2

CONF-9104321--1

LBL-30104

Lawrence Berkeley Laboratory

UNIVERSITY OF CALIFORNIA

Information and Computing Sciences Division

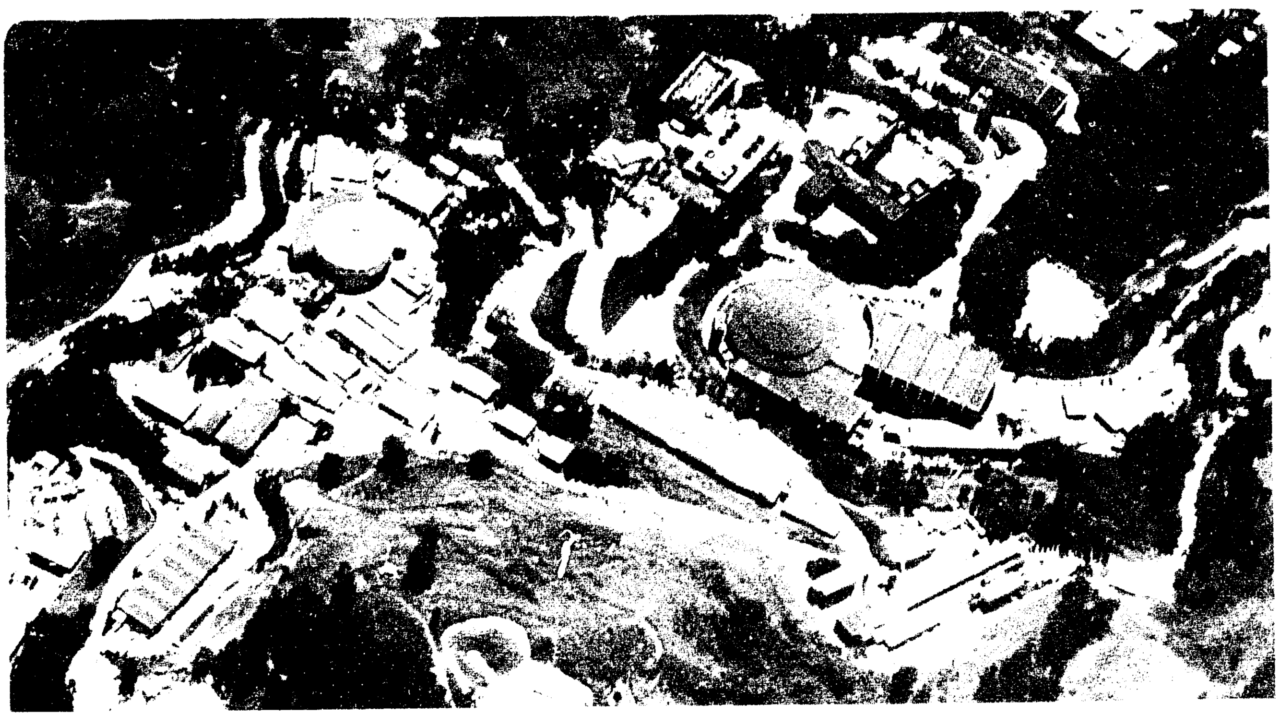
DEC 17 1991

Presented at the 7th IEEE International Conference on Data Engineering, Kobe, Japan, April 8–12, 1991, and to be published in the Proceedings

Problems Underlying the Use of Referential Integrity Mechanisms in Relational Database Management Systems

V.M. Markowitz

December 1990



Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098

DISCLAIMER

LBL--30104

DE92 004304

Problems Underlying the Use of Referential Integrity Mechanisms in Relational Database Management Systems

Victor M. Markowitz

Computing Science Research & Development
Information & Computing Sciences Division
Lawrence Berkeley Laboratory
1 Cyclotron Road
Berkeley, California 94720

December 1990

Proceedings of the 7th IEEE International Conference on Data Engineering, Kobe, Japan, April 8-12, 1991

This work was supported by the Director, Office of Energy Research, Applied Mathematics Sciences Research Program and the Office of Health and Environmental Research Program of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

PROBLEMS UNDERLYING THE USE OF REFERENTIAL INTEGRITY MECHANISMS IN RELATIONAL DATABASE MANAGEMENT SYSTEMS *

Victor M. Markowitz

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

ABSTRACT

Referential integrity is used in relational databases for expressing existence dependencies between tuples. Relational database management systems (RDBMS) provide diverse referential integrity capabilities. Thus, in some RDBMSs referential integrity constraints can be specified non-procedurally (declaratively), while in other RDBMSs they must be specified procedurally. Moreover, some RDBMSs restrict the class of allowed referential integrity constraints. We examine in this paper the main problems underlying the use of referential integrity mechanisms in three representative RDBMSs, DB2, SYBASE, and INGRES.

1. INTRODUCTION

In relational databases existence dependencies between tuples are expressed using *referential integrity constraints* [1]; referential integrity constraints are specified by associating a *foreign-key* in one relation with the *primary-key* of another relation [3]. Referential integrity constraints are usually associated with *referential integrity rules* that define the behavior of the relations involved in these constraints under insertion, deletion, and update.

Presently, several relational database management systems (RDBMS), notably IBM's DB2, SYBASE, and INGRES, support the specification of referential integrity constraints. The referential integrity mechanisms provided by these systems are different and difficult to use. Thus, SYBASE [11] and INGRES [7] provide mechanisms (*triggers* in SYBASE and *rules* in INGRES) for specifying referential integrity constraints procedurally. Conversely, DB2 [6] supports non-procedural (declarative) specifications of referential integrity constraints, but with restrictions on the structure of such constraints. In this paper we examine and compare the referential integrity mechanisms of DB2, SYBASE, and INGRES, and discuss the main problems underlying their use.

We examine the mechanisms provided by SYBASE and INGRES for the procedural specification of referential integrity constraints. We show that although conceptually similar, these mechanisms are technically different, with the INGRES *rule* mechanism being more flexible and less restrictive than the SYBASE *trigger* mechanism. The task of specifying procedurally referential integrity constraints in SYBASE and INGRES is tedious and labor-intensive, and therefore likely to be avoided by most users. Moreover, SYBASE and INGRES leave to users the task of specifying correct referential integrity structures.

Compared to the complexity of the procedural referential integrity mechanisms of SYBASE and INGRES, the non-procedural referential integrity mechanism of DB2 is significantly simpler. Furthermore, DB2 has been unique among RDBMSs in addressing data manipulation problems caused by certain referential integrity structures. DB2 attempts to avoid such problems by imposing restrictions on the structure of referential integrity constraints it allows. We show that these restrictions are too stringent and do not prevent certain data manipulation problems.

The rest of the paper is organized as follows. In section 2 we briefly review the relational concepts used in this paper. In section 3 we examine the mechanisms provided by SYBASE and INGRES for the procedural specification of referential integrity constraints. The DB2 mechanism supporting the declarative specification of referential integrity constraints is examined in section 4. Section 5 concludes this paper with a summary and a brief discussion of further issues. A generic procedural definition for referential integrity constraints is given in the appendix.

* Issued as technical report LBL-30104. 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.

2. Preliminary Definitions

We use in this paper some graph-theoretical concepts. Any textbook on graph theory (e.g. [5]) can provide the necessary reference. 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, from vertex v_i to vertex v_j; h is said to be *incident* from v_i to 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. [8]) for the basic concepts, and in [2] for inclusion dependencies. 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 dependencies over R. We consider relational schemas with $\Delta = F \cup I \cup N$, where F, I, and N denote sets of functional dependencies, inclusion dependencies, 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. 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 *functional dependency* over R_i is a statement of the form $R_i\colon Y \rightarrow Z$, where Y and Z are subsets of X_i; $R_i\colon Y \rightarrow Z$ is *satisfied* by r_i iff for any two tuples of r_i, t and t', $t[Y]=t'[Y]$ implies $t[Z]=t'[Z]$. A *key* associated with R_i is a subset of X_i, K_i, such that $R_i : K_i \rightarrow X_i$ is satisfied by any r_i associated with R_i 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. An *inclusion dependency* is a statement of the form $R_i[Y] \subseteq R_j[Z]$, where Y and Z are compatible subsets of X_i and X_j, respectively; $R_i[Y] \subseteq R_j[Z]$ is *satisfied* by r_i and r_j iff $\pi{\downarrow}_Y(r_i) \subseteq \pi{\downarrow}_Z(r_j)$. If Z is the primary-key of R_j then $R_i[Y] \subseteq R_j[Z]$ is said to be *key-based*, and Y is called a *foreign-key* of R_i. Key-based inclusion dependencies are *referential integrity* constraints ([1], [3]).

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

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* [3]. 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 (resp. update) of t' cannot be performed if there exist tuples in r_i referencing t'; a *cascades* delete (resp. update) rule asserts that the deletion (resp. update)

---

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*)

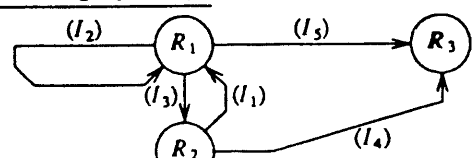
EMPLOYEE: $\emptyset \overset{EX}{\rightarrow}$ E_SSN MANAGER: $\emptyset \overset{EX}{\rightarrow}$ M_SSN
PROJECT: $\emptyset \overset{EX}{\rightarrow}$ P_NR

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*	*cascades*
(I_2, I_5)	*restricted*	*nullifies*	*cascades*

ii. Referential Integrity Graph :



iii. Database State :

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

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

Figure 1. A Relational Database Example.

of 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 (resp. update) of t' implies setting to null the subtuple $t[Y]$ in all the tuples t of r_i referencing t'.

A *null constraint* is a restriction on the way nulls appear in relations [8]. 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: $Y \rightrightarrows Z$, where Y and Z are subsets of X_i; R_i: $Y \rightrightarrows Z$ 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 *nulls-not-allowed* type of null constraint. A *nulls-not-allowed* constraint has the form R_i: $\emptyset \rightrightarrows Z$; R_i: $\emptyset \rightrightarrows Z$ 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 dependencies, referential integrity constraints, and nulls-not-allowed constraints is shown in figure 1(i); the referential integrity graph corresponding to this schema is shown in figure 1(ii).

3. Referential Integrity in Sybase and Ingres

SYBASE [11] and INGRES [7] do not allow declarative specifications of referential integrity constraints. Instead, they provide mechanisms for specifying such constraints procedurally. In this section we examine the main problems underlying the use of these mechanisms.

Referential integrity constraints can be enforced in a database by executing a *referential integrity procedure* whenever a relation is affected by a data manipulation consisting of tuple insertions, deletions, or updates. Given a data manipulation δ involving one or several tuples of a relation r_i associated with relation scheme R_i, the referential integrity procedure corresponding to r_i must:

(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.

The definition of a generic referential integrity procedure called ***RefProc*** is given in the appendix. ***RefProc*** assumes that for every relation r_i there exists a relation called *change*$_i$, that records how a given data manipulation δ would affect the tuples of r_i, without actually applying δ on r_i. Every tuple $\bar{t}$ of *change*$_i$ consists of the concatenation of two tuples, t and t', where t is an existing tuple of r_i that is going to be deleted or updated, and t' is a new tuple that is either going to be inserted in r_i, or is going to replace t in r_i.

The mechanism provided by SYBASE for the procedural specification of referential integrity constraints involves a special kind of stored procedures, called ***triggers*** that are activated (***fired***) when a relation is affected by a data manipulation. A trigger procedure 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 relation r_i then following a data manipulation the ***deleted*** relation consists of the r_i tuples that are going to be deleted or updated; the ***inserted*** relation consists of tuples that are going to be inserted into r_i, or of newly updated tuples of r_i. SYBASE allows the specification of three trigger procedures per relation: an ***insert***, a ***delete***, and an ***update*** trigger procedure. These procedures can be derived straightforwardly from ***RefProc*** as follows:

```
create trigger insertMANAGER on MANAGER for insert as
begin
  declare @row int, @insPROJECT int, @nullPROJECT int,
          @insEMPLOYEE int, @nullEMPLOYEE int
  select @row = @@rowcount
  select @nullPROJECT = count(*) from inserted
                  where inserted.P_NR = null
  select @insPROJECT = count(*) from inserted, PROJECT
                  where inserted.P_NR = PROJECT.P_NR
  select @insEMPLOYEE = count(*) from inserted, EMPLOYEE
                  where inserted.M_SSN = EMPLOYEE.E_SSN
  if @nullPROJECT + @insPROJECT + @insEMPLOYEE
                            != 2 * @row
  begin
        print "Failed insertion into MANAGER because of"
        if @nullPROJECT + @insPROJECT != @row
              print "Missing reference to PROJECT"
        if @nullEMPLOYEE + @insEMPLOYEE != @row
              print "Missing reference to EMPLOYEE"
        rollback transaction
  end
end

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
        print "Failed deletion from MANAGER because of
              existing reference from EMPLOYEE"
        rollback transaction
  end
end
```

Note: @@rowcount = number of tuples affected by insertion

Figure 2. SYBASE Trigger Examples.

- *insert* trigger procedures correspond to *RefProc* [I(1, 2.a, 3)]; *delete* trigger procedures correspond to *RefProc* [I(1, 2.b, 3), II(1, 2.a)]; and *update* trigger procedures correspond *RefProc* [I(1, 2.a, 2.c, 3), II(1, 2.b)];
- relations *deleted* and *inserted* replace relation $change_i$ as follows: *deleted* is the projection of $change_i$ that includes existing tuples of r_i that are affected by the data manipulation under consideration; and *inserted* is the projection of $change_i$ that includes new tuples that are going to be inserted into r_i following the data manipulation under consideration;
- the relational algebra expressions in the definition of *RefProc* are translated into *SQL* expressions.

Trigger procedures are specified in SYBASE's *Transact−SQL* which allows in addition to the standard SQL the specification of control-flow statements. For example, the *insert* and *delete* trigger procedures for relation-scheme MANAGER of the relational schema of figure 1(i) are shown in figure 2.

The SYBASE trigger mechanism has the following limitations:

1. the number of levels allowed for nesting triggers is limited to 16;
2. a trigger cannot be fired more than once for a given data manipulation; thus, if deleting a tuple t in a relation r_i leads (cascades) to the deletion of another tuple, t', in r_i then the delete trigger associated with r_i is not activated by the deletion of t';
3. the employment of the system provided relations *inserted* and *deleted* does not provide a way of keeping track of how new tuples replace existing tuples in a relation.

Restriction (2) above means that cyclic referential integrity structures involving referential integrity constraints associated with *cascades* delete-rules cannot be correctly specified in SYBASE. Restriction (3) above means that a *cascades* update-rule can be implemented only if updates of primary-key values referenced by other tuples, are limited to single tuples at a time, that is, only if the *inserted* and *deleted* relations consist of at most one referenced tuple (see the note in the appendix). Finally, *Transact−SQL* includes an operation called TRUNCATE TABLE that deletes all the tuples in a relation without activating the delete triggers, and thus potentially undermining the referential integrity of the database.

The mechanism provided by INGRES for the procedural specification of referential integrity constraints is conceptually similar to the SYBASE trigger mechanism. Instead of triggers INGRES allows the specification of *rules*. Like the triggers, rules are activated when relations are affected by data manipulations. However, while triggers embody the referential integrity procedures, rules are employed only as a mechanism for invoking the referential integrity procedures which must be specified separately. While SYBASE triggers are set-oriented (i.e. are activated for sets of tuple manipulations), INGRES rules are tuple-oriented (i.e. are activated for single tuple manipulations). Accordingly, the *inserted* and *deleted* relations provided by SYBASE are replaced in INGRES by 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

```
create procedure
     p_insertMANAGER (n_P_NR char(20), n_M_SSN int) as
declare msg varchar(80) not null; check_val integer;
begin
   if n_P_NR is not null then
         select count (*) into :check_val from PROJECT
            where P_NR = :n_P_NR;
         if check_val = 0 then
           msg = 'Failed insertion into MANAGER because
                     of missing reference to PROJECT';
           raise error 1 :msg;
         endif;
   endif;
   if n_M_SSN is not null then
         select count (*) into :check_val from EMPLOYEE
            where E_SSN = :n_M_SSN;
         if check_val = 0 then
           msg = 'Failed insertion into MANAGER because
                     of missing reference to EMPLOYEE';
           raise error 2 :msg;
         endif;
   endif;
end;

create rule r_insertMANAGER after insert into MANAGER
   execute procedure p_insertMANAGER (n_P_NR = new.P_NR,
                                 n_M_SSN = new.M_SSN);
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 from 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 3. INGRES Rule Examples.

newly updated tuple of r_i. Although INGRES, unlike SYBASE, allows the specification of any number of rules per relation, it is enough to specify an *insert*, a *delete*, and an *update* rule for each relation. The referential integrity procedures associated with the rules can be derived from *RefProc* in a similar way to the derivation of trigger procedures mentioned above. The procedures associated with rules are specified in INGRES's *(Extended) SQL*, which is richer and more flexible than SYBASE's *Transact–SQL*. For example, the *insert* and *delete* rules and referential integrity procedures for relation-scheme MANAGER of the relational schema of figure 1(i) are shown in figure 3.

The INGRES rule mechanism does not have the limitations of the SYBASE trigger mechanism. However, both SYBASE and INGRES have two important flaws in their referential integrity mechanisms. First, both in SYBASE and INGRES the removal of a relation-scheme R_i leads to the removal of the triggers and rules associated with R_i, but not of the triggers and rules referring to R_i, thus allowing syntactically incorrect trigger and rule specifications. Second, both SYBASE and INGRES provide *data loading* facilities that bypass the triggers and rules, thus allowing the introduction of data that is inconsistent with respect to the referential integrity constraints expressed by triggers and rules. Moreover, SYBASE and INGRES do not provide any mechanism for detecting or removing such inconsistent data.

4. Referential Integrity in DB2

Referential integrity constraints in IBM's DB2 database management system are specified declaratively (i.e. non-procedurally). In this section we examine the main characteristics and limitations of the DB2 referential integrity mechanism.

Referential integrity specifications in DB2 are coupled with the specifications for relation-schemes, primary-keys, and nulls-not-allowed constraints; thus, the DB2 specification 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 specification for relation-scheme EMPLOYEE of the relational schema of figure 1(i) is shown in figure 4.

```
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 4. Example of a Relation Definition in DB2.

Referential integrity constraints are associated in DB2 by default with *restricted* update-rules; DB2 does not support *nullifies* and *cascades* update-rules.

Example 1. Suppose that in the relational database of figure 1(iii) data manipulation δ consists of changing from a to d the value of attribute P_NR in tuple (a) of relation r_3. If the referential integrity constraints are associated with *cascades* update-rules (as they actually are in the schema of figure 1(i)) then δ implies changing from a to d the P_NR values in tuple (4 a) of relation r_2 and in tuples (1 4 4 a) and (4 – – a) of relation r_1. These changes would be carried out automatically while enforcing the referential integrity constraints. Conversely, if the referential integrity constraints are associated with *restricted* update-rules then δ cannot be executed.

Allowing only *restricted* update-rules is misplaced because restricting updates of attribute values should be a property of the attributes, rather than depend on the tuple references. Thus, while there is no reason for restricting updates of regular (key or non-key) relational attributes, updates of *surrogate* attributes are not allowed by definition [1]. Consequently, if (primary and foreign) key attributes are surrogate attributes then update-rules are not needed; however, if key attributes are regular (non-surrogate) attributes then the referential integrity constraints should not be associated with *restricted* update-rules. For updates such as that in example 1 above, DB2 proposes an unreasonably complex alternative: tuples affected by primary-key changes together with all the tuples referencing them must be manually deleted and then reinserted with the new values.

Certain referential integrity structures may have unpredictable effects on the outcome of tuple deletions.

Example 2. 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. The outcome of δ depends on the order in which I_1, I_4, and I_5 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 deletion of tuple (4 a) from r_2 while enforcing I_1; or (ii) if I_4 is enforced first then δ is blocked by tuple (4 a) of r_2.

Example 3. 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. 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 both tuples involved in δ are deleted; or (ii) if $(2 - - b)$ is accessed first then δ is blocked by tuple $(3\ 2 - b)$.

The following restrictions imposed by DB2 on the structure of referential integrity constraints are intended to avoid problems such as those exemplified above.

Definition †. Let $RS = (R, F \cup I)$ be a relational schema, where F and I denote sets of key dependencies and referential integrity constraints, respectively. Let $G_I = (R, H)$ 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 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 $Restr(R_i)$, $Null(R_i)$, and $(Casc(R_i) - \{R_i\})$ 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: if I' consists of a single constraint then this constraint must be associated with a *cascades* delete-rule; otherwise at least two constraints of I' must be associated with *restricted* or *nullifies* delete-rules. ■

For example, the referential integrity structures of examples 2 and 3 above do not satisfy conditions T1, respectively T2. Conditions T1 and T2, however, disallow not only problematic referential integrity structures, but non-problematic ones as well.

Example 4. If in the relational schema of figure 1(i) referential integrity constraints I_3 and I_5 are associated with *nullifies* delete-rules, and I_4 is associated with a *cascades* delete-rule then condition T1 is not satisfied. However, it can be verified that in this case the outcome of deletions does not depend on the sequence in which I_3, I_4, and I_5 are enforced.

The extra restriction imposed by T1 is meant to avoid the effect of null constraints on deletions.

Example 5. 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 $\xrightarrow{EX}$ P_NR. Let deletion δ involve tuple (a) of relation r_3. Note that without N_1 δ would imply nullifying (via I_5) the P_NR values in tuples $(1\ 4\ 4\ a)$ and $(4 - - a)$ of relation r_1, and nullifying (via I_4 and I_3) the M_SSN value in tuple $(1\ 4\ 4\ a)$ of relation r_1. However, N_1 makes the outcome of δ depend on the order in which I_3, I_4, and I_5 are enforced: (i) if I_4 is enforced first then tuple $(4\ a)$ is deleted from r_2, thus leading to the nullification of the M_SSN value in tuple $(1\ 4\ 4\ a)$ of r_1 while enforcing I_3; the subsequent enforcement of I_5 results in nullifying the P_NR values 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 the P_NR value cannot be nullified because of N_1.

Although DB2 does not support declarative specifications of general null constraints such as N_1 above, such constraints can be specified procedurally using a special *Validproc* procedure which is activated (triggered) by every tuple manipulation. However, even when null constraints are involved condition T1 is still too restrictive.

Example 6. Consider the relational schema of figure 1(i), and suppose that referential integrity constraint I_3 is associated with a *nullifies* delete-rule, I_4 is associated with a *cascades* delete-rule, and I_5 associated with a *restricted* delete-rule. If relation-scheme R_1 is associated with null constraint R_1: P_NR $\xrightarrow{EX}$ M_SSN ‡ then condition T1 is not satisfied. However, the outcome of deletions does not

† Our notations differ from the notations used in [6].

‡ Disregard the database state of figure 1(iii) which does not satisfy this constraint.

depend on the sequence in which I_3, I_4, and I_5 are enforced, because the null constraint overrides the *nullifies* delete-rule associated with I_4, thus making I_4 to behave as if it is associated with a *restricted* delete-rule.

Condition T2 ensures that deletions do not depend on the access sequence selected by the query optimizer (e.g. see example 3 above). However, the restriction of not allowing *nullifies* delete-rules for referential integrity constraints such as I_2 of figure 1(i) is misplaced.

Example 7. Suppose that the relational schema of figure 1(i) includes only referential integrity constraint I_2 associated with a *nullifies* delete-rule, and 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 deleting from r_1 tuples that represent employees without supervisors. ***DM*** has two possible executions depending on the order in which the tuples of r_1 are accessed: (i) if tuples $(2 - - b)$ and $(4 - - a)$ are accessed first, then tuples $(3\ 2 - b)$ and $(1\ 4\ 4\ a)$ are also deleted since the S_SSN values in these tuples turn to nulls while enforcing I_2 following the first deletions; or (ii) if tuples $(2 - - b)$ and $(4 - - a)$ are accessed last, then no other tuples are deleted.

The problem illustrated above, 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, the second execution interprets the WHERE condition as a postcondition for the deletion. Accordingly, instead of not allowing *nullifies* delete-rules for referential integrity constraints such as I_2 above, ambiguous deletions such as ***DM*** should be rejected.

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

Example 8. Suppose that the relational schema shown in figure 1(i) is transformed as follows:

(a) relation-scheme EMPLOYEE is split into two relation-schemes: EMPLOYEE (E_SSN, M_SSN, P_NR)

and SUPERVISE (E_SSN, S_SSN);

(b) SUPERVISE is associated with null constraint

$\varnothing \stackrel{EX}{\rightarrow}$ E_SSN, S_SSN;

(c) SUPERVISE is involved in two referential integrity constraints associated with *cascades* delete-rules:

SUPERVISE [S_SSN] $\subseteq$ EMPLOYEE [E_SSN]

and SUPERVISE [E_SSN] $\subseteq$ EMPLOYEE [E_SSN].

It can be verified that this transformation results in a schema equivalent to the schema of figure 1(i), and that the following data manipulation expressed over the new schema 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 by DB2 as ambiguous and therefore rejected.

While examples 4, 6, 7, and 8 above illustrate how the conditions imposed by DB2 on the structure of referential integrity constraints can be excessively restrictive, the example below involves a data manipulation problem that, although caused by a referential integrity structure, is not prevented by DB2.

Example 9. Consider relation-schemes R_1 and R_2 of the relational schema of figure 1(i), and suppose that referential integrity constraint I_2 is associated with a *cascades* delete-rule, so that conditions T1 and T2 are both satisfied. 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 insertion of tuples $(5\ 2\ 6\ b)$ and $(6\ 5\ 2\ a)$ in r_1, and of tuple $(6\ b)$ in r_2, although once inserted these tuples satisfy I_1, I_2, and I_3.

5. CONCLUSION.

We have examined the referential integrity mechanisms of three relational database management systems (RDBMS), DB2, SYBASE, and INGRES. DB2 supports the declarative specification of referential integrity constraints, but imposes restrictions on the structure of referential integrity constraints. We have shown that some of these restrictions limit unreasonably the specification of referential integrity constraints in DB2; conversely, DB2 allows the specification of some referential integrity structures that cause data manipulation problems. We have also shown that ambiguous data manipulations are not treated uniformly in DB2.

We have examined the mechanisms provided by SYBASE and INGRES for the procedural specification of referential integrity constraints. We have shown that although conceptually similar, these mechanisms differ, with the INGRES rule mechanism being more flexible and less restrictive than the SYBASE trigger mechanism. Unlike DB2, SYBASE and INGRES do not provide any mechanism for detecting erroneous referential integrity structures.

Compared with the relative simplicity of specifying declarative referential integrity constraints in DB2, specifying SYBASE triggers and INGRES rules is a tedious and error-prone process. Triggers and rules can be made transparent by providing users with a language for the declarative specification of referential integrity constraints, and a compiler for generating code for trigger and rule procedures. Such a compiler has been incorporated into the *Schema Design* and *Translation* (*SDT*) tool described in [10]. *SDT* supports the design of both conceptual (Extended Entity-Relationship) schemas and abstract (i.e. RDBMS independent) relational schemas, from which it can generate schema specifications for DB2, SYBASE, and INGRES. The difficulty of specifying SYBASE triggers and INGRES rules is illustrated by the amount of code (over three thousand lines) generated by *SDT* for the trigger and rule procedures involved in the definition of relational schemas with thirty relation-schemes.

The concept of referential integrity is still surrounded by confusion, as illustrated by the successive modifications of the original definition of [1] (see [3], [4]). Thus, although it is known that certain referential integrity structures may cause data manipulation problems (see [4]), the nature of these problems has not been explored and conditions for avoiding them have not been formally developed. *Safeness* conditions necessary for avoiding such data manipulation problems are formally developed in [9]. In [9] we have shown that while some DB2 restrictions are more stringent than the safeness conditions, DB2 allows the specification of certain unsafe referential integrity structures.

REFERENCES

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

[2] M.A. Casanova, R. Fagin, and C.H. Papadimitriou, "Inclusion dependencies and their interaction with functional dependencies", *Journal of Computer and System Sciences* 28,1 (Feb. 1984), pp. 29-59.

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

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

[5] S. Even, *Graph Algorithms* , Computer Science Press, 1979.

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

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

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

[9] V.M. Markowitz, "Safe referential integrity structures in relational databases", TR LBL-28363, Dec. 1990.

[10] V.M. Markowitz and W. Fang, "*SDT* 3.1. Reference manual", TR LBL-27843, May 1990.

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

APPENDIX. A GENERIC REFERENTIAL INTEGRITY PROCEDURE

Input: A relational schema $RS = (R, F \cup I \cup N)$, where R, F, I, and N denote sets of relation-schemes, key dependencies, safe referential integrity constraints, and nulls-not-allowed constraints, respectively;

Outline: Procedure ***RefProc***(R_i) is associated with relation-scheme $R_i\,(X_i)$ of R; ***RefProc***(R_i) must be executed whenever a data manipulation (i.e. insertion, deletion, or update) affects a relation r_i associated with R_i.

Notations:

- r_i is the relation currently associated with R_i;
- δ is the data manipulation applied on r_i: $\delta \in \{\mathit{insert}, \mathit{delete}, \mathit{update}\}$;
- K_i is the primary-key associated with R_i;
- FK_{i_j}, FK_i are a foreign-key, respectively the union of all foreign-keys, associated with R_i;
- $To(R_i)$ is the set of referential integrity constraints involving R_i in their right-hand sides: $\{R_k[FK_{k_i}] \subseteq R_i[K_i] \mid R_k[FK_{k_i}] \subseteq R_i[K_i] \in I\}$;
- $From(R_i)$ is the set of referential integrity constraints involving R_i in their left-hand sides: $\{R_i[FK_{i_j}] \subseteq R_j[K_j] \mid R_i[FK_{i_j}] \subseteq R_j[K_j] \in I\}$;
- r_j is the relation currently associated with R_j, where R_j is involved in a referential integrity constraint of $From\ (R_i)$;
- r_k is the relation currently associated with R_k, where R_k is involved in a referential integrity constraint of $To\ (R_i)$;
- $change_i$ is a relation associated with attribute set $X_i X'_i$, where the attributes of X'_i are renamed attributes of X_i; every tuple $\bar{t}$ of $change_i$ consists of the concatenation of two tuples, t and t', where t is an old (existing) tuple of r_i, that is deleted or updated following δ, and t' is a new tuple that is inserted in r_i, or replaces t in r_i

following δ; for insertions $\bar{t}[X_i]$ is null, and for deletions $\bar{t}[X'_i]$ is null;

$refin_{i_k}$ consists of foreign-key values of r_i that do not have references to existing primary-key values in r_j: $refin_{i_k} \triangleq \pi\downarrow_{FK'_{i_k}}(change_i) - \pi\downarrow_{K_j}(r_j)$;

$refdel_{k_i}$ consists of foreign-key values of r_k that reference deleted or updated primary-key values of r_i: $refdel_{k_i} \triangleq \pi\downarrow_{FK_{k_i}}(r_k) \cap (\pi\downarrow_{K_i}(change_i) - \pi\downarrow_{K'_i}(change_i))$.

RefProc (R_i) :

I. 1 *error* := 0;
2. *case* (δ) *of*
a. (*insert*, *update*):
for each $R_i[FK_{i_k}] \subseteq R_j[K_j]$ in $From(R_i)$
having *restricted* insert-rule *do*
if ($refin_{i_k} \neq \emptyset$) *then* *error* := *error*+1 ;
print 'r_i tuples have no references in r_j';
endif
enddo
b. (*delete*):
for each $R_k[FK_{k_i}] \subseteq R_i[K_i]$ in $To(R_i)$
having *restricted* delete-rule *do*
if ($refdel_{k_i} \neq \emptyset$) *then* *error* := *error*+1 ;
print 'r_i tuples are referenced by r_k tuples';
endif
enddo
c. (*update*):
for each $R_k[FK_{k_i}] \subseteq R_i[K_i]$ in $To(R_i)$
having *restricted* update-rule *do*
if ($refdel_{k_i} \neq \emptyset$) *then* *error* := *error*+1 ;
print 'primary-keys in r_i tuples are referenced by r_k tuples';
endif
enddo
endcase
3. *if* (*error* > 0) *then* *revoke* δ *endif*

II. 1 *if* (*error* = 0) *then*
2. *case* (δ) *of*
a. (*delete*):
for each $R_k[FK_{k_i}] \subseteq R_i[K_i]$ in $To(R_i)$
having *nullifies* delete-rule *do*
replace in r_k the tuples of
$\Lambda = \{t \mid t \in r_k, \exists t' \in refdel_{k_i} \text{ s.t. } t[FK_{k_i}] = t'\}$
by $\{\bar{t} \mid \bar{t}[X_k - FK_{k_i}] = t[X_k - FK_{k_i}], \bar{t}[FK_{k_i}] = null,$ where $t \in \Lambda\}$
enddo
for each $R_k[FK_{k_i}] \subseteq R_i[K_i]$ in $To(R_i)$
having *cascades* delete-rule *do*
delete from r_k the tuples of
$\{t \mid t \in r_k, \exists t' \in refdel_{k_i} \text{ s.t. } t[FK_{k_i}] = t'\}$
enddo
b. (*update*):
for each $R_k[FK_{k_i}] \subseteq R_i[K_i]$ in $To(R_i)$
having *nullifies* update-rule *do*
replace in r_k the tuples of
$\Lambda = \{t \mid t \in r_k, \exists t' \in refdel_{k_i} \text{ s.t. } t[FK_{k_i}] = t'\}$
by $\{\bar{t} \mid \bar{t}[X_k - FK_{k_i}] = t[X_k - FK_{k_i}], \bar{t}[FK_{k_i}] = null,$ where $t \in \Lambda\}$.
enddo
for each $R_k[FK_{k_i}] \subseteq R_i[K_i]$ in $To(R_i)$
having *cascades* update-rule *do*
replace in r_k the tuples of
$\Lambda = \{t \mid t \in r_k, \exists t' \in refdel_{k_i} \text{ s.t. } t[FK_{k_i}] = t'\}$
by* $\{\bar{t} \mid \bar{t}[X_k - FK_{k_i}] = t[X_k - FK_{k_i}],$
$\bar{t}[FK_{k_i}] = t_{upd}[K'_i]$, where $\exists (t \in \Lambda$ and $t_{upd} \in change_i)$ s.t. $t[FK_{k_i}] = t_{upd}[K_i]\}$.
enddo
endcase
endif ■

Note: * can be replaced by:
$\{\bar{t} \mid \bar{t}[X_k - FK_{k_i}] = t[X_k - FK_{k_i}], \bar{t}[FK_{k_i}] = t_{new}[K'_i],$
where $\exists (t \in \Lambda, t_{new} \in \pi_{X'_i}(change_i)$ and
$t_{old} \in \pi_{X_i}(change_i))$ s.t. $t[FK_{k_i}] = t_{old}[K_i]\}$
iff $(|refdel_{k_i}| = 0)$ or
$(|refdel_{k_i}| = |\pi_{X_i}(change_i)| = |\pi_{X'_i}(change_i)| = 1)$.

This condition underlies the enforcement of referential integrity constraints associated with *cascades* update-rules in SYBASE, where: $deleted = \pi_{X_i}(change_i)$ and $inserted = \pi_{X'_i}(change_i)$.

DATE FILMED

01/24/92