SAFE REFERENTIAL INTEGRITY STRUCTURES IN RELATIONAL DATABASES *

Victor M. Markowitz

Information and Computing Sciences Division
Lawrence Berkeley Laboratory
1 Cyclotron Road, Berkeley, CA 94720

Abstract

Referential integrity constraints express in relational databases existence dependencies between tuples. Although it is known that certain referential integrity structures may cause data manipulation problems, the nature of these problems has not been explored and the conditions for avoiding them have not been formally developed. In this paper we examine these data manipulation problems and formally develop *safeness* conditions for avoiding them. Next, we discuss the problem of specifying safe referential integrity constraints in three representative relational database management systems, IBM's DB2, SYBASE, and INGRES.

Key Words : null constraint, referential integrity constraint, relational database management system, safe referential integrity structure.

1. Introduction

Commercial relational database management systems (RDBMS) provide mechanisms for maintaining key and restricted (*nulls not allowed*) null constraints. Several commercial RDBMSs, notably IBM's DB2, SYBASE 4.0, and INGRES 6.3 also provide mechanisms for maintaining referential integrity constraints. Referential integrity constraints are used in relational databases for expressing existence dependencies between tuples [1]: such constraints are specified by associating a *foreign-key* in one relation with the *primary-key* of another relation [2]. Referential integrity constraints are usually associated with rules that define the behavior of the relations involved in these constraints under insertion, deletion, and update of tuples.

The concept of referential integrity is still surrounded by confusion, as illustrated by the successive modifications of the original definition of [1] (see [2], [3]). Thus, although it is known that certain referential integrity structures may cause data manipulation problems (e.g. see [3]), the nature of these problems has not been explored and the conditions for avoiding them have not been formally developed. In particular, problems created by the interaction of referential integrity and null constraints have not been investigated. In this paper we examine these data manipulation problems and develop *safeness* conditions for avoiding them. It is worth noting that the approach of this paper is different from that of [7], where it is shown that relational schema translations of object-oriented schemas have referential integrity structures with desirable properties.

The referential integrity mechanisms provided by various RDBMSs are different and difficult to use. Thus, SYBASE 4.0 [10] and INGRES 6.3 [5] provide procedural mechanisms (*triggers* in SYBASE and *rules* in INGRES) for maintaining referential integrity constraints. Conversely, DB2 [4] allows non-procedural (declarative) specifications of referential integrity constraints, but imposes restrictions on the structure of these constraints. Problems underlying the use of the referential integrity mechanisms of DB2, SYBASE, and INGRES have been examined in [9]. In this paper we examine these mechanisms in the context of safe referential integrity structures.

In SYBASE 4.0 and INGRES 6.3 the task of specifying correctly referential integrity constraints is left to users. Thus, no mechanism is provided by these systems for detecting unsafe or even not well-defined referential integrity constraints. DB2 has been unique among RDBMSs in addressing the data manipulation problems caused by certain referential integrity structures. DB2 attempts to avoid these problems by imposing restrictions on the structure of referential integrity constraints it allows. We compare the DB2 restrictions with the safeness conditions and show that while some DB2 restrictions are implied by the safeness conditions, other

* Also issued as technical report LBL-28363. This work was supported by the Office of Health and Environmental Research Program and the Applied Mathematical Sciences Research Program, of the Office of Energy Research, U.S. Department of Energy, under Contract DE-AC03-76SF00098.

restrictions are too stringent or misplaced. Moreover, DB2 allows the specification of certain unsafe referential integrity structures.

The rest of the paper is organized as follows. The relational concepts used in this paper are reviewed in section 2. In section 3 we discuss briefly the SYBASE, INGRES, and DB2 mechanisms for maintaining referential integrity and null constraints. In section 4 we examine the data manipulation problems caused by certain referential integrity and null constraint structures, and develop the safeness conditions required for avoiding these problems. The specification of safe referential integrity structures in SYBASE, INGRES, and DB2 is examined in section 5. The paper concludes with a summary.

2. Preliminary Definitions

We use in this paper some graph-theoretical concepts. We denote by $G = (V, H)$ a directed graph with set of vertices V and set of edges H, and by $v_i \rightarrow v_j$ a directed edge, h, *incident* from vertex v_i to vertex v_j. A directed *path* from (*start*) vertex v_{i_0} to (*end*) vertex v_{i_m} is a sequence of alternating vertices and edges, $v_{i_0}\, h_{j_1}\, v_{i_1} \ldots h_{j_m}\, v_{i_m}$, such that h_{j_k} is incident from $v_{i_{k-1}}$ to v_{i_k}, $1 \le k \le m$. A directed *cycle* is a directed path whose start vertex is also its end vertex.

We review briefly below the relational concepts used in this paper. Details can be found in any textbook (e.g. [6]) for the basic concepts, and in [2] for referential integrity constraints. We denote by t a tuple and by $t[W]$ the subtuple of t corresponding to the attributes of W. A tuple is said to be *total* if it has only non-null values.

A *relational schema* RS is a pair (R, Δ), where R is a set of relation-schemes and Δ is a set of constraints over R. We consider relational schemas with $\Delta = F \cup I \cup N$, where F, I, and N denote sets of key, referential integrity, and null constraints, respectively. A *relation-scheme* is a named set of attributes, $R_i(X_i)$, where R_i is the relation-scheme name and X_i denotes the set of attributes. Every attribute is assigned a *domain*, and every relation-scheme, $R_i(X_i)$, is assigned a *relation* (value), r_i. A *database state* associated with (R, Δ) is defined as $r = < r_1 \ldots r_k >$, where r_i is equal to a subset of the cross-product of the domains corresponding to attributes of $R_i(X_i)$. A database state r associated with a relational schema $RS = (R, \Delta)$ is said to be *consistent* if it satisfies the constraints of Δ. Two attributes are said to be *compatible* if they are associated with the same domain, and attribute sets X and Y are said to be *compatible* iff there exists a one-to-one correspondence of compatible attributes between X and Y.

Let $R_i(X_i)$ be a relation-scheme associated with relation r_i. The *total projection* of r_i on a subset W of X_i is denoted $\pi\downarrow_W(r_i)$, and is equal to $\{t[W] \mid t \in r_i$ and $t[W]$ is total$\}$.

Let $R_i(X_i)$ be a relation-scheme associated with relation r_i. A *key constraint* over R_i is a statement of the form $R_i: K_i \rightarrow X_i$, where K_i is a subset of X_i, called *key*; $R_i: K_i \rightarrow X_i$ is *satisfied* by r_i iff for any two tuples of r_i, t and t', $t[K_i] = t'[K_i]$ implies $t = t'$, and there does not exist any proper subset of K_i having this property. A relation-scheme can be associated with several *candidate keys* from which one *primary-key* is chosen.

Let $R_i(X_i)$ and $R_j(X_j)$ be two relation-schemes associated with relations r_i and r_j, respectively. A *referential integrity constraint* is a statement of the form $R_i[Y] \subseteq R_j[K_j]$, where Y and K_j are compatible subsets of X_i and X_j, respectively, K_j is the primary-key of R_j, and Y is called a *foreign-key* of R_i; $R_i[Y] \subseteq R_j[K_j]$ is *satisfied* by r_i and r_j iff $\pi\downarrow_Y (r_i) \subseteq \pi\downarrow_{K_j} (r_j)$.

A referential integrity constraint $R_i[Y] \subseteq R_j[K_j]$ is associated with an *insert–rule*, a *delete–rule* and an *update–rule* [2]. There is a unique insert-rule, *restricted*, which asserts that inserting a tuple t into r_i can be performed only if the tuple of r_j referenced by t already exists. The delete and update rules define the effect of deleting (resp. updating the primary-key value in) a tuple t' of r_j : a *restricted* delete (resp. update) rule asserts that the deletion of (resp. update of the primary-key value in) t' cannot be performed if there exist tuples in r_i referencing t'; a *cascades* delete (resp. update) rule asserts that the deletion of (resp. update of the primary-key value in) t' implies deleting (resp. updating the subtuple $t[Y]$ in) the tuples of r_i referencing t'; and a *nullifies* delete (resp. update) rule asserts that the deletion of (resp. update of the primary-key value in) t' implies setting to null the subtuple $t[Y]$ in all the tuples t of r_i referencing t'.

Let $RS = (R, \Delta)$ be a relational schema, so that Δ includes referential integrity constraints. The referential integrity (directed) graph associated with RS, $G_I = (V, H)$, is defined as follows: $V = R$, and $H = \{R_i \rightarrow R_j \mid R_i[Y] \subseteq R_j[K_j] \in I\}$. The set of referential integrity constraints of RS is said to be *acyclic* iff G_I does not have directed cycles.

A *null constraint* is a restriction on the way nulls appear in relations [6]. Let $R_i(X_i)$ be a relation-scheme associated with relation r_i. A *null constraint* is a statement of the form $R_i: Y \overset{EX}{\rightarrow} Z$, where Y and Z are subsets of X_i; $R_i: Y \overset{EX}{\rightarrow} Z$ is *satisfied* by r_i iff for every tuple t of r_i, $t[Y]$ is total only if $t[Z]$ is total. All relational database management systems support the specification of *nulls-not-allowed* constraints. A *nulls-not-allowed* constraint

has the form $R_i: \emptyset \overset{EX}{\rightarrow} Z$; $R_i: \emptyset \overset{EX}{\rightarrow} Z$ is satisfied by r_i iff for every tuple t of r_i, the subtuple $t[Z]$ is total.

An example of a relational schema involving key, referential integrity, and null constraints is shown in figure 1(i); the referential integrity graph corresponding to this schema is shown in figure 1(ii), and a database state that satisfies the constraints involved in this schema is shown in figure 1(iii).

3. Referential Integrity and Null Constraints in DB2, SYBASE and INGRES

Relational database management systems (RDBMS) support the specification of relation-schemes, keys, and nulls-not-allowed constraints. In this section we overview briefly the mechanisms provided by three representative commercial RDBMSs, namely DB2, SYBASE 4.0, and INGRES 6.3 for maintaining referential integrity and general null constraints. These mechanisms are examined in more detail in [9]. We illustrate our discussion with the relational schema shown in figure 1(i).

Referential integrity and nulls-not-allowed constraints are specified in DB2 [4] declaratively (i.e. non-procedurally). Referential integrity constraints are associated in DB2 by default with *restricted* update-rules. Referential integrity specifications in DB2 are coupled with the specifications for relation-schemes, primary-keys, and nulls-not-allowed constraints; thus, the DB2 definition for a relation-scheme R_i includes the specification of all the referential integrity constraints that involve R_i in their left-hand sides. For example, the DB2 definition for relation-scheme EMPLOYEE of the schema in figure 1(i) is shown in figure 2.

SYBASE 4.0 [10] and INGRES 6.3 [5] provide procedural mechanisms for specifying referential integrity constraints. Thus, referential integrity constraints are maintained in these systems by executing *referential integrity procedures* whenever tuples are inserted, deleted or updated in a relation. Given a data manipulation δ involving one or several tuples of a relation r_i associated with relation-scheme R_i, a referential integrity procedure associated with r_i must [9]:

(i) revoke δ if the relation that would result by applying δ on r_i, r'_i does not satisfy the referential integrity constraints involving R_i and associated with *restricted* insert, delete, or update rules;

(ii) initiate additional (corrective) data manipulations if r'_i does not satisfy the referential integrity constraints involving R_i and associated with *nullifies* or *cascades* delete or update rules.

In SYBASE the definition of referential integrity procedures involves specifying a special kind of procedures called *triggers* that are activated (*fired*) when a relation is affected by a data manipulation. A trigger is associated with a unique relation-scheme, say R_i, and employs two system provided relations, called *deleted* and *inserted*: if R_i is associated with a relation r_i, then following a data manipulation, relation *deleted* consists of

i. **Relation–Schemes** (*Keys are underlined*)

(R_1) EMPLOYEE (E_SSN, S_SSN, M_SSN, P_NR)
(R_2) MANAGER (M_SSN, P_NR)
(R_3) PROJECT (P_NR)

Null Constraints (*Nulls-Not-Allowed*)

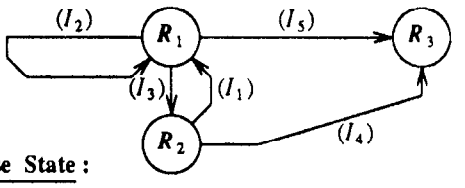
EMPLOYEE: $\emptyset \overset{EX}{\rightarrow}$ E_SSN MANAGER: $\emptyset \overset{EX}{\rightarrow}$ M_SSN
PROJECT: $\emptyset \overset{EX}{\rightarrow}$ P_NR

Referential Integrity Constraints

(I_1) MANAGER [M_SSN] $\subseteq$ EMPLOYEE [E_SSN]
(I_2) EMPLOYEE [S_SSN] $\subseteq$ EMPLOYEE [E_SSN]
(I_3) EMPLOYEE [M_SSN] $\subseteq$ MANAGER [M_SSN]
(I_4) MANAGER [P_NR] $\subseteq$ PROJECT [P_NR]
(I_5) EMPLOYEE [P_NR] $\subseteq$ PROJECT [P_NR]

Rules	insert	delete	update
(I_1, I_3, I_4)	*restricted*	*restricted*	*restricted*
(I_2, I_5)	*restricted*	*nullifies*	*restricted*

ii. **Referential Integrity Graph :**



iii. **Database State :**

(R_1): $r_1 = \begin{bmatrix} 1 & 4 & 4 & a \\ 2 & - & - & b \\ 3 & 2 & - & b \\ 4 & - & - & a \end{bmatrix}$ (R_2): $r_2 = \begin{bmatrix} 2 & - \\ 4 & a \end{bmatrix}$ (R_3): $r_3 = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$

Abbr. : E(MPLOYEE), M(ANAGER), P(ROJECT), S(UPERVISOR)

Figure 1. A Relational Database Example.

```
CREATE TABLE EMPLOYEE (
        PRIMARY KEY (E_SSN),
        E_SSN CHAR(12) NOT NULL, S_SSN CHAR(12),
        M_SSN CHAR(12), P_NR INTEGER,
        FOREIGN KEY (S_SSN) REFERENCES EMPLOYEE
                                ON DELETE SET NULL,
        FOREIGN KEY (M_SSN) REFERENCES MANAGER
                                ON DELETE RESTRICT,
        FOREIGN KEY (P_NR)  REFERENCES PROJECT
                                ON DELETE SET NULL)
```

Figure 2. Example of a DB2 Relation Definition.

the r_i tuples that are going to be deleted or updated, and relation *inserted* consists of tuples that are going to be inserted into r_i, or newly updated tuples of r_i. SYBASE allows the specification of three triggers per relation: an *insert*, a *delete*, and an *update* trigger that are fired when tuples are inserted into, deleted from, or updated in r_i, respectively. Triggers are specified in SYBASE's dialect of SQL, that allows the specification of control-flow statements in addition to standard SQL statements. For example, the *delete* trigger for relation-scheme MANAGER of the relational schema in figure 1(i) is shown in figure 3.

In INGRES the specification of referential integrity procedures is supported by a mechanism similar to the SYBASE trigger mechanism. Instead of triggers INGRES allows associating *rules* with relations. Like a trigger, a rule is activated when the associated relation is affected by a data manipulation, but while triggers can be activated by manipulations involving multiple tuples, rules are activated by single tuple manipulations. Accordingly, instead of the *inserted* and *deleted* relations provided by SYBASE, INGRES provides two tuples, called *new* and *old*: following a data manipulation involving a relation r_i, the *old* tuple contains the r_i tuple that is going to be deleted or updated, and the *new* tuple is the tuple that is going to be inserted into r_i, or the newly updated tuple of r_i. The rule procedures are specified in INGRES's dialect of SQL, that, like SYBASE's SQL, allows the specification of control-flow statements in addition to standard SQL statements. For example, the *delete* rule for relation-scheme MANAGER of the relational schema in figure 1(i) is shown in figure 4.

Regarding null constraints, DB2, SYBASE, and INGRES allow declarative specifications of nulls-not-allowed constraints. General null constraints can be maintained using triggers in SYBASE and rules in INGRES, by embedding the procedural specification for such null constraints with the procedural specification for referential integrity constraints. In DB2 general null constraints can be maintained using special *Validproc* procedures. Every relation in a DB2 database can be associated with a *Validproc* procedure, and these procedures are activated by tuple manipulations in a way similar to the activation of SYBASE triggers and INGRES rules.

```
create trigger deleteMANAGER on MANAGER for delete as
begin
  declare @delEMPLOYEE int
  select @delEMPLOYEE = count(*) from deleted, EMPLOYEE
                  where deleted.M_SSN = EMPLOYEE.M_SSN
  if @delEMPLOYEE > 0
  begin
        raiserror 1 "Failed deletion in MANAGER because of
                        existing reference from EMPLOYEE"
        rollback transaction
  end
end
```

Figure 3. A SYBASE Delete Trigger Example.

```
create procedure
    p_deleteMANAGER (o_P_NR char(20), o_M_SSN int) as
declare msg varchar(80) not null; check_val integer;
begin
   select count(*) into :check_val from EMPLOYEE
                              where M_SSN = :o_M_SSN;
   if check_val > 0 then
        msg = 'Failed deletion in MANAGER because
                of existing reference from EMPLOYEE';
        raise error 1 :msg;
   endif;
end;
create rule r_deleteMANAGER after delete from MANAGER
   execute procedure p_deleteMANAGER (o_P_NR = old.P_NR,
                                   o_M_SSN = old.M_SSN);
```

Figure 4. An INGRES Delete Rule Example.

4. Safe Referential Integrity Structures

Certain referential integrity constraints cause data manipulation problems. In this section we examine these problems and specify conditions for avoiding them; referential integrity structures that satisfy these conditions are said to be *safe*. In the next section we discuss the referential integrity mechanisms of DB2, SYBASE, and INGRES in the context of safe referential integrity structures. We assume below that the referential integrity constraints are specified correctly, that is, are *well-defined*. We illustrate our discussion with the relational schema shown in figure 1(i), and the database state shown in figure 1(iii).

The data manipulations considered in this paper are insertions, deletions or updates of one or several tuples, where a data manipulation δ (i) involves only insertions, only deletions, or only updates, (ii) refers to a unique relation, and (iii) involves a set of tuples that does not change during the execution of δ. We assume that the constraints in a database are verified after every single-tuple data manipulation. A data manipulation δ is considered to *succeed* iff all its single-tuple data manipulations are carried out, otherwise (i.e. if at least one single-tuple manipulation of δ cannot be carried out) it is considered to *fail*. The safeness conditions developed in this section ensure that:

1. For every data manipulation δ, the overall effect of δ does not depend on the order in which the referential integrity constraints are enforced, nor on the order in which the tuples involved in δ are accessed; if such an independence is not ensured then the result of some data manipulations may be unpredictable.
2. For every two consistent database states, r and r', there exists a sequence of data manipulations δ_1 , ... , δ_m that maps r into r', so that every data manipulation in the sequence maps a consistent database state into another consistent database state.

The first safeness condition refers to the relationship between referential integrity constraints and tuple deletions. Let δ denote a deletion (of one or several tuples) in a relation r_i. δ can trigger: (i) the deletion of tuples that reference tuples involved in δ via referential integrity constraints associated with *cascades* delete-rules; or (ii) the update of foreign-key values in tuples that reference tuples involved in δ via referential integrity constraints associated with *nullifies* delete-rules. Conversely, if a tuple t references a tuple involved in δ via a referential integrity constraint associated with a *restricted* delete-rule then t blocks (the execution of) δ. Similarly, a tuple t can affect or be affected by a deletion δ if t references a tuple involved in δ via several (transitive) referential integrity constraints. For example, consider the database state of figure 1(iii) associated with the relational schema of figure 1(i); tuple (1 4 4 a) in relation r_1 blocks (directly) the deletion of tuple (4 a) in relation r_2 (via I_3), and blocks (transitively) the deletion of tuple (4 – – a) in r_1 (via I_3 and I_1).

The outcome of a deletion δ is unpredictable when enforcing referential integrity constraints following δ implies triggering the deletion or update of tuples that, in turn, can block δ.

Example 4.1. Suppose that the relational schema of figure 1(i) includes only three referential integrity constraints, I_1 and I_5 associated with *cascades* delete-rules, and I_4 associated with a *restricted* delete-rule. Let deletion δ involve tuple (a) of relation r_3. Note that tuple (4 a) of relation r_2 can block δ via I_4, while δ can trigger the deletion of this tuple via I_5 and I_1. The outcome of δ depends on the order in which the referential integrity constraints involving R_3 are enforced: (i) if I_5 is enforced first, then tuples (1 4 4 a) and (4 – – a) are deleted from r_1, thus leading to the enforcement of I_1 which results in deleting tuple (4 a) from r_2; or (ii) if I_4 is enforced first, then δ is blocked by tuple (4 a) of r_2.

Example 4.2. Suppose that the relational schema of figure 1(i) includes only referential integrity constraint I_2 associated with a *restricted* delete-rule. Let deletion δ involve tuples (2 – – b) and (3 2 – b) of relation r_1. Note that tuple (3 2 – b) in relation r_1 can block δ via I_2, while δ includes the deletion of this tuple. The outcome of δ depends on the order in which the tuples involved in δ are accessed: (i) if (3 2 – b) is accessed first, then δ can be carried out and both tuples are ultimately deleted; or (ii) if (2 – – b) is accessed first, then δ is blocked by tuple (3 2 – b).

Example 4.3. Suppose that the relational schema of figure 1(i) includes only two referential integrity constraints, I_1 associated with a *cascades* delete-rule and I_3 associated with a *restricted* delete-rule. Let deletion δ involve tuples (1 4 4 a) and (4 – – a) of relation r_1. Note that tuple (1 4 4 a) of r_1 can block δ via I_3 and I_1, while δ includes the deletion of this tuple. The outcome of δ depends on the order in which the tuples involved in δ are accessed: (i) if (1 4 4 a) is accessed first, then δ can be carried out and both tuples are ultimately deleted; or (ii) if (4 – – a) is accessed first, then δ is blocked by tuple (1 4 4 a).

Null constraints may conflict with referential integrity constraints associated with *nullifies* delete-rules. For example, null constraint R_i: $Y \overset{EX}{\rightarrow} Z$ conflicts with referential integrity constraint $R_i[Z] \subseteq R_j[K_j]$ associated with a *nullifies* delete-rule; thus, deleting a tuple in the relation associated with R_j that is referenced by a tuple t of r_i, in which subtuples $t[Y]$ and $t[Z]$ are both total, would imply setting to null subtuple $t[Z]$, while such an update is not allowed by the null constraint. If a tuple t references a tuple involved in a deletion δ via a referential integrity constraint associated with a *nullifies* delete-rule that conflicts with a null constraint involving t, then t blocks (the execution of) δ.

Example 4.4. Suppose that the relational schema of figure 1(i) includes only three referential integrity constraints, I_3 and I_5 associated with *nullifies* delete-rules, and I_4 associated with a *cascades* delete-rule. Suppose also that relation-scheme R_1 is associated with null constraint (N_1) R_1: M_SSN $\overset{EX}{\Rightarrow}$ P_NR. Let deletion δ involve tuple (a) of relation r_3. Note that without N_1 δ would imply nullifying (via I_5) subtuples t[P_NR] in tuples (1 4 4 a) and (4 – – a) of relation r_1, and nullifying (via I_4 and I_3) subtuple t[M_SSN] in tuple (1 4 4 a) of relation r_1. However, when N_1 is considered, the outcome of δ depends on the order in which the referential integrity constraints involving R_3 are enforced: (i) if I_4 is enforced first, then tuple (4 a) is deleted from r_2, thus leading to the enforcement of I_3 which results in nullifying subtuple t[M_SSN] in tuple (1 4 4 a) of r_1; subsequently, enforcing I_5 results in nullifying subtuples t[P_NR] in tuples (1 4 – a) and (4 – – a) of r_1; or (ii) if I_5 is enforced first,

then δ is blocked by tuple (1 4 4 *a*) of r_1, where subtuple t[P_NR] cannot be nullified because of N_1.

Note that because of the unique *restricted* insert-rule, problems such as those discussed above cannot be caused by insertions. Updates, however, can cause similar problems. For the sake of simplicity we assume in this paper that in a relational schema all the referential integrity constraints are associated with identical *restricted* update-rules. The safeness condition specified below ensures that the overall effect of a deletion δ involving one or several tuples of a given relation, is independent of both the order in which the referential integrity constraints are enforced, and of the order in which the tuples involved in δ are accessed.

Definition 4.1 – *Safeness Condition* **S1**.

Let $RS = (R, F \cup I \cup N)$ be a relational schema, where F, I, and N denote sets of key, referential integrity, and null constraints, respectively. Let $G_I = (R, I)$ be the referential integrity graph associated with RS. Given a relation-scheme R_i of R, sets $Casc(R_i)$ and $Null(R_i)$ defined below consist of the relation-schemes whose associated relations may contain tuples that can be deleted, respectively updated, as a result of deleting tuples in a relation associated with R_i; and set $Restr(R_i)$ defined below consists of the relation-schemes whose relations may contain tuples that can block the deletion of tuples in a relation associated with R_i:

- $Casc(R_i)$ is the subset of R consisting of R_i and the relation-schemes that are connected in G_I to R_i by a directed path consisting of edges that correspond to referential integrity constraints associated with *cascades* delete-rules;
- $Null(R_i)$ is the subset of R consisting of relation-schemes R_j, where R_j is connected in G_I to a relation-scheme R_k of $Casc(R_i)$, by an edge that corresponds to a referential integrity constraint $R_j[Y] \subseteq R_k[K_k]$ associated with a *nullifies* delete-rule, such that none of the attributes of foreign-key Y belongs to a set of attributes Z that is involved in a null constraint of the form R_j: $W \xrightarrow{EX} Z$;
- $Restr(R_i)$ is the subset of R consisting of relation-schemes R_j, where R_j is connected in G_I to a relation-scheme R_k of $Casc(R_i)$, by an edge that corresponds to a referential integrity constraint $R_j[Y] \subseteq R_k[K_k]$ associated with either (i) a *nullifies* delete-rule, such that at least one of the attributes of foreign-key Y belongs to a set of attributes Z that is involved in a null constraint of the form $R_j : W \xrightarrow{EX} Z$, or (ii) a *restricted* delete-rule.

Relational schema RS is said to satisfy safeness condition **S1** iff for every relation-scheme R_i of R, set $Restr(R_i)$ is disjoint with both $Casc(R_i)$ and $Null(R_i)$. ■

The relational schemas of examples 4.1, 4.2, 4.3, and 4.4 above, for instance, do not satisfy condition S1: in example 4.1 relation-scheme R_2 belongs to both $Restr(R_3)$ and $Casc(R_3)$, in examples 4.2 and 4.3 relation-scheme R_1 belongs to both $Restr(R_1)$ and $Casc(R_1)$, and in example 4.4 relation-scheme R_1 belongs to both $Restr(R_3)$ and $Null(R_3)$. Conversely, the relational schema of figure 1(i) satisfies condition **S1**.

Proposition 4.1. Let $RS = (R, F \cup I \cup N)$ be a relational schema, where F, I, and N denote sets of key, referential integrity, and null constraints, respectively. If RS satisfies condition **S1** then for every relation r_i associated with a relation-scheme R_i of R, and for every deletion δ involving one or several tuples of r_i, the effect of δ is independent of both the order in which the referential integrity constraints of I are enforced, and of the order in which the tuples involved in δ are accessed.

Proof Sketch. Let $T(\delta)$ denote the set of tuples either involved in δ or potentially affected by δ by enforcing referential integrity constraints associated with *cascades* or *nullifies* delete-rules. Let $\bar{T}(\delta)$ denote the set of tuples potentially blocking the deletion or update of tuples in $T(\delta)$ following the enforcement of either null constraints, or referential integrity constraints associated with *restricted* delete-rules. Clearly, if $\bar{T}(\delta) \not\subseteq T(\delta)$ then δ either fails (for $\bar{T}(\delta) - T(\delta) \neq \emptyset$) or succeeds (for $\bar{T}(\delta) = \emptyset$). However, if $\bar{T}(\delta) \subseteq T(\delta)$ then different sequences of accessing the tuples involved in δ or of enforcing the referential integrity constraints, can lead to different results. The proof shows that condition **S1** ensures that $\bar{T}(\delta) \not\subseteq T(\delta)$. ■

The second safeness condition refers to data manipulation deadlocks caused by *cyclic* referential integrity structures; such structures involve referential integrity constraints that correspond to directed cycles in the associated referential integrity graphs.

Example 4.5. Consider relation-schemes R_1 and R_2 of the relational schema of figure 1(i), and suppose that referential integrity constraints I_1, I_2, and I_3 are all associated with *restricted* delete-rules. Suppose also that in the database state of figure 1(iii) all the tuples of relation r_1 are total, that relation r_1 includes two additional tuples, (5 2 6 *b*) and (6 5 2 *a*), and relation r_2 includes the additional tuple (6 *b*). If foreign-keys S_SSN and M_SSN associated with R_1 are not allowed to have null values, then referential integrity constraints I_1, I_2, and I_3 prevent the deletion of these three additional tuples from r_1 and r_2, although their deletion results in a consistent

database state, namely that shown in figure 1(iii).

The safeness condition specified below ensures that the null and referential integrity constraints do not cause data manipulation deadlocks such as that discussed in example 4.5 above.

Definition 4.2 – *Safeness Condition* **S2**.

Let $RS = (R, F \cup I \cup N)$ be a relational schema, where F, I, and N denote sets of key, referential integrity, and null constraints, respectively, and let G_I be the referential integrity graph associated with RS. Relational schema RS is said to satisfy safeness condition **S2** iff for every directed cycle of G_I, at least one of the referential integrity constraints that correspond to an edge of this cycle, involves a foreign-key whose attributes are allowed to have null values. ■

The relational schema of example 4.5, for instance, does not satisfy condition **S2**, while the relational schema of figure 1(i) satisfies condition **S2**.

Example 4.6. Consider the relational schema of example 4.5, and suppose that in relation-scheme R_1 foreign-key S_SSN is allowed to have null values, while foreign-key M_SSN is not allowed to have null values. Suppose that in the database state of figure 1(iii) relations r_1 and r_2 include the additional tuples mentioned in example 4.5. Then the deletion of these tuples, namely of tuples (5 2 6 *b*) and (6 5 2 *a*) from r_1 and of tuple (6 *b*) from r_2, can be performed as follows: (i) subtuple t[S_SSN] in tuple (6 5 2 *a*) of r_1 is nullified; (ii) tuple (5 2 6 *b*) is deleted from r_1, (iii) tuple (6 *b*) is deleted from r_2, and (iv) tuple (6 – 2 *a*) is deleted from r_1. Note that every manipulation in this sequence results in a consistent database state.

Proposition 4.2. Let $RS = (R, F \cup I \cup N)$ be a relational schema, where F, I, and N denote sets of key, referential integrity, and null constraints, respectively. If RS satisfies condition **S2** then for every consistent database state associated with RS, r, there exists a sequence of data manipulations that map the empty database state (resp. r) associated with RS into r (resp. the empty database state), so that every data manipulation results in a consistent database state of RS.

Proof Sketch. We consider below only the mapping of the empty database state into r. The proof for the reverse mapping, of r into the empty database state, is similar (example 4.6 illustrates this reverse mapping). G_I denotes below the referential integrity graph associated with RS.

(i) If G_I is acyclic then G_I defines the following partial order for the relation-schemes of R: for every pair of relation-schemes of R, R_i and R_j, $R_i > R_j$ iff $R_j \rightarrow R_i$ is an edge in G_I. Mapping the empty database state into r can be achieved by following this order for inserting tuples (i.e. if $R_i > R_j$ then all the tuples of the relation associated with R_i are inserted before the tuples of the relation associated with R_j).

(ii) If G_I has cycles then the subgraph G'_I of G_I is defined as follows: G'_I results by removing from G_I edges that belong to cycles, so that every removed edge corresponds to a referential integrity constraint involving a foreign-key whose attributes are allowed to have null values. Condition **S2** ensures that G'_I is acyclic. Consequently, G'_I defines the following partial order for the relation-schemes of R: for every pair of relation-schemes of R, R_i and R_j, $R_i > R_j$ iff $R_j \rightarrow R_i$ is an edge in G'_I. Then mapping the empty database state into r can be achieved by: (a) following this order for inserting tuples, as discussed above in (i), where the values of the attributes of foreign keys involved in referential integrity constraints that correspond to edges of G_I that do not appear in G'_I, are replaced by null values; and (b) by replacing the null values introduced in (a) with the actual values for these foreign keys attributes. ■

5. Safe Referential Integrity Structures in DB2, SYBASE, and INGRES

In this section we discuss problems underlying the specification of safe referential integrity structures in DB2, SYBASE 4.0, and INGRES 6.3. Additional details regarding the referential integrity mechanisms of these systems can be found in [9].

The mechanisms provided by SYBASE and INGRES for maintaining referential integrity constraints are general mechanisms that can be used for maintaining other (e.g. null) constraints as well. In SYBASE and INGRES databases the specifications of referential integrity (and/or other) constraints are *encoded* in (trigger or rule) procedures, and it is very hard, if not impossible, to *decode* these constraint specifications by parsing these procedures. Consequently, it is not surprising that SYBASE and INGRES do not provide any mechanism for detecting problematic (e.g. unsafe) or even not well-defined referential integrity constraints. In systems such as SYBASE and INGRES, users are solely responsible for both the syntactic and semantic correctness of the procedural specifications for referential integrity constraints.

In RDBMSs that support declarative specifications of referential integrity constraints, the structure of the constraints can be easily analyzed. Accordingly, in such systems problematic referential integrity structures can be detected and disallowed. The only (to our knowledge) RDBMS providing such a capability is DB2. The following two restrictions imposed by DB2 are meant to avoid

the same problems as safeness condition S1 defined in section 4. Note that the notations in the definition below differ from the notations used in [4].

Definition 5.1.

Let $RS = (R, F \cup I \cup N)$ be a relational schema, where F, I, and N denote sets of key, referential integrity, and null constraints, respectively. Let G_I be the referential integrity graph associated with RS. Let $Casc(R_i)$ be defined as in section 4, and let $Null'(R_i)$, $Restr'(R_i)$, and $Null''(R_i)$ be defined as follows:

- $Null'(R_i)$ is the subset of R consisting of relation-schemes R_j, where R_j is connected in G_I to a relation-scheme of $Casc(R_i)$, by an edge that corresponds to a referential integrity constraint associated with a *nullifies* delete-rule;
- $Restr'(R_i)$ is the subset of R consisting of relation-schemes R_j, where R_j is connected in G_I to a relation-scheme of $Casc(R_i)$, by an edge that corresponds to a referential integrity constraint associated with a *restricted* delete-rule;
- $Null''(R_i)$ is the subset of $Null'(R_i)$ consisting of relation-schemes R_j, where R_j is connected in G_I to relation-schemes of $Casc(R_i)$ by at least two edges corresponding to referential integrity constraints associated with *nullifies* delete-rules.

In DB2 the referential integrity constraints must satisfy the following two restrictions:

- **T1**: For every relation-scheme R_i of R, sets $(Casc(R_i) - \{R_i\})$, $Null'(R_i)$, and $Restr'(R_i)$ are pairwise disjoint, and set $Null''(R_i)$ is empty.
- **T2**: For every subset I' of I that consists of referential integrity constraints corresponding to edges forming a directed cycle in G_I: (i) if I' consists of a single constraint, then this constraint must be associated with a *cascades* delete-rule; (ii) if I' consists of two or more constraints, then at least two constraints of I' must be associated with *restricted* or *nullifies* delete-rules. ■

Note that the DB2 restrictions above do not require cyclic referential integrity structures to involve at least one foreign-key consisting of attributes that are allowed to have null values (safeness condition S2), and therefore data manipulations deadlocks such as those discussed in section 4 are not prevented in DB2. We compare below conditions T1 and T2 with safeness condition S1.

Condition T1 is more restrictive than condition S1.

Example 5.1. Suppose that in the relational schema of figure 1(i) referential integrity constraints I_3 and I_5 are associated with *nullifies* delete-rules, while referential integrity constraint I_4 is associated with a *cascades* delete-rule. Then condition T1 is not satisfied (set $Null''(R_3)$ is not empty), while condition S1 is satisfied (set $Restr(R_3)$ is empty).

The additional restriction imposed by T1 is meant to avoid the effect of null constraints on the outcome of deletions, as discussed in section 4 (see example 4.4). Recall that DB2 provides a mechanism for maintaining general null constraints using special *Validproc* procedures that are activated (triggered) by every tuple manipulation. Thus, in example 5.1 above, a *Validproc* procedure associated with relation-scheme R_1 can be used in order to maintain a null constraint R_1: P_NR $\stackrel{EX}{\rightarrow}$ M_SSN, that disallows M_SSN values to be null in tuples where P_NR values are not null.

Example 5.2. Consider the relational schema of figure 1(i), and suppose that referential integrity constraints I_3 and I_5 are associated with *nullifies* delete-rules, while referential integrity constraint I_4 is associated with a *cascades* delete-rule. If relation-scheme R_1 is associated with null constraint R_1: P_NR $\stackrel{EX}{\rightarrow}$ M_SSN, then this schema does not satisfy both condition T1 (again, set $Null''(R_3)$ is not empty), and condition S1 (set R_1 belongs to both $Restr(R_3)$ and $Null(R_3)$).

However, even when null constraints are involved condition T1 is still more restrictive than condition S1.

Example 5.3. Consider the relational schema of figure 1(i) as specified in example 5.2 above, but with referential integrity constraint I_5 associated with a *restricted* delete-rule. Then this schema does not satisfy condition T1 (R_1 belongs to both $Null'(R_3)$ and $Restr'(R_3)$), while it satisfies condition S1 (R_1 belongs to $Restr(R_3)$, but does not belong to either $Null(R_3)$ or $Casc(R_3)$).

Condition T2 treats cycles involving single referential integrity constraints differently from cycles involving multiple referential integrity constraints. This apparent contradiction does not exist for safe referential integrity structures.

Example 5.4. Consider the relational schema of figure 1(i). If I_2 is associated with a *nullifies* delete-rule then condition T2 is not satisfied, while condition S1 is satisfied; both T2 and S1 are satisfied if I_2 is associated with a *cascades* delete-rule. Similarly, if referential integrity constraints I_1 and I_3 are both associated with *cascades* delete-rules, then condition T2 is not satisfied, while condition S1 is satisfied; both T2 and S1 are satisfied if I_1 and I_3 are associated with *restrict* or

nullifies delete-rules.

According to [4], condition T2 ensures that deletions do not depend on the access sequence selected by the query optimizer. Note that this is the same goal as that of condition S1 (see proposition 4.1). However, while condition S1 is based on the assumption that the set of tuples involved in a deletion does not change during its execution, DB2 allows such sets of tuples to change during deletion executions, that is, allows *ambiguous* deletions.

Example 5.5. Suppose that the relational schema of figure 1(i) includes only one referential integrity constraint, I_2, that is associated with a *nullifies* delete-rule, and suppose that relation-scheme R_1 (EMPLOYEE) is associated with relation r_1 of figure 1(iii). Consider the following data manipulation:

DM : DELETE FROM EMPLOYEE WHERE S_SSN IS NULL

which requires the deletion from relation r_1 of the tuples that represent employees without supervisors. ***DM*** has two possible executions, depending on the order in which the tuples of r_1 is accessed:

1. if tuples $(2 - - b)$ and $(4 - - a)$ are accessed first, then tuples $(3\ 2 - b)$ and $(1\ 4\ 4\ a)$ are also deleted, because the S_SSN values in these tuples turn to nulls after the first two deletions;
2. if tuples $(2 - - b)$ and $(4 - - a)$ are accessed last, then none of the tuples in r_1 are deleted.

The problem here, however, is not caused by the existence of multiple access sequences for ***DM***, but by the ambiguity of ***DM***. Thus, the two executions above correspond to different interpretations of ***DM***: while the first execution interprets the WHERE condition as a *precondition* for the deletion (i.e remove only tuples that represent employees without supervisors at the time of expressing, but before carrying out, ***DM***), the second execution interprets the WHERE condition as a *postcondition* for the deletion (i.e following ***DM***, none of the tuples should represent employees without supervisors). Accordingly, not the structure of the referential integrity constraints should be restricted, but instead ambiguous data manipulations should be rejected.

Interestingly, while ***DM*** is allowed by DB2, a deletion equivalent to ***DM*** expressed over a relational schema equivalent to the schema of figure 1(i), is not allowed by DB2, as illustrated by the following example.

Example 5.6. Consider the relational schema shown in figure 5(i). As explained later in this section, this schema is equivalent to the relational schema of figure 1(i). The following data manipulation expressed over the relational schema of figure 5(i) is equivalent to ***DM***:

DM' : DELETE FROM EMPLOYEE WHERE E_SSN NOT IN
(SELECT E_SSN FROM SUPERVISE)

Like ***DM***, ***DM'*** is ambiguous and has two possible executions. However, deletions such as ***DM'*** are detected as ambiguous, and therefore are rejected by DB2 (for more details see the section on DML restrictions in [4]).

It can be easily verified that DB2 conditions T1 and T2 are equivalent to safeness conditions S1 and S2 for relational schemas of the following form:

$RS = (R, F \cup I \cup N)$, where R is a set of relation-schemes, F is a set of key constraints, I is an *acyclic* set of referential integrity constraints that are associated either with *restricted* or *cascades* delete-rules and *restricted* update-rules, and N consists only of

i. Relation-Schemes (*Keys are underlined*)

(R'_1) EMPLOYEE (E_SSN, P_NR)
(R'_2) MANAGER (M_SSN, P_NR)
(R'_3) PROJECT (P_NR)
(R'_4) SUPERVISE (E_SSN, S_SSN)
(R'_5) LEAD (E_SSN, M_SSN)

Null Constraints (*Nulls-Not-Allowed*)

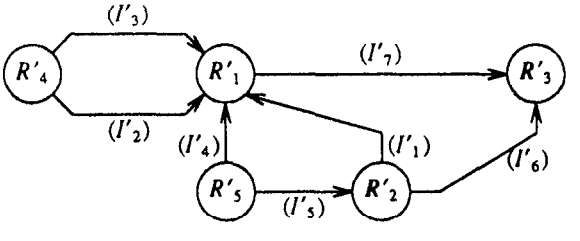
EMPLOYEE: $\varnothing \overset{EX}{\rightarrow}$ E_SSN SUPERVISE: $\varnothing \overset{EX}{\rightarrow}$ E_SSN, S_SSN
MANAGER: $\varnothing \overset{EX}{\rightarrow}$ M_SSN LEAD: $\varnothing \overset{EX}{\rightarrow}$ E_SSN, M_SSN
PROJECT: $\varnothing \overset{EX}{\rightarrow}$ P_NR

Referential Integrity Constraints

(I'_1) MANAGER	[M_SSN]	$\subseteq$ EMPLOYEE	[E_SSN]
(I'_2) SUPERVISE	[S_SSN]	$\subseteq$ EMPLOYEE	[E_SSN]
(I'_3) SUPERVISE	[E_SSN]	$\subseteq$ EMPLOYEE	[E_SSN]
(I'_4) LEAD	[E_SSN]	$\subseteq$ EMPLOYEE	[E_SSN]
(I'_5) LEAD	[M_SSN]	$\subseteq$ MANAGER	[M_SSN]
(I'_6) MANAGER	[P_NR]	$\subseteq$ PROJECT	[P_NR]
(I'_7) EMPLOYEE	[P_NR]	$\subseteq$ PROJECT	[P_NR]

Rules	insert	delete	update
(I'_1, I'_4, I'_5, I'_6)	*restricted*	*restricted*	*restricted*
(I'_2, I'_3)	*restricted*	*cascades*	*restricted*
(I'_7)	*restricted*	*nullifies*	*restricted*

ii. Referential Integrity Graph :



Abbr. : E(MPLOYEE), M(ANAGER), P(ROJECT), S(UPERVISOR)

Figure 5. A Relational Schema Equivalent to the Relational Schema of Figure 1(i).

nulls-not-allowed constraints, that disallow (primary and foreign) key attributes to have null values.

Certain relational schemas can be transformed into schemas of the form specified above. A transformation that, under certain conditions, can remove cyclic referential integrity structures, referential integrity constraints associated with *nullifies* delete-rules, and foreign-key attributes that are allowed to have null values, is given in [8]. This transformation is exemplified in figure 5: the relational schema of figure 5(i) results from the relational schema of figure 1(i) by splitting relation-scheme R_1 into relation-schemes R'_1, R'_4, and R'_5, while adapting accordingly the key, referential integrity and null constraints of the relational schema of figure 1(i). The result of this transformation is a schema equivalent with (i.e. having the same information-capacity as) the schema of figure 1(i) (see [8] for more details). Transformations such as that exemplified above, can be used in order to transform a relational schema involving a safe referential integrity structure into a relational schema that involves a referential integrity structure that is both safe and complying with the restrictions imposed by DB2.

6. Summary

We have examined the data manipulation problems caused by certain structures of referential integrity and null constraints, and developed *safeness* conditions for avoiding these problems. These conditions should complement the *well-definedness* conditions that ensure that referential integrity constraints are specified correctly.

We have examined the problems underlying the specification of safe referential integrity structures in three relational database management systems, DB2, SYBASE, and INGRES. DB2 allows declarative specifications of referential integrity constraints, but imposes restrictions on the structure of referential integrity constraints. We have shown that some of these restrictions limit the capability of specifying safe referential integrity structures in DB2; conversely, DB2 allows the specification of some unsafe referential integrity structures. We have also shown that ambiguous data manipulations are not treated uniformly in DB2.

The procedurality of the referential integrity mechanisms provided by SYBASE and INGRES makes very hard, if not impossible, the task of detecting unsafe or even not well-defined referential integrity constraints. Therefore, it is not surprising that SYBASE and INGRES do not provide mechanisms for detecting erroneous referential integrity constraints.

References

[1] E.F. Codd, "Extending the relational database model to capture more meaning", *ACM TODS* **4**, 4 (Dec 1979), pp. 397-434.

[2] C.J. Date, "Referential integrity", in *Relational Database-Selected Writings*, Addison-Wesley, 1986.

[3] C.J. Date, "Referential integrity and foreign keys: Further considerations", in *Relational Database-Writings 1985-1989*, Addison-Wesley, 1990.

[4] IBM Corporation, "IBM DATABASE 2 Referential Integrity Usage Guide", June 1989.

[5] Ingres, Inc., "INGRES/SQL Reference Manual", Release 6.3, Alameda, California, Nov. 1989.

[6] D. Maier, *The theory of relational databases*, Computer Science Press, 1983.

[7] V.M. Markowitz, "Referential integrity revisited: An object-oriented perspective", Proc. of the *16th VLDB Conference*, Brisbane, Australia, 1990, pp. 578-589.

[8] V.M. Markowitz and J.A. Makowsky, "Identifying extended entity-relationship object structures in relational schemas", *IEEE Trans. on Software Engineering*, **16**, 8, (Aug. 1990), pp. 777-790.

[9] V.M. Markowitz, "Problems underlying the use of referential integrity mechanisms in relational database management systems", Proc. of the *7th Int. Conf. on Data Engineering*, Japan, 1991.

[10] Sybase, Inc., "Transact-SQL User's Guide", Release 4.0, Emeryville, California, Oct. 1989.