Lawrence Berkeley National Laboratory
Recent Work

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

Permalink
https://escholarship.org/uc/item/9qd3b9sb

Author
Markowitz, V.M.

Publication Date
1990-12-01
Problems Underlying the Use of Referential Integrity Mechanisms in Relational Database Management Systems

V.M. Markowitz

December 1990
DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. Neither the United States Government nor any agency thereof, nor The Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or The Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or The Regents of the University of California and shall not be used for advertising or product endorsement purposes.

Lawrence Berkeley Laboratory is an equal opportunity employer.
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

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

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.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED
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, at 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 $G=(V,E)$ a directed graph with set of vertices $V$ and set of edges $E$, 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 $v_{i_n}$ is a sequence of alternating vertices and edges, $v_{i_0}, h_{i_1}, v_{i_1}, ..., h_{i_n}, v_{i_n}$, such that $h_{i_k}$ is incident from $v_{i_{k-1}}$ to $v_{i_k}$. 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 $(R, \Delta)$, where $R$ is a set of relation-schemes and $\Delta$ is a set of dependencies over $R$. We consider relational schemas with $\Delta \subseteq F \cup I \cup N$, 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. Each 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 if 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 $\pi_W(r_i)$, and is equal to $\{t[W] | t \in r_i$ 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 $R_i : Y \rightarrow Z$, where $Y$ and $Z$ are subsets of $X_i$; $R_i : Y \rightarrow Z$ 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 $\pi_Y(r_i) \subseteq \pi_Z(r_j)$. 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 $\Delta$ involving referential integrity constraints. The referential integrity (directed) graph associated with $RS$, $G_i=(V,E)$, is defined as follows: $V=R$, and $E \subseteq \{(r_i \rightarrow r_j) | R_i[Y] \subseteq R_j[Z] \in \Delta\}$. The set of referential integrity constraints of $RS$ is said to be cyclic (resp. acyclic) if $G_i$ has (resp. does not have) directed cycles.

A referential integrity constraint $R_i[Y] \subseteq R_j[Z]$ 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)
of $t'$ implies deleting (resp. updating the subtree $t[Y]$ 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 subtree $t[Y]$ 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 $Y \subseteq Z$, where $Y$ and $Z$ are subsets of $X_i$; $R_i: Y \subseteq Z$ is satisfied by $r_i$ if 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 $R_i: \emptyset \subseteq Z$; $R_i: \emptyset \subseteq Z$ is satisfied by $r_i$ iff for every tuple $t$ of $r_i$, the subtree $t[Z]$ 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 $\delta$ 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 $\delta$ if the relation that would result by applying $\delta$ 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 $\text{RefProc}$ is given in the appendix. $\text{RefProc}$ assumes that for every relation $r_i$ there exists a relation called $\text{change}_i$, that records how a given data manipulation $\delta$ would affect the tuples of $r_i$, without actually applying $\delta$ on $r_i$. Every tuple $\bar{t}$ of $\text{change}_i$ 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 $\text{RefProc}$ 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 @nullEMPLOYEE = count(*) from inserted, EMPLOYEE
    where inserted.M_SSN = EMPLOYEE.M_SSN
  if @nullPROJECT + @nullPROJECT = @insEMPLOYEE
    != 2 * @row
    print "Failed insertion into MANAGER because of"
    if @nullPROJECT + @nullPROJECT != @row
      print "Inserting reference to PROJECT"
    if @nullEMPLOYEE + @nullEMPLOYEE != @row
      print "Missing reference to EMPLOYEE"
  end
end
```

```
create trigger deleteMANAGER on MANAGER for delete as
begin
  declare @deEMPLOYEE int
  select @deEMPLOYEE = count(*) from deleted, EMPLOYEE
    where deleted.M_SSN = EMPLOYEE.M_SSN
  if @deEMPLOYEE > 0
    begin
      print "Failed deletion from MANAGER because of"
      if @deEMPLOYEE != 0
        print "Existing reference from EMPLOYEE"
    end
end
```

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

Figure 2. SYBASE Trigger Examples.
-insert trigger procedures correspond to \textit{RefProc} \{ I(1, 2.a, 3); delete trigger procedures correspond to \textit{RefProc} \{ I(1, 2.b, 3), II(1, 2.a); and update trigger procedures correspond \textit{RefProc} \{ I(1, 2.a, 2.c, 3), II(1, 2.b); \\
-relations \textit{deleted} and \textit{inserted} replace relation change; \\
-as follows: \textit{deleted} is the projection of \textit{change} that includes existing tuples of \textit{r}\textsubscript{i} that are affected by the data manipulation under consideration; and \textit{inserted} is the projection of \textit{change}; that includes new tuples that are going to be inserted into \textit{r}\textsubscript{i} following the data manipulation under consideration; \\
-the relational algebra expressions in the definition of \textit{RefProc} are translated into SQL expressions.

Trigger procedures are specified in SYBASE’s \textit{Transact-SQL} which allows in addition to the standard SQL the specification of control-flow statements. For example, the \textit{insert} and \textit{delete} trigger procedures for relation-scheme \textit{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 \textit{t} in a relation \textit{r}\textsubscript{i} leads (cascades) to the deletion of another tuple, \textit{t}' in \textit{r}\textsubscript{i}, then the delete trigger associated with \textit{r}\textsubscript{i} is not activated by the deletion of \textit{t}’;
3. the employment of the system provided relations \textit{inserted} and \textit{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 \textit{cascades} delete-rules cannot be correctly specified in SYBASE. Restriction (3) above means that a \textit{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 \textit{inserted} and \textit{deleted} relations consist of at most one referenced tuple (see the note in the appendix). Finally, \textit{Transact-SQL} includes an operation called \textit{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 \textit{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 \textit{inserted} and \textit{deleted} relations provided by SYBASE are replaced in INGRES by two tuples, called \textit{new} and \textit{old}: following a data manipulation involving a relation \textit{r}\textsubscript{i}, the \textit{old} tuple contains the \textit{r}\textsubscript{i} tuple that is going to be deleted or updated, and the \textit{new} tuple is the tuple that is going to be inserted into \textit{r}\textsubscript{i}, or the

\begin{verbatim}
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;
    end;
  end;

  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;
    end;
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;
  end;
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.
\end{verbatim}
newly updated tuple of \( r_j \). 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 \textit{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 \textit{Transaction-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.

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

Example 1. Suppose that in the relational database of figure 1(ii) data manipulation \( \delta \) 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 \textit{cascades} update-rules (as they actually are in the schema of figure 1(i)) then \( \delta \) 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 \textit{restricted} update-rules then \( \delta \) cannot be executed.

Allowing only \textit{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 \textit{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 \textit{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 \textit{cascades} delete-rules, and \( I_4 \) associated with a \textit{restricted} delete-rule. Let deletion \( \delta \) involve tuple (a) of relation \( r_3 \). The outcome of \( \delta \) 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 \( I_4 \); or (ii) if \( I_4 \) is enforced first then \( \delta \) 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 \textit{restricted} delete-rule. Let deletion \( \delta \) involve tuples (2 \( - - b \)) and (3 \( 2 - b \)) of relation \( r_1 \). The outcome of \( \delta \) depends on the order in which the tuples involved in \( \delta \) are accessed: (i) if (3 \( 2 - b \)) is accessed first then both

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.
tuples involved in \( \delta \) are deleted; or (ii) if \((2-b)\) is accessed first then \( \delta \) 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 1.** 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 \( G_1=(R, H) \) be the referential integrity graph associated with \( RS \). Given a relation-scheme \( R_i \) of \( R \), sets \( \text{Casc}(R_i) \) and \( \text{Null}(R_i) \) 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 \( \text{Restr}(R_i) \) defined below consists of the relation-schemes whose relations may contain tuples that can block the deletion of tuples in a relation associated with \( R_i \):

- \( \text{Casc}(R_i) \) is the subset of \( R \) consisting of \( R_i \) and the relation-schemes that are connected in \( G_1 \) to \( R_i \) by a directed path consisting of edges that correspond to referential integrity constraints associated with \textit{cascades} delete-rules;
- \( \text{Null}(R_i) \) is the subset of \( R \) consisting of relation-schemes \( R_j \), where \( R_j \) is connected in \( G_1 \) to a relation-scheme of \( \text{Casc}(R_i) \) by an edge that corresponds to a referential integrity constraint associated with a \textit{nullifies} delete-rule;
- \( \text{Restr}(R_i) \) is the subset of \( R \) consisting of relation-schemes \( R_j \), where \( R_j \) is connected in \( G_1 \) to a relation-scheme of \( \text{Casc}(R_i) \) by an edge that corresponds to a referential integrity constraint associated with a \textit{restricted} delete-rule;
- \( \text{Null}'(R_i) \) is the subset of \( \text{Null}(R_i) \) consisting of relation-schemes \( R_j \), where \( R_j \) is connected in \( G_1 \) to relation-schemes of \( \text{Casc}(R_i) \) by at least two edges corresponding to referential integrity constraints associated with \textit{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 \( \text{Restr}(R_i) \), \( \text{Null}(R_i) \), and \( \{\text{Casc}(R_i)\} \cup \{\text{Null}(R_i)\} \) are pairwise disjoint, and set \( \text{Null}'(R_i) \) is empty.

**T2:** For every subset \( I' \) of \( I \) that consists of referential integrity constraints corresponding to edges forming a directed cycle in \( G_1 \); if \( I' \) consists of a single constraint then this constraint must be associated with a \textit{cascades} delete-rule; otherwise at least two constraints of \( I' \) must be associated with \textit{restricted} or \textit{nullifies} delete-rules. \( \blacksquare \)

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 \textit{nullifies} delete-rules, and \( I_4 \) is associated with a \textit{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 \textit{nullifies} delete-rules, and \( I_4 \) associated with a \textit{cascades} delete-rule. Suppose also that relation-scheme \( R_1 \) is associated with null constraint \((N_1)\) \( R_1; \text{M}\_\text{SSN} \leq \text{P}\_\text{NR} \). Let deletion \( \delta \) involve tuple \((a)\) of relation \( r_3 \). Note that without \( N_1 \) \( \delta \) would imply nullifying (via \( I_3 \)) the \text{P}\_\text{NR} values in tuples \((1 4 4 a)\) and \((4-\text{-}a)\) of relation \( r_1 \), and nullifying (via \( I_4 \) and \( I_5 \)) the \text{M}\_\text{SSN} value in tuple \((1 4 4 a)\) of relation \( r_1 \). However, \( N_1 \) makes the outcome of \( \delta \) depend on the order in which \( I_3, I_4, \) and \( I_5 \) are enforced: (i) if \( I_4 \) is enforced first then tuple \((4 \text{-}a)\) is deleted from \( r_2 \), thus leading to the nullification of the \text{M}\_\text{SSN} value in tuple \((1 4 4 a)\) of \( r_1 \) while enforcing \( I_5 \); the subsequent enforcement of \( I_5 \) results in nullifying the \text{P}\_\text{NR} values in tuples \((1 4 -a)\) and \((4-\text{-}a)\) of \( r_1 \); or (ii) if \( I_5 \) is enforced first then \( \delta \) is blocked by tuple \((1 4 4 a)\) of \( r_1 \), where the \text{P}\_\text{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 \textit{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 \textit{nullifies} delete-rule, \( I_4 \) is associated with a \textit{cascades} delete-rule, and \( I_5 \) associated with a \textit{restricted} delete-rule. If relation-scheme \( R_1 \) is associated with null constraint \( R_1; \text{P}\_\text{NR} \leq \text{M}\_\text{SSN} \) then condition \( T1 \) is not satisfied. However, the outcome of deletions does not
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:

$DM : \text{DELETE FROM EMPLOYEE WHERE S_SSN IS NULL}$

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:

and $\text{SUPERVISE (S_SSN, S_SSN)}$;

(b) $\text{SUPERVISE}$ is associated with null constraint $\text{\emptyset \not\subseteq E_SSN, S_SSN}$;

(c) $\text{SUPERVISE}$ is involved in two referential integrity constraints associated with cascades delete-rules:

$\text{SUPERVISE \{S_SSN\} \subseteq EMPLOYEE \{E_SSN\}}$ and $\text{SUPERVISE \{E_SSN\} \subseteq EMPLOYEE \{E_SSN\}}$.

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 $DM$:

$\text{DM': DELETE FROM EMPLOYEE WHERE E_SSN NOT IN (SELECT E_SSN FROM SUPERVISE)}$

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


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 relationschemas, key dependencies, safe referential integrity constraints, and nulls-not-allowed constraints, respectively.

Outline: Procedure \( \text{RefProc}(R_i) \) is associated with relation-scheme \( R_i \); \( \text{RefProc}(R_i) \) 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 \( R_i \);

\( \delta \) is the data manipulation applied on \( r_i \):

\( \delta \in \{ \text{insert}, \text{delete}, \text{update} \} \);

\( K_i \) is the primary-key associated with \( R_i \);

\( FK_i,FK_j \) are a foreign-key, respectively the union of all foreign-keys, associated with \( R_i \);

\( To(R_j) \) is the set of referential integrity constraints involving \( R_j \) in their right-hand sides:

\[ \{ R_k[FK_k] \subseteq R_j[K_j] \mid R_k[FK_k] \subseteq R_i[K_i] \in I \} \]

\( From(R_j) \) is the set of referential integrity constraints involving \( R_j \) in their left-hand sides:

\[ \{ R_i[FK_i] \subseteq R_j[K_j] \mid R_i[FK_i] \subseteq R_j[K_j] \in I \} \]

\( r_i \) is the relation currently associated with \( R_i \), where \( R_i \) is involved in a referential integrity constraint of \( \text{From}(R_j) \);

\( r_k \) is the relation currently associated with \( R_k \), where \( R_k \) is involved in a referential integrity constraint of \( \text{To}(R_j) \);

\( change_i \) is a relation associated with attribute set \( X,X' \), where the attributes of \( X' \) are renamed attributes of \( X \); every tuple \( t \) 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 \( \delta \), and \( t' \) is a new tuple that is inserted in \( r_i \), or replaces \( t \) in \( r_i \).
following δ; for insertions \( \tilde{r}(X_i) \) is null, and for deletions \( \tilde{r}(X_i) \) is null;

\( \text{refin}_k \) consists of foreign-key values of \( r_i \) that do not have references to existing primary-key values in \( r_j \); \( \text{refin}_k = \pi_{FK_k}(\text{change}_i) - \pi_{FK}(r_i) \);

\( \text{refdel}_k \) consists of foreign-key values of \( r_i \) that reference deleted or updated primary-key values of \( r_j \); \( \text{refdel}_k = \pi_{FK_k}(r_i) \cap (\pi_{FK}(\text{change}_i) - \pi_{FK}(r_i)) \).

\( \text{RefProc}(R_i) \):

1. Let \( \text{error} := 0; \)

2. Case (\( \delta \)) of

   (a) (insert, update):
   
   (i) for each \( R_i[FK_1] \subseteq R_j[K_j] \) in \( \text{From}(R_i) \)
   
   (ii) having restricted insert-rule do
   
   if \( \text{refin}_k \neq \emptyset \) then \( \text{error} := \text{error} + 1; \)
   
   print \( r_i \) tuples have no references in \( r_j \);
   
   endif
   
   enddo

   (b) (delete):
   
   (i) for each \( R_k[FK_k] \subseteq R_j[K_j] \) in \( \text{To}(R_i) \)
   
   (ii) having restricted delete-rule do
   
   if \( \text{refdel}_k \neq \emptyset \) then \( \text{error} := \text{error} + 1; \)
   
   print \( r_i \) tuples are referenced by \( r_k \) tuples;
   
   endif
   
   enddo

   (c) (update):
   
   (i) for each \( R_k[FK_k] \subseteq R_j[K_j] \) in \( \text{To}(R_i) \)
   
   (ii) having restricted update-rule do
   
   if \( \text{refdel}_k \neq \emptyset \) then \( \text{error} := \text{error} + 1; \)
   
   print \( \text{primary-keys in } r_i \) tuples are referenced by \( r_k \) tuples;
   
   endif
   
   enddo

endcase

3. If \( \text{error} > 0 \) then revoke \( \delta \) endif

II.1 If \( \text{error} = 0 \) then

2. Case (\( \delta \)) of

   (a) (delete):
   
   (i) for each \( R_i[FK_1] \subseteq R_j[K_j] \) in \( \text{To}(R_i) \)
   
   (ii) having nullifies delete-rule do

   replace in \( r_k \) the tuples of

   \( \Lambda = \{ t \mid \exists t' \in \text{refdel}_k \text{ s.t. } t[FK_k] = t' \} \)

   by \( \{ \tilde{r}[X_k - FK_k] = t[X_k - FK_k], \tilde{r}[FK_k] = \text{null}, \text{null} \} \) where \( t \in \Lambda \)

   enddo

   for each \( R_k[FK_k] \subseteq R_j[K_j] \) in \( \text{To}(R_i) \)
   
   (iii) having cascades delete-rule do
   
   delete from \( r_k \) the tuples of

   \( \{ t \mid r_k \exists t' \in \text{refdel}_k \text{ s.t. } t[FK_k] = t' \} \)

   enddo

   (b) (update):
   
   (i) for each \( R_k[FK_k] \subseteq R_j[K_j] \) in \( \text{To}(R_i) \)
   
   (ii) having nullifies update-rule do

   replace in \( r_k \) the tuples of

   \( \Lambda = \{ t \mid \exists t' \in \text{refdel}_k \text{ s.t. } t[FK_k] = t' \} \)

   by \( \{ \tilde{r}[X_k - FK_k] = t[X_k - FK_k], \tilde{r}[FK_k] = \text{null}, \text{null} \} \) where \( t \in \Lambda \)

   enddo

   for each \( R_k[FK_k] \subseteq R_j[K_j] \) in \( \text{To}(R_i) \)
   
   (iii) having cascades update-rule do

   replace in \( r_k \) the tuples of

   \( \Lambda = \{ t \mid \exists t' \in \text{refdel}_k \text{ s.t. } t[FK_k] = t' \} \)

   by \( \{ \tilde{r}[X_k - FK_k] = t[X_k - FK_k], \tilde{r}[FK_k] = \text{null}, \text{null} \} \) where \( t \in \Lambda \)

   enddo

   endcase

endif

Note: * can be replaced by:

\[ \{ \tilde{r}[X_k - FK_k] = t[X_k - FK_k], \tilde{r}[FK_k] = \text{null}, \text{null} \} \]

where \( \exists (t \in \Lambda, \text{null} \in \pi_{FK}(\text{change}_i)) \) and \( \text{null} \in \pi_{FK}(\text{change}_i) \text{ s.t. } t[FK_k] = \text{null} \).

iff \( |\text{refdel}_k| = 0 \) or

\( |\text{refdel}_k| = 1 \neq |\pi_{FK}(\text{change}_i)| = 1 \neq |\pi_{FK}(\text{change}_i)| = 1 \).

This condition underlies the enforcement of referential integrity constraints associated with cascades update-rules in SYBASE, where: deleted = \( \pi_X(\text{change}_i) \) and inserted = \( \pi_X(\text{change}_i) \).