A new look at existence dependency in databases

by T. C. CHIANG*

American Bell Inc.
Piscataway, New Jersey

ABSTRACT

To ensure data consistency, existence dependencies between records must be preserved when the records are updated. It is necessary to identify such dependencies when the system is designed, so that update operations can be performed correctly.

This paper presents a new look at existence dependencies in databases. It identifies a set of basic update rules that can be incorporated into a database management system (DBMS) to preserve existence dependencies between records. The paper shows that the existence dependencies supported by other existing commercially available DBMSs can be defined precisely in terms of the basic rules. Furthermore, a combination of these basic rules captures new existence dependency semantics that are not handled by the commercially available systems. This paper also discusses solutions to the “interference” problems associated with closely related records. These interference problems have never been discussed in the literature before, but are important to system operations.

*This paper was written while T. C. Chiang was working at Bell Laboratories, Whippany, New Jersey.
A. INTRODUCTION

In an enterprise that has complicated relationships among entities, the existence of one entity often depends on that of others. Since a database represents a real world, existence dependencies in that world are reflected on the database as existence dependencies among data records. For an application that requires frequent deletion and insertion of records, existence dependency in the database must be handled correctly to avoid data inconsistency.

A reasonable place in a database system to handle existence dependency is in the database management system (DBMS) (as opposed to in the user’s programs). Most of the commercial DBMSs such as IBM/IMS or other CODASYL-based systems handle a few kinds of existence dependency. For example, in IMS, a deletion of a root segment instance will trigger the deletions of all child segments instances under the root segment. In CODASYL based systems, there are AUTOMATIC and MANUAL, and MANDATORY and OPTIONAL declarations for memberships in a CODASYL set, which define certain kinds of dependency among records. For example, if a member is inserted into a set, it cannot be deleted independent of the set. These mechanisms are not rich enough to capture other kinds of existence dependency in most databases.

In the research community, little attention has been given to the theory of existence dependency. For example, in the early work on relational data model, much attention was given to functional dependency and normalization. The term update anomaly was used to define all the problems related to existence dependency. However, in reality, many of the cases of anomaly are not really anomalous but are expected existence dependency and should be handled as such. In the recent years, some attention has been paid to the problems of existence dependency. 1,5,7-9 Chen defined the concepts of regular and weak entities to capture some semantics of existence dependency among entities. 1 In a later paper, Dogac and Chen mentioned the concepts of update propagation.2 Similarly, Smith and Smith talked about the ideas of triggered updates that involve automatic updating of dependent records.3 Keller and Wiederhold listed a number of existence dependencies that were considered important in keeping the database consistent.4 None of these papers treated the problems of existence dependency extensively enough to capture most of the dependency semantics. Chiang and Bergeron described a system that handles a set of existence dependencies.5 However, no detail on other problems related to existence dependency has been presented. Navathe and Schkolnick 7 talked about update rules in the framework of view representation, and Date defined a rich language to describe existence dependencies as referential integrity for the relational data model. 6 However, neither of these papers broke down the existence dependency semantics into atomic units or mentioned the interference problems.

In this paper, existence dependency and its related problems are intensively discussed. Existence dependency is viewed as a property of a relationship among entities. A coupling factor is defined as a set of update rules for handling existence dependencies between entities. This paper shows that update rules supported by the commercial systems (e.g., IMS and CODASYL systems) and by others can be described in terms of the coupling factors. 2,5 Furthermore, the new look of the existence dependency problems enables us to discover new dependency semantics. A DBMS will enforce the rules for handling existence dependency, so that record deletions and insertions will not turn the database into an inconsistent state. Problems arise when two relationships exist within the same set of entities and the existence dependencies of the relationships interfere with each other. Later sections of this paper will define the sets of various kinds of existence dependency and of their interferences, and will present an algorithm to detect the interferences.

The result described in this paper is actually implemented in a DBMS for telephone business applications in the Bell telephone companies over the United States.5

B. EXISTENCE DEPENDENCY

An extended entity-relationship (E-R) data model is used as a basis for dealing with existence dependency. Roughly speaking, a basic E-R model views a database consisting of files that are sets of records and relationships among records.1 In the extended E-R model, a set of relationships is referred to as an association.2 An association has a coupling factor as one of its properties. A coupling factor is a set of existence dependencies. To simplify the representation, only the existence dependencies of binary associations are discussed in the rest of this paper. Therefore, the term association will mean binary association from now on. Also, an association will be viewed as a binary relation with the two files involved as its domains.

Although there may exist many kinds of existence dependency, four basic ones are identified; the definitions are given as follows. Let E1 and E2 be two files, and A an association between E1 and E2. Let e1 be any record in E1 and e2 a record in E2 type associated with e1 via A; then e1 and e2 are referred to as A-associated. The basic existence dependencies can be defined as the following update rules:

1. An e2 cannot be inserted unless there already exists an
A-associated $e_1$. An insertion of $e_2$ implies establishing a relationship between $e_1$ and $e_2$.

2. The deletion of an $e_1$ implies the deletions of all $A$-associated $e_2$s.

3. An $e_2$ cannot be deleted, if there exist an $A$-associated $e_1$.

4. A relationship cannot be deleted, unless $e_1$ or $e_2$ is deleted.

Note that normally deletion of $e_1$ or $e_2$ implies the deletion of the relationship between $e_1$ and $e_2$. The insertion of a relationship is allowed only if both $e_1$ and $e_2$ exist. Also, the dependencies specified by the rules are directional and transitive. For example, in the descriptions of the rules above, $E_2$ depends on $E_1$. Furthermore, if $E_2$ depends on $E_1$ and $E_1$ depends on $E_3$, it implies that $E_2$ depends on $E_3$.

A set of coupling factors can be defined as the power set of the basic rules. Thus, there are 16 possible coupling factors, including the null coupling factor that has none of the above dependencies. By Chiang and Bergeron, a “very tight” coupling factor is defined as one that has all four dependencies. The other coupling factors represent various kinds of dependencies between records. The coupling factors are unidirectional, that is, all the rules of the coupling factor have the same direction, which is defined as the direction of the coupling factor. Coupling factors are also transitive. The transitive property of coupling factors is the key to the propagation of updates.

With these definitions of coupling factor, we can show how update rules supported by others can be described in terms of coupling factors.

### B.1 IMS

The update rules for IMS are defined by its hierarchical structure. Therefore, the deletion of one node in the hierarchy implies the deletion of all its child nodes. An insertion of a node into the hierarchy requires the existence of its parent node.

The IMS update rules can be viewed as the coupling factor consisting of rule 1, rule 2, and rule 4 above. For example, consider a subset of the education database presented by Date. Figure 1 shows the structure for the educational database.

A student segment can be inserted only if there is an offering of a course (i.e., rule 1), and the deletion of an offering of a course implies the deletion of all student segments associated with the offering (i.e., rule 2). Furthermore, IMS does not allow the breaking up of hierarchical relationships (e.g., rule 4). Note that IMS does not support rule 3.

### B.2 CODASYL Systems

In CODASYL systems, membership class in a set is used to describe the update rules. Let Owner ($O$) and Member ($M$) be two record types, and $OM$ be the set defined between $O$ and $M$. The membership for $m$ of $M$ in $OM$ is said to be mandatory, if $m$ cannot be removed from the set and deletion of an owner $o$ implies the deletion of all members. If the membership is optional, $m$ can be removed from $OM$ without deleting $m$ from the database. If the membership of $M$ in $OM$ is automatic, the DBMS will automatically establish the membership in $OM$ when an $m$ of $M$ is inserted. If the membership is manual, then the programmer has to establish the membership explicitly. There are four kinds of memberships: (1) mandatory-automatic, (2) optional-automatic, (3) mandatory-manual, and (4) optional-manual. We can use coupling factors to describe these memberships precisely. Mandatory-automatic is the coupling factor containing rules 2, 4, and 1; optional-automatic is the coupling factor containing rules 2 and 1; mandatory-manual is the coupling factor containing rules 2 and 4; and optional-manual is rule 2.

### B.3 Relational Systems

In relational systems, there are no comparable update rules. Existence dependency is treated as anomaly in the context of normalization. For example, consider a supplier relation:

$$S' (S\#, \text{ SNAME}, \text{ STATUS}, \text{ CITY})$$

$S'$ is in 2nd normal form, because all non-key attributes depend on the key and STATUS depends on CITY. This implies that information about STATUS cannot be inserted until some supplier in the city is in the database, and deletion of the only supplier in a CITY will delete all STATUS and CITY information. These are considered to be anomalies by Date. However, they may be valid existence dependencies for some applications. The coupling factor in this case contains rule 1 and the rule to be described in Section D.

To show the problems in dealing with existence dependency in a relational system, let us consider another example...
Consider a database consisting of three relations: (1) suppliers (SUPP), (2) parts (PART), and (3) shipments (SHIP). Each of the relations has a set of attributes and a set of tuples as follows:

**SUPP (S#, SNAME, STATUS, CITY)**

<table>
<thead>
<tr>
<th>S#</th>
<th>SNAME</th>
<th>STATUS</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>adams</td>
<td>20</td>
<td>Chicago</td>
</tr>
<tr>
<td>s2</td>
<td>baker</td>
<td>10</td>
<td>Newark</td>
</tr>
<tr>
<td>s3</td>
<td>chang</td>
<td>30</td>
<td>L.A.</td>
</tr>
</tbody>
</table>

**PART (P#, PNAME, COLOR)**

<table>
<thead>
<tr>
<th>P#</th>
<th>PNAME</th>
<th>COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>chip</td>
<td>red</td>
</tr>
<tr>
<td>p2</td>
<td>LED</td>
<td>green</td>
</tr>
<tr>
<td>p3</td>
<td>fan</td>
<td>red</td>
</tr>
</tbody>
</table>

**SHIP (S#, P#, QTY)**

<table>
<thead>
<tr>
<th>S#</th>
<th>P#</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>p1</td>
<td>10</td>
</tr>
<tr>
<td>s1</td>
<td>p3</td>
<td>5</td>
</tr>
<tr>
<td>s2</td>
<td>p2</td>
<td>20</td>
</tr>
<tr>
<td>s2</td>
<td>p3</td>
<td>8</td>
</tr>
</tbody>
</table>

Suppose we want to delete all red parts from the database (from both PART and SHIP). Using a pseudorelational language, we can express the deletion operations as follows:

1. delete SHIP tuples, where SHIP.p# = PART.p#, and
2. delete PART tuples, where PART.color = "red".

Now, if we reversed the order to do step 2 first, then step 1, the deletion of the SHIP tuples would not be possible, because the information about red parts would not be available at that point. Currently, there is no relational system to prevent deletions in a wrong order. If we consider SHIP as an association between PART and SUPP, then the deletion of red parts will trigger the deletions of the SHIP relationship tuples.

We have shown that the idea of coupling factor can be used to describe the existence dependency supported by some existing systems and to capture other dependency semantics that are not recognized by the existing systems. Coupling factors have many other interesting properties; one of them will be discussed in the next section. The problems discussed in the next section have never been handled by the existing systems.

### C. INTERFERENCES

An interference occurs when two adjacent associations have coupling factors that interfere with each other. For example, suppose that we have a database consisting of a department file, an employee file, and a project file, and that the update rules for the database are as follows:

1. An employee record cannot be inserted, if there is no department that the employee can work for
2. A project record cannot be inserted if there is no employee in department that can handle the project
3. A department record cannot be inserted, if there is no project for it.

If these update rules are enforced, then no record can be inserted into the database. In a system that supports more complex relationships between records, the interference between two coupling factors becomes a problem. In this section, we shall talk about this kind of interference. We shall use the first three coupling factors that were presented in Section B to illustrate the problem. More detailed definitions are given as follows.

**Adjacency.**—An association A is said to be adjacent to another association B, and vice versa, if and only if A and B have a domain in common.

**Parallel adjacency.**—Associations A and B are said to be parallel adjacent if and only if they have the same set of two perhaps distinct domains.

**Serial adjacency.**—Associations A and B are said to be serial adjacent if and only if A and B have exactly one domain in common.

**Interference state.**—Two adjacent associations are said to be in an interference state if and only if no record can be inserted in or deleted from any of their domains.

**Interfering coupling factors.**—Two coupling factors are said to be interfering with each other if and only if assigning them to two adjacent associations would cause an interference state.

Now we are ready to look at all possible interference states generated by pairings of the eight possible coupling factors.

### C.1 Parallel Interferences

Parallel interference occurs when two parallel adjacent associations have interfering coupling factors. It can occur only when two coupling factors have opposite directions. Therefore, all theorems in this section assume that the coupling factors are in opposite directions. e1 and e2 are used to represent records in the two domains, E1 and E2, of an association. The rules are to be assigned to two distinct associations.

**Theorem 1.** Rule 1 interferes with rule 1.

*Proof.* Since the insertion of e1 depends on the existence of e2, and vice-versa, the associations would be in an interference state.

**Theorem 2.** Rule 3 interferes with rule 3.

*Proof.* Since e1 cannot be deleted without first deleting e2, and vice-versa, the association would be in an interference state.
Clearly two coupling factors are interfering with each other if they contain one interfering rule.

To identify all the possible interference states, we use three Boolean variables, $X$, $Y$, and $Z$, to represent the presence of rules 3, 2, and 1, respectively, in a coupling factor. A Boolean variable has the value 1, if a rule is present. Thus, the eight coupling factors can be represented as

$$XYZ$$

$$\begin{array}{cccccccc}
000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
000 & 001 & x & x & x & x & x & x \\
010 & x & x & x & x & x & x & x \\
011 & x & x & x & x & x & x & x \\
100 & x & x & x & x & x & x & x \\
101 & x & x & x & x & x & x & x \\
110 & x & x & x & x & x & x & x \\
111 & x & x & x & x & x & x & x \\
\end{array}$$

A matrix that represents all the possible states is shown in Figure 2, where an $x$ indicates an interference state. A null coupling factor, which contains none of the rules, does not interfere with any other coupling factors. From Figure 2, it can be seen that there are in total 28 interference states, which are easy to detect: one performs an intersection on the coupling factors involved; if the resulting set contains either rule 1 or rule 3, there is an interference. By considering each coupling factor to be represented by three Boolean variables, one can even set up a hardware machine to realize the Boolean function represented by the matrix in Figure 2.

C.2 Circuit Interference

The concept of parallel interference can be generalized to produce concepts of other interferences. Defining an E-R diagram with respect to existence dependency as a labeled directed graph, we use nodes to represent files and edges to represent associations. A label on an edge represents the association name (which uniquely identifies the edge), and the coupling factor. Figure 3 shows an example of an E-R diagram conveying dependency information, where E1 and E2 are nodes representing files, A is an association, and C is the coupling factor of A. The arrow on an edge represents the direction of the coupling factor. Since the arrow is pointing from E1 to E2, E1 depends on E2 via A/C. E1 is referred to as the initial node and E2 the final node.

A path in an E-R diagram is a sequence of serial adjacent edges, where the final node of one edge is the initial node of another. Figure 4 shows an example of a path, where A1, A2, ..., An are names of the associations. A circuit is a path A1, A2, ..., An in which the initial node of A1 is the final node of An.

A circuit interference is defined as an interference-state caused by the interfering coupling factors of a circuit of associations. With the algorithm in Section 1 and the transitive property of the dependency rules, we can reduce the problems of circuit interference to those of parallel interference. For example, Figure 5 shows a circuit interference.

To detect such a state, one could

1. Detect a circuit.
2. Starting at a node of the circuit, reduce two serial adjacent edges and the common node to a "virtual" edge, $V$, with a coupling factor that is the intersection of the two original associations.
3. Repeat step 2 until two parallel adjacent associations result.
4. Check parallel interference using the algorithm in Section C.1.

D. FUTURE WORK

There are many interesting problems related to existence dependency.

1. To identify more existence dependence rules. For example, a variation of rule 2 can be stated as follows: The deletion of an $e_1$ implies the deletion of all A-associated $e_2$s, if $e_1$ is the only A-associated $e_1$ of the $e_2$s. This rule captures new semantics of dependency and will generate more interference states when it is paired with other rules.
2. To extend the definitions of coupling factors to $n$-ary associations.
3. To apply the concept of coupling factor to update dependency in general. Existence dependency is one kind of update dependency. The other kind is modification dependency. Modification dependency occurs when the modification of a data item depends on the modification or existence of other data items. For example, a deletion of an employee record may trigger the modification of the total number of employees in a company.

D. CONCLUSION

We have considered existence dependency as a set of data semantics that has the same importance as that of functional dependency has in relational theory. We have described the problems of existence dependency in databases. We have identified a set of existence dependency rules from which a coupling factor of an association can be defined. We have described some interesting properties of the coupling factors and identified many interference states and the algorithms to detect them. We believe that there are still many interesting problems in existence dependency left to be discovered.

REFERENCES
