]$ where $i=1, \ldots, n$ must also hold (by IND2). Although the converse result is not true in general, we can make use of the unary INDs (which are easier to identify) to help identify and validate the n-ary INDs.

4. Logical Database Design within Classical Relational Design Theory

Prior to discussing how INDs can have an impact on the design of "good" database schemes, we will first examine the classical framework for logical database design. More importantly, we address the question, "What is a *good* database scheme?" and introduce some classical results.

4.1. Characterizing a "Good" Database Scheme

In the seminal paper [3], Beeri et al. suggested that the logical database design problem can be summarized as follows: for a given set of attributes (the *initial database scheme* which comprises of a *universal relation scheme* R_0) and a given set of data dependencies Σ, find an *equivalent* database scheme $\{R_1, \ldots, R_n\}$ that is *better* in some sense. The goal here is to replace a relation r_0 over the universal relation scheme R_0 with a multi-relation database $\{r_1, \ldots, r_n\}$ where r_i is a relation over R_i. The relations $r_1, \ldots, r_n$ in this case is said to be a *decomposition* of r_0. This is desirable from the point of view of storage minimization as well as avoidance of updating anomalies. With these objectives in mind, Beeri et al. proposed the following design principles: representation, separation, and minimality.

4.1.1. The Principle of Representation

The *principle of representation* suggests that any database scheme derived from the first must be capable of representing the same information and no more. Since we assumed that the initial specification, comprising of a universal relation scheme R and a set of dependencies Σ, is a complete and error-free rendition of the real world information requirements, any further refinement of this model must therefore be a total representation of this initial scheme.

A first requirement for representation is *injectiveness* or *reconstructibility*, i.e., it should be possible to reconstruct the relation r_0 over R_0 from the relations over $\{R_1, \ldots, R_n\}$. The reconstruction operator for this purpose is usually the *join* operator. Suppose r_0 is a relation of R_0 satisfying Σ, and $r_i = r_0[R_i]$. We say that the database $\{r_1, \ldots, r_n\}$ is a *lossless* (or *nonloss*) decomposition of r_0 if $r_0 = \bowtie_1^n r_i$. The mapping from R_0 to $\{R_1, \ldots, R_n\}$ is injective if for every r_0 of R_0 satisfying Σ, $\{r_1, \ldots, r_n\}$ is a nonloss decomposition of r_0.

A second criterion for representation is *surjectiveness*. It has been suggested that a decomposition must represent an instance of the real world; i.e., any database $\{r_1, \ldots, r_n\}$ over $\{R_1, \ldots, R_n\}$ must result in a relation r_0 over R_0. When only FDs are present in Σ, this is to say that $\Sigma^+ = (\bigcup_1^n \Sigma_i)^+$, where Σ_i is the set of FDs which holds in R_i and F^+ denotes the closure of the set of FDs given by F. We sometimes refer to this by saying that $\{R_1, \ldots, R_n\}$ is a *dependency preserving* decomposition of R. Unfortunately, it is not clear how this characterisation can be extended to include other types of data dependencies such as (embedded) MVDs or JDs.

The two criteria taken together defines a surjective and injective mapping (i.e., a *bijection*) [21] between the databases of $\{R_0\}$ and that of $\{R_1, \ldots, R_n\}$. In this instance, we say that two database schemes are *equivalent*.

4.1.2. The Principle of Separation

The *principle of separation* suggests that "independent relationships" among attributes should be represented separately by different relation schemes. This offers several advantages [24]. First, it decrease the necessities of update propagations for maintaining given dependencies. Second, it makes it possible to insert and delete information without having to prepare data which are yet unavailable or without losing information (insertion and deletion anomalies). These difficulties are in fact the ones which first motivated the introduction of the various *normal forms*. It is conceived that by separating these "independent relationships" into basic information units, less effort is needed when updating the database. The relation schemes in good normal form thus collectively forms a "better" representation than the initial universal relation scheme.

4.1.3. The Principle of Minimal Redundancy

The *principle of minimal redundancy* is strongly related to the criteria of representation. If our concern is to preserve the information content, we may strive for minimal data representation. This can take two forms: first, there may exist a relation scheme R_j which is redundant, i.e., $r_0 = \bowtie_{i=1, i \neq j}^n r_i$; second, certain attributes may be redundant, i.e., there exists some $R_j' \subset R_j$ where $r_0 = (\bowtie_{i=1, i \neq j}^n r_i) \bowtie r_j'$.

On the other hand, if the concern is the representation of dependencies, then we similarly define an attribute to be redundant if its removal does not upset the fact that the FDs embodied in the database scheme forms a cover for Σ.

4.2. Logical Design Through Normalization

The desire to avoid update propagations and various other updating anomalies prompted the introduction of data dependencies and normal forms. The first data dependency known as *functional dependency (FD)* was introduced by Codd [9]. This is followed by the introduction of *multivalued dependency (MVD)* [12], and later on, *join dependency (JD)* [1] and many others (see [14]). These data dependencies give rise to a number of normal forms, such as "third" [9], "Boyce-Codd" [10], "fourth" [12], "projection-join" [13], etc. In the last two decades, research in relational database design has focused largely on the introduction of various data dependencies and "higher" normal forms. The "faith" in normal form is so strong that a relation scheme is often considered to be "good" if it is one of these normal forms. Logical database design thus became synonymous with *normalization*: the process of arriving at relation schemes in good normal form.

Ling et al. [16] showed that a collection of relation schemes in good normal form (in the classical sense) is not devoid of redundancies. This is due to the fact that classical design theory focused only on redundancies within a relation scheme and failed to observe that redundancies may continue to exist on a "global" level even though none exists in any relation by itself. To remedy this oversight, an *Improved Third Normal Form* was proposed.

Definition 3. Given a universal relation scheme R_0, and a set of FDs Σ (we sometimes denote this by writing $<R_0,\Sigma>$), a database scheme $D = \{R_1(U_1), \ldots, R_n(U_n)\}$ is said to be a *preparatory database scheme* (abbreviated PDS) derived from R_0 and Σ if

(i) the set of FDs implied by the keys of the relation schemes (i.e., the set of all FDs $\{K \rightarrow U_i \mid K$ is a key of $R_i(U_i)\})$ forms a minimal cover for Σ, and

(ii) no two distinct relation schemes have equivalent keys (i.e., whenever K_i and K_j are keys of R_i and R_j respectively, it is not the case that $K_i \rightarrow K_j$ and $K_j \rightarrow K_i$).

(iii) the relation schemes $R_1, \ldots, R_n$ form a nonloss decomposition of R_0.

A PDS of $<R_0,\Sigma>$ can be obtained by using a *Preparatory Algorithm* [16]. This algorithm is similar to Bernstein's synthesis method [5] except that the output is augmented to guarantee reconstructibility. Intuitively, the Preparatory Algorithm separates the attributes in the universal relation scheme, such that each relation scheme now embodies an independent relationship among its attributes. Clearly, a PDS $D = \{R_1, \ldots, R_n\}$ obtained in this manner is equivalent to the original database scheme described by the universal relation scheme R_0 since both covering and reconstructibility are guaranteed.

Give a PDS $D=\{R_1(U_1), \ldots, R_n(U_n)\}$, we denote the FDs implied by the keys of the relation schemes, excluding B in R_i, by $G'_i(B)$; i.e., $G'_i(B) = \bigcup_{j=1, j\neq i}^{n} \{K \rightarrow U_j - K \mid K$ is a key in $R_j\} \cup \{K \rightarrow U_i - K - B \mid K$ is a key in R_i and $B \notin K\}$.

Definition 4. Let R_i be a relation scheme in a PDS. An attribute B in R_i is said to be *restorable* if there exists a key K of R_i, not containing B, such that $K \rightarrow B$ can be inferred from $G'_i(B)$. In other words, the B-value in a relation over R_i can always be recovered from the other relations in a database.

Definition 5. Let R_i be a relation scheme in a PDS. An attribute B in R_i is said to be *nonessential* if whenever B is contained in a key K of R_i, there exists another K' of R_i not containing B such that $K \rightarrow K'$ can be inferred from $G'_i(B)$. In other words, B is not needed to derive any attribute in R_i.

Definition 6. An attribute B in R_i is said to be *superfluous* if it is both restorable and nonessential.

Definition 7. A preparatory database scheme $D=\{R_1, \ldots, R_n\}$ is in *Improved Third Normal Form* if there are no superfluous attributes in any of the relation schemes in D.

The Improved 3NF can be achieved in practice using the *Deletion Normalization Algorithm* proposed by Ling et al. This algorithm comprises of the following steps.

Step One. (Generate a Preparatory Schema.) Given $<R_0,\Sigma>$, we make use of the Preparatory Algorithm to arrive at a PDS D.

Step Two. (Delete all superfluous attributes.) For each superfluous attribute that is in D, delete that attribute from the corresponding relation scheme, resulting in a new database scheme. Repeat this step until no more superfluous attributes can be found.

Clearly, an attribute which is restorable in R_i can be eliminated from the relation scheme without affecting reconstructibility. It can also be shown that if B is both restorable and nonessential in R_i, then B can be eliminated from R_i without affecting dependency preservation. The deletion of a superfluous attribute (in step two) therefore result in a database scheme which is equivalent to the original one.

Example 2. Suppose the initial database scheme comprises of the attributes `Stu#,SName, Course#,CName,Mark,Year` and we are given the following FDs

```
Stu# → SName
Course# → CName
CName → Course#
Stu#,Course# → Mark
Stu#,CName → Year
```

The Preparatory Algorithm produces the following relation schemes:

```
R1(Stu#,SName)
R2(Course#,CName)
R3(Course#,Stu#,CName,Mark,Year)
```

Either `Course#` or `CName` is superfluous in `R3`, since its removal does not affect covering or reconstructibility. If either `Course#` or `CName` is removed from `R3`, then the resulting database scheme is in Improved 3NF. □

Theorem 1. If a database scheme D is in Improved 3NF, then every relation scheme in D is in Codd 3NF.

Proof: See [16].

5. A Normal Form for Functional and Inclusion Dependencies

While it has been generally acknowledged that FDs and INDs together constitute the most commonly occurring constraints in a relational database [2], many researchers held the notion that INDs do not play any role in determining the logical structure of a database [18]. With the introduction of the Improved 3NF, it therefore appears that the problem of logical database design is well-solved (at least for most practical purposes in which "higher" dependencies such as MVDs and JDs are deemed unrealistic).

In this paper, we contend that this conclusion is fallacious. This is because INDs in fact have an important impact on the traditional design goals of achieving minimality and avoidance of updating anomalies. We substantiate this statement with the following example.

Example 3. Consider the database scheme presented in Example 1. Each of the relation schemes are clearly in 5NF. Furthermore, the database scheme is also in Improved 3NF. Nevertheless, the database is certainly not devoid of redundancies. For instance, the attribute `MN` in `R3` is clearly redundant since it can always be derived from `R1`. By eliminating this attribute we obtain a database scheme which is clearly better than the original one from the view point of storage minimization. Furthermore, the IND `R3[M,MN#]` $\subseteq$ `R1[E,EN]` is now reduced to a unary IND involving only `M` and `E`, which is probably cheaper to enforce. □

The above example suggests clearly that classical normal forms (including the Improved 3NF) are not adequate in characterizing a database which is nonredundant. Furthermore, since INDs are qualitatively different from FDs, the redundancies induced by the interaction between INDs and FDs cannot be circumvented by simply identifying all the FDs that are implied, and afterwhich applying classical synthesizing or decomposition methods for normalizing the database. (In this respect, we remark that undue attention has been accorded to identifying a complete axiomatization of FDs and INDs [6, 7, 19], which have yet to yield any success.) In Example 3 for instance, all relevant FDs have in fact been identified (right from the start of the design process) based on the semantics of the data elements involved; nevertheless, the database scheme is not free from redundancies.

In the following definitions (8 through 11), we assume that $D = \{R_1, \ldots, R_n\}$ is a PDS, and we let Φ be the set of INDs which hold in D.

Definition 8. An attribute B in a relation scheme $R_i \in D$ is said to be *weakly restorable* if there exists a key K of R_i, not containing B, such that $K \rightarrow B$ can be derived from $G'_i(B) \cup \Phi$ using Armstrong's axioms and the Pullback Rule.

Clearly, whenever B in R_i is restorable, it is also weakly restorable. However, we need to show that whenever B in R_i is weakly restorable, it can indeed be removed without affecting reconstructibility as is the case when it is restorable.

Lemma 2. Suppose D′ is the database scheme obtained from D by removing the attribute B from $R_i \in D$. If B in R_i is weakly restorable, then every database of D can be reconstructed from a database of D′.

Proof. Due to space limitation we will give only a sketch of the proof here. It has been shown that if B in R_i is restorable, then it can be removed without affecting reconstructibility. If B in R_i is weakly restorable but not restorable, then it must be the case that there exists a relation scheme S pj-derived from some relation schemes (not including R_i) such that the IND $R[XB] \subseteq S[YA]$ holds, and $Y \rightarrow A$ can be inferred from $G'_i(B)$. It is now sufficient to show that whenever $R_i[XB] \subseteq S[YA]$ and $Y \rightarrow A$ hold, then B can be eliminated from R_i without affecting reconstructibility. Let R'_i be the relation scheme obtained from R_i after removing B. Suppose r_i and s be relations over R_i and S respectively. It can be shown that, $r_i[XB] = r_i[XB] \bowtie r_i[X] = \pi_{XA}(s[YA] \bowtie_{Y=X} r'_i[X])$, where r'_i is the corresponding relation over R'_i, since $r_i[XB] \subseteq s[YA]$ and $Y \rightarrow A$. □

Definition 9. An attribute B in a relation scheme $R_i \in D$ is said to be *weakly nonessential* if whenever B is contained in a key K of R_i, there exists another key K′ of R_i not containing B such that $K \rightarrow K'$ can be derived from $G'_i(B) \cup \Phi$ using Armstrong's axioms and the Pullback Rule.

It is again clear that whenever an attribute B is R_i is nonessential, it is weakly nonessential.

Definition 10. An attribute B in R_i is said to be *weakly superfluous* if it is both weakly restorable and weakly nonessential.

Theorem 2. If D′ is a database scheme obtained from D by removing from it $R_i.B$ that is weakly superfluous, then the database schemes D and D′ are equivalent.

Proof. We have shown (in Lemma 2) that every weakly restorable attribute can be removed without affecting reconstructibiliy. While (FD) covering is not guaranteed to be preserved when a weakly superfluous attribute is being removed, we will show that every database d′ of D′ cannot violate any FDs implied by the keys of D. The sketch of the proof is as follows.

Since B in R_i is weakly restorable, then there exists a key K_1 of $R_i(U_i)$ not containing B such that $K_1 \rightarrow B$ can be inferred from $G'_i(B) \cup \Phi$. In other words, this constraint can be enforced in the database scheme independently of B in R_i. Let K be any key of R_i. If B is not in K, then obviously $K \rightarrow B$ can be enforced independently of B in R_i since $K \rightarrow K_1$ and $K_1 \rightarrow B$ follows from $G'_i(B) \cup \Phi$. Suppose now B is in K. Since B is weakly nonessential, there exists a key K_2 of R_i not containing B such that the FD $K \rightarrow K_2$ can be inferred from $G'_i(B) \cup \Phi$. Since K_2 is a key of $R'_i(U'_i)$ (obtained from $R_i(U_i)$ by removing from it the attrbute B), $K_2 \rightarrow U'_i$ and thus $K \rightarrow U'_i$ is enforced independently of B in R_i. Thus, B can be removed from R_i without introducing a database state which violates some FDs implied by the keys in D. Removing a weakly superfluous attribute from its relation scheme is therefore equivalence preserving. □

Definition 11. A PDS $D=\{R_1, \ldots, R_n\}$ is in *Inclusion Normal Form (IN-NF)* if there are no weakly superfluous attributes in any of the relation schemes.

It is worthwhile pointing out that Mannila and Raiha have proposed an *Inclusion Dependency Normal Form (IDNF)* [17] which suggests that database designers should strive for database schemes in which INDs are *key-based* and *noncircular*. Their IDNF however did not address the classical relational design goals (of minimality and avoidance of updating anomalies) and also assumed that all INDs which are not key-based are irrelevant to logical design. The IN-NF proposed in this paper on the other hand is consistent with classical relational framework, and give rise some very nice results, as seen in the next theorem.

Theorem 3. Every database scheme in IN-NF is also in Improved 3NF.

Proof: It suffices to prove that every superfluous attribute in any R in a database scheme D is weakly superfluous. This is trivial since every restorable attribute is weakly restorable and every nonessential is also weakly nonessential. □

We emphasize that the converse of this theorem is not true; i.e., a database scheme in Improved 3NF is not necessarily in IN-NF. Thus IN-NF is a "higher" normal form than Improved 3NF analogous to saying that 4NF is a higher normal form than Codd 3NF. Since relation schemes in a database scheme in Improved 3NF are guaranteed to be in 3NF (see Theorem 1), we have the following corollary.

Corollary 1. If D is a database scheme in IN-NF, then every relation scheme in D is in Codd 3NF.

Database schemes in the proposed IN-NF can be achieved by extending the the Deletion Normalization Algorithm:

Step One. Given a universal relation scheme and a set of functional dependencies, we obtain a PDS using the Preparatory Algorithm. Having done this, it is possible for certain real world semantics to be attributed to each relation scheme. For instance, a data analyst might be able to identify relation schemes representing entity, properties, and relationships. Intuitively, each relation scheme in this database scheme embodies an independent relationship.

Step Two. Nontrivial INDs which hold in this database can now be identified by the data analyst using guidelines suggested in Section 3.

Step Three. The database scheme can be transformed into one in IN-NF by eliminating those attributes which are weakly superfluous.

Ling et al. [16] have shown that the Deletion Normalization Algorithm has a polynomial execution time. We expect the same bounds to apply here.

Example 4. We will make use of the scenario given in Example 1 to illustrate how database schemes in IN-NF can be achieved in practice. Using the Preparatory Algorithm, we would have obtained the same relation schemes as given earlier. It is possible to attribute certain semantics to each relation scheme. For instance, it is clear that R1 embodies information of the EMPLOYEE entity type, R2 of OFFICE, and R3 of MANAGER. As before, we can identify the INDs

$$R3[M,D] \subseteq (R1 \bowtie R2)[E,D]$$
$$R3[M,MP] \subseteq (R1 \bowtie R2)[E,P]$$
$$R3[M,MN] \subseteq R1[E,EN]$$

It can be shown easily that both R3.MN and R3.MP are weakly superfluous since each is (weakly) nonessential and weakly restorable. Although R3.D is restorable, it is not weakly nonessential ($D \rightarrow M$ cannot be inferred from $G'_3(D)$ and the INDs identified above). Thus the database scheme obtained from {R1,R2,R3} after removing R3.MN and R3.MP is in IN-NF.

Suppose for the sake of illustration, every employee can be identified uniquely by their name. Hence $EN \rightarrow E$

and MN→M hold. The Preparatory Algorithm results in the same database scheme as before, except that EN and MN are now keys of the relation schemes R1 and R3. Clearly, R3.MN is weakly restorable as before. Although MN is not nonessential (MN→M cannot be implied from G'_3(MN), it is weakly nonessential since R3[M,MN]⊆R1[E,EN] and EN→E is implied by the key of R1. Hence R3.MN is weakly superfluous and should be removed to result in a database scheme which is in IN-NF. □

6. Conclusion

While Inclusion Dependencies (INDs) have been recognized as important constraints in a relational database, it is traditionally seen as irrelevant to logical database design. In this paper, we show that contrary to this commonly held notion, INDs play an important role in accomplishing logical relational design objectives of minimality and avoidance of updating anomalies. Classical normal forms as well as design approaches are found to be inadequate in view of this finding. To remedy this deficiency, we proposed an Inclusion Normal Form (IN-NF). It is shown that a database scheme in IN-NF is also in Improved 3NF (while the converse is not true), and consequently, every relation scheme in such a database scheme is also in Codd 3NF. The Deletion Normalization Algorithm is extended to facilitate the design of a database scheme in IN-NF.

Much of the difficulty in designing a database scheme in IN-NF lies with problems in identifying the INDs which hold. Since INDs dependencies are inherently inter-relational, they cannot be readily identified in the the universal relation scheme (which forms the starting point for logical design in the classical relational design framework). We suggest that this problem can only be overcomed by injecting greater semantics into the design process. We are currently examining how this can be accomplished using an Entity-Relationship model, in which INDs have a natural interpretation.

References:

[1] Aho, A.V., C. Beeri, and J.D. Ullman, "The theory of joins in relational data bases," *ACM TODS*, Vol 4, 1979, pp. 297-314.

[2] Atzeni, P., and E.P.F. Chan, "Independent database schemes under functional and inclusion dependencies," *Proc. 13th VLDB*, Brighton, 1987, pp. 159-166.

[3] Beeri, C., P.A. Bernstein, and N. Goodman, "A sophisticate's introduction to database normalization theory," Proc. 4th VLDB, 1978, pp. 113-124.

[4] Beeri, C., and H.F. Korth, "Compatible attributes in a universal relation," *Proc. ACM PODS*, 1982, pp. 55-62.

[5] Bernstein, P. A., "Synthesizing Third Normal Form Relations from Functional Dependencies," *ACM TODS*, 1:4, Dec 76, pp. 277-298.

[6] Casanova, M.A., R. Fagin, and C.H. Papadimitriou, "Inclusion dependencies and their interaction with functional dependencies," (*Extended Abstract*), *Proc. ACM PODS*, 1982, pp. 171-176.

[7] Casanova, M.A., R. Fagin, and C.H. Papadimitriou, "Inclusion dependencies and their interaction with functional dependencies," *J. of Comp. and Sys. Sc.*, 28, 1984, pp. 29-59.

[8] Chen, P. P., "The entity-relationship model: towards a unified view of data," *ACM TODS*, 1:1, 1976, pp. 9-36.

[9] Codd, E.F., "A relational data model for large shared data banks," *CACM 13:6*, 1970, pp. 377-387.

[10] Codd, E.F., "Further normalization of the data base relational model," *Data Base Systems* (R. Rustin, ed.), Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 33-64.

[11] Codd, E.F., "Extending the database relational model to capture more meaning," *ACM TODS* 4:4, 1979, pp. 397-343.

[12] Fagin, R., "Multivalued dependencies and a new normal form for relational databases," *ACM TODS*, 2:3, 1977, pp. 262-278.

[13] Fagin, R., "Normal forms and relational database operators," *Proc. ACM SIGMOD*, 1979.

[14] Fagin, R., "The theory of data dependencies -- a survey," *Proc. of Sym. in Applied Maths*, Vol 34, 1986, pp. 19-71.

[15] Hull, R., and R. King, "Semantic database modeling: survey, applications, and research issues," *ACM Comp. Surveys*, 19:3, Sept 1987, pp. 201-260.

[16] Ling, T.W., F. W. Tompa, and T. Kameda, "An improved third normal form for relational databases," *ACM TODS*, 6:2, 1981, pp. 329-346.

[17] Mannila, H., and K.-J. Raiha, "Inclusion dependencies in database design," *Proc. 2nd ICDE*, 1986, pp. 713-718.

[18] Missaoui, R., and R. Godin, "The implication problem for inclusion dependencies: a graph approach," *SIGMOD RECORD*, 19:1, 1990, pp. 36-40.

[19] Mitchell, J.C., "Inference rules for functional and inclusion dependencies," *Proc. ACM PODS*, 1983, pp. 58-69.

[20] Peckham, J., and F. Maryanski, "Semantic data models," *ACM Comp. Surveys*, 20:3, Sep 1988, pp. 153-189.

[21] Rissanen, J., "Independent components of relations," *ACM TODS* 2:4, 1977, pp. 317-325.

[22] Sciore, E., "Inclusion dependencies and the universal instance," *Proc. ACM PODS*, 1983, pp. 48-57.

[23] Smith, J. M., and D. C. P. Smith, "Database abstractions: aggregation and generalization," *ACM TODS* 2:2, 1977, pp. 105-133.

[24] Ullman, J. D., *Principles of Database and Knowledge-Base Systems*, Vol I, Computer Science Press, 1988.