PROCEEDINGS

Very Large Data Bases

FIFTH INTERNATIONAL CONFERENCE
ON VERY LARGE DATA BASES

Edited By Antonio L. Furtado & Howard L. Morgan

RIO DE JANEIRO, BRAZIL
OCTOBER 3-5, 1979

SPECIAL CONFERENCE PROCEEDINGS
FOR CONFERENCE PARTICIPANTS ONLY

THIS EDITION WAS PREPARED AND PRINTED
BY COURTESY OF SUCESU-RJ

Very Large Data Bases

FIFTH INTERNATIONAL CONFERENCE

ON VERY LARGE DATA BASES

RIO DE JANEIRO, BRAZIL

OCTOBER 3-5, 1979

Sponsors and Supporters

The following societies and institutions have approved sponsorship or support of the Fifth International Conference on Very Large Data Bases at the time of the publication of the advanced program.

Association for Computing Machinery (ACM)-SIGMOD, SIGBDP, SIGIR
International Federation for Information Processing
IEEE Computer Society
SUCESU Users Association, Brazil
IBM, Brazil

MONITORING INTEGRITY CONSTRAINTS IN A CODASYL-LIKE DBMS

Rubens N. Melo

Departamento de Informática
Pontifícia Universidade Católica
Rio de Janeiro

ABSTRACT

This paper proposes that a DBMS of the kind typically used in a COBOL application environment should have facilities to implement automatically the monitoring of the integrity constraints specified at the conceptual level. In particular it is suggested that within certain limitations a preprocessor can cheaply implement such monitoring. Such facilities which are being implemented in a CODASYL-like DBMS are shown and the implementation techniques are outlined.

INTRODUCTION

A database is not just a static collection of data. As a model of some real world system a database must reflect both the static and dynamic aspects of the system it models. At a certain point of time the contents of a database represent a snapshot of a possible state of the application world. At successive time instants this world may change in certain limited ways and the database is correspondingly moved to a new permitted state.

In this paper we will deal with the problem os specifying the permitted states of a database by introducing 'integrity' constraints in the definition of a database. First, however, we must make clear what we mean by 'integrity' because in this context this term still deserves some explanation.

Database integrity, in its broadest sense, implies the completeness, soundness, purity, veracity and confidentiality of data. In this sense, 'integrity' encompasses other disciplines such as security, privacy, access control and consistency.

There is some confusion of terms in the literature when one refers to this subject.[1] Security, for example, is sometimes used instead of privacy, to mean the prevention of unauthorized access and modification of data. In other context, specially in the context of commercial DBMS[2], 'security' includes integrity, recovery and access control. The term 'consistency' is often used when one refers to the prevention of semantic errors which may arise due to the interaction of two or more processes in concurrent use of shared data[1]. Some authors[3,4] however, use 'consistency constraint' to mean assertions on the database contents to reflect a legitimate configuration of its application domain.

In this paper we use the term 'integrity constraint' in the same sense as 'consistency constraint' mentioned above. A semantic integrity constraint is therefore, an expression which specifies that a predicate holds for one or more elements of the database structure. Such predicates define the 'correctness' of data in such a way that the violation of one of them indicates that the state of the database is illegitimate and that some of its contents are erroneous.

Examples of integrity constraints are:

'The salary of an employee must not be less than 1500 and not greater than 200000'

'No employee can earn more than his manager'

Integrity constraints play an important role in database design. The conceptual Schema[5] of a database must include not only a specification of the

CH1406-8/79/0000-0209$00.75 © 1979 IEEE

data structure but also the integrity constraints. The Conceptual Schema should be specified in a language where the naturally occurring structure and situations could be described without resorting to artificial constructs. For this purpose some recent works on the semantics of database have developed 'semantic data models'[6,7] and languages[8,9,10,12].

It is commonly assumed that the conceptual specification will serve as a guide to the database administrator (DBA) for constructing a schema in terms of the lower level data model of an available DBMS. Typically the DBA has some canonical sets of representations for each construct of the conceptual structure of the database. In this fashion the design task of mapping the conceptual structure to lower level concepts of a DBMS is made more systematically. However it is tacitly assumed that the semantic integrity constraints defined at the conceptual level are going to be represented separately by DBA procedures or consciously considered in the application programs.

The purpose of this paper is to propose that a DBMS should have tools to implement both the data structure and the integrity constraints specified at the conceptual level. In particular we show how these tools are incorporated in a CODASYL-like DBMS being implemented[15]

The remainder of this paper is organized as follows. In the next section the specification of the integrity constraints at the level of a data model used in a CODASYL-like DBMS is shown. In Section 3, comments on the implementation of the integrity constraints are made. The paper ends with some concluding remarks.

SPECIFICATION OF INTEGRITY CONSTRAINTS ON A CODASYL-LIKE MODEL

In this section we discuss the aspects related to the description of integrity constraints at the internal level, i.e., in terms of the data model of a CODASYL-like DBMS. First we briefly review the concepts of schema and subschema, then we show by means of examples an outline of the language used in the schema to describe the database structure together with its integrity constraints. The section ends with a presentation of the main data manipulation operations and a discussion of the possible violations of integrity constraints when these operations are performed.

Schema and Subschema

A schema is a representation of the database structure and its integrity constraints in terms of certain data model concepts. The main concepts of our data model are the record-type and the set-type[16]. Even without explicit integrity constraints a schema by itself already imposes constraints on the possible contents of the database. This kind of constraints is not suficient as far as limiting the database definition to the DBA's intention is concerned. The integrity constraints try to complete the database definition by avoiding the specification of undesirable data.

A database is to be used by different applications. Each class of application usually is concerned with only a subset of the data contained in the database. A subschema is a description of this subset in terms of the same data model used in the schema. In our system we also allow the definition of access constraints which select the occurrences of record and set types that can be accessed through a subschema.

As an example consider a database about employees (emp) departments (dept) and projects (prj). Figure 1 illustrates a structure diagram for this database and figure 2 shows an outline of a corresponding schema definition. For economy of space, we do not show a complete schema. Some clauses are enough to show differences between CODASYL DDL[11] and the one used in our system. Examples of integrity constraints are also shown and commented.

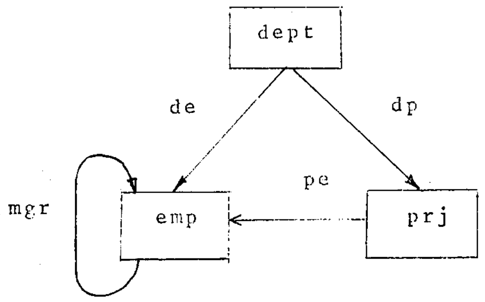

Figure 1: Structure diagram of example database

```
SCHEMA NAME IS sch1.
RECORD NAME IS dept,/* Department */
ITEM dn CHAR(2); /* Department number */
ITEM budget INTEGER;
--------------------
IDENTIFIER  (dn);
CONSTRAINTS:
   dn = 'D1' OR 'D2' OR 'D3'.

RECORD NAME IS prj; /* project */
ITEM pn CHAR(2); /* project number */
ITEM dn CHAR(2); /* Department number */
ITEM status CHAR(1);
ITEM budget INTEGER;
IDENTIFIER (pn);
CONSTRAINTS:
   status = 'A' OR budget > 200000.
   /* If status 'A' then budget>200000 */
RECORD emp; /* employee */
ITEM en INTEGER, /* employee number */
ITEM dn CHAR(2), /* emp's dept number */
ITEM salary INTEGER;
--------------------
IDENTIFIER (en);
CONSTRAINTS:
   salary > 1500 AND < 100000;
   dn = 'D2' OR salary < 50000.
   /*Employees from dept D2 earn less than
                                  50000 */
SET NAME IS dp; /* projects of a dept */
OWNER IS dept;
MEMBER IS prj FIXED, AUTOMATIC
USING dn OF prj = dn OF OWNER;
ORDER IS FIFO;/*projects of a department
                are in chronological order*/
CONSTRAINTS:
   SUM(budget of MEMBER)<budget of OWNER.
SET de; /* employees of a dept */
--------------------
SET pe; /* employees in a project */
OWNER prj;
MEMBER empt OPTIONAL,MANUAL;
ORDER IS SORTED ASCENDING BY salary;
CONSTRAINTS:
   dn OF MEMBER = dn OF OWNER,
SET mgr;/*employees of the same manager*/
OWNER emp;
MEMBER emp OPTIONAL MANUAL;
CONSTRAINTS:
   dn OF MEMBER = dn OF OWNER
      AND salary OF MEMBER<=salary OF
                                OWNER.
   /*an employee's manager works in the
     same dept and earns not less than
     him*/
```

Figure 2: Example of Schema Definition

Record Types and Their Integrity Constraints

A record type is described in this model by:

- its name
- its data items
- its identifier (there can be more than one identifier)
- the integrity constraints which apply to this record type independently of its participation in any set type.

Identifiers. An identifier is a sub-list of the list of items of a record type the value of which cannot be duplicated in the record type. Different identifiers may be specified for the same record type. For example in a record type

Emp(en, firstname, lstname)

either the item en(employee's number) or the pais (firstname, lstname) could be specified as different identifiers.

Record Integrity Constraints. The integrity constraints on a record type may be classified as:

- Intra-record constraints
- Inter-record constraints
- Constraints on aggregation of records

a) Intra-record constraints. These are predicates that the items values of a record occurrence must satisfy. The sintax of these predicates is similar to that of a valid logical expression in COBOL. For instance the constraints on a record Person (name, height, weight, eye, age) could be:

```
1.00 + 0.7 * weight < height
AND height < 1.4 * weight + 1.00;
age > 18 AND < 35,
eye = 'green' OR 'blue' OR 'brown';
```

b) Inter-record constraints. These constraints are predicates that must hold between two or more records of the same type. A typical constraint of this types is:

'No employee can earn more than his manager'

supposing that the manager is identified by an item in the employee record. For instance the employee record could be Emp(en, name, mgr, salary) where mgr is the manager's number as an employee.

To describe this constrain we usually have to resort to range variables or synonims to selct the manager record occurrence.

Ex. Range x is emp
$\forall$ x (salary OF emp < = salary OF x OR mgr OF emp $\neq$ en OF x)

In this model we have two ways to specify this constraint:

1. Using a synonim x of emp

'salary OF emp < = salary OF emp(x)
WHERE (mgr OF emp = en OF x)';

2. Via a set between emp and emp and a set constraint (see below). Probably the relationship of employee with his manager would be better modelled by a set between emp and emp (a construct allowed in our system) which also indicate that each employee has only one manager. In this case this constraint would be specified very easily by a set constraint:

'salary OF OWNER $\geqslant$ salary OF MEMBER;'

c) Constraint on Aggregation of Records. These constraints are predicates on 'aggregation' of records because typically they express that the value of some aggregation function (AVG, SUM, etc) applied to a subset of the record type is restricted.

For example:

'The average salary of employees whose age is more than 30 must be greater than 100000'.

The specification of this constraint could be stated as:

'AVG(salary OF emp RECORD WHERE
(age OF emp > 30)) > 100000'

These constraint are very difficult to monitor because they may imply many database accesses. Again, in most cases the aggregation of records are implemented as sets, and these constraints are expressed as constraints on the members of a set.

Set Types and Their Integrity Constraints

A set type is described in this model by

- its name
- its owner record type and member record type
- the storage class of its member (AUTOMATIC or MANUAL)
- the retention class of its member (OPTIONAL, MANDATORY and FIXED)
- the (permanent) order of record member in its set occurrences (FIFO, LIFO, NEXT, PRIOR, IMMATERIAL, SORTED by ...)
- the additional integrity constraints that owner record and its members must satisfy in the context of a set occurrence.

A set type is a function f: MEMBER $\rightarrow$ OWNER from the MEMBER record type into the OWNER record type.

The same record type may play the role of OWNER and MEMBER in a set type. Singular set types owned by the special record SYSTEM are also allowed.

An AUTOMATIC member of a set is automatic and immediately inserted in the set when it is stored in the database. To select the appropriate set occurrence on storage of a record the system uses the USING subclause in the AUTOMATIC clause.

A record can only be removed from a set if it is an OPTIONAL member. The difference between MANDATORY and FIXED is that a MANDATORY member can be moved from a set occurrence to another and a FIXED member cannot.

The order of member records in a set is automatically maintained when a member is inserted or updated.

Set Integrity Constraints. The OWNER and MEMBER clause define which records can participate in a valid association between records. The set integrity constraints are the predicates which give meaning to these associations. These constraints are additionally imposed to the records participating in a set besides the constraints on the record types to which they belong.

The set constraints may be classified as:

- Constraints between owner and member
- Inter member constraints
- Constraints on aggregations of members

a) Constraints between owner and member. These constraints are predicates involving items of both owner and member record types.

```
Ex. Set NAME is mgr;
    OWNER is emp;
    MEMBER is emp;
    CONSTRAINT:
        salary of MEMBER < salary of
                                OWNER;
        sex of MEMBER = 'female'
```

b) Inter member constraints. These are predicates that must hold between two or more members of a set. These constraints may be expressed in terms of the relative position of members in the set. For instance, a further constraint on the set type mgr of the last example could be:

FETCH PRIOR salary of MEMBER <=salary of MEMBER;

```
salary of MEMBER < = FETCH NEXT
salary of MEMBER
```

In other words the above constraints are equivalent to

```
'ORDER IS SORTED ASCENDING BY salary'
```

In fact the ORDER clause is an 'implicit' integrity constraint. However it is considered a 'hard' constraint and is not specified in the constraint section because it is inherent to the model. In this example if the ORDER clause is specified the order of members is automatically maintained after an insertion or update. In this case these operations are not conditional and they cannot violate the order constraint because the check of order and the necessary internal modifications to maintain it are performed as part of the operation.

If the explicit constraints were used as in the last example and no ORDER clause was given (which is equivalent to IMMATERIAL ORDER) then any member insertion or update would be conditional. The operation would only be concluded if the specified integrity constraints were not violated. Then the check would not be a part of the operation.

c) Constraints on aggregation of members. This type of constraint specifies that when some aggregation function (AVG, SUM, etc) is applied to the member records of a set occurrence,it must yield some limited value.

Ex. In a set between Dept (deptno, budget) and Emp (empno, salary) a constraint of this type could be:

```
SUM(salary of MEMBER) <=budget
of OWNER
```

to express that 'the sum of salary of employees of each department must be limited by the department budget'.

Data Manipulation vs Integrity Constraints

The integrity constraints are concerned with the meaning of data. The meaning of data can only be achieved by defining not only its structure but also its uses (operations on data). As we are dealing with databases where the data is shared by many users, the complete specification of the use of the database includes many aspects such as access constraints, authorization constraints besides the semantic integrity constraints. This paper however deals only with the latter aspect.

The use of a database is through subschemas. As we mentioned before, a subschema in our system besides the common subsetting transformation usually possible in a CODASYL subschema, the record and set types in our subschema consist only of selected occurrences of the corresponding schema types.

For example a subschema of the schema illustrated in figure 2 could be defined as follows:

```
SUBSCHEMA sub1 OF SCHEMA sch1
-----------------------------
RECORD teacher FROM emp WHERE
                  (dn = 'D1');
ITEM teacher-no FROM en;
ITEM salary,

RECORD e-project FROM prj
          WHERE (budget > 100000).

SET pp FROM pe WHERE (dn of OWNER
                          = 'D1');
OWNER is e-project;
MEMBER is teacher.
```

The applications which use this subschema refer to 'employee of department D1' as teacher and the relevant items of teacher records for these applications are teacher-no and salary. An expensive project (with budget > 100000) is renamed as e-project and the only set occurrences of set type pe (renamed as pp) considered in this subschema are those where the owner is an e-project which has the item dn equal to 'D1'.

It could be the case that, by an error of design, these access constraints, may be specified inconsistently with some other constraint given in the schema. In this case, the access constraint implies empty record or set types.

Operations. Before we discuss the possible violations of integrity constraints when an operation is performed we must distinguish between the different levels of constraints that are imposed on the database by the system.

Some constraints are considered inherent to the data model. The data structure by itself already imposes constraints. For instance the rule 'a member record cannot be simultaneously in two different occurrences of the same set type' can be seen and used as an 'implementation' of an integrity constraint like 'an employee cannot work simultaneously for two departments'.

Some clauses like the IDENTIFIER, AUTOMATIC, MANDATORY, FIXED and ORDER clauses can be seen as 'implicit' integrity constraints. The check of these constraints are however, considered part of the database operations.

The user applications are supposed to be written in extended COBOL or FORTRAN. The data manipulation is expressed in the programs by means of high level commands incorporated in the host language. A preprocessor analyzes and transforms the user program into a standard COBOL or FORTRAN program with some extra code corresponding to the invocation of these high level commands.

The record and set types referred to by these commands must be defined in the subschema used by this application.

In the sequel the relevant commands as used in COBOL are briefly introduced and their implication on the integrity constraints are discussed.

a) Retrieval Operations.

```
         NEXT
         PRIOR  rec-type RECORD
     FINDFIRST
         LAST   set-type MEMBER
         ANY

     WHERE (logical expression)
```

A record occurrence (or member) of the specified record (or set) type which satisfies the positional and WHERE conditions is set as the current record (or member) of the record (or set) type.

```
                          rec-type RECORD
   .GET|i1,i2,...OF|set-type MEMBER
                          set-type OWNER

     |INTO d1, d2, ...|
```

The specified items i_1, i_2, ... of current record (or member or owner) of the specified record (or set) type are brought to the UWA (user work area). If no item is specified a complete subschema record is brought to the UWA. If the INTO clause is given then the items are further moved to data names i_1, d_2, ...

```
                               rec-type RECORD
   .FETCH|i1, i2, ... OF|set-type MEMBER
                               set-type OWNER

     WHERE (logical expression)

     |INTO d1, d2, ... |
```

This is a combined form of FIND and GET.

These three operations do not change the database state. Therefore they are not concerned with integrity constraints. By hypothesis, the database is 'correct' when they are applied. They are concerned, however, with the access constraints specified in the subschema.

b) Modification Operations. The following operations change the database state.

```
   .STORE rec-type |FROM d|
```

If the FROM clause is used then the contents of (data name) d are first moved to the record in the UWA as the data to be stored.

If this operation is allowed in a subschema, the schema items not included in the subschema record are set to NULL.

The record in the UWA corresponding to the specified record type is stored in the database and made the current of the record type.

Moreover, for each set type where this record type is an AUTOMATIC member type, the new record is inserted in the appropriate set occurrence determined by the USING clause of the MEMBER definition.

As part of the STORE operation the system has to check some implicit constraints. The identifiers can be neither set to NULL nor duplicated. The selection of the appropriate set occurrence in each set type where the record is AUTOMATIC member and is to be inserted, must be possible. Otherwise the operation fails. These checks and the proper insertion of the record in the sets are considered part of the operation when applied to any database.

The explicit integrity constraints are specific to the database in question. These constraints will be checked separately from the operation. These checks determine if the operation will violate the 'correctness' of data of this particular database.

The integrity constraints possibly affected by a STORE are:

Intra-record constraints. The items of the new record (including those with NULL values) must satisfy the intra-record constraints of the record type to which it belongs.

Inter-record constraints. The new record must also satisfy any inter-record constraint of its record type.

Constraints on aggregation of records These constraints may also be violated by the arrival of a new record occurrence.

Set constraints of the set types where the record is an AUTOMATIC member . As the new record must be inserted in the sets all the set constraints checked on an insertion operation are also checked here.

DELETE rec-type. The current record of the specified record type is to be deleted from the database. First, it is removed from all sets containing it as a member. For each set where the record is owner, if the members are OPTIONAL then these are removed, otherwise they are deleted. In this case, each deletion follows the same rules above.

Several integrity constraints have to be checked in a delete operation.

Constraints of the record type. The delete operation may possibly violate the inter-record constraints and the constraints on aggregation of records.

Constraints of the set types where the record is a member. Because the record is to be removed from these sets, the delete operation may violate some set constraints.

The above rules have to be recursively applied for each implied deletion(for example, of MANDATORY or FIXED members).

. INSERT rec-type

INTO set-type1, set-type2,...

The current of the specified record type is to be inserted in the current occurrences of the specified set types. The specified record type must have been defined in the subschema as member type of the given set types. The implicit order constraint is considered by the operation and the new member is inserted in the proper position in each set. The explicit set integrity constraints have to be checked for each set type involved in the operation.

. REMOVE rec-type

FROM set-type1, set-type2, ...

The current of the specified record type is to be removed from the current occurrences of the specified set types that contain it as a member. The record still exists in the database but not more as a member of these sets. Only OPTIONAL members can be removed. This is an implicit constraint that has to be checked. The explicit set integrity constraints can be violated by the removal of a member and have to be checked before the operation is completed.

. MODIFY |i_1, i_2, ...OF|

rec-type RECORD
set-type MEMBER |FROM d_1, d_2, ..|
set-type OWNER

First, if the FROM clause is used then the contents of data names d_1, d_2,.. are moved into the corresponding items i_1, i_2, ... in the UWA. If no item is specified a complete record is considered. The specified items (or the complete record) in the UWA replace the corresponding items (or all) of the specified record (or set) type.

Some implicit integrity constraints are checked as part of this operation. For example, an identifier cannot be modified. The order of the sets where this record is a member is also maintained by this operation. The constraint defined by the USING clause is also maintained. An attempt to change some item in the record constrained to be equal to some item of its owner in some set containing it is an error. In order to be possible to change these items constrained by the USING clause, a variante of the MODIFY operation is used.

```
                        rec-type RECORD
MODIFY|i1,i2,...OF|set-type MEMBER
                        set-type OWNER
```

|FROM d_1,d_2,...|

USING set-type1, set-type 2,...

MEMBERSHIP

In this case the record automatically changes to the appropriate set occurrence where the USING constraint holds between owner and member.

Note that this is an implicit constraint. Its check and maintenance are part of the operation.

The explicit constraints which can be violated by this operation are checked separately before the operation is really done. This operation may violate both record type constraints and set type constraints.

ON THE IMPLEMENTATION OF INTEGRITY CONSTRAINTS

In this section we outline the implementation techniques for monitoring the integrity constraints. The discussion shall concentrate on the simpler constraints. The implementation of the more complex ones are still under investigation.

There are three phases when an integrity constraint can be checked:

a) During the processing of the schema-DDL description of the database. During this phase only 'syntatical' checks can be made. For example, checks for invalid qualifications, inexistent items, record or set types used in the logical expression defining the constraint.

b) During the preprocessing of user programs. Some of the implicit integrity constraints violations can be detected. For instance a MODIFY operation applied to an identifier or a REMOVE applied to a MANDATORY or FIXED member type.

c) During the execution of user programs. The explicit integrity constraints involve the values of data which are only known, during execution time. These constraints have to be checked in some way during the execution of the user program but only the possibly affected constraints need be checked for a given operation invocation.

Although the modification operations are known and fixed (only the parameters vary) the problem of discovering at preprocessing time which tests should be inserted into the user program can become a very difficult task specially if we consider the more complex integrity constraints. An interesting investigation of this kind for higher level operation which may cause multiple changes to a database is done in[13].

The implementation technique proposed here is based on the use of a preprocessor which extends the base language (COBOL or FORTRAN) with high level operations of data manipulation. During the preprocessing phase the operations and their parameters are analyzed and transformed into a series of tests to detect any violation of the possibly affected constraints, followed by the 'internal operation' corresponding to the high level operation invoked by the user program.

Note that the user program refers to the records by their subschema names. The preprocessor transforms the invocation of the subschema in the program DATA DIVISION into a set of record definitions. For each record, member and owner type defined in the subschema the preprocessor defines a record in the DATA DIVISION. Therefore, references to items in integrity constraints are easily transformed into the corresponding data names in the user program.

For example, consider a subschema of schema in figure 2 involving only the emp record and the mgr set between emp and emp. The preprocessor will generate the following records in the DATA DIVISION of the user program:

```
01 EMP.
   02 EN----
   02 DN----
   02 SALARY----
   -------------
01 MGR.
   02 OWNER.
      03 EN----
      03 DN----
      03 SALARY----
      -------------
   02 MEMBER.
      03 EN----
      03 DN----
      03 SALARY----
      -------------
```

Therefore the integrity constraint for the set mgr:

```
'dn OF MEMBER = dn OF OWNER

    AND salary OF MEMBER
    < = salary OF OWNER'
```

is easily transformed into an unambiguous COBOL logical expression

```
dn OF MEMBER OF mgr = dn OF OWNER OF mgr
     AND salary OF MEMBER OF mgr
     NOT > salary OF OWNER OF mgr
```

Check of Intra-record Constraints

An intra-record constraint has to be checked when a record is stored or modified. The new contents of the record, however are always available as data names in the user program. The preprocessor can, therefore, generate the IFs corresponding to the checks of integrity constraints of this kind.

For example consider, for the moment, only the record type emp and its integrity constraints of figure 2, then the operation

MODIFY dn OF emp

would correspond to the following text in the user program:

```
      MOVE modify-error TO failure-code
      IF NOT (dn OF emp NOT = 'D2'
             OR salary OF emp < 50000)
             GO TO LABEL
      MOVE ZERO TO failure-code
      CALL 'internal modify operation'
            USING 'emp record in UWA'.
      LABEL. ----
```

Note that only the record type constraints which involve the items mentioned in the MODIFY operation produce IFs in the user program. Note also that no database access is needed to check this type of constraint..

Check of Constraints Between OWNER and MEMBER of a Set

Now consider as the subschema invoked by the user program, both the record type emp and the set type mgr, as defined in the schema of figure 2. In this case a modification of emp may violate both record and set constraints involving the record type emp.

For example the operation:

```
MODIFY salary OF emp
```

then would correspond to the following text:

```
      MOVE modify-error TO failure-code.
      NOTE check if new record is a
        'correct' emp record.
      IF NOT (salary OF emp > 1500
             AND < 100000) GO TO LABEL.
      IF NOT (dn OF emp NOT = 'D2'
             OR salary OF emp < 50000)
             GO TO LABEL.
      (Save member and owner currency of
       mgr set).
      NOTE check if new record would be a
        'correct' member of mgr.
      MAKE emp CURRENT MEMBER OF mgr.
      IF error-status = ZERO THEN
        IF NOT (dn OF MEMBER OF mgr
                   = dn OF OWNER OF mgr
              AND salary OF MEMBER OF mgr
              NOT > salary OF OWNER OF mgr)
              GO TO LABEL.
      NOTE check if new record would be a
        'correct' owner of mgr.
      MAKE emp CURRENT OWNER OF mgr.
      (For each member of mgr set)
      IF NOT (dn OF MEMBER OF mgr
             = dn OF OWNER OF mgr
             AND salary OF MEMBER OF mgr
             NOT>salary OF OWNER OF mgr)
             GO TO LABEL.
      (End for)
      (Restore member and owner currency
         of mgr set)
      MOVE ZERO TO failure-code.
      CALL 'internal modify operation'
           USING 'emp record in UWA'
    LABEL.----
```

Note that in this case some database accesses may be needed. The MAKE command which appears in the above example is a data manipulation operation that explicitly sets currency indicators and leaves the corresponding records available in the UWA. The currency indicators are explicitly handled in our system.

Other possibility for implementing the integrity constraints checks is to interpret the constraints during execution time. In this case, both the operation and its parameters are passed to a subroutine that checks for possible constraint violations.

The advantage of the former technique is that we can use the base language to check the logical expressions. This is a cheap way of implementing the constraint monitoring. Furthermore, with a careful analysis of which constraints should be checked the amount of extra-code inserted in the user program is limited.

The monitoring of other constraints such as the access constraints of the subschema and the authorization constraints may be implemented in a similar way. The cost of constraint monitoring by this technique then becomes small compared to the benefits. A similar techniques has already been used by Stonebracker in[14].

CONCLUSION

This paper proposes that a DBMS of the kind typically used in a COBOL application environment should have facilities to implement automatically the integrity constraints specified at the conceptual level. Such facilities which are being implemented in a CODASYL-like DBMS were shown and the implementation techniques were outlined.

It is also claimed that the high cost usually associated with the implementation of such facilities may not be so high if one considers that the same mechanisms used to implement integrity constraint monitoring also be employed to implement other facilities such as access control and authorization. Furthermore it is suggested that within certain limitations a preprocessor can very cheaply implement the constraint monitoring.

Aknowlegements

The author wish to thank Antonio Fernando C. Gomes and Helena Barbosa for their useful comments on this paper.

REFERENCES

1. Eswaran, K. P. and Chamberlin, D. D. "Functional Specification of a Subsystem for Database Integrity" Proc Int. Conf. on Very Large Databases. Framingham, Massachussetts (Sept. 1975).

2. Davis B. "The Selection of Database Software", NCC Publication, 1977

3. Florentin, J. J., "Consistency Auditing of Databases", Comput. J. 17, 1 (1974).

4. Huits, M. H. H., "Requirements for Languages in Database Systems" Database Description, North-Holland, Elsevier, Amsterdam, 1975.

5. ANSI/X3/SPARC, "Study Group on Database Management System" Interim Report ANSI 75-02-08.

6. Hammer, M. M., "The Semantic Data Model: A Modelling Mechanism for Database Applications" Proc. Int. Conf. on Management of Data, Austin, Texas (June, 1978).

7. Chen, P. P., "The Entity-Relationship Model: Towards a Unified View of Data" ACM Trans. Database System 1, 1 (Mar. 1976).

8. Biller, H. and Neuhold, E. J. "Semantics of Databases: The Semantics of Data Models" Information Systems, Vol. 3, Number 1, 1978.

9. Cabral, V., "Em Busca de uma Linguagem para a Especificação Conceitual de Sistemas Apoiados em Banco de Dados". Tese de Mestrado - PUC-RJ,1978.

10. Teixeira, G. A., "Especificação Formal de Banco de Dados e suas Restrições de Integridade". Tese de Mestrado, PUC-RJ, 1979.

11. CODASYL DDL Journal of Development, June 1973 Report

12. Steel, T. B.,"Formalization of Conceptual Schemas" Proc 2nd SHARE Working Conference on DBMS, Montreal, Canada, April, 1976.

13. Hammer, N. M. and Sarin, S. K., "Efficiente Monitoring of Database Assertions" Proc. ACM SIGMOD (1978).

14. Stonebraker, M., "Implementation of Integrity Constraints and Views by Query Modification" Proc. ACM SOGMOD Conf. San Jose, Calif. (May, 1975).

15. Melo, R. N., "Projeto MIDAS: Relatório Intermediário", PUC-RJ (1979) (To be published)

16. CODASYL Database Task Group, April, 1971, Report.