A Generalisation of Entity and Referential Integrity in Relational Databases

Mark Levene
Department of Computer Science
University College London
Gower Street
London WC1E 6BT, U.K.
Email: mlevene@cs.ucl.ac.uk
Phone: 0171 419 3684
Fax: 0171 387 1397

Corresponding author: George Loizou
Department of Computer Science
Birkbeck College
Malet Street
London WC1E 7HX, U.K.
Email: george@dcs.bbk.ac.uk

November 17, 2000

Abstract

Entity and referential integrity are the most fundamental constraints that any relational database should satisfy. We re-examine these fundamental constraints in the context of incomplete relations, which may have null values of the types "value exists but is unknown" and "value does not exist". We argue that in practice the restrictions that these constraints impose on the occurrences of null values in relations are too strict. We justify a generalisation of the said constraints wherein we use key families, which are collections of attribute sets of a relation schema, rather than keys, and foreign key families which are collections of pairs of attribute sets of two relation schemas, rather than foreign keys. Intuitively, a key family is satisfied in an incomplete relation if each one of its tuples is uniquely identifiable on the union of the attribute sets of the key family, in all possible worlds of the incomplete relation, and, in addition, is distinguishable from all the other tuples in the incomplete relation by its nonnull values on some element in the key family. Our proposal can be viewed as an extension of Thalheim's key set, which only deals with null values of type "value exists but is unknown". The intuition behind the satisfaction of a foreign key family in an incomplete database is that each pair (F_i, K_i) of attribute sets in the foreign key family takes the foreign key attribute values over F_i of a tuple in one incomplete relation referencing the key attribute values over K_i of a tuple in another incomplete relation. Such referencing is defined only in the case when the foreign key attribute values do not have any null values of the type "value does not exist"; we insist that the referencing be defined for at least one such pair. We also investigate some combinatorial properties of key families, and show that they are comparable to the standard combinatorial properties of keys.

1 Introduction

The notions of *entity integrity* arising from the choice of a *primary key* and *referential integrity* arising from the choice of *foreign keys* are the most fundamental constraints in a relational database [Cod79, Cod90].

A primary key, which is a set of attributes designated by the user, is satisfied in a relation, if each tuple in the relation is uniquely identified by the primary key values and, in addition, the primary key must be a minimal set of attributes for which this uniqueness property holds. We quote from Codd's definition of entity integrity [Cod90]:

"No component of a primary key is allowed to have a missing value of any type".

Thus entity integrity can be viewed as a restriction on the occurrence of null values in a relation which may be incomplete. To our knowledge there has been no attempt to date, apart from [Tha89], to re-examine the validity of entity integrity, despite the fact that there is a growing literature on incomplete information in relational databases (see [Mai83, AHV95]; for alternative treatments to dealing with incomplete information we refer the reader to [LL99, Chapter 5]). We think that this is mainly due to the fact that in most of the theory of relational database design it is implicitly assumed that relations do not contain null values, in which case entity integrity is enforced by default. Moreover, support for null values in current relational database management systems is still in its infancy. Our generalisation of entity integrity can be viewed as an extension of Thalheim's formalism [Tha89] to the case when relations may contain occurrences of both categories of null value defined below.

A foreign key, which is a set of attributes in one relation schema referencing the primary key attributes of another relation schema, is satisfied in a database, if whenever the foreign key values of a tuple in the referencing relation do not contain null values, then these values are the primary values of some tuple in the referenced relation. Referential integrity enforces the satisfaction of such foreign key constraints [Dat86, HR96]. We continue to quote from Codd's definition of entity integrity [Cod90]:

"No component of a foreign key is allowed to have an I-marked value",

where an I-marked value in our terminology is a null value of the type "value does not exist". Thus foreign key values also restrict the occurrence of null values in a relation which may be incomplete. A recent formalisation of the notion of foreign key, which caters for incomplete relations, can be found in [LL97].

Missing information generally falls into two categories: applicable information, for example, the name of a person may be unknown, and inapplicable information, for example, a person may not have a spouse. In both cases the missing information can be modelled by special values, called *null values*, which act as place holders for the information that is missing. Varied interpretations of null values within these two categories were listed in [ANS75].

In order to model these two categories of missing information we add to the database domains two types of null value:

1) "value at present exists but is unknown" [Cod79], which is denoted in the database by the distinguished value *unk*, and

2) "value does not exist", or equivalently, "value is inapplicable" [LL86], which is denoted in the database by the distinguished value *dne*.

We note that *dne* is very useful when filling in forms where some of the categories in the form may be filled in as inapplicable. As opposed to *unk*, the null value *dne* does not arise due to incompleteness of information. Despite this fact *dne* cannot be treated as just another

nonnull value; for example, we can record the fact that a person is unmarried by having *dne* as their spouse attribute value but we cannot say that two unmarried people have the same spouse. Moreover, entity integrity forbids primary key values to be null values of any type, and referential integrity forbids foreign key values to be of the null type *dne*.

To motivate our approach we give four examples showing that entity and referential integrity are too strict in practice. Consider the relation, say r, shown in Table 1 over a relation schema containing the two attributes NAME and ADDRESS, and assume that {NAME, ADDRESS} is the primary key of R. It can easily be seen that r violates entity integrity. Despite this fact all the tuples in r are uniquely identifiable, since the problematic third tuple is the only tuple having the name Sue Jones. Thus in all possible worlds, say s, of r, where the occurrence of *unk* is replaced by a nonnull value, {NAME, ADDRESS} uniquely identifies each tuple in s, and also there is a one-to-one correspondence between the tuples of r and s. So as long as each tuple in a relation r is uniquely identifiable as a distinct entity by the nonnull portion of its primary key values, we can consider the relation to satisfy entity integrity.

NAME	ADDRESS
John Smith	Hampstead Way
John Smith	Harold Rd
Sue Jones	*unk*

Table 1: A relation showing that entity integrity is too strict

As a second motivating example consider a relation schema R containing the attributes NAME, ADDRESS and DOB (date of birth), and assume that {NAME, ADDRESS, DOB} is the primary key of R. It can easily be seen that the relation, say r, over R shown in Table 2 violates entity integrity. Despite this fact all the tuples in r are uniquely identifiable by the nonnull portion of their primary key values. As in the previous example, in all possible worlds, say s, of r, where each occurrence of *unk* is replaced by a nonnull value, {NAME, ADDRESS, DOB} uniquely identifies each tuple in s, and also there is a one-to-one correspondence between the tuples of r and s.

NAME	ADDRESS	DOB
John Smith	*unk*	13/6/95
Sue Jones	Harold Rd	*unk*
unk	Hampstead Way	17/12/96

Table 2: Another relation showing that entity integrity is too strict

As a third motivating example for generalising entity integrity, consider a relation schema S containing the attributes SS# (social security number), P# (passport number) and NAME, and assume that SS# and P# are candidate keys of S. It is evident that the relation, say s, over S, shown in Table 3, violates entity integrity, since either SS# or P# is the primary key of S. Despite this fact, due to the semantics of *dne*, the first and second tuples of s represent distinct entities, since s is the only possible world of itself. Thus if, at some later stage, John

Smith of the first tuple acquires a P# it cannot be 2, and if, at some later stage, John Smith of the second tuple acquires a SS# it cannot be 1. This would not have been the case had the null values in s been of type *unk*, since then there would be a possible world of s in which both the tuples in s represent the same entity. Moreover, in this example every tuple in s is distinguishable by nonnull values either by SS# or P#, since each value of SS# or P# is unique.

SS#	P#	NAME
1	*dne*	John Smith
dne	2	John Smith

Table 3: Yet another relation showing that entity integrity is too strict

As a final example, we motivate the generalisation of referential integrity, by considering a relation schema R containing the attributes DEPT_NAME, MGR_SS# and MGR_P#, and assuming that MGR_SS# is a foreign key for R referencing the candidate key SS# for S, and MGR_P# is a foreign key for R referencing the candidate key P# for S. The need for alternative foreign keys, one for each candidate key, is due to the null values, as we cannot be sure whether SS# or P# is the primary key of S. It is evident that the database $d = \{r, s\}$, where r over R is shown in Table 4 and s over S is shown in Table 3, violates referential integrity. Despite this fact, the MGR_SS# value of the first tuple in r, i.e. <1>, references the SS# value of the first tuple in s, and the MGR_P# value of the second tuple in r, i.e. <*unk*>, references the P# value of the second tuple in s. Thus for each tuple in r, whenever the values of a foreign key do not have an occurrence of *dne*, then these values reference the corresponding candidate key values of a tuple in s. Moreover, for each tuple in the referencing relation, there is at least one foreign key whose values do not include *dne*.

DEPT_NAME	MGR_SS#	MGR_P#
Computing	1	*dne*
Mathematics	*dne*	*unk*

Table 4: A relation demonstrating that referential integrity is too strict

The approach we take in order to solve the problems raised in the above examples is to relax the notions of a key and foreign key thus allowing us to generalise entity integrity and referential integrity.

Given a relation r over a relation schema R, we define a *key family* $\mathbf{K}$ to be a family of subsets of R. A key family $\mathbf{K}$ is satisfied in the relation r, if the set of all attributes appearing in the elements of $\mathbf{K}$ is a key in all possible worlds of r, and, in addition, each tuple $t \in r$ is distinguishable from all the other tuples in r by the nonnull values of the attributes of some $K_i \in \mathbf{K}$. As an example, the key family {{NAME}, {ADDRESS}} is satisfied in the relation r shown in Table 1. This key family is also a *minimal key family* in r, since it has no redundant element and no element in it has redundant attributes. As another example, the key family {{NAME, ADDRESS}, {NAME, DOB}, {ADDRESS, DOB}} is satisfied in the relation r shown in Table 2. This key family is not minimal in r, since the key family

{{NAME}, {ADDRESS}, {DOB}} is satisfied in r; however, it is minimal in the relation resulting from adding the tuple <John Smith, Harold Rd, 17/12/96> to r. Finally, the key family {{SS#}, {P#}} is a minimal key family in the relation shown in Table 3.

Our generalisation of entity integrity is now evident. We define a *primary key family* to be a key family which is designated by the user. Given a primary key family $\mathbf{K}$, a relation r satisfies *generalised entity integrity* if $\mathbf{K}$ is a minimal key family in r.

Suppose that a database schema contains the relation schemas R and S, and that d is a database over this database schema containing two relations r over R and s over S. A *foreign key family* $\mathbf{F}$ for R is a family of pairs, such that for each pair $(F_i, K_i) \in \mathbf{F}$, F_i is a subset of R, K_i is a subset of S, and F_i and K_i have the same cardinalities; the set of all K_i such that $(F_i, K_i) \in \mathbf{F}$, say $\mathbf{K}$, is called the key family for S referenced by $\mathbf{F}$. The foreign key family $\mathbf{F}$ is satisfied in the database d, if $\mathbf{K}$ is a key family in s, for all tuples $t \in r$ there is some $(F_i, K_i) \in \mathbf{F}$ such that the F_i values of t do not have an occurrence of *dne*, and for all tuples $t \in r$, for all $(F_i, K_i) \in \mathbf{F}$, if the F_i values of t do not have an occurrence of *dne*, then there is some tuple $u \in s$ such that the F_i values of t reference the K_i values of s. As an example, the foreign key family {(MGR_SS#, SS#), (MGR_P#, P#)} is satisfied in $d = \{r, s\}$, where r is the relation shown in Table 4 and s is the relation shown in Table 3. We note the fact that the MGR_P# value of the second tuple of r is *unk* does not cause a violation of the foreign key family.

Our generalisation of referential integrity is now evident. Given a foreign key family $\mathbf{F}$ for R referencing a primary key family $\mathbf{K}$ of S, a database d containing relations over R and S satisfies *generalised referential integrity* if $\mathbf{F}$ is a foreign key family in d.

The layout of the rest of the paper is as follows. In Section 2 we formalise the notion of incomplete relations. In Section 3 we formalise the fundamental notions of distinguishability and identifiability, which are the necessary ingredients of our generalisation of the concept of a key. In Section 4 we formalise the concepts of key family and minimal key family in a relation and as a result we suggest a generalised definition of entity integrity. In Section 5 we study the relationship between the concepts of key family and Thalheim's key set. In Section 6 we investigate some combinatorial problems relating to key families and minimal key families. In Section 7 we generalise the notion of foreign key to that of a foreign key family and as a result we suggest a generalised definition of referential integrity. Finally, in Section 8 we give our concluding remarks.

2 Relations that model incomplete information

Herein we formalise the notion of an incomplete relation, which allows us to model incomplete information of the form "value at present exists but is unknown" and of the form "value does not exist", or equivalently, "value is inapplicable".

We use the notation |S| to denote the cardinality of a set S. A *family* of sets is a collection of sets indexed by the set of natural numbers $I = \{1, 2, \ldots, n\}$. If S is a subset of T we write $S \subseteq T$ and if S is a proper subset of T we write $S \subset T$. We often denote the singleton {A} simply by A, when no ambiguity arises, and the union of two sets S, T, i.e. $S \cup T$, simply by ST. The nonempty power set of a set S is denoted by $\mathcal{P}(S)$. Finally, we will refer to the cardinality of some standard encoding [GJ79] of S as the *size* of S.

Definition 2.1 (Relation and database) A *relation schema* R is a finite set of attributes and a *database schema* $\mathbf{R}$ is a finite collection $\{R_1, R_2, \ldots, R_n\}$ of relation schemas.

We assume a countably infinite domain of constants, *Dom*, containing the distinguished constants *unk* and *dne*, denoting the null values unknown and does not exist, respectively.

A tuple over R is a total mapping t from R into *Dom* such that $\forall A_i \in \mathrm{R}$, $t(A_i) \in Dom$. The *projection* of a tuple t over R onto a set of attributes $\mathrm{Y} \subseteq \mathrm{R}$, denoted by $t[\mathrm{Y}]$, is the restriction of t to Y. The projection $t[\mathrm{Y}]$ is said to be *complete*, if $\forall A_i \in \mathrm{Y}$, $t[A_i] \neq unk$, and $t[\mathrm{Y}]$ is said to be *nonnull* if, in addition, $\forall A_i \in \mathrm{Y}$, $t[A_i] \neq dne$. Given a tuple t, we let $t\downarrow$ denote the maximal subset Y of R such that $t[\mathrm{Y}]$ is nonnull.

An *incomplete relation* (or simply a relation) r over R is a finite set of tuples over R. A relation whose tuples are all complete is called a *complete relation*. An *incomplete database* (or simply a database) d over $\mathbf{R}$ is a finite collection $\{r_1, r_2, \ldots, r_n\}$ of relations such that for each $i \in I$, $r_i \in d$ is a relation over $R_i \in \mathbf{R}$. A *complete database* is a database whose relations are all complete.

Our justification for allowing occurrences of *dne* in a complete relation is that, although *dne* is a distinguished value, it does not arise due to incompleteness of information. For example, given a relation schema PERSON having an attribute SPOUSE and a relation r over PERSON, the fact that for a tuple $t \in r$ we have $t[\mathrm{SPOUSE}] = dne$ does not signify any incompleteness of information.

From now on we let R *be a relation schema and* r *be a relation over* R.

Definition 2.2 (Less informative constants and tuples) We define a partial order in *Dom*, denoted by $\sqsubseteq$, as follows:

$$u \sqsubseteq v \text{ if and only if } u = v \text{ or } (u = unk \text{ and } v \neq dne), \text{ where } u, v \in Dom.$$

We extend $\sqsubseteq$ to be a partial order in the set of tuples over R as follows: where t_1 and t_2 are tuples over R, t_1 is *less informative than* t_2, written $t_1 \sqsubseteq t_2$, if $\forall A_i \in \mathrm{R}$, $t_1[A_i] \sqsubseteq t_2[A_i]$.

Informally, $\mathrm{POSS}(r)$ is the set of all complete relations, which are more informative than r, that is, they do not contain nulls of type *unk*.

Definition 2.3 (The set of possible worlds of a relation) The set of all *possible worlds* relative to a relation r (or the set of all complete relations emanating from r), denoted by $\mathrm{POSS}(r)$, is defined by

$$\begin{aligned} \mathrm{POSS}(r) \quad = \quad & \{s \mid s \text{ is a relation over R and there exists a total and onto mapping} \\ & f : r \to s \text{ such that } \forall t \in r, t \sqsubseteq f(t) \text{ and } f(t) \text{ is complete}\}. \end{aligned}$$

3 Distinguishability and identifiability

The notions of distinguishability and identifiability are fundamental to our generalisations of the notions of key and foreign key. Distinguishability of a tuple by a set of attributes X implies that the tuple is nonnull on X and that we can distinguish it from other tuples, regardless of whether they have null values or not. Identifiability of a relation on a set of attributes X implies that in all possible worlds s of r, each tuple of r is uniquely identifiable by the X values of some tuple in s. The formal definitions are now given.

Definition 3.1 (Distinguishable tuple) A tuple $t \in r$ is *distinguishable* by a set of attributes X, with X $\subseteq$ R, if

1) t[X] is nonnull, and

2) there does not exist a tuple $t' \in r - \{t\}$, with t[X] = t'[X].

Definition 3.2 (Identifiable relation) A relation r over R is called an *identifiable* relation on a set of attributes X $\subseteq$ R, if for all complete relations $s \in$ POSS(r), the cardinality of the projection of s onto X is equal to the cardinality of r, i.e. $|\pi_X(s)|=|r|$.

The next lemma characterises identifiable relations in terms of inequalities between pairs of tuples in these relations.

Lemma 3.1 A relation r over R is identifiable on X $\subseteq$ R if and only if for all pairs of distinct tuples $t_1, t_2 \in r$, there exists an attribute $A_i \in$ X such that $t_1[A_i]$ and $t_2[A_i]$ are complete and $t_1[A_i] \neq t_2[A_i]$. □

An alternative characterisation of identifiable relations is given in the next corollary.

Corollary 3.2 A relation r over R is identifiable on X $\subseteq$ R if and only if for all pairs of distinct tuples $t_1, t_2 \in r$, t_1[X] $\not\sqsubseteq$ t_2[X]. □

4 Key families and entity integrity in incomplete relations

Herein we define the notions of key family and minimal key family being satisfied in a relation. We show that the notion of a minimal key family in a relation is *faithful* to the standard notion of minimal key in a relation, and, in the absence of *dne* in relations, is a *precise* generalisation of the notion of a key in a complete relation. (See [Mai83] for the concepts of faithfulness and preciseness.) Finally, we define generalised entity integrity using the concept of a primary key family.

Definition 4.1 (Key family) A *key family* **K** for R (or simply a key family, if R is understood from context) is a family $\mathbf{K} = \{K_1, K_2, \ldots, K_n\}$ having $n \geq 1$ subsets of R.

Given a key family **K**, a relation must be identifiable on the union of the elements $K_i \in \mathbf{K}$. Due to the possible presence of *dne* in relations, we need the further condition that each tuple in a relation be distinguishable by at least one element $K_i \in \mathbf{K}$.

Definition 4.2 (Satisfaction of a key family) A relation r over R satisfies a key family **K** (or alternatively, **K** is a key family in r), written $r \models\!\!\approx \mathbf{K}$, if the following two conditions are satisfied:

1) r is identifiable on $\bigcup_{i \in I} K_i$, and

2) every tuple in r is distinguishable by some set of attributes $K_i \in \mathbf{K}$.

The reader can verify that for the boundary cases of a relation, say r, containing no tuples or a single tuple, the key family consisting of the empty set, i.e. $\{\emptyset\}$, is always satisfied in r. To avoid these special cases we assume for the rest of the paper that *relations contain at least two tuples*. When r contains two or more tuples, then $\{\emptyset\}$ is not satisfied in r.

We further observe that when at least some key family $\mathbf{K}$ is satisfied in r, where $r = \{t_1, t_2, \ldots, t_m\}$, then $\{t_1\downarrow, t_2\downarrow, \ldots, t_m\downarrow\}$ is also a key family in r. This key family is the coarsest possible key family in r, which motivates the next definition.

Definition 4.3 (Minimal key family) A key family $\mathbf{K}$ is said to be *minimal* in r (or alternatively, $\mathbf{K}$ is a *minimal key family* in r), if

1) $r \not\approx \mathbf{K}$,

2) for no proper subset $\mathbf{K}' \subset \mathbf{K}$, does r satisfy $\mathbf{K}'$, i.e. the cardinality of $\mathbf{K}$ is minimal, and

3) for no proper subset $K'_i \subset K_i$, with $K_i \in \mathbf{K}$, does r satisfy $(\mathbf{K} - \{K_i\}) \cup \{K'_i\}$, i.e. the cardinalities of all the elements in $\mathbf{K}$ are minimal.

The proof of the following proposition, which provides justification for key families, is straightforward.

Proposition 4.1 The following two statements are true:

1) The notion of a minimal key family is *faithful* to the notion of a minimal key in a relation r, in the sense that if $\mathbf{K} = \{K\}$ is a minimal key family in r, then K is a minimal key in r.

2) In the absence of *dne* in relations, the notion of a key family is a *precise* generalisation of the notion of a key in a complete relation, in the sense that if $\mathbf{K}$ is a key family in r, then for all complete relations $s \in \mathrm{POSS}(r)$, $\bigcup_{i\in I} K_i$ is a key in s. $\square$

We observe that we cannot strengthen part (2) of Proposition 4.1 to say that, in the absence of *dne* in relations, the notion of a minimal key family is a precise generalisation of the notion of a minimal key. As a counterexample, {{NAME}, {ADDRESS}} is a minimal key family in the relation shown in Table 1. However, if we replace the occurrence of *unk* in the third tuple by Asmuns Hill, then {NAME, ADDRESS} is a key but not a minimal key in the resulting complete relation. Moreover, as the relation shown in Table 3 demonstrates, in the presence of *dne*, a key may be undefined according to the standard definition of a key, assuming that the values of at least one of the candidate keys cannot be null. We are now ready to generalise entity integrity.

Definition 4.4 (Generalised entity integrity) A *primary key family* is a key family, which is designated by the user. Given a primary key family $\mathbf{K}$, a relation r satisfies *generalised entity integrity*, if $\mathbf{K}$ is a minimal key family in r.

We observe that the notion of generalised entity integrity reduces to the standard notion of entity integrity when $\mathbf{K}$ is a singleton.

5 A comparison to Thalheim's notion of a key set

We now compare our proposal of key family and minimal key family to Thalheim's proposal [Tha89] of key set and minimal key set. Since Thalheim's approach was defined only for the case when incomplete relations are allowed to have null values of type *unk*, we will assume in this section that incomplete relations do not have nulls of type *dne*.

Definition 5.1 (Thalheim's key set) A *key set* $\mathbf{K}$ for R (or simply a key set if R is understood from context) is a family of singleton attributes from R. A relation r over R satisfies a *key set* $\mathbf{K} = \{\{A_1\}, \{A_2\}, \ldots, \{A_n\}\}$, if r is identifiable on X, where $\mathrm{X} = \bigcup_{i \in I} \{A_i\}$.

A key set $\mathbf{K}$ is *minimal* in r if for no proper subset $\mathbf{K}' \subset \mathbf{K}$ does r satisfy the key set $\mathbf{K}'$.

The next proposition shows that key sets and key families are defined on exactly the same class of relations.

Proposition 5.1 Given a relation r over R, some key set $\mathbf{K}$ is satisfied in r if and only if some key family $\mathbf{K}'$ is satisfied in r, where $\bigcup_{K \in \mathbf{K}} K = \bigcup_{K' \in \mathbf{K}'} K'$.

Proof. If. Suppose that some key family $\mathbf{K}'$ is satisfied in r. Let $\mathrm{X} = \bigcup_{K' \in \mathbf{K}'} K'$, where $\mathrm{X} = \{A_1, A_2, \ldots, A_q\}$. It follows that the key set $\{\{A_1\}, \{A_2\}, \ldots, \{A_q\}\}$ is also satisfied in r.

Only if. Suppose that some key set $\mathbf{K}$ is satisfied in r. Let $\mathrm{X} = \bigcup_{K \in \mathbf{K}} K$, where $\mathrm{X} = \{A_1, A_2, \ldots, A_q\}$, and $s = \pi_{\mathrm{X}}(r)$, where $s = \{t_1, t_2, \ldots, t_m\}$. The result follows, since the key family $\{t_1{\downarrow}, t_2{\downarrow}, \ldots, t_m{\downarrow}\}$ is also satisfied in r. □

When we fix $\mathbf{K}$, then the concepts of key set and key family are incomparable. Firstly, {{NAME}, {ADDRESS}} is a key set in r but not a key family in r, where r is the relation shown in Table 1, together with the occurrence of *unk* in the third tuple replaced by Harold Rd. Secondly, {{NAME, ADDRESS}, {NAME, DOB}, {ADDRESS, DOB}} is a key family in r but, since the elements of the key family are not singletons, not a key set in r, where r is the relation shown in Table 2.

We observe that when we allow null values of type *dne* in relations, then distinguishability, as opposed to identifiability, is also important. For example, if we treat *dne* as just another nonnull value, then {{SS#}} and {{P#}} will be minimal key sets for the relation, say r, shown in Table 3, which is contrary to Codd's assertion that tuples should be distinguishable by their nonnull values. On the other hand, in this case there is one minimal key family in r, namely {{SS#}, {P#}}.

6 Combinatorial problems relating to key families

Herein we investigate some combinatorial properties of key families and show that the problem of finding a minimal key family in a relation can be solved in polynomial time. There remain the problems of deciding whether there exists a minimal key family in a relation which either has an element of cardinality no greater than some natural number k or is itself of cardinality no greater than k; these problems are both NP-complete. We also show that a relation can be constructed having $\mathcal{P}(\mathrm{R})$ as its minimal key family.

Proposition 6.1 Given a relation r over R the problem of finding whether there exists a key family $\mathbf{K}$ such $r \not\approx \mathbf{K}$ can be solved in polynomial time in the sizes of r and R.

Proof. By Proposition 5.1 to test for identifiability it is sufficient to test whether $\{\{A_1\}, \{A_2\}, \ldots, \{A_n\}\}$ is a key set in r, where $\mathrm{R} = \{A_1, A_2, \ldots, A_n\}$, treating *dne* as a nonnull value. If this test is positive, then we can test distinguishability by testing whether each tuple $t \in r$ is distinguishable by some $\mathrm{X} \in \{t_1\downarrow, t_2\downarrow, \ldots, t_m\downarrow\}$, where $r = \{t_1, t_2, \ldots, t_m\}$. By Lemma 3.1 and Definition 3.1 both these tests can be computed in polynomial time in the sizes of r and R, which concludes the result. □

Proposition 6.2 Given a relation r over R that satisfies at least one key family for R, the problem of finding a key family $\mathbf{K}$, such that $\mathbf{K}$ is a minimal key family in r, can be solved in polynomial time in the sizes of r and R.

Proof. By Proposition 5.1 $\mathbf{K} = \{K_1, K_2, \ldots, K_n\}$ is a key family in r, where $r = \{t_1, t_2, \ldots, t_n\}$ and for all $i \in I$, $K_i = t_i\downarrow$. We note that, by Lemma 3.1 and Definition 3.1, $r \not\approx \mathbf{K}$ can be checked in polynomial time in the sizes of r and R.

We next minimise the cardinality of each element in $\mathbf{K}$ as follows. For each $K_i \in \mathbf{K}$, remove any attribute $A_j \in K_i$, if $r \not\approx (\mathbf{K} - \{K_i\}) \cup \{K_i - \{A_j\}\}$; such an attribute A_j is called a *redundant* attribute in $K_i \in \mathbf{K}$ with respect to r. We repeat the process of eliminating redundant attributes A_j in $K_i \in \mathbf{K}$ with respect to r until no further changes to $\mathbf{K}$ are possible. Since the cardinality of $\mathbf{K}$ is at most $|r|$ and the cardinality of each $K_i \in \mathbf{K}$ is at most $|\mathrm{R}|$, this step takes polynomial time in the sizes of r and R.

In the final step we check if $\mathbf{K}$ is of minimal cardinality by removing from $\mathbf{K}$ any element K_i if $r \not\approx \mathbf{K} - \{K_i\}$; such a set K_i of attributes is called *redundant* in $\mathbf{K}$ with respect to r. We repeat this process of eliminating from $\mathbf{K}$ redundant elements in $\mathbf{K}$ with respect to r until no further changes to $\mathbf{K}$ are possible. Since the cardinality of $\mathbf{K}$ is at most $|r|$, this step also takes polynomial time in the sizes of r and R.

This concludes the proof, since $\mathbf{K}$ now satisfies the three conditions of Definition 4.3. □

Proposition 6.3 The problem of deciding whether a relation r over R satisfies a key family having an element of cardinality less than or equal to some natural number k is NP-complete.

Proof. To show that the problem is in NP we simply guess a key family $\mathbf{K}$ for R and then check, in polynomial time in the sizes of r and R, whether the cardinality of some $K \in \mathbf{K}$ is less than or equal to k and whether $r \not\approx \mathbf{K}$.

To show that the problem is NP-hard, a polynomial-time transformation from the problem of whether a relation r, whose tuples are all nonnull, has a key of cardinality k or less, which is known to be NP-complete [DT88] (cf. [LO78]), is trivial. In particular, if r is a relation over R, whose tuples are all nonnull, then, trivially, $r \not\approx \{\mathrm{R}\}$, and thus the result follows. □

Proposition 6.4 The problem of deciding whether a relation r over R satisfies a key family of cardinality less than or equal to some natural number k is NP-complete.

Proof. To show that the problem is in NP we simply guess a key family $\mathbf{K}$ for R and then check in polynomial time in the sizes of r and R, whether $\mathbf{K}$ has k or less elements and whether $r \not\approx \mathbf{K}$.

To show that the problem is NP-hard a polynomial-time transformation from the vertex cover problem of cardinality k or less, which is known to be NP-complete [GJ79], is given.

Given a graph (N, E) we construct a relation schema R having the same cardinality, say n, as N such that each node in N corresponds to an attribute in R; for the rest of the proof we do not distinguish between the nodes in N and their corresponding representative attributes in R. In addition, we construct a complete relation r over R having the same cardinality as E such that for each $e_i = \{A_i^1, A_i^2\} \in$ E we add the following tuple t_i to r: $t_i[A_i^1] = t_i[A_i^2] = i$ and $t_i[\text{R} - \{A_i^1, A_i^2\}] = < dne, \ldots, dne >$. It remains to show that r satisfies a key family of cardinality k or less if and only if (N, E) has a vertex cover of cardinality k or less.

We observe that by the construction of r the key family $\{\{A_1\}, \ldots, \{A_n\}\}$ is satisfied in r, since each tuple $t_i \in r$ is distinguishable by both A_i^1 and A_i^2, where the attribute set $\{A_i^1, A_i^2\} \subseteq$ R corresponds to the edge $e_i \in$ E; moreover, r is identifiable on $\bigcup_{i \in I}\{A_i\}$. (We assume without loss of generality that (N, E) is connected and has no loops.) It remains to show that (N, E) has a vertex cover whose cardinality is less than or equal to k if and only if r satisfies a key family whose cardinality is less than or equal to k.

If (N, E) has a vertex cover, say $\mathbf{V}$, whose cardinality is not greater than k, then, by the construction of r, $\mathbf{V}$ is a key family that is satisfied in r. Conversely, if r satisfies a key family $\mathbf{K}$, whose cardinality is not greater than k, then by the above observation we can assume without loss of generality that the elements of $\mathbf{K}$ are singletons and thus, by the construction of r, $\mathbf{K}$ is a vertex cover of (N, E). □

The next proposition shows that we can construct a relation over R such that it has a key family of cardinality $2^n - 1$, where $|\text{R}| = n$, which is a minimal key family in this relation.

Proposition 6.5 There is a relation r over R such that $\mathcal{P}$(R) is a minimal key family in r.

Proof. For each X $\in \mathcal{P}$(R) we construct a complete relation r_X as follows, with the proviso that the nonnull values in r_X are distinct from the nonnull values in any other relation r_Y, where Y $\in \mathcal{P}$(R)−{X}. If X is a singleton, say {A}, then $r_A = \{t\}$, where t[A] is nonnull and t[R−{A}] = $< dne, \ldots, dne >$. Otherwise, if X = $\{A_1, \ldots, A_n\}$, with $n > 1$, then r_X contains $n + 1$ tuples $t_0, t_1, \ldots, t_n$ such that t_0[X] = $< 0, \ldots, 0 >$ and t_0[R−X] = $< dne, \ldots, dne >$, and for all $i \in I$, $t_i[A_i] = < 1 >$, t_i[X−{A_i}] = $< 0, \ldots, 0 >$ and t_i[R−X] = $< dne, \ldots, dne >$.

It can now be verified that $r = \bigcup_{X \in \mathcal{P}(\text{R})} r_X$ is the desired relation. □

7 Foreign key families and generalised referential integrity

In the context of incomplete relations the fundamental constraint of referential integrity [Cod79, Dat86, HR96] also needs to be re-evaluated. Herein we define the notion of foreign key family being satisfied in a database. We show that the notion of a foreign key family in a database is *faithful* to the standard notion of foreign key in a database, and, in the absence of *dne* in relations, is a *precise* generalisation of the notion of a foreign key in a complete database. Finally, we define generalised referential integrity using the concepts of foreign key family and primary key family.

Definition 7.1 (Foreign key family) Let R and S be relation schemas in a database schema $\mathbf{R}$. Then a *foreign key family* $\mathbf{F}$ for R referencing a key family $\mathbf{K}$ for S (or simply a foreign key family $\mathbf{F}$, if R and $\mathbf{K}$ are understood from context) is a family of pairs, $\mathbf{F}$ =

$\{(F_1, K_1), (F_2, K_2), \ldots, (F_n, K_n)\}$, where for all $i \in I$, $F_i \subseteq$ R, $K_i \subseteq$ S, and $|F_i|{=}|K_i|$; $\mathbf{K} = \{K_1, K_2, \ldots, K_n\}$ is called the key family for S referenced by $\mathbf{F}$.

Definition 7.2 (Satisfaction of a foreign key family) A database d over $\mathbf{R}$ containing relations r over R and s over S, where R, S $\in \mathbf{R}$, satisfies a foreign key family $\mathbf{F}$ (or alternatively, $\mathbf{F}$ is a foreign key family in d), written $d \models\!\!\approx \mathbf{F}$, if the following three conditions are satisfied:

1) $s \models\!\!\approx \mathbf{K}$, where $\mathbf{K}$ is the key family for S referenced by $\mathbf{F}$,

2) for all $t \in r$, there is some $(F_i, K_i) \in \mathbf{F}$ such that for all A $\in F_i$, t[A] $\neq$ *dne*, and

3) for all $t \in r$, for all $(F_i, K_i) \in \mathbf{F}$, if for all A $\in F_i$, t[A] $\neq$ *dne*, then there is some $u \in s$ such that $t[F_i] \sqsubseteq u[K_i]$.

The first condition in Definition 7.2 guarantees that $\mathbf{K}$ is a key family in s. The second condition guarantees that the foreign key family is well defined for at least one pair $(F_i, K_i) \in \mathbf{F}$. Finally, the third condition guarantees that whenever the projection of a tuple $t \in r$ onto F_i does not have occurrences of *dne*, then it is less informative than the projection onto K_i of some tuple $u \in s$. This is the standard requirement of a foreign key constraint, namely that only entities which exist are referenced.

We note that given a database d and a foreign key family $\mathbf{F}$, it can easily be tested in polynomial time in the sizes of d and $\mathbf{R}$ whether $\mathbf{F}$ is a foreign key family in d over $\mathbf{R}$ or not.

The proof of the following proposition, which provides justification for foreign key families, is straightforward. It is assumed that R, S $\in \mathbf{R}$.

Proposition 7.1 The following two statements are true:

1) The notion of a foreign key family is *faithful* to the notion of a foreign key in a database d containing relations r over R and s over S, in the sense that if $\mathbf{F} = \{(F, K)\}$ is a foreign key family in d, then F is a foreign key for R in r referencing the key K for S in s.

2) In the absence of *dne* in relations, the notion of a foreign key family is a *precise* generalisation of the notion of a foreign key in a complete database, in the sense that if $\mathbf{F}$ is a foreign key family in d, containing relations r over R and s over S, then for all complete relations $r' \in \mathrm{POSS}(r)$, there exists a complete relation $s' \in \mathrm{POSS}(s)$ such that for all $t' \in r'$, there exists $u' \in s'$ such that t'[X] $= u'$[Y], where X $= \bigcup_{i \in I} F_i$ and Y $= \bigcup_{i \in I} K_i$. □

As is the case for Proposition 4.1 we cannot strengthen the statements of Proposition 7.1. We are now ready to generalise referential integrity.

Definition 7.3 (Generalised referential integrity) Let $\mathbf{F}$ be a foreign key family for R referencing a primary key family $\mathbf{K}$ of S. A database d over $\mathbf{R}$ containing relations r over R and s over S, where R, S $\in \mathbf{R}$, satisfies *generalised referential integrity*, if $\mathbf{F}$ is a foreign key family in d.

We observe that the notion of generalised referential integrity reduces to the standard notion of referential integrity when $\mathbf{F}$ is a singleton.

8 Concluding remarks

Entity and referential integrity in their classical form were shown to be overly restrictive for practical applications and as a result we proposed a generalisation of them. Our proposal caters for the two types of null value, *unk* and *dne*, which should be supported by any modern relational DBMS [Cod90]. We have generalised entity and referential integrity via the notions of a key family being satisfied in an incomplete relation and a foreign key family being satisfied in an incomplete database. The intuition behind this generalisation is based on the concepts of distinguishability and identifiability. In order to uniquely identify a tuple in a relation we need to be able to distinguish this tuple from all the other tuples in the relation, and, in addition, its identification must be established in all possible worlds. The classical notions of entity and referential integrity are special cases of our generalisation, so upgrading a DBMS to provide this facility would be an upwards compatible feature.

The support of generalised entity and referential integrity will ease the task of the database practitioner in situations such as those demonstrated via the examples given in the introduction. In the absence of such support database designers will usually apply the following solution, involving the addition of a *surrogate* [Dat92] as an additional attribute to the relation schema. A *surrogate* is an attribute with *no* intrinsic meaning whose values are nonnull and unique for each tuple in a relation. The values of a surrogate are generated either by the database system (if it supports such a mechanism) or by an application program, and should be concealed from the user, since they are not real-world identifiers and thus have no meaning to the user. Although such a solution is viable, there are various overheads in maintaining surrogates (see [Dat92] for a discussion on the adverse effects of surrogates on database design, and on queries and updates to the database). We stipulate that entity and referential integrity should be generalised even in the presence of surrogates. Consider, for example, the relation, say r over R, shown in Table 3. If we add a surrogate attribute to R, then it would be possible to have in r a tuple of the form <sno, *unk*, *dne*, *unk*>, where sno is a surrogate value. In such a case when a tuple is null on both SS# and P#, users have no *meaningful* way to uniquely identify the tuple. Thus surrogates do not solve the problem that users may have of being able to identify in incomplete relations one tuple from another by some set of meaningful values.

References

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, Reading, Ma., 1995.

[ANS75] ANSI/X3/SPARC. Study group on database management systems, interim report. *Bulletin of ACM SIGFIDET*, 7(2), 1975.

[Cod79] E.F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4:397–434, 1979.

[Cod90] E.F. Codd. *The Relational Model for Database Management: Version 2.* Addison-Wesley, Reading, Ma., 1990.

[Dat86] C.J. Date. Referential integrity. In *Relational Database: Selected Writings*, pages 41–63. Addison-Wesley, Reading, Ma., 1986.

[Dat92] C.J. Date. Composite keys. In *Relational Database Writings 1989-1991*, pages 467–474. Addison-Wesley, Reading, Ma., 1992.

[DT88] J. Demetrovics and V.D. Thi. Relations and minimal keys. *Acta Cybernetica*, 8:279–285, 1988.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, New York, 1979.

[HR96] T. Härder and J. Reinert. Access path support for referential integrity in SQL2. *The VLDB Journal*, 5:196–214, 1996.

[LL86] N. Lerat and W. Lipski Jr. Nonapplicable nulls. *Theoretical Computer Science*, 46:67–82, 1986.

[LL97] M. Levene and G. Loizou. Null inclusion dependencies in relational databases. *Information and Computation*, 136:67–108, 1997.

[LL99] M. Levene and G. Loizou. *A Guided Tour of Relational Databases and Beyond*. Springer-Verlag, London, 1999.

[LO78] C.L. Lucchesi and S.L. Osborn. Candidate keys for relations. *Journal of Computer and System Sciences*, 17:270–279, 1978.

[Mai83] D. Maier. *The Theory of Relational Databases*. Computer Science Press, Rockville, Md., 1983.

[Tha89] B. Thalheim. On semantic issues connected with keys in relational databases permitting null values. *Journal of Information Processing Cybernetics*, 25:11–20, 1989.