DESIGN OF RELATIONAL VIEWS OVER NETWORK SCHEMAS

Carlo Zaniolo

Sperry Research Center, Sudbury, MA 01776

ABSTRACT

An algorithm is presented for designing relational views over network schemas to : (1) support general query and update capability, (2) preserve the information content of the data base and (3) provide independence from its physical organization. The proposed solution is applicable to many existing CODASYL databases without data or schema conversion. The particular declarations of a CODASYL schema which supply sources of logical data definition are first identified. Then the view design algorithm is derived on the basis of a formal analysis of the semantic constraints established by these declarations. A new form of data structure diagram is also introduced to visualize these constraints.

1. INTRODUCTION

This paper presents a rigorous solution to the problem of designing relational views which support general query and update capabilities over network schemas. Three objectives are of paramount concern in our approach. They are:

1. information preservation
2. updatability
3. data independence.

Let us consider information preservation first. This is needed for supporting general purpose data manipulation capability. Indeed a user must be capable of accessing through views all the information of interest (within his authorization domain). Thus the view must be information-wise equivalent to the underlying schema or that portion of interest.

Let us consider now the problem of specifying updates through a view (these include insert delete and modify operations). The simple data organization displayed by a view is often very different from the physical organization of data. This fact must be made totally transparent to a user who regards his view as real data. The property of a view to conform to real data behavior is here called updatability because updates supply the critical test case for it. To assess what the behavior of a view should be, let us consider the case of a user who is operating alone or has temporarily locked out updates to his data by other sources. Assume that this user first obtains a snapshot of his view content, then tells the system to carry out some updates and finally asks for a second snapshot of his view content to verify that the requested changes were made. What the user sees are the two snapshots and the constraints embodied in the view definition (e.g. keys of relations). Based on these, the user expects the system to perform as follows:

(a) Updates which respect the semantic constraints embodied in the view are accepted and carried to completion. However, updates which violate those constraints are flagged erroneous and rejected by the system.

(b) Updates accepted by the system produce the expected result in terms of view content. Therefore, the second snapshot differs from the first only for those additions, deletions or changes specified by the user.

The final topic in the list of objectives is data independence. This is a very important and pervasive concept in database systems. In this paper we address what can be regarded as the ultimate level of data independence: complete visibility of external data and their structure along with invisibility of internal data. External data are those of interest in the world outside of the data processing department, such as entities and attributes taken from the enterprise environment (e.g. employees, their salaries, their employment history). For the purpose of storing and processing external data efficiently, reliably and securely, the database system supplements it with internal data, internal structures and protocols. In a CODASYL system for instance, external data appear as values of data items in records. Internal data are supplemental data required by the CODASYL implementation. A view which hides internal data adds to user convenience and ensures that application programs written against this view remain valid independent of the underlying implementation.

2. BUILDING UPON EXISTING SCHEMAS

Basically, there are two approaches to the problem of defining relational views over CODASYL schemas. One is the top-down approach: starting from a relational view one designs a CODASYL schema to support this view. The second is a bottom-up approach; here one accepts the existing schema as the source of data definition and designs a set of relations which recasts this definition in terms of the relational model. This second approach was chosen here since it enables the addition of relational facilities to existing systems without converting the present databases or compromising ongoing applications. In conformity with this approach we will use the framework of the '73 DDL [CODA 73] on which current DBMS's are based. Thus when we speak about CODASYL we implicitly refer to the '73 J.O.D. However, much of what is said here can be carried over to the '78 specifications [CODA 78]. A CODASYL schema contains information about:

(a) external data and their structure
(b) physical organization of data
(c) processing and manipulation of data
(d) privacy of data.

A first source of external data definition in a CODASYL schema is the description of record types and of the data items making up these records. A second source is the association between owner record types and member record types established by the various set types in the schema. The nature of this association depends on whether this set was declared as (insertion is) MANUAL or AUTOMATIC and (retention is) MANDATORY or OPTIONAL. A third source is the declaration of certain data item combinations to be DUPLICATES NOT ALLOWED (DNA) in the location mode of a record. This establishes that no two record occurrences can have the same value for a DNA combination at any given time. A fourth source is the DNA declaration for sets. This establishes that no two member records in the same occurrence of a DNA set have identical values for the specified combinations of data items in the member record.

In order to visualize these four sources of data definition we have augmented the well known data structure diagram to a form which we call the data structure Z-diagram. Figure 1 gives the Z-diagram for the data base discussed in [TAYL 76]. Rectangular contours are used to represent record types; the record name appears next to the contour. The (elementary) data items are displayed inside the contour. Set types are represented by directed arcs from owner record types to member record types. If sets are AUTOMATIC MANDATORY we use solid arcs, otherwise we use dashed arcs. Data item combinations which are specified DNA in the location mode of a record are underlined inside the contour. For instance, this is the case for (PLN, PFN) of record PRES. A bar outside a contour denotes that all the sets incident to the bar are specified with the option DNA for the data item combination spanned by the bar. For instance, ADM# was declared DNA for set AH. For simplicity some of the data items appearing in [TAYL 76] have been omitted and names have been abbreviated. The structural complexity of the original example, however, is faithfully captured by Figure 1.

A NULL IS ALLOWED phrase can be attached to certain DNA set key declarations in the schema[1]. Then the uniqueness requirement for the DNA combination is waived when some data items in this combination are null[2]. Although such declarations have not been considered for our presidential database schema it is clear how they could easily be included in a Z-diagram. A simple solution, for instance, is to use dashed bars for DNA combinations where NULL IS ALLOWED and solid bars for others.

Our design algorithm is driven by the external data definition contained in CODASYL schemas, as captured by Z-diagrams. The objective of this algorithm is to make the information of type (a) (external data and their structure) totally visible in the view while hiding the schema information which falls under types (b) and (c). Some privacy information -type (d)- may also be attached to the view; however a discussion of this subject is outside the scope of this paper.

For the purpose of simplifying our discussion the following assumptions are made:

1. Records do not contain SOURCE or RESULT data items.
2. Singular sets are AUTOMATIC MANDATORY.
3. Repeating groups do not occur.
4. Multimember sets do not occur.
5. No record has a DIRECT location mode nor does it contain data items of type DATA-BASE-KEY.

Later we indicate how the design algorithm can be extended to remove these restrictions.

In our approach we start by modeling the CODASYL database by a set of time-varying relations, which capture content and structure of data via the explicit use of data base keys. Then we apply a succession of information preserving transformations to factor out data base keys and derive a set of relations in which only external data are visible.

3. SYNONYM STRUCTURE

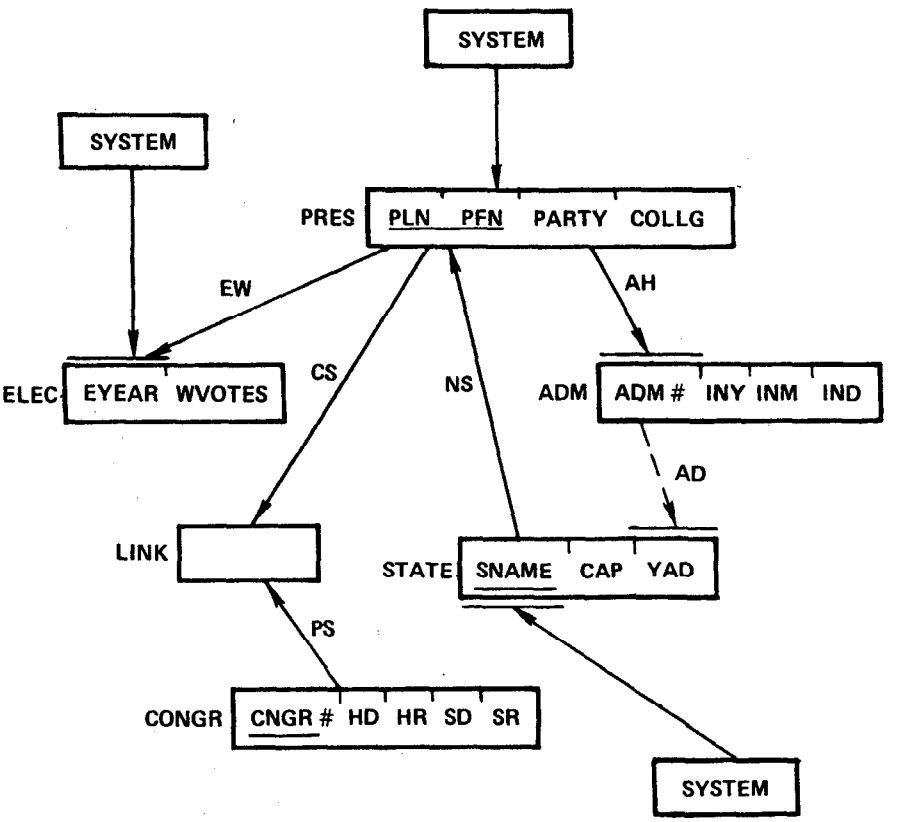
Every record occurrence in a CODASYL data base is uniquely identified by its data base key value. The set of data base key values for record type R will be denoted @R and called the data base key of R. A synonym of @R is a combination of data items from R and its predecessors (i.e. an owner of R or an owner of an owner and so on) which uniquely and non-redundantly identifies the occurrences of R. Thus synonyms can be used as surrogates for data base keys. Synonyms are defined by the DNA declarations in the schema. For instance, referring to Figure 1, we find that (PLN, PFN) is a

---

(1) This is the only instance in the CODASYL '73 DDL where a special treatment is prescribed for null data item values. Thus these are the only null values with which we need to concern ourselves in our relational views.

(2) Under a consistent interpretation of paragraph 7, page 3.69, and paragraph 11, page 3.77, of [CODA 73].

Figure 1. PRESIDENTIAL DATA BASE



LEGEND

PRES: president's record

- PLN: president's last name
- PFN: president's first name
- PARTY: president's party
- COLLG: president's college

ADM: administration record

- ADM#: administration number (relative to a given president)
- INY: administration inauguration year
- INM: administration inauguration month
- IND: administration inauguration day

STATE: state record

- SNAME: state name
- CAP: state capital
- YAD: year admitted (into the U.S.A.)

ELEC: election record

- EYEAR: election year
- WVOTES: election winner's votes

CONGR: congress record

- CNGR: congress number
- HD: house democrats
- HR: house republicans
- SD: senate democrats
- SR: senate republicans

LINK: link record

- EW: election won set
- CS: congress served set
- PS: president served set
- NS: native son set
- AH: administration headed set
- AD: admitted during set

synonym of the data base key of PRES. In fact no two occurrences of PRES records can have the same (PLN, PFN) value. Moreover, nothing in the schema declaration prevents PRES records from having the same PLN value or PFN value. Also we find that (PLN, PFN, ADM#) non-redundantly identifies occurrences of ADM since (PLN, PFN) identifies the occurrence of set AH while ADM# identifies the occurrence of ADM within this set. This triplet therefore is a synonym of @ADM.

The previous examples do not expose the kind of ambiguity which occurs in connection with schemas of the kind shown in Figure 2. The value of the pair (E#, EDT) qualifies at most one EHR record if set HS is searched but could qualify more than one EHR record if set SP is searched. Thus a statement of the type "(E#, EDT) is a synonym of @EHR" would be ambiguous. In order to remove this ambiguity we augment the data items from an owner record with the name of the set along which they were migrated. The set name serves as a "role name" and is separated from the owner item by a dot. Thus we write,

(E#.HS, EDT) is a synonym for @EHR,
(E#.SP, EDT) is not a synonym for @EHR.

The composition of an attribute A with a role name S, denoted (A).S is defined as follows:

$$(A).S = \begin{cases} A.S \text{ if A does not contain a role name} \\ A \text{ otherwise .} \end{cases}$$

This rule avoids the cascading of role names. If X=(A1, A2, ..., Ap) is a set of attributes then X.S = ((A1).S, (A2).S, ..., (Ap).S). Also if X and Y are two sets of attributes, for convenience we may write X,Y to denote X U Y. Also, we write X,A to denote X U{A} . Thus we have that

((PLN, PFN).AH, ADM#) = (PLN.AH, PFN.AH, ADM#)

is a synonym of @ADM. Also this combination migrated down AD and combined with YAD will be represented as:

((PLN.AH, PFN.AH, ADM#).AD, YAD) =
= (PLN.AH, PFN.AH, ADM#.AD, YAD)

Note that as long as items from each record type along the migration path are included, the role names uniquely identify the migration path. In figure 3 we give an algorithm for deriving synonyms.

SYNONYM DERIVATION ALGORITHM

1. [FIRST LEVEL SYNONYMS] The data item combination X is a synonym for @A iff
   (a) X is DNA in the location mode of A, or in a singular set having A as member.
   (b) no proper subset of X has property (a). If the DNA on X for the singular set has the phrase NULL IS ALLOWED attached, then X is a pseudo-synonym, otherwise it is a proper synonym.

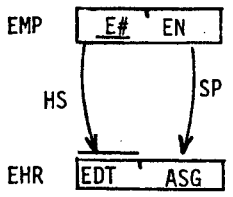
2. [ELIMINATION OF REDUNDANT DNA DECLARATIONS] Remove the DNA specification for each item combination X declared DNA for a set S if either of the following is true:
   (a) There exists a Y $\subseteq$ X which is specified DNA for the set S,
   (b) The data base key of the member record of S has a first level synonym Z $\subseteq$ X.

3. [LOWER LEVEL SYNONYMS] (X.S, Y) is a synonym for @B if the following two conditions are both satisfied:
   (a) The set S with owner A and member B has Y declared DNA and X is a synonym of @A
   (b) The role name S does not already appear in X (This condition pertains to schema cycles).

   (X.S, Y) is a pseudo-synonym when Y is a pseudo-synonym or when S is not AUTOMATIC MANDATORY or when a NULL IS ALLOWED phrase is attached to the DNA declaration. In every other case it is a proper synonym.

Figure 3. The Synonym Derivation Algorithm

The algorithm separates synonyms into two distinct classes: proper synonyms and pseudo-synonyms.



Legend

EMP: employee record
E#: employee number
EN: employee name

EHR: employee history record
EDT: effective date
ASG: assignment

HS: history set
SP: supervisor set

Figure 2. Employees, their previous assignments and their supervisors at each assignment.

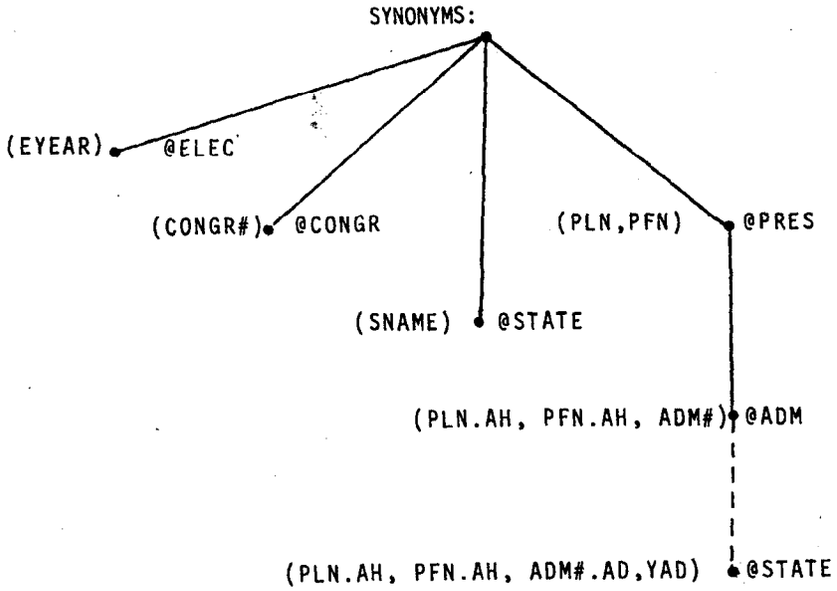


Figure 4. The synonym structure of the Presidential Database Schema

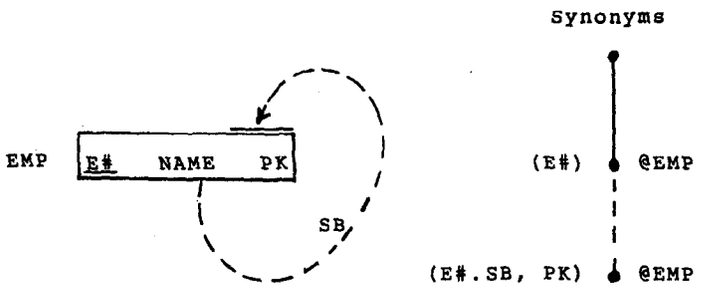


Figure 5. The ultimate hierarchy - For each employee his number (E#), his name (NAME) and all his subordinates (set SB). Each subordinate is ranked in a pecking order (PK) where there are no ties.

Pseudo-synonyms are those which have been contributed by set DNA declarations where the NULL IS ALLOWED phrase is attached or where the set is not AUTOMATIC MANDATORY; the remaining synonyms are called proper. The algorithm consists of three steps. The first step is concerned with the definition of first level synonyms. The second step eliminates redundant DNA declarations from non-singular sets. The third step recursively defines the lower level synonyms. We also assume that no inconsistency occurs in the declaration of NULL [NOT] ALLOWED for keys of set types. Thus if a combination X is specified NULL NOT ALLOWED in some set declaration then no subset of X is specified NULL ALLOWED somewhere else in the schema.

With reference to step 3 of the algorithm we say that X is a direct predecessor of (X.S, Y). Then a predecessor is defined as a direct predecessor or a predecessor of a direct predecessor. Figure 4 illustrates the hierarchial representation which we use to depict the synonym structure of a schema. Hanging from a common root we find the first level synonyms. At the lower level we find their successors. Solid lines lead down to proper synonyms; dashed lines lead down to pseudosynonyms. Condition (b) of step 3 avoids endless composition of synonyms which would otherwise occur in schemas with direct cycles such as the one of figure 5 (This schema actually uses an extension to CODASYL 73 included in CODASYL 78. It is used here since it supplies a simple and clear example of the effect of DNA in schemas with cycles). In this schema (E#), (E#.SB, PK), (E#.SB, PK.SB, PK),... all supply valid synonyms for @EMP. However only the first two need to be considered since the view design algorithm never uses synonyms which employs the same set more than once in the migration path.

4. THE RELATIONAL ANALOG

We begin by modeling a CODASYL database by a relational analog. This consists of a set of relations where both data items and database keys of records are visible. For each record type A in the schema the relational analog contains a relation (named after the record):

A (@A,A1,A2,...,Ap,@B1.S1,@B2.S2,...,@Bn.Sn) (4.1)

where: @A denotes data base keys of record-type A,

A1, A2, ..., Ap ($p \geq 0$) denote the data items of record-type A,

S1, S2, ..., Sn($n \geq 0$) denote the non-singular sets of which record-type A is a member,

B1, B2, ..., Bn denote the record-types owning these sets,

@B1, @B2, ..., @Bn denote the data base keys of these owners.

Since record-types A, B1, B2, ..., Bn need not be distinct the set name has been appended as role name to the data base keys of the owner records. Naming ambiguities are thus avoided.

The relational analog for the examples of figure 1, 2, and 5 are given in figures 6, 7 and 8, respectively (the meaning of the underlinings and

PRES (@PRES, PLN, PFN, PARTY, COLLG, @STATE)

ADM (@ADM, ADM#, INY, INM, IND, @PRES)

STATE (@STATE, SNAME, CAP, YAD, @ADM)
x x x x -

ELEC (@ELEC, EYEAR, WVOTES, @PRES)

LINK (@LINK, @PRES, @CONGR)

CONGR (@CONGR, CNGR#, HD, HR, SD, SR)

Figure 6. The relational analog for the presidential data base.

EMP (@EMP, E#, EN)

EHR (@EHR, EDT, ASG, @EMP.HS, @EMP.SP)

Figure 7. The relational analog for the example of figure 2.

EMP (@EMP, E#, NAME, PK, @EMP.SB)
X X X X -

Figure 8. The relational analog for the example of figure 5.

the rows of "x" and "-" will be given later). The time-varying content of relation A (see (4.1)) is defined as follows. There is a tuple t for each occurrence r of record A in the data base. The @A value of t is the data base key value of r. The A1, A2, ..., Ap values of t are the values of the corresponding data items in r. For each $0 \leq i \leq n$ the t-value of attribute @Bi.Si is the database key value of the Si-owner if one exists, the null value otherwise. Figure 9 gives the Z-diagram and sample content for a simple data base. Figure 10 gives the relational analog and corresponding content. Positive integers have been used to represent data base key values. In figure 10 a null value is displayed by a dash "-". A tuple or a subtuple of a relation is fully defined when it contains no null value.

The time-varying contents of relations in our analog obey some time-independent constraints dictated by the data base schema declarations. The concept of (candidate) key for a relation supplies a useful construct to express these constraints [CODD 72, DATE 77]. An attribute combination X of a relation A is said to be a candidate key for A when the following two conditions are satisfied:

1. Uniqueness: no two tuples in A have the same fully defined X-value.

2. Minimality: No proper subset of X has this property.

Note that the uniqueness requirement is waived for a tuple in which the value of one or more key attributes has a null value. This policy reflects the constraints generated by schema DNA options for sets which are not AUTOMATIC MANDATORY or where the NULL IS ALLOWED phrase is attached. For instance

in figure 9 the set S is DNA on F. Thus (@R1.S, F) is a candidate key for R2; no two tuples of R2 can have the same value for both @R1.S and F when neither is null. However, when @R1.S is null (the R1-owner is missing) two or more tuples could share the same F-value. Also two or more tuples can share the same @R1.S value if F is null. Thus the previous definition of candidate keys ensures total consistency between the treatment of null values modeling a missing owner in MANUAL or OPTIONAL sets and the treatment of null values occuring in DNA set keys where NULL IS ALLOWED is specified. We represent both kind of null values with the same distinguished symbol "-". Since such null values are allowed only for certain attributes in our relation, one must specify them as part of the integrity constraints of the relational analog.

A combination X of attributes of a relation R is an allowable null pattern for R if R can contain tuples where all and only the X-attributes have null values [ZANI 77]. An allowable null pattern X is a minimal null pattern for R if no subset of X is also an allowable null pattern. We only consider relations where every null pattern is either minimal or a union of minimal patterns. Thus we give only the minimal null patterns, the other patterns being immediately inferrable from these. In relation A of (4.1) there is a minimal null pattern @Bi.Si for each set which is MANUAL and/or OPTIONAL. There is also a minimal null pattern for every attribute in a DNA combination where the NULL IS ALLOWED phrase is attached. Thus @ADM is a minimal null pattern for relation STATE of figure 6 and @EMP.SB is a minimal null pattern for EMP in figure 8. Then there are two minimal patterns, @R1 and F for R2 in figure 10. The other relations in our examples have no null patterns. As seen in these figures minimal null patterns are described by rows where "x" stands for any non-null value and "-" for the null values. The null patterns for our relational analog reduces to a single attribute. This is not true for the relational views which we consider later.

Candidate keys will be called proper if their value is always fully defined (i.e. they do not intersect any null pattern); they will be called pseudo-keys otherwise. Directly from our definition of keys we have that the relational

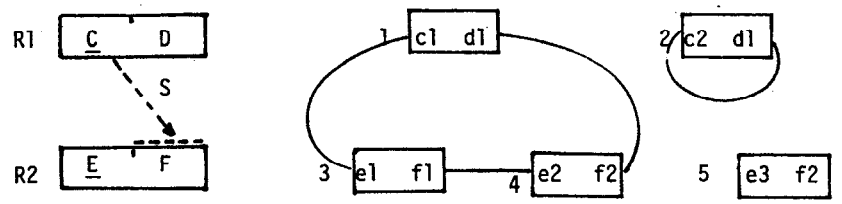


Figure 9. The Z-diagram and sample content for a CODASYL data base

Relational Analog: R1 (@R1, C, D)

R2 (@R2, E, F, @R1.S)

x x x -

x x - x

Sample Content: R1 (@R1, C, D)

1 c1 d1

2 c2 d1

R2 (@R2, E, F, @R1.S)

3 e1 f1 1

4 e2 f2 1

5 e3 f2 -

Figure 10: The relational analog and its content for the example of Figure 9.

analog contain three types of keys, each defined by the following rules:

(a) @A is a proper key for relation A,

(b) every attribute combination of a relation A which is a proper synonym or a pseudo-synonym for @A is, respectively, a proper key or a pseudo-key for A.

(c) if X is a DNA declaration for the non-singular set S with owner B (step 2 of the synonym derivation algorithm has removed redundant DNA declarations), then (@B.S,X) is a candidate key for A. This is a proper key if S is AUTOMATIC MANDATORY and the NULL IS ALLOWED phrase is not present. Otherwise this is a pseudo-key.

Note how the keys for the analogs in figures 6, 7, 8 and 10 obey these rules. As usual keys are displayed by suitable underlining of the attributes comprising the key; distinct keys are denoted by different style of underlining.

The content of our relational analog obeys an important constraint which we call the foreign key constraint. A foreign key for a relation R is simply an attribute combination of R which is a key for another relation. Now, candidate keys can be regarded as unique identifiers of real world objects. In a system such as our relational analog where no two relations share the same candidate key one can view each relation's role as defining and describing the objects for which the candidate key of the relation serves as unique identifier. Thus foreign key values become references to objects described and defined in other relations. Null foreign key values can be used to model undefined references. Yet when a reference to a unique object is made through a fully defined foreign key value one expects this object to be defined and described in a relation with such a key. Formally therefore:

Foreign Key Constraint: If a candidate key X of relation R1 is also an attribute combination of relation R2 then every X-value appearing in R2 must also appear in R1 (i.e. the X-projection of R2 must be a subset of the X-projection of R1).

The foreign key constraint was first advocated in [SMITH 77] for the purpose of making the relational data model suitable for data base abstraction and aggregation. Our relational analog, where foreign keys of relation A are simply data base keys of owners of A, uses this constraint.

5. THE DESIGN ALGORITHM

Our design algorithm performs a two-step transformation on the relational analog. We refer to the relational analog described in the last section as the r0-analog, to the one obtained at the end of step 1 as the r1-analog and to the one at the end of step 2 as the r2-analog. The view design algorithm is given in figure 11. Associated with the transformation of attribute sets of our relations there is a corresponding transformation on (1) the content of these relations and (2) the keys and the minimal null patterns of these relations. Thus if t denotes a tuple of A before the replacement of @B by its synonym X and t' a tuple of A after this replacement, t' is equal to t

VIEW DESIGN ALGORITHM

Step 1: [top-down synonym substitution]. If a relation A contains an owner data base key @B.S, and relation B contains a proper synonym of @B, say X, then in relation A replace the attribute @B.S by the attribute(s) X.S. Repeat this operation until no more such replacement can take place.

Step 2: For each relation A project out the attribute @A.

Figure 11: The algorithm which transforms the relational analog into the final view.

in every attribute except the attributes in X.S. The X.S-value of t' is simply the X-value of the tuple of B with a @B-value equal to the @B.S-value of t if this is not null; otherwise every X.S-value of t' is null. Moreover if (@B.S,Y) is a key of A before the transformation then (X.S,Y) is a key of A after the transformation. If @B.S was a minimal null pattern before the transformation then X.S is a minimal null pattern after the transformation (either all or none of the X.S-values can be null at once).

Because of the one-to-one correspondence between the X-values and the @B values these transformations are clearly correct and invertible. Thus there exists a one-to-one correspondence between the r0-analog and the r1-analog. Note that this correspondence may be lost if data base keys are replaced by their pseudosynonyms. Also note that the rules (a),(b) and (c) to compute the candidate keys of relations remain valid after each replacement step has taken place. Thus they hold for the r1-analog as well as for the r0-analog.

Now observe that step 1 ensures that synonyms are always used in a top down fashion in the synonym tree. A synonym of @A is used only after it has become a proper key for A; thus all its predecessors have become keys for their respective relations. As shown in [ZANI 79] this is essential to preserve the foreign key constraints. Clearly if a data base key of some owner record has more than one synonym there may exist more than one r1-analog for any given schema.

Let us now assume that there exists a proper synonym for the data base key of each owner record in the schema (more about this assumption will be said later). Then the only data base key appearing in a relation A is @A. The second step of the design algorithm simply removes this attribute. The content of A in the r2-analog is therefore the projection of the content of A in the r1-analog. X is a minimal null pattern in the new A relation if and only if it was a minimal null pattern in the old A. A proper key or a pseudo-key for the old A, except @A of course, becomes a proper key or a pseudo-key of the new A. If the old A had no key except @A then the whole attribute set of the new A becomes its key. The r2-analogs for our examples are given in figures 12, 13, 14 and 15. Role names were dropped when no ambiguity occured.

6. RELATIONAL VIEWS

The r2-analog supplies our basic relational view.

PRES (PLN, PFN, PARTY, COLLG, SNAME)

ADM (PLN, PFN, ADM#, INY, INM, IND)

STATE (SNAME, CAP, YAD, PLN, PFN, ADM#)
x x x - - -

ELEC (EYEAR, WVOTES, PLN, PFN)

LINK (PLN, PFN, CNGR#)

CONGR (CNGR#, HD, HR, SD, SR)

Figure 12. The final view for the Presidential Data Base of figure 1.

EMP (E#, EN)

EHR (E#.HS, EDT, ASG, E#.SP)

Figure 13. The final view for the schema of figure 2.

EMP (E#, NAME, PK, E#.SB)
x x x -

Figure 14. The final view for the schema of figure 5.

R1 (C, D)

R2 (E, F, C)
x - x
x x -

Figure 15. The final view for the schema of figure 9.

Relations in these views cut across record boundaries to capture logically related data items. In general a view relation A has as attributes the data items from record A plus data items migrated down from its predecessors (owner, owners of owners, etc.); the design algorithm selects the data items to be migrated on the basis DNA declarations in the schema. For instance in the STATE relation of figure 12 we find SNAME, CAP and YAD from the STATE record, plus (PLN, PFN) from the president record and ADM# from the administration record. The presence of a STATE record without a ADM owner is denoted by null values for PLN, PFN, ADM#.

These relations supply the user with a congenial interface to support high-level relational data manipulation languages (DMLs) such as SEQUEL or Query by Example [DATE 77]. Since access paths and physical organization of data is invisible in the view, responsibility for efficient execution is taken over by the system. DML optimization routines will take into account the characteristics of the requested transactions and the organization of the underlying database (including the clustering of records in pages, the presence of indexes, the availability of owner pointers and prior pointers in records and other features specified by schema declarations ignored in our Z-diagrams), to dynamically select access paths and perform aggregate operations to ensure efficient processing. Further discussion of the DML optimization problem is outside the scope of this paper.

Relational languages have the closure property which ensures that the result of queries on relations is still a relation. Thus a query statement can be interpreted as the definition of a view. This enables a relational DML user to derive other views from the basic ones generated by our design algorithm. Also the user may want to include in his relations some data items in the schema whose value was specified to be the RESULT of some data base procedure. There is no guarantee, however, that these other views will support update requests correctly [DAYA 78]. A simple example will help to illustrate some of the problems in this area. Say for instance that our view over the schema of figure 9 consists of a single relation R12. Such a view could have been constructed directly from the network schema as described in [ZANI 77]. Alternatively, the basic relational view of figure 15 could have been derived first, then R12 constructed from this using an or-join; this second approach is also described in [ZANI 77]. In either case the content of R12 is filled as per figure 16. R12 contains a tuple for for each occurrence of R2; the E and F value of this tuple are taken from this record occurrence, the C and D values are taken from its owner record; they are null if such an owner does not exist. Also R12 contains a tuple such as (-, -, c2, d1), for each memberless owner in R12. Thus R12 preserves the information content of the underlying database but it is not updatable. The keys for R12 are E and (F, C). These fail to specify that R12 also obeys the constraint that every two tuples having the same C-value must also have the same D-value. Now say that the insertion of a new tuple (e4, f4, c1, d3) is requested. Although such a request does not violate the constraints embodied

in the view (i.e. those expressed by the keys and the allowable null patterns of R12), serious problems occur if one tries to carry it out. If no previous record with a C-value of c1 exists then one can simply add a new record R1 with content (c1, d3), then add a new record R2 with content (e4, f4) and finally let the former own the latter in an occurrence of S. However, since an R1 occurrence with a C-value equal to c1 already exists, such a policy would result in an error message for the violation of a DNA condition on C. As an alternative one could try to see if a R1 occurrence with a C-value of c1 already exists and

R12	(E,	F,	C,	D)
	e1	f1	c1	d1
	e2	f2	c1	d1
	e3	f2	-	-
	-	-	c2	d1

Figure 16. A non-updatable view relation.

then let it be the owner of a new R2 occurrence with content (e4, f4). The result of this policy upon the view would be the addition of the tuple (e4, f4, c1, d1) instead of the requested (e4, f4, c1, d3). For this last tuple to appear in R12 the content of the R1 record with a C-value of c1 must be updated to (c1, d3) and this must become the owner of the R2 record with content (e4, f4). This, however, would have the effect of changing the first two tuples in the old view (figure 16) from a D-value of d1 into a D value of d3. In summary the seemingly correct user request will either be rejected or it will produce unexpected results. In order to predict and/or understand the update behavior of this view the user must look at the underlying schema and possibly understand the internal behavior of the view support subsystem. Such a view would fail to insulate the user from the complexity of the underlying system.

Clearly the previous problems could be cured by employing more powerful and complex primitives to define logical constraints (e.g. allowing the specification of functional dependencies other than those implied by keys). But this approach leads to more complex and less friendly views. A much better solution, instead, is to use the view of figure 15. This is free of the previous update problems since the keys of these relations clearly indicate that there is at most one C-value associated with any given D-value. Indeed the view design algorithm ensures that all the integrity constraints defined by the Z-diagram are fully captured by the allowable null patterns and the keys of the relations. According to these constraints, moreover, when all tuples are fully defined then the view relations are in Fourth Normal Form [FAGI 77] (a definition of Fourth Normal Form in the presence of null values is not available). Moreover it is always possible to translate any correct update on these views into an equivalent sequence of DML commands executable against a COBOL or a PL/I subschema. Algorithms for single tuple DML translation have been obtained and tested on a CODASYL DBMS. A more detailed discussion on this topic exceeds the scope of this paper.

In implementing delete and update requests a choice can be made between two policies which are discussed next. Assume for instance that deletion of a tuple from relation STATE in figure 12 is requested. If the name of this STATE is referenced by some tuple of PRES then one can either disallow the request (rejection) or delete all such tuples from PRES (propagation). If propagation is chosen then deletions from PRES may in turn propagate to relations LINK, ELEC, ADM; however a deletion on ADM does not propagate to STATE since null values are allowed for its foreign key (PLN, PFN, ADM#). As discussed in [SEVC 78] a proper choice between rejection and propagation can only be based on the semantics of the case at hand in relation to the enterprise environment. For the presidential data base, for instance, a propagation policy is probably meaningless. For the example of figure 13 instead it could mean that when an employee leaves the company his employee-record is also discarded. However, a rejection strategy reduces the risk of jeopardizing the integrity of the system and also ensures that deletion and insertion are inverse operations as in the usual relational framework. The propagation policy is, in fact, not generally applicable to insertion. For instance assume that the insertion in ELEC of a new president name with the year in which he was elected is requested. If no PRES record with such a name is available one could consider creating a new one, but two problems would occur. The first is that every remaining data item in the record would have to be null, and such an option might not be available. The second problem is that we need to find an owner for this record in NS. Since all we know about this president is his name, there is no reasonable way to choose an owner in NS (i.e. his native state).

The foreign key constraint limits the choice of fully updatable subviews that one can construct by subsetting the given set of relations in the r2-analog. If a subview contains along with relation A all the relations corresponding to owners of record A then every constraint restricting addition of new tuples to A is captured by this subview. Likewise, if all relations which correspond to members of A in AUTOMATIC, MANDATORY sets are also part of the subview then every constraint which may limit the deletion of tuples from A, under the rejection policy, is captured by the subview. However these member relations need not be included if a propagation policy is followed. Similar rules can be derived for operations which modify tuples in relations.

7. DISCUSSION

In this section we review the assumptions which limit the generality of the previous approach and propose extensions to overcome them.

Firstly let us reassess the performance of our views in terms of information preservation. As described in the introduction the only information of interest is external data. Now a relation A in the r0-analog totally preserves the information associated with each record type of the schema. Moreover it identifies the owners of each occurrence of the A-record, for each set in which this record serves as a member. Thus the composition of the various set occurrences is also preserved by the r0-analog. However, a programmer through the COBOL subschema can also determine the

order of the member records in a set occurrence. If the set was declared SORTED by the defined DNA keys then the order can be derived from the content of the member records. If the order is declared IMMATERIAL or by DATA-BASE-KEY no external significance can be attached to the order of the members in the set. In other cases (e.g. those denoted by the key words FIRST, LAST, NEXT, PRIOR) the order reflects the processing history of the set. Thus it could be used to convey external information not inferrable from other data explicitly stored in the data base (e.g. the temporal sequence in which invoices were received where no timestamp for these invoices is recorded). It is hard to believe that under these premises such information can be regarded as reliable and that there is a stringent need to preserve it in the view. However, if the designer decides that this is the case, then he can use the solution discussed later in this section.

The first step in the view-design algorithm is clearly information preserving since it is reversible. After the second step, however, the values of data base keys are lost. Since we have excluded DIRECT records we can safely assume that the data-base-key values are of no direct interest to external users; thus no information is lost. Notice, however, that more than the value of data base keys may be lost where for some record A no proper synonym exists for @A. Indeed any two tuples of A in the r1-analog which are identical except in the @A value will collapse into one tuple in the r2-analog. This is the case of our relation LINK which fails to distinguish between multiple occurrences of a LINK record with identical PRES and CONGR owner records. Once again it takes the designer's judgement to decide whether these duplicate LINK records are or are not supposed to be in the actual data base. If no duplicate records are expected in the data base since the current applications implicitly respect this constraint, then two courses of action are possible. The first is to leave the schema as it is and ensure that the relational DML support routines enforce this constraint. The second alternative is to include this constraint explictly in the schema. For records such as LINK, where the synonym data items are drawn from more than one owner record, one needs to use SOURCE items. Thus, for example, CNGR# from CONGR can be added as SOURCE item to the LINK record and set CS can be specified DNA on this data item (or conversely (PLN, PFN) can be included in LINK and a DNA on it can be specified for set PS). Thus SOURCE data items, although not included in many present system implementations, would greatly enhance the data definition power of CODASYL schemas and make them more amenable to support multiple views. This was noted and used in [JOHN 78] to derive an architecture which ensures commonality for relational and network DDL and DML. Note that if VIRTUAL SOURCE items are used, no conversion would be required for present data bases.

The approach proposed in this paper is robust as it can be extended to remove simplifications and limitations which were previously introduced. Let us, for example, assume that the designer wants to preserve information regarding the order of member records in occurrences of set S. Say that A is a member of S and B is its owner. Then relation A of the r0-analog will contain the attribute O.S along with B.S, where B denotes the owner of S and O.S denotes the position of an A-record among its cohorts in a occurrence of S, unless S is SORTED on a DUPLICATES ALLOWED combination X, in which case O.S denotes the position of the A-record relative to its set occurrence cohorts sharing the same X-value. In the first case, then, S will be assumed DNA on O.S; in the second case S will be assumed DNA on (X,O.S).

Repeated applications of this technique can be used to ensure that a proper synonym exists for each record in the schema (for records whose location mode is CALC using data item combination Y with DUPLICATES ALLOWED (Y,O) can also be used as the synonym; O denotes the position of the record in the calc chain). The conditions which limit the applicability of step 2 of the design algorithm are thus removed. The proposed solution reflects the current usage of data where a record which cannot be uniquely identified by value will be identified by the order in which it appears in the processing sequence. After adding the various O-columns in the r0-analog the view design algorithm can be applied without any modification. Since the various O-columns are now part of the final view the user is exposed to some implementation dependent information. Thus there is a potential loss of data independence. Moreover when these relations are updated the O-columns behave as contiguous sequences of numbers (e.g. deletion of a row may cause an automatic decrease of the O-values of the following rows). Thus syntactic constructs must be included to differentiate the O-columns from the others. In brief these views lack some of the qualities posessed by the views described in the previous section. Nevertheless they are simpler than network subschemas and they can support high level relational DML. Thus they can be used to model recalcitrant portions of existing schemas when a modification of these schemas is not acceptable. Extensions to remove assumptions 1 through 5 of section 2 are also available. For brevity we will not discuss them here as they are documented in [ZANI 78] and [GOLD 79].

8. CONCLUSION

In this paper we have presented a viable solution to the problem of designing relational views over network schemas to support general query and update DML. This capability brings significant benefits to a CODASYL DBMS in the area of data independence, high level DML and ease of use. The approach taken is totally evolutionary in as much as it does not require conversion of present data bases or translation of current application programs. This view design approach has been applied to a number of existing data bases with encouraging results [GOLD 79].

An improved understanding of the role of the various CODASYL declarations in defining the logical structure of external data has also followed from this work. The designer can benefit by improving his logical schema design (if nothing else by eliminating redundant DNA declarations which only slow DML execution).

This research brings the relational and the network

model one step closer. It also indicates how CODASYL can evolve into a system which effectively supports multimodel external schemas; this topic is treated more extensively in [ZANI 79]. Finally this work confirms the important role, recognized by previous authors, that null values and foreign key constraints can play in data base relations.

ACKNOLEDGMENTS

I would like to aknowledge the important contribution of Jay Goldman who, among other things, has proposed the simplified design approach used in this paper. I am also grateful to Murray Edelberg for constant encouragment, penetrating discussions and numerous improvements. Finally I would like to thank Dan Connelly for many consultations and discussions on CODASYL DBMS's.

REFERENCES

CODA 73 "CODASYL Data Description Language Journal of Development, June 1973," NBS Handbook 113.

CODA 78 "CODASYL 78 Data Description Language Journal of Development, January 1978," Information Systems, Vol. 3, No. 4, 1978.

CODD 72 Codd, E. F. "Further Normalization of the Data base Relational Model," Database Systems, Courant Computer Science Series, Vol. 6, Prentice Hall, New York, 1972.

DATE 77 Date, C. J. "An Introduction to Data Base Systems," Addison-Wesley, 2nd Ed., New York 1977.

DAYA 78 Dayal, U. and P. Bernstein, "On the Updatability of Relational Views," Very Large Data Base Conference, Sept. 13-15, 1978, West Berlin, Germany.

FAGI 77 Fagin, R. "Multivalued Dependencies and New Normal Form for Relational Databases," ACM Trans. on DataBase Systems, Vol. 2 ,No. 3, Sept. 77.

GOLD 79 Goldman, J. "Automated Generation of Relational Schemas for CODASYL Databases," Sperry Research Center Research Paper SCRC-RR-79-13, March 79 (submitted to VLDB '79).

JOHN 78 Johnson H. R., Larson J. A. and J. D. Lawrence, "A Common Data Base Architecture," IEEE Workshop on Data Management and Storage Hierarchies, Sept. 6-8 1978, Lake Arrowhead, California.

SEVC 78 Sevcik, K. C. and A. L. Futardo "Complete and Compatible Sets of Update Operations," Proceedings of the ICMOD 78 Conference on DBMS, June 29-30 1978, Milan, Italy.

SMIT 77 Smith J.M. and C.P. Smith "Database Abstractions: Aggregation," CACM Vol.20, no.6, June 1977.

ZANI 77 Zaniolo, C. "Relational Views in Data Base Systems: Support for Queries," COMPSAC77, November 8-11, 1977, Chicago, Illinois.

ZANI 78 Zaniolo, C. "Relational Views over a Network Schema," Sperry Research Center Report, SCRC-RR-78-37, July 1978.

ZANI 79 Zaniolo, C. "Multimodel External Schemas for CODASYL Data Base Management Systems," IFIP TC-2 Working Conference on Data Base Architecture June 26-29, 1979, Venice, Italy (North Holland).