Updating defined relations

by I. M. OSMAN
University of Khartoum
Khartoum, Sudan

INTRODUCTION

Some relational data base systems such as PRTV and SEQUEL allow the user to create his own relations (logical files) as subsets of the main data. With such a facility the user may express his own view of the data in terms of the relations he creates. If a large number of such relations are created, disc storage space problems and other maintenance difficulties can arise.

A relation which is expressed in terms of other relations is called a defined (derived or implied) relation or view. The user submits to the system a definition of these relations expressed in terms of data base relations using relational operators. For example, consider relations MALE, FEMALE whose domains (fields) are name, age and income. The user may define the relation PEOPLE as the union of relations MALE and FEMALE. He may also define the relation OLD as a subset of relation PEOPLE where (age > 50).

The data base system decodes the definition and stores it in a retrievable form. The defined relation is then available to the user at the same logical level as the other data base relations. The defined relation remains merely a stored definition (i.e. implicit) until it is requested by a query. The implicit form takes negligible disc space.

When the defined relation in its implicit form is referenced by a query, e.g. list the people whose income is greater than 5,000, it is then created, i.e. made explicit, by carrying out on the stored data the operations indicated in the definition. For example, to create relation OLD relation PEOPLE is assembled first. The tuples (records) of relation PEOPLE are then accessed and those matching the selection criterion are written back to disc as tuples of the relation OLD.

The explicit forms of relations PEOPLE and OLD may stay in the data base and they will not have to be recreated if they are requested by other queries. In this way defined relations may substantially improve the response time of recurring queries.

We may consider the data base storage space as made up of two parts. One part is for storing the base relations (e.g. MALE and FEMALE) and another part whose content changes dynamically with the needs of the users. The defined relations facility enables the system to make efficient use of the dynamic storage space. At some stage in the process of creating and querying relations the dynamic storage space may be consumed. One or more explicit relations will then have to be made implicit in order to free space for other requested relations. The relations in the dynamic storage switch between being explicit and implicit. The replacement algorithm which picks the relations to be made implicit has been discussed in a previous paper.

Notation

In this paper the set notation is adopted for relations. This is explained in the following table:

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>union</td>
</tr>
<tr>
<td>∩</td>
<td>intersection</td>
</tr>
<tr>
<td>(filter)</td>
<td>selection, the filter is a boolean expression</td>
</tr>
<tr>
<td>% (list)</td>
<td>projection, the list is a string of selectors</td>
</tr>
<tr>
<td>*</td>
<td>join (cartesian product or equijoin)</td>
</tr>
<tr>
<td>-</td>
<td>difference</td>
</tr>
<tr>
<td>←</td>
<td>is defined as</td>
</tr>
<tr>
<td>: =</td>
<td>assigned to</td>
</tr>
</tbody>
</table>

e.g. the definitions of relations PEOPLE and OLD are expressed as follows:

PEOPLE ← MALE U FEMALE
OLD ← PEOPLE: (age > 50)

The hierarchy of defined relations

In Figure 1 a data base having a hierarchy of defined relations is shown. In such a hierarchy the user may not be aware of the relationships and the dependencies that exist among the data base relations. It is therefore advantageous to provide the facilities by means of which the user can perform all the operations on a defined relation without restrictions. Ideally, the user will treat the defined relations in exactly the same way as any other base relations, without restrictions forcing him to be aware of the dependencies between relations. That is, the system should possess a higher degree of data independence. It is therefore desirable to have the user activities (queries, application programs,
Figure 1—Supply data base. In these hierarchies of defined relations, OLDST, SUPPLIER, OLDSP and SUPPLY are base relations. SC is defined on SUPPLIER:
FULLST is defined on SC and OLDST.
Definitions: SC ← SUPPLIER % (City, Status)
FULLST ← OLDEST ∩ SC
NEWSP ← OLDSP ∩ SUPPLY

updates, etc.) independent of the logical representation, the access path and the level in the hierarchy of the data. The user need not know where a relation is a base or a defined relation, is implicit or explicit, nor will he need to know whether it is stored as a file or a mere collection of pointers to some other files. However, giving all this freedom to the user is desirable only as long as the consistency of the definitions and the defined relations is preserved.

The problems of the management of defined relations

In a data base with a defined relations capability such as the one shown in Figure 1, some of the system operators (e.g. updates, removal of relations from the data base and the redefinition of relations) may lead to logical problems. These operators must be adapted to take care of the hierarchical structure and the dependencies that exist among the relations. This paper is concerned only with the update problems.

Examples of these problems are:

- In Figure 1, if relation SUPPLIER is updated and relation SC is implicit, how can we update an explicit copy of relation FULLST? This is an efficiency problem.
- If relation FULLST is updated, how can we update SUPPLIER and OLDST? This is a logic problem.

Objectives

This paper explains the update problems and offers solutions for some of these problems. Examples are given to illustrate these solutions. The paper also identifies the unsolved problems which require further research and investigation.

Some update algorithms are suggested. They aim at preserving the consistency of the information in a data base having a high degree of data independence. These update algorithms deal with the logical part of the problem and hence pave the way for other algorithms to care for the particular details in particular implementations.

UPDATES

Updates may be divided into:

a. Insertions
b. Deletions
c. Changes in object values.

In the following discussions, the tuples which will update a relation are termed the updating relation. The insertion is seen as a union of the updating relation and the relation to be updated. The deletion is seen as the difference of the relation to be updated and the updating relation. The
changes in object values are taken conceptually as deletions followed by insertions.

The update algorithms

Let us divide the updates into two:

1. Update at higher levels of the hierarchy.
2. Update at lower levels of the hierarchy.

Referring to Figure 1, updating relation SUPPLY or SUPPLIER is an update at a higher level of the hierarchy with respect to NEWSP and SC, FULLST respectively. Updating FULLST or NEWSP is an update at a lower level. Updating SC is an update at a higher level with respect to FULLST and an update at a lower level with respect to SUPPLIER.

Update at a higher level

In this type the update is filtered down the hierarchy and reflected on all the relations defined on the updated relation. The corresponding definition is applied to the updating relation successively at each level of the hierarchy. The problem here is how to pass updates down efficiently and in particular when intermediate levels are implicit.

---

<table>
<thead>
<tr>
<th>DEFINITION</th>
<th>PROOF</th>
<th>IN</th>
</tr>
</thead>
<tbody>
<tr>
<td>X← A ∪ R</td>
<td>A ∪ (R ∪ I) = (A ∪ R) ∪ I = (A ∪ R) ∪ (A ∪ I)</td>
<td>I or A ∪ I</td>
</tr>
<tr>
<td>X← A ∩ R</td>
<td>A ∩ (R ∪ I) = (A ∩ R) ∪ (A ∩ I)</td>
<td>A ∩ I</td>
</tr>
<tr>
<td>X← R - A</td>
<td>(R ∪ I) - A = (R - A) ∪ (I - A)</td>
<td>I - A</td>
</tr>
<tr>
<td>X← A - R</td>
<td>A - (R ∪ I) = (A - R) - I = (A - R) - (I ∩ A)</td>
<td>I or I ∩ A</td>
</tr>
<tr>
<td>X← R% (list)</td>
<td>(R ∪ I)% (list) = R%(list) ∪ I%(list)</td>
<td>I%(list)</td>
</tr>
<tr>
<td>X← R:(filter)</td>
<td>(R ∪ I) : (filter) = (R:(filter) ∪ I:(filter))</td>
<td>I : (filter)</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>DEFINITION</th>
<th>PROOF</th>
<th>IN</th>
</tr>
</thead>
<tbody>
<tr>
<td>X← A ∪ R</td>
<td>A ∪ (R - I) = (A ∪ R) - (I - A)</td>
<td>I - A</td>
</tr>
<tr>
<td>X← A ∩ R</td>
<td>A ∩ (R - I) = (A ∩ R) - (I ∩ A)</td>
<td>I or A ∩ I</td>
</tr>
<tr>
<td>X← R - A</td>
<td>(R - I) - A = (R - A) - I = (R - A) - (I - A)</td>
<td>I or I - A</td>
</tr>
<tr>
<td>X← A - R</td>
<td>A - (R - I) = (A - R) ∪ (I ∩ A)</td>
<td>I ∩ A</td>
</tr>
<tr>
<td>X← R% (list)</td>
<td>(R - I)% (list) ≠ R%(list) - I%(list)</td>
<td>R must be made explicit</td>
</tr>
<tr>
<td>X← R:(filter)</td>
<td>(R - I) : (filter) = R:(filter) - I:(filter)</td>
<td>I or I : (filter)</td>
</tr>
</tbody>
</table>

---

**Insertions**

Given relations A and R, let R∪I be the updated value of R. Let X=f(A,R) be a defined relation. Relation IN is sought such that X∪IN is the updated value for X which corresponds to the update of R. Such IN can be found for all operators (f) in case of insertions at high levels. This is shown in table CHANGE (Figure 2).

When f is union, intersection, projection or selection IN=f(A,I) i.e. X need not be present in order to evaluate IN.

The mechanism of the insertion is essentially carried out at each level as follows:

a. Given the updating relation (I) and a definition, relation IN is found from table CHANGE. Relation IN is now the updating relation which will be passed to the following levels down the hierarchy. (i.e. IN will be I of the lower level).

b. If the relation to be updated (X) is explicit, the updated value will be X∪IN. This applies to all relations at lower levels. If X is implicit, updates are passed down the hierarchy without materializing X.

Examples 1 and 3 in the Appendix illustrate this type of insertion.

The exception to Rules a and b is the definition containing the difference operator when the second relation is to be

---

From the collection of the Computer History Museum (www.computerhistory.org)
updated i.e. definitions of the type \( X \leftarrow A - R \), where \( R \) is the relation to be updated. In this type the insertion is carried out as follows:

1. Either:
   \[ \text{IN} = I \] is passed to lower levels.
   Or:
   Alternatively, \( \text{IN} = I \setminus A \) will be passed to lower levels (as in table \text{CHANGE}).
   The updating relation \( \text{IN} \) is passed to the lower levels as a deletion and the process continues as a deletion operation.

2. If the relation to be updated \( (X) \) is explicit, the updated value will be \( X - \text{IN} \). This applies to all the relations at lower levels. If \( X \) is implicit, updates are passed down the hierarchy without materializing \( X \).

Example 2 in the Appendix illustrates this type of definition.

### Deletions

In this case the update may be specified by providing either a relation \( I \) containing the tuples to be deleted or boolean filter which selects the tuples to be deleted from the relation to be updated. These extracted tuples constitute the updating relation \( I \).

In a fashion similar to the previous section’s, given relations \( A \) and \( R \), let \( R - I \) be the updated value of \( R \). Let \( X = f(A,R) \) be a defined relation. Relation \( 
\text{IN} \) is sought such that \( X - \text{IN} \) is the updated value for \( X \) which corresponds to the update of \( R \). Such \( \text{IN} \) can be found for all operators, \( f \), as in table \text{CHANGE} (Figure 2).

When \( f \) is intersection, join or projection \( \text{IN} = f(A,I) \), i.e. \( X \) need not be present in order to evaluate \( \text{IN} \). The mechanism of the deletion is the same as that of the insertion (described in the previous section) except for the following:

- In (b) the union is replaced by the difference.
- In (1) only the alternative method is applicable. The updating relation is passed to lower levels as an insertion.
- In (2) the difference is replaced by the union.

### Change in object value

A change in object value (modification) can be conceptually considered as deletion followed by insertions.

### The algorithm

The algorithm for insertions and deletions from higher levels follows:

```plaintext
procedure alter (R, I, DELT) recursive:
    boolean DELT, DELN; relations R, I, IN;
    comment R is the relation to be altered

I is the relation of tuples to be inserted or deleted from R
n is the number of relations defined on R
Dj is the jth definition on R
Xj is the subject of Dj, i.e. the jth relation defined.

DELT, DELN true if deletions are to be made from R false if insertions are to be made to R.

DELj is true if an insertion to R implies a deletion from Xj

CHANGE is a procedure which calls table CHANGE. Given the definition Dj, the updating relation I and DELT, the procedure returns the tuples to be inserted or deleted:
```

```plaintext
if \( R \) is explicit then begin
    if \( \text{DELT} \) then \( R:=R-I \)
    else \( R:=R\cup I \)
end:
for \( j:=1 \) step 1 until \( n \) do
begin
    \( \text{DELN}:=\text{DELT} \neq \text{DELj} \)
    \( \text{IN}:=\text{CHANGE} (Dj, I, \text{DELT}); \)
    if \( \text{IN} \neq \text{null} \)
        then alter (Xj, IN, \text{DELN})
end alter;
```

The algorithm for changes in object value follows from the previous algorithm.

### Discussion

The great advantage of the approach described above is that only the explicit forms are updated. Implicit relations will not be recreated just for the purpose of having them updated. This saves the cost of creating all the relations down the hierarchy. Hence, in very large data bases the advantages of defined relations will not be outweighed by the substantial cost of recreating implicit relations whenever an update occurs. In such an environment, the overhead of storing table \text{CHANGE} and executing the update algorithm is negligible.

It is noteworthy that table \text{CHANGE} will have entries for all the user-defined operators. One problem, however, is to find rules for dealing with the complex definitions. No attempt has been made to explore this subject in the present paper.

### Update at a lower level

Here the problem is how to pass the update up the hierarchy without introducing inconsistency. It is a logic problem. In addition to the conditions imposed on ordinary updates, e.g. the compatibility of the updating relation and the
relation to be updated, the following conditions should be satisfied for updating relations at lower levels:

1. The update should not result in incomplete information at upper levels. Consider the following example: Given relations X, Y and R.

    \[
    \begin{array}{ccc}
    Y & X & R \\
    2 & 5 & 3 & A & 8 & 3 & A \\
    3 & A & 8 & 6 & B & 8 & 6 & B \\
    6 & B & 8 \\
    \end{array}
    \]

    Definitions:
    \[
    X = Y: (\text{field}(1)>2) \\
    R = X\% (\text{fields} 1 \text{ and } 2)
    \]

    To update R by I
    \[
    4 & M \\
    5 & T
    \]

    and X by I'
    \[
    9 & K & 3
    \]

    When relation R is updated, the union of the updating tuples (I) and relation X cannot be formed because these tuples supply information for only two of the components of X. In such cases the update will provide incomplete information and should therefore be prohibited. Updating X, however, does not lead to missing information in upper levels. The system should prompt the user to update the relation at the lowest level whose update does not violate Conditions 1 and 2.

2. No ambiguity should result at higher levels due to the update.

    \[
    S & Q & Z & I \\
    1 & 4 & 1 & 9 \\
    6 & 5 & 6 & 3 \\
    2 & 2 & 2 \\
    4 & \\
    5 &
    \]

    Definition: \( Z = S \cup Q \)

e.g. if a relation Z is defined as the union of relation S and Q.

    When Z is updated the data base system cannot readily know which of the updating tuples should update each of S and Q and which should update both relations. Whenever such an ambiguity exists, the update operation must be prohibited.

3. The updating tuples must satisfy the definition of the relation to be updated. In the example in (1) above, if relation X is updated by relation P,

    \[
    P \\
    9 & K & 3 \\
    1 & T & 6
    \]

    the system must reject the last tuple because it contradicts the definition of the relation to which it will belong. Similarly, if a tuple is to be deleted from such a relation, an equivalent tuple must exist in the relation; otherwise the deletion is meaningless.

    With these constraints, more weight is attached to the consistency of the data base information at the expense of the user convenience. Indeed, inconsistency in itself, regardless of any other repercussions, may perhaps lead to more inconvenience to the user. With updating at a lower level, almost every operator in the definition requires different updating algorithms as will be explained in Figure 4. In all cases the update can be passed up without explicit values of the relations involved. For joins, the upper level relations must be implicit if side effects of updates are to be avoided.

    A more formal but less complete discussion of these problems, based in part on an earlier version of this paper, is given by Todd. The relational operators may be divided into regular and irregular operators according to their performance when relations having these operators in their definitions are updated. Regularity is a property of the type of update as well as of the form of operation.

The regular and irregular operators

    Regular operators have two major properties when they appear in the definition of the relation to be updated:

a. They do not lead to ambiguity.
b. The updating information can be passed to higher levels.

    When the operator is regular we will have the advantages of the updates at higher levels. When a definition contains an irregular operator, the problem of ambiguity arises. It is interesting to note (Figure 3) that with the exception of the selection operator, those operators that are irregular with insertion are regular with deletion and vice versa.

Validity of updates

    In certain cases of join and selection, even regular updates can make the data base invalid or have undesirable side effects. These can only be checked given particular data values.

    Consider a relation I inserted into relation X.

    X = A:(filter). If the tuples of I do not satisfy the filter an inconsistency will be introduced. If such a set I is deleted
Insert:

<table>
<thead>
<tr>
<th>Definition</th>
<th>Proof</th>
<th>Irregularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-AUB</td>
<td>XUI = (AUIU(BU))</td>
<td>regular</td>
</tr>
<tr>
<td>X-AUB</td>
<td>XUI = (AUIBU)</td>
<td>regular</td>
</tr>
<tr>
<td>X-A-B</td>
<td>XUI = (AUB)-(B-I)</td>
<td>regular</td>
</tr>
<tr>
<td>X-A%[filter]</td>
<td>XUI = A(UQ%[filter]), many Q</td>
<td>irregular</td>
</tr>
<tr>
<td>X-A+B</td>
<td>XUI = (AUI%[(fields(A))])</td>
<td>regular</td>
</tr>
<tr>
<td>X-A:(filter)</td>
<td>XUI = A(U: (filter))</td>
<td>regular</td>
</tr>
</tbody>
</table>

Delete:

<table>
<thead>
<tr>
<th>Definition</th>
<th>Proof</th>
<th>Irregularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-AUB</td>
<td>X-I = (A-I)(B-I)</td>
<td>regular</td>
</tr>
<tr>
<td>X-A%B</td>
<td>X-I = (A-I)%B</td>
<td>regular</td>
</tr>
<tr>
<td>X-A-B</td>
<td>X-I = (A-I)-B</td>
<td>irregular</td>
</tr>
<tr>
<td>X-A%[filter]</td>
<td>X-I = (A-I)%[filter]</td>
<td>irregular</td>
</tr>
<tr>
<td>X-A+B</td>
<td>X-I = A(B-I)%[fields(B)]</td>
<td>irregular</td>
</tr>
<tr>
<td>X-A:(filter)</td>
<td>X-I = (A-I):(filter)</td>
<td>regular</td>
</tr>
</tbody>
</table>

Figure 3—Regular and irregular operators for updates at a lower level.

\* A-I, generalized difference, those tuples of A that do not project into I.

from X, only tuples satisfying the filter must be deleted from A (the remainder cannot be deleted from X in any case, as they could not have been X to start with).

If a relation I is inserted into a relation X, X-A*B, tuples are inserted into A and B. This may have the side effect of inserting tuples (E) into X. (E=I%[fields(A)]*BUA*I%[fields(B)].) Whether or not this is desirable depends on the application. It can be checked by reference to A and B without materializing X.

Other cases of regular updates cannot directly cause validity or side effect problems.

Discussion

The table in Figure 4 shows the update form of the defining relations. Each defining relation has its own form of update. Updates from lower levels of the hierarchy should be carefully checked to ensure that the operators in the definitions involved are regular and that the updating information is logically valid. With large data bases such overheads are tolerable because of the probable avoidance of recreating intermediate implicit relations. However, with small sized data bases it may be sensible to prohibit updates from lower levels.

CONCLUSIONS

This paper has attempted to investigate the major aspects of the update problem for data bases with defined relations capabilities. It has been shown that in the majority of the cases implicit relations can be updated at higher levels of the hierarchy without the need to create their explicit forms. No ambiguity or inconsistency will arise from updates at higher levels.

However, updates at lower levels may lead to ambiguity in some situations. Checks are also needed to establish the validity of the updating information. In very large data bases updating the defined relations in the manner described in this paper will lead to substantial saving in the computer system resources. It is hoped that this paper will initiate some research in this interesting and practical problem. Thorough investigation is needed for the whole problem of lower level updates. Algorithms for handling complex definitions are also required.

ACKNOWLEDGMENTS

The author is indebted to Barry Aldred of IBM Scientific Research Center (UK), to the referees for their valuable suggestions and to Stephen Todd for editing this paper.

REFERENCES

APPENDIX

Relation SUPPLIER

<table>
<thead>
<tr>
<th>S#</th>
<th>SNAME</th>
<th>STATUS</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>SMITH</td>
<td>20</td>
<td>LONDON</td>
</tr>
<tr>
<td>S2</td>
<td>JONES</td>
<td>10</td>
<td>PARIS</td>
</tr>
<tr>
<td>S3</td>
<td>BLAKE</td>
<td>10</td>
<td>PARIS</td>
</tr>
<tr>
<td>S4</td>
<td>CLARK</td>
<td>20</td>
<td>LONDON</td>
</tr>
<tr>
<td>S5</td>
<td>ADAMS</td>
<td>30</td>
<td>ATHENS</td>
</tr>
</tbody>
</table>

Relation SUPPLY

<table>
<thead>
<tr>
<th>S#</th>
<th>P#</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>P1</td>
<td>3</td>
</tr>
<tr>
<td>S1</td>
<td>P2</td>
<td>2</td>
</tr>
<tr>
<td>S1</td>
<td>P3</td>
<td>4</td>
</tr>
<tr>
<td>S1</td>
<td>P4</td>
<td>2</td>
</tr>
<tr>
<td>S1</td>
<td>P5</td>
<td>1</td>
</tr>
<tr>
<td>S2</td>
<td>P1</td>
<td>3</td>
</tr>
<tr>
<td>S2</td>
<td>P2</td>
<td>4</td>
</tr>
<tr>
<td>S3</td>
<td>P3</td>
<td>4</td>
</tr>
<tr>
<td>S3</td>
<td>P3</td>
<td>4</td>
</tr>
<tr>
<td>S3</td>
<td>P5</td>
<td>2</td>
</tr>
<tr>
<td>S4</td>
<td>P2</td>
<td>2</td>
</tr>
<tr>
<td>S4</td>
<td>P4</td>
<td>3</td>
</tr>
<tr>
<td>S4</td>
<td>P5</td>
<td>4</td>
</tr>
<tr>
<td>S5</td>
<td>P4</td>
<td>3</td>
</tr>
</tbody>
</table>

The hierarchy of the defined relations mentioned in the following examples has been shown in Figure 1.

Relation OLDST

<table>
<thead>
<tr>
<th>CITY</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LONDON</td>
<td>20</td>
</tr>
<tr>
<td>KHARTOUM</td>
<td>15</td>
</tr>
</tbody>
</table>

Relation OLDSP

<table>
<thead>
<tr>
<th>S#</th>
<th>P#</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>P1</td>
<td>3</td>
</tr>
<tr>
<td>S2</td>
<td>P2</td>
<td>4</td>
</tr>
<tr>
<td>S6</td>
<td>P1</td>
<td>3</td>
</tr>
<tr>
<td>S7</td>
<td>P4</td>
<td>1</td>
</tr>
</tbody>
</table>

Example 1—(Insertion at a higher level)

Definitions:
(i) SC~SUPPLIER%(CITY,STATUS)
(ii) FULLST~OLDST∩SC

Before Update:

Relation SC

<table>
<thead>
<tr>
<th>CITY</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LONDON</td>
<td>20</td>
</tr>
<tr>
<td>PARIS</td>
<td>10</td>
</tr>
<tr>
<td>ATHENS</td>
<td>30</td>
</tr>
</tbody>
</table>

Relation FULLST

<table>
<thead>
<tr>
<th>CITY</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LONDON</td>
<td>20</td>
</tr>
</tbody>
</table>

Relation I (The updating tuples to be inserted in relation SUPPLIER)

<table>
<thead>
<tr>
<th>S#</th>
<th>SNAME</th>
<th>STATUS</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S6</td>
<td>AHMED</td>
<td>15</td>
<td>KHARTOUM</td>
</tr>
<tr>
<td>S7</td>
<td>KIM</td>
<td>35</td>
<td>TOKYO</td>
</tr>
</tbody>
</table>

After Update:

(1) Update relation SUPPLIER

SUPPLIER: =SUPPLIER∪I

(2) Compute new I by substituting I in the definition for the relation to be updated.

I: =I%(CITY,STATUS)

Update SC.

SC: =SC∪I

SC

LONDON  20
PARIS    10
ATHENS  30
KHARTOUM 15
TOKYO   35

(3) I: =OLDST∩I

I

KHARTOUM 15

Update FULLST

FULLST: =FULLST∪I

FULLST

LONDON  20
KHARTOUM 15

Example 2—(Insertion and deletion at a higher level)

Definitions:
(i) NEWSP~OLDST~SUPPLY
(ii) R~NEWSP%(1)

Before Update:

Relations SUPPLY and OLDSP are as above

NEWSP

<table>
<thead>
<tr>
<th>S#</th>
<th>P#</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S6</td>
<td>P1</td>
<td>3</td>
</tr>
<tr>
<td>S7</td>
<td>P4</td>
<td>1</td>
</tr>
</tbody>
</table>

R

S6
S7

After Update:

update SUPPLY by I

I

S# | P#   | QTY |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S6</td>
<td>P1</td>
<td>3</td>
</tr>
<tr>
<td>S8</td>
<td>P2</td>
<td>1</td>
</tr>
</tbody>
</table>

(1) SUPPLY: =SUPPLY∪I

(2) following the proposed method, I either remains unchanged, or I: =OLDSP∩I.

I

S6 | P1 | 3 |
S8 | P2 | 1 |

NEWSP: =NEWSP−I

(3) the update continues as deletion.

I: =D: = 1%(1)

D

S6
S8

R: =R−I

R

S7
Example 3—(Insertion at a higher level, with implicit relations)

Definitions—Same as Example 1. Assume that relation SC is implicit.

Before Update—As in Example 1

(1) SUPPLIER := SUPPLIER ∪ I
(2) I := I ∪ {(CITY, STATUS)}

<table>
<thead>
<tr>
<th>I</th>
<th>FULLST</th>
</tr>
</thead>
<tbody>
<tr>
<td>KHARTOUM 15</td>
<td>FULLST</td>
</tr>
<tr>
<td>TOKYO 35</td>
<td></td>
</tr>
</tbody>
</table>

SC is implicit. We do not create it for the update. Continue to the next level.

(3) I := OLDST ∩ I

<table>
<thead>
<tr>
<th>I</th>
<th>FULLST</th>
</tr>
</thead>
<tbody>
<tr>
<td>KHARTOUM 15</td>
<td>FULLST</td>
</tr>
<tr>
<td>LONDON 20</td>
<td></td>
</tr>
<tr>
<td>KHARTOUM 15</td>
<td></td>
</tr>
</tbody>
</table>