PROCEEDINGS

9

October 9–11, 1985

compsac 85

The IEEE Computer Society's Ninth International

Computer Software & Applications Conference

Americana Congress Hotel, Chicago, Illinois

ISSN 0730-3157
IEEE CATALOG NO. 85CH2221-0
LIBRARY OF CONGRESS NO. 83-640060
IEEE COMPUTER SOCIETY ORDER NO. 643
ISBN 0-8186-0643-6

COMPUTER SOCIETY PRESS

IEEE COMPUTER SOCIETY

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

IMPLICIT JOINS IN THE MULTIDATABASE SYSTEM MRDSM

Witold LITWIN

INRIA, 78153 Le Chesnay, France

ABSTRACT

Implicit joins are joins that determination is left to the system. This functionality simplifies query formulation. We present a method for determination of implicit equijoins and the corresponding implementation within MRDSM. The method is more general than others with similar aims, in the sense that it works for usual relational schemas and avoids the logical navigation in more cases.

1. INTRODUCTION

When a query involves attributes from different relations, a typical relational system requires one to indicate all the corresponding interrelational joins. In particular, one has to indicate the equijoins. One frequently considers that this property of the relational mode, typically called logical navigation, annoys users. First, since it is unnatural to explicit natural equijoins and since shorter queries are usually felt simpler. Next, since the query expression depends then on the database structure i. e. its decomposition into relations. However, the ability to define the logical navigatation may also be considered as an interesting property of the model, as it allows users to express their informal queries (wishes) particularly precisely /COD79/.

A user may, in particular, wish data from a collection of usually independently administrated databases, managed by a multidatabase system, like MRDSM /WON84/, /LIT85/. Such databases may be relational, as in MRDSM or MERMAID /TEM84/, or at least conceptually relational while internally heterogeneous /BRE84/, /DAY84/, /LIT83/. They may model similar universes or even the same one, through to some extent different schemas. A user wish may then correspond to a set of queries, one per database, as in examples in Section 2 below. If each query has to explicit a particular logical navigation, one may need to formulate as many different queries as there are databases. Otherwise, one formulation may suffice and as it is shown below this formulation may become the *unique* query to be specified by the user. Since many databases may be involved, savings in user effort may be considerable. To avoid the logical navigation in the multidatabase environment should thus be of high interest.

While the multidatabase case is new, the (mono)database one already generated an important research work. This work is mainly based on the idea that a relational database should be considered as only one, so-called universal relation /SAG81/, /KUC82/, /MAI83/, /KEN84/, /MAI84/, /ULL84/, /KOR84/, /IMI84/, /WAL84/,... The need for the logical navigation disappears then automatically. However, other inconveniences appear /KEN84/. In particular, the following ones :

- a monorelational database may be considered as a messy model of the universe.
- one may need to rename many attributes since all attributes must have different names.
- one may frequently need to enumerate in a query a long list of attributes, that otherwise may be all attributes of a (smaller) single relation. In a typical multirelational system it suffices then to invoke only the relation name or a special character like '*' in SQL meaning "all attributes".
- some queries lead to unnatural results, unless one reintroduces several relations (maximal objects). Even so, result of some queries still remain unnatural. Also, the logical navigation reappears for queries encompassing several relations.
- there is no, at least yet, possibility of updates.

Below, we present an alternative method. We do not assume anything particular about the database schema that is, thus, a *usual (multi)relational one*. The idea is therefore that the system determines by itself some of interelational joins that should otherwise be in the query. We call such joins implicit joins. We show how the implicit equijoins may be found through an analysis of semantic dependencies among attributes. We discuss the corresponding algorithmic for both the database and the multidatabase cases. We also overview the implementation within MRDSM.

We called our method Implicit Joins method. Its following properties should reveal interesting :

- a user may consider a the database as one or several relations,
- relation names may be used,
- attributes in different relations may have the same names,
- updates may be performed,
- multidatabase queries may be formulated,
- queries that lead to unnatural results and/or to additional constructs in a universal relation database schemas, do not lead to such side effects in our approach,
- query language may be fully compatible with any typical relational language,
- if the way in which the system determined the implicit joins is not the one the user had in mind, the query may always be formulated in the usual precise manner.
- joins other than those corresponding to the logical navigation may potentially be rendered implicit as well.

0730-3157/85/0000/0495$01.00 © 1985 IEEE

Section 2 recalls some features of the multidatabase manipulation language of MRDSM, called MDSL /LIT84b/, that are needed by the method. Section 3 discusses the concept of implicit joins and presents algorithms that determine them in various cases. Section 4 discusses the implementation aspects. Section 5 compares our method to related work. Section 6 concludes the paper.

2. MULTIDATABASE MANIPULATION LANGUAGE

A multidatabase system allows users to formulate multidatabase queries and, eventually, to define multidatabase dependencies /LIT82/... /LIT85/. Such queries and dependencies address more than one database. For instance the following queries to databases presented in Annex are multidatabase queries /LIT84a/ :

(1) - Select from the database **michelin** and from the database **cinemas** all restaurants and cinemas at the same street.
(2) - Select from the databases **michelin** and **gault-m** all restaurants that are chinese according to at least one database.
(3) - If the value of the attribute **street** is 'Etoile' in all the above databases, then update it to 'Ch. de Gaulle'.

In order to allow users to formulate multidatabase queries, current (monodatabase) manipulation languages have to be extended. Manipulation languages for multidatabase queries were called multidatabase manipulation languages (MML) /LIT82/. Features of relational MMLs are analysed in /LIT84a/. Most of these features characterize the manipulation language MDSL of the prototype relational multidatabase system MRDSM /LIT84b/. The overall goal of MDSL was to render *simple* the expression of multidatabase wishes. The following meaning was adopted for the concept of simplicity :

- one wish should lead to one query.
- a query is simpler when it is shorter, provided it stays understandable to the user.

This meaning seems well correspond to the natural perception of the concept. For instance, the relational queries are simpler than navigational ones, the universal relation interface may be simpler than the corresponding (typical) relational one etc.

The analysis show in particular that in the relational multidatabase environment it is useful to introduce the following possibilities for an MML :

(a) - the concept of multiple identifiers. While a unique identifier designates in the selection expression of a query one one attribute or one relation, a multiple identifier designates several objects bearing the same name. Current relational languages know only unique identifiers. MDSL allows also queries with multiple ones /ABD83/. Such a query, let it be Q, is considered as the set of all queries, let them be q that :

- q basically correspond to all possible replacements of the all multiple identifiers with the underlying unique ones,
- however, only the replacements called in /LIT84/ pertinent to a database are taken to the account (usual relational queries are always pertinent).
- also, replacements leading to queries called incomplete queries in the next section, are taken to the account only if there is no replacement leading to, so-called below, complete queries (usual relational queries are always complete)
- in particular, if Q asks for a modification, then, as usually, -select clause of q must concern one relation at a time.

(b) - the possibility of usage of attribute name alone. This, either as a unique identifier or as a multiple one.

Ex 1. The wish (2) may be formulated in MDSL as follows :

```
open michelin gault-m                                (4)
-select * -where (type = 'chinese')
retrieve
```

In the selection expression of this query, type is a multiple identifier that designates both michelin.r.type and gault-m.r.type. The system considers therefore (4) as the set of two "normal" queries :

```
 open michelin                                     (4.1)
-select r -where (r.type = 'chinese')
retrieve

open gault-m                                      (4.2)
-select r -where (r.type = 'chinese')
retrieve
```

(4) is in our sense, as well as for most users, simpler than (4.1) plus (4.2). The difference would increase with the number of databases involved. Multiple identifiers avoid here one kind of logical multidatabase navigation. Note that a restaurant may figure in only one of databases and it may be chinese for only one of databases. Note also that the relations produced by (4) would not be union compatible, as their attributes would differ.

Ex 2. Consider bank database (Annex and Fig 3) that is that from /MAI83/, but without the universal relation interface. Consider the wish "retrieve all customers" noted (Q2) in /MAI83/ that had no expression in the universal relation model. In MDSL, the corresponding query would simply be :

```
-select c
retrieve
```

The response would be the set of relations, indicating not only all values of c, but also, through the value position, the kind of the customer (having a loan, or, might be also, an account etc.). If one does not wish to know these details, one should apply the standard function union :

```
-select (union (c*))
```

In this case only one relation being the union of the above ones will appear.

Consider on the other hand the database s-p (see Annex) and the wish : "delete the supplier with s# = '123'. This wish may be formulated as follows :

```
-select s# -where (s# = "123")                 (5)
delete
```

6

s# is here a multiple identifier of s.s# and of sp.s#. (5) replaces two usual DSL queries. Note that (5) is not only simpler, but that it also preserves the referential integrity between s and sp.

A query where all the identifiers are unique is called, in our terminology, elementary query. If some identifier are multiple (or the query involves so-called in /LIT84b/ semantic variables), then we speak about a multiple query. Both, elementary and a multiple queries may be monodatabase queries, or multidatabase queries.

3. IMPLICIT JOINS

3.1 The idea

We call explicit joins all joins that figure in a query, typically through the corresponding join clauses within -where clause. In contrast, a join is implicit if one does not formulate it, although it should exist if the query was a typical one. Implicit joins simplifies the formulation of both mono and multidatabase query. In particular the logical navigation, corresponding basically to explicit equijoining of key attributes, may be avoided.

Ex 3. The wish "select (from the database s-p) the cities of suppliers of red parts" may in MDSL be formulated as follows :

```
-select (city)                                  (6.1)
-where  (color = "red") &
(s.s# = sp.s#)  &  (sp.p# = p.p#)
```

If the last two joins could be implicit, the expression would be simpler :

```
-select (city) -where (color = "red")     (6.2)
```

Ex. 4. Consider a database s-p' where the attributes of relations s, p, sp of s-p are all in one (universal) relation, let it be s. Note that at least one of attributes name must then be renamed. Consider then that one wishes to select all cities of suppliers of red parts, known to s-p and/or to s-p'.

The query (6.2) would then be meaningful for s-p'. For s-p in contrast, if the joins had to be explicit, one would need (6.1). The wish expression would then need two different queries. If joins in (6.1) could in contrast be implicit, then (6.2) would be an adequate expression also for s-p. It would then suffice for the whole wish as only one multiple query,.

3.1 Basic concepts

3.1.1 Relational database

As usually, we consider that a relational database is a set of relations R_i ; i=1..m. The number of elements of each Ri is assumed finite and typically varying. Each relation relates some attributes ($a_{i1}, .. a_{ij}$). Each attribute takes values within a set called domain d_k ; k=1..n. A domain is an atomic model of some real concept. It thus is characterized by some semantic and is not only "simply a set of values" /ULL80/. Attribute names are sometimes called domain role names /COD71/.

For each relation, some of its attributes constitutes its primary key. Generally one defines this concept as the set of attributes whose values that are minimal (unique) identifiers of tuples, for all possible relation contents. Another definition of this concept may be that that the primary key is the set of values that are definitions *in intension* of distinct real objects (things) which some properties are modelled by other attributes values in each tuple. Below, if we wish to insist on the second meaning, we speak about natural key. One may easily see that both meanings are not always equivalent.

3.1.2 Semantic dependencies

The database relations are implicitly assumed to be semantically depended. Only such dependencies allow to consider that the same value of an attribute in different relations, corresponds the same thing (ex. same values of p#). Or, that attributes model the same concept (ex. attributes name) etc.

Let a and b be two attributes from, respectively, relations R_1 and R_2. We will call them naturally dependent if (i) they share a domain, (ii) at least one belongs to the primary key. A natural dependency between R_1 and R_2 will consist of all those between pairwise naturally dependent attributes, let them be $(a_1, b_1), .. (a_k, b_k)$; where $k \geq 1$ and $i \neq j \rightarrow a_i \neq a_j \wedge b_i \neq b_j$. If some a is in natural dependency with more than one b or vice versa, as it may happen, then each possible choice leads to one distinct natural dependency between R_1 and R_2. While many natural dependencies may then in principle appear between two relations, the case study through the litterature shows only two practical cases :

(a) - Only one a in natural dependency with some $b_1, \dots b_k$; $k \geq 1$ with typically k=1. R_1 and R_2 present then k natural dependencies, corresponding to couples $(a, b_1) \dots (a, b_k)$.

(b) - Some $a_1, \dots . a_k$ are parwise in natural dependencies with some $b_1, \dots b_k$ and no other attributes of R_1 and of R_2 are in mutual naturally dependency. R_1 and R_2 are then naturally dependent through the dependency corresponding to $(a_1, .. b_1), \dots (a_k, \dots b_k)$ together.

Attributes may be semantically dependent without being naturally dependent. Such dependencies exist in particular between attributes that do not belong to the primary key, but share a domain and a name, or share a domain, but their names differ or share the name only..

A natural dependency will be denoted :

- (a) or (R1.a ; R2.a) if it involves two attributes with the same name 'a',
- (a ; b) or (R1.a ; R2.b) if it involves attributes a and b,
- (a1,.. an) or (R1.(a1,.. an) ; R2.(a1,.. an)) if it involves several attributes in both relations named in the same manner.
- ((a1,.. an) ; (b1,.. bn)) or (R1.(a1,..an) ; R2.(b1,.. bn)) if some attribute names differ.

In /LIT84/ and /SHI84/ natural dependencies are called (in french) canonical connections.

Ex 5. In the database s-p there are following dependencies :

- natural dependencies (s#), (sp.p# ; p.p#), (comp.pmaj* ; p.p#), (comp.pmin* ; p.p#), (comp.pmaj* ; sp.p#) and (comp.pmin* ; p.p#).
- a dependency (name).

Suppose now that s-p contains also a relation, let it be sp', that has the attributes of sp. Then, between sp and sp' there is a natural dependency (s#, p#).

3.1.3 Database graph.

We call dependency graph the graph where :

- nodes are relation names,
- edges are semantic dependencies between nodes.

A database graph is the dependency graph that involves all relations. We will say that a graph is natural, if it deals only with natural dependencies. Unless the contrary is stated, below we consider only the natural and *connected* graphs. The latter property is the formal expression of the characteristic property of any database that it should model *one* universe.

Fig 1 shows the s-p graph. Edges are named as the corresponding dependencies. The purpose of edge numbers will be explained later on.

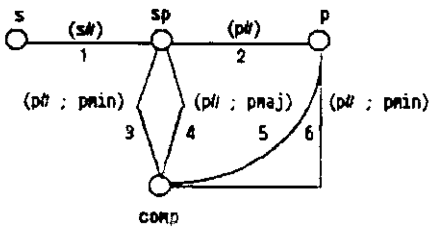

Fig 1 s-p database graph

3.1.4 Conjunctive query graph

For a while the term *query* will mean *conjunctive monodatabase query*. We consider that any such query involving more than one relation and/or tuple variable, typically defines some dependencies between these relations. In SQL, QUEL etc. like queries, these dependencies correspond either to join clauses in the -where clause or to cartesian product. The dependencies defined by join clauses may be natural or not depending on the schema and on the type of the join. The cartesian product dependency is inferred by the system when no join clause relates a relation enumerated in the -select clause to others. This is a trivial and meaningless dependency, since it relates everything to everything.

The (conjunctive) query graph is the graph where :
- each node, corresponds bijectively either to a relation named in the query or to a tuple variable, if more than one variable share a relation.
- edges are dependencies.

3.1.5 Complete and incomplete queries

We call a query complete if it is a (well formed) SQL, QUEL etc. query. The graph of a complete query is always connected. If the lack of a connection is considered as the trivial dependency i. e. the cartesian product. However, we disregard below this dependency, unless the contrary is stated. An incomplete query is then a query whose graph is not connected. Some clauses lack then with respect to a complete query. The algorithm finding missed clauses will be called completion algorithm.

3.2. Completion algorithm for implicit joins

3.2.1 The idea

An incomplete query may a priori be completed in many ways. Below, we consider that the incompleteness is the formal counterpart of the common expression "corresponds to". This expression denotes in general the relationship that is the simplest and the most natural for most users. Usually, anyone understands the choice. A misfit is nevertheless possible. One has then to precise more what was meant. Or, one may even have to define the meaning completely.

The aim of a completion algorithm is then to find the complete query that is the most natural and the simplest completion. The notion of natural completion is interpreted as meaning the *natural dependencies only*. The one of simplicity is interpreted as meaning *as few additional dependencies as possible*. The user should typically be satisfied with the algorithm output, but a misfit is possible. The user has then either to indicate more clauses or, may even need to specify all the dependencies he ment. This may always be done, since both incomplete and complete queries *address the same and usual schema*. Next, since the capability of dealing with incomplete queries does not impede the system to understand the complete ones.

3.2.2 Basic case

We consider as the basic case the one of an incomplete monodatabase, elementary and conjunctive query, with at most one tuple variable per relation. The completion algorithm works then as follows :

Completion algorithm CA1 :

```
1 - Find from the database graph all the connected
graphs let them be G such that each G :
  - includes all the nodes of the query graph,
  - is minimal in number of edges.
2 - For each G :
  2.1 - pick-up all and only the edges that added to
  the query graph render it connected.
  2.2 - add to the incomplete query as the conjunctions
  all the clauses corresponding to the new edges.
3 - Let q denote each query resulting from (2.2).
Consider as the complete query the union of all q.
```

Since G are all minimal in the number of edges, they have all the same number of edges. For the same reason, all G are acyclic, i. e. they are all minimal trees. The number of edges of G will be called its size. In practice the union of q is the query with the selection expression unioning through the key word union the selection expressions of each q.

Ex 6.

1. Consider the query (6.2). The only G is the path of size 2 between s and p. The completion leads to (6.1), as most users would wish.

2. Consider the query :

```
-select (s.name)                          (q1)
-where (tass = "screw")
retrieve
```

This query is incomplete. It is then basically considered as expressing the wish "Select the names of suppliers corresponding to the assembling type "screw". The graph of s-p shows that there are then two minimal trees (pathes in this case) of size 2 connecting the nodes s and comp. Both include the edge (s#), but differ by the choice of either the edge (p# ; pmin) or of (p# ; pmaj). The first choice leads to the expression :

```
-select (s.name)                              (q1)
-where (tass = "screw") &
(s.s# = sp.s#) & (sp.p# = pmin)
```

The second choice leads to :

```
-select (s.name)                              (q2)
-where (tass = "screw") &
(s.s# = sp.s#) & (sp.p# = pmaj)
```

(q1) means that "correspond to" was interpreted as "suppliers of minor parts assembled with screws". (q2) means that one considered the suppliers of such major parts. Since, there is no indication in favour of any of these interpretations, the natural approach is to assume them both. The assumed wish becomes "suppliers of minor or of major parts ...". That is what CA1 provides through the union in the step (3). The final query is then :

```
-select (s.name)                              (qc)
-where (tass = "screw") &
(s.s = sp.s) & (sp.p = pmin)
union
-select (s.name) -where (tass = "screw") & (s.s =
sp.s) & (sp.p = pmaj)
retrieve
```

(qi) will effectively appear to most users rather (much) simpler than (qc) .

3. Consider the query :

```
-select (s.name p.name) -where (tass = "screw")
retrieve
```

Since this query is incomplete its meaning is "retrieve names of suppliers and of parts corresponding to assembling with screw". Two relationships between parts and suppliers are then intuitively most natural ones :

(a) - the part is supplied by the supplier,
(b) - the part is assembled (through screws) with a part supplied by the supplier.

The completion algorithm produces now eight minimal trees of size 3, shown at the Fig 2. These trees are, by the way, no more pathes and correspond to all trees of the s-p graph at Fig 1. They thus lead to the union of eight subqueries. One may easily see that the result correctly expresses the considered relationships.

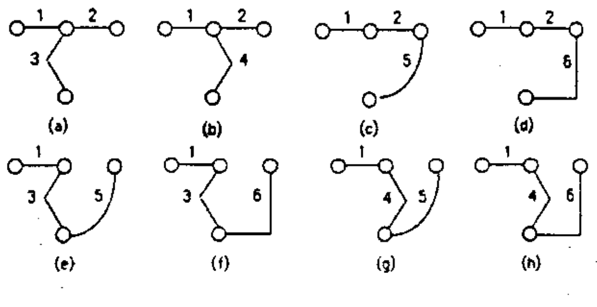

Fig 2 Trees within s—p database graph

Note that, because of equijoin associativity, several of the eight trees lead to equivalent queries. For instance (a) and (d) as well as, on the other hand, (b) and (c). Thus it would suffice to take to the account only two of the corresponding four queries, (a) and (b) for instance. While to produce more queries has no consequence on the result (unioning) correctness, it shows that there is room for CA1 optimizing.

5. Consider now that a user is interested only in the relationship (a) above. Then he has to explicit in the query the corresponding dependency, living implicit the others. The corresponding (incomplete) selection expression may then be :

```
-select (s.name p.name) -where (tass = "screw") &
(s.s# = sp.s#) & (sp.p# = p.p#)
```

CA1 will then produce the corresponding intuitively correct complete query, since involving the part *either* as the minor *or* as the major one.

4. Consider the relation sp' (ex. 5) and the selection :

```
-select sp'.qty -where (sp.qty = "10")
```

We showed in ex. 5 that there is a natural dependency between sp and sp' that involves jointly two attributes. CA1 will thus lead to the following completion :

```
-select sp'.qty -where (sp.qty = "10") & (sp.s# =
sp'.s#) & (sp.p# = sp'.p#)
```

This expression clearly corresponds to the natural meaning that one could expect, i. e. sp'.qty of parts that sp.qty is 10.

3.2.2 Several variables over the same relation.

Let R be a relation over which one defined several tuple variables X_i. Each X_i constitutes then in the query graph one node named $R.X_i$. The algorithm CA1 expands as follows :

Completion algorithm CA2 :

```
1 - Consider that all nodes corresponding to R
    constitute one (multi)node. Then apply step (1) of
    CA1.
2 - For each G :
    2.1 - apply the step (2.1) of CA1,
    2.2 - Let multiedge be any new edge which one or
          both extremities are multinodes. Replace each
          multiedge with all edges which one or both
          extremities are nodes within the corresponding
          multinodes.
    2.3 - Apply step (2.2) of CA1.
3 - Apply step (3) of CA1.
```

Ex 7. Consider the following query :

```
-range (x s) (y s)
-select x.name y.name
-where (x.city ≠ y.city ) & (qty = "20")
retrieve
```

This query is incomplete. It therefore basically means "select couples of suppliers that, although in different cities, correspond to parts in quantity equal to 20". The query graph will contain one multiedge that one may note ({s.x, s.y}.s# ; sp.s#). It will lead to two natural equi-join clauses, one for each variable. The complete query generated by CA2 will be :

```
-range (x s) (y s)
-select x.name y.name
-where (x.city ≠ y.city ) & (qty = "20") & (x.s#
= sp.s#) & (y.s# = sp.s#)
retrieve
```

The meaning of "correspond to" is now " both supply".

3.2.3 Set type and/or disjunctive queries.

Set type queries involve operations like union, intersection, etc., between -select clauses. Each -select clause corresponds to a full subquery, i.e may be followed by its own -where clause. One may in particular replace any disjunctive query with a set type query where (i) to each conjunction corresponds exactly one -select and -where clauses (ii) the select clauses are unioned. Set type and/or disjunctive queries are completed as follows :

Completion algorithm CA3 :

1 - If a query is a disjunctive query, replace it with the corresponding set type query.
2 - Apply CA1 or CA2 to any incomplete subquery.

Ex 8. Retrieve names of suppliers that supply 'red' parts but not 'yellow' parts.

```
-select s.name -where (colour = 'red')
differ
-select s.name -where (colour = 'yellow')
retrieve
```

Each -where clause will be completed.

3.2.4 Multiple queries

Multiple queries are completed as follows :

Completion algorithm CA4 :

1 - apply the adequate CA to each elementary query that reveals incomplete.
2 - if several such queries exist for a database, then consider only the ones minimizing the query graph.

Ex 9. The multidatabase wish "retrieve cities of red part suppliers, known to databases s-p and/or s-p'" from the example 4 may be effectively expressed through (6.2), namely :

```
-select (city) -where (color = "red")
retrieve
```

This query will first be considered as one incomplete elementary query to s-p and one complete elementary query to s-p'. The final query would then be the set of two queries :

```
-select (s-p.s.city)
-where (s-p.p.color = "red") & (s-p.s.s# =
s-p.sp.s#) & (s-p.sp.p# = s-p.p.p#)
retrieve
```

```
-select (s-p'.s.city) -where (s-p'.s.color =
"red")
retrieve
```

Note how simpler (6.2) is.

Example 10

Consider again the bank database. Consider the following selection expressions :

```
-select bnk -where (c = "Jones")        (Q'2)
-select acc -where (l = "4-326")        (Q'3)
```

They are formulated almost like queries noted Q2 and Q3 in /MAI83/. These queries led in the universal relation model to intuitively wrong interpretations. While one could expect that (Q2) meant "banks where Jones has *either* an account *or* a loan", its actual interpretation was "banks where Jones has both an account *and* a loan". For (Q3), it was suggested that one could expect the meaning "print accounts that are either at the same bank as loan 4-326 or are owned by a customer who also holds loan 4-326". However, not only it was not the case, but even it was unclear whether any natural interpretation of (Q3) existed.

In order to get responses, at first one had then to define in addition to the univesal relation two maximal objects. These are the largest sets of smaller objects in which the user is willing to navigate (note that one may consider that the corresponding interface is then no more the universal relation interface since it consists again from several relations). Then one could express the wishes corresponding to (Q2) and (Q3). However, one had to use two queries per wish. In particular, each query corresponding to (Q3) had to involve an additional equijoin on, respectively, c and bnk. They corresponded to the logical navigation between the maximal objects (for details see /MAI83/).

In appears easily from the Fig 3b that for (Q'2), the replacement of the multiple identifiers and the evaluation of implicit joins through CA4 leads to two complete queries :

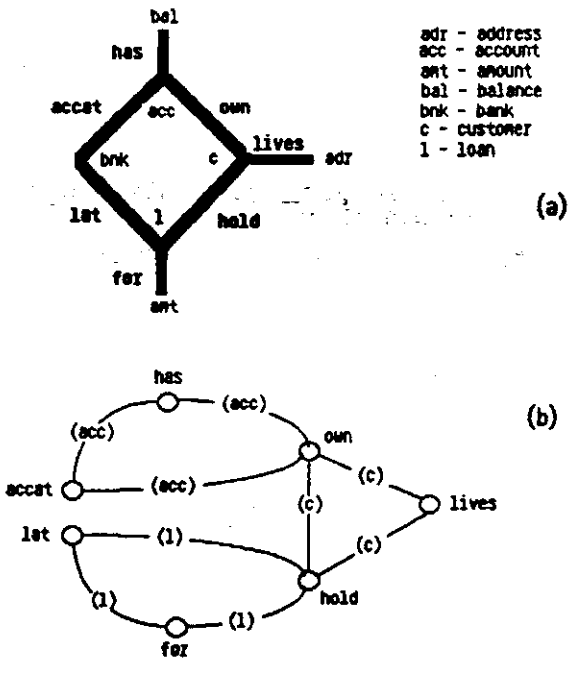

Fig 3 bank database : (a) hypergraph, (b) graph.

```
-select accst.bnk -where (own.c = "Jones") &
(accst.acc = own.acc)
retrieve
```

```
-select lat.bnk -where (hold.c = "Jones") &
(lat.l = hold.l)
retrieve
```

The corresponding meaning will be thus the natural one. In addition one would know whether the bank corresponds to the account or to a loan. If one is not interested by these details, one should ask for :

```
-select union (bnk) -where (c = "Jones")
retrieve
```

With respect to (Q'3), the only complete query would be :

```
-select own.acc -where (hold.l = "4-326") &
(own.c = hold.c)
```

The corresponding interpretation of (Q'3) would thus be "Accounts that are are owned by a customer who holds loan 4-326". It would thus be be only the second subinterpretation within the one expected in /MAI83/. It would also correspond then to one of the queries that one could finally formulate when the maximal objects were defined, the one with the explicit join over c (but without this join). For the first subinterpretation, one has to formulate a query similar to the one with the maximal objects, i.e. with the explicit join over bnk.

Technically, the first subinterpretation was disregarded, since there is no natural dependency between accat.bnk and lat.bnk. The corresponding background reason is that banks are then less important as relating concepts in this schema than the customers. This, as there is no definition *in intension* of banks through some natural key. They are then not (in this schema) real objects by themselves, but only properties of some objects, of loans and of accounts in the occurrence. The dependency through properties is less important, as, for similar reasons, in s-p it the dependency between parts and suppliers through the same name.

Note that CA4 would provide as the meaning of (Q'3) both subinterpretations, if banks were considered as important as customers, that is they would have a natural key in bnk. This, in contrast to /MAI83/, where two queries would still be required.

3.2.5 Updates

Update queries may be incomplete as well. According to the type of their selection expression, they are completed as we discussed above.

Ex 11. Delete suppliers corresponding to red parts.

```
-select s -where (colour = "red")
delete
```

3.2.6 Interdatabase joins

A multidatabase query may involve interdatabase join clauses. This is for instance the case of the query 1 in Section 2 and in general of any elementary multidatabase query. Such queries may in particular be incomplete /LIT84b/. Two cases may be distinguished :

(i) - all implicit clauses are monodatabase. This means that one may connect the disconnected parts of the (multidatabase) query graph using edges of database graphs only.

(ii) - some implicit clauses are interdatabase. The completion must use edges encompassing different databases.

Queries of type (i) are completed in the manner analogous to the one developed above for the monodatabase queries. Queries of type (ii) lead to a new situation since they span over two separated sets of domain. No database graph contain therefore the corresponding edges.

For such queries we apply the concept of interdatabase dependencies /LIT82/ that may either be defined within the multischema of a collection of databases or may be dynamically inferred from database schemas through expert system capabilitites. In the case under the analysis we consider the equivalence dependencies. These dependencies relate the sets of domains in different databases where identity of values implies the same real object. One may thus consider them as multidatabase natural dependencies. For instance, in the case of michelin and gault-m one may consider the following dependencies :

```
michelin (rname, street) = gault-m (rname,
street)
michelin (cname) = gault-m (cname)
```

They would mean that the same name and address in both databases identify a restaurant and that course names identify courses. Note that, in contrast, there is no equivalence dependency between names and addresses within michelin and within cinemas.

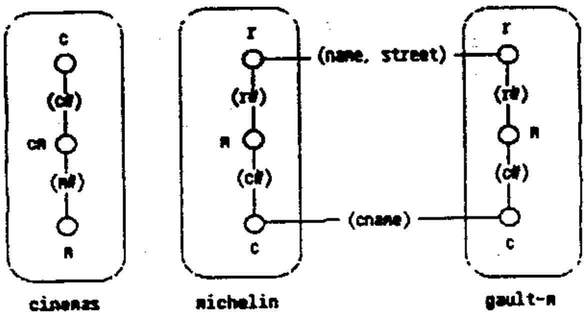

Fig 4 Multidatabase graph of cinemas, michelin and gault-m with equivalence dependencies.

If equivalence dependencies are not known in the case of a query of type (ii), we consider that either the trivial (cartesian product) dependency is generated or the query is rejected. Otherwise, one applies the above CA algorithms, except that instead of the database graph one refers to the multidatabase graph. The latter is simply the union of the former ones, connected through the equivalence dependencies, if known.

Ex 12. Assume known the above equivalence dependencies (Fig 4). The selection expression of the query "retrieve gault-m comments corresponding to '***' restaurants in michelin may then be expressed as follows :

```
open michelin gault-m
-select comment -where (stars = "***")
```

The completion, will add the join clauses :

```
(michelin.r.rname = gault-m.r.rname) &
(michelin.r.street = gault-m.r.street)
```

3.2.7 Implementation aspects

The prototype implementation within MRDSM is described in /SCH84/. The completion algorithm leaded mainly to the following graph manipulation problems :
(i) - test of the connectiveness,
(ii) - determination of all connected minimal trees including a given set of nodes (nodes of the query graph).

In order to solve (i) it suffices to apply the matrix representation of graphs and the corresponding algorithm, used in particular by INGRES /STO76/. One way to solve (ii) is as follows :

(1) - union the database graph and the query graph.
(2) - if two nodes are then linked by an edge of the query graph and some edges of the database graph, then delete the database graph edges (priority to query edges).
(3) - find all trees of the resulting graph.
(4) - for each tree delete recursively all the edges which one extremity is a terminal node (node that is not within any other edge) that is not within the query graph.

It is easy to see that the result of (4) is the solution to (ii). The only non obvious step above is (3). While several algorithms may be devised for this purpose, a particularly elegant solution is provided by the <u>algebra of structural numbers</u> /LIT72/, /BEL69/, /DUF59/. Since this algebra is not largely known, we will shortly present its aspects relative to our purpose.

If X is some set, then <u>structural numbers</u> over X are finite sets of finite subsets of X. A structural number is usually represented as a table where the subsets are columns which lengths may differ. X may in particular be the set of (names of) all edges of a connected graph let it be D. If t is then the set of names of all edges of a maximal tree in D (a tree on all nodes of D), then one may characterize D with a structural number let it be T where each column is exactly one t and to any t corresponds one column. It may be shown that T may be computed in the following way :

<u>Computation of maximal trees within a connected graph (TA) :</u>

- let n be the number of nodes in D. Define for n-1 nodes the structural numbers where each column denotes one edge adjacent to the node and all edges are denoted. Let S_i be these numbers called <u>node (structural) images</u>.
- let S' and S'' be structural numbers. Let the structural number denoted S' x S" be the number whose columns are all possible unions of a column of S' with a column of S'', provided the exclusion : (i) of unions of equal columns, (ii) of the maximal even number of equal results. Then :

$T = S1 \times \ldots \times Sn\text{-}1.$

The operation 'x' is called <u>multiplication</u>. Its result is called <u>product</u>. T, called <u>(structural) image</u> of D, is thus simply the product of n-1 node images. The choice of the nodes has no influence on T value, but may influence the speed of computing. The reason is that it may lead to less or more unuseful unions. The cardinal of T is larger when D has more cycles.

If X are natural numbers, storage representation of a column may consist of a bit string where the bit b_i = 1 iff i belongs to the column. Storage representation of T may thus be compact even for D with large number of cycles. Structural numbers represented in this form are called <u>binary structural numbers</u>. See /LIT72/ for the corresponding algorithms for the binary multiplication and other operations on structural numbers.

<u>Ex 13.</u> For simplicity, the edges of s-p graph on Fig 1 are also named 1,...6. The trees of s-p , shown at the fig 2, may be computed as follows :

$$T = [1] \quad [1\,2\,3\,4] \quad [3\,4\,5\,6] = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 3 & 3 & 3 & 4 & 4 \\ 3 & 4 & 5 & 6 & 4 & 5 & 6 & 5 & 6 \end{bmatrix}$$

In order to compute then, for instance, all the minimal trees connecting the nodes s and p, one (i) deletes from the columns the elements 3, 4, 5 and 6, since they denote edges leading to the node comp that is a terminal one in trees, (ii) deletes, if any, the resulting columns that are empty or are shorter than the longest columns (shorter columns correspond to incomplete trees), (iii) unions equal columns, if any. The result as expected is :

$$T(s,\ p) = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

4. RELATED WORK

4.1 Universal relation model

Most related proposals concern the universal relation model. Main practical differences between these proposals and ours are that in our case (i) there is no additional requirements on the database schema and/or need for additional types of objects ; (ii) logical navigation is eliminated or shortened in more cases (queries in Ex. 7, updates, multiple queries and multidatabase queries). These properties of our method as well as those listed in the introduction, seem to result from the fact that we take to the account the semantic aspects of the concept of a domain.

Within these proposals and the discussed limitations, many of ideas within /WAL84/ are particularly close to ours. However, the methods being developed independently, one should be careful since the precise meaning of the same terms usually differ (ex. the "database graph"). Next, graphs in /WAL84/ correspond to the entity-relationship model. Also, /WAL84/ looks only for retrievals and for a way to choose only one "query tree" if several correspond to a wish. The result should frequently coincide with ours for acyclic graphs. In contrast, it should be typically different otherwise. /WAL84/ should then usually provide as result only one of our subqueries, (the one that minimizes the cost function of /WAL84/), while we provide them all, considering that there is no further objective criteria in favour of any of them. In particular, if several queries minimize the /WAL84/ cost function, user intervention is assumed, unlike in our method.

70

4.2 Semantic model

A less known approach, based no more on the idea of universal relation, but on a particular semantic data model is described in /SUG84/. One proposes to show this model to users of a relational database, instead of the relational one. The model allows then a query language, named DOMCALC, intended as largely logical navigation free extension of QBE. The model is used in order to define by an administrator a collection of "domains" and a list of possible interconnections among them. The "domains" are not the ones of the relational model, but "may be regarded as collections of entities of attributes or of relationships".

DOMCALC allows then the user to formulate retrieval only queries referring to at most two "domains". The result may be ours, if the completion algorithms were applied to QBE or vice versa (both cases seem possible). However, the DOMCALC user must learn a new data model. Next, the administrator has to identify (i) the lists of all the joinable attributes and (ii) all generalizations of the similar attributes into a common "domain". The "domains" have then to be, in addition, adequately named. Our method is free of such constraints, and again deals with more types of queries.

4. CONCLUSION

We have presented a method simplifying relational queries expression. The method allows to leave implicit natural equijoins, avoiding in this way the corresponding logical navigation. This property should reveal important in practice. In particular it should allow to formulate new types of multidatabase queries.

Implicit joins are implemented on the prototype relational multidatabase system MRDSM. The implementation took about six months of student work. The corresponding principles may be extended or improved in several ways. First, one may optimize the completion algorithm, in particular in order to eliminate equivalent queries. Then, one may provide to the user not only the union of results of subqueries, but also, upon demand, the set of results corresponding each to a nonequivalent completion. Furthermore, one may include cost considerations or select also nonminimal trees. Then, one may take to the account statistics about a user behaviour, infering therefore completions particularly adapted to habit of this user. These completions may then include even joins other than equijoins and/or dependencies other than natural ones.

Acknowledgements

We thank Y. Sagiv for fruitful discussions.

References

/ABD83/ Abdellatif, A. Traitement de requêtes multiples par un système de gestion multibase (MRDSM). Th. Fac. des Sc. de Tunis. INRIA, (June 83), 80.
/ANS75/ ANSI-SPARC interim report on data base management system. Doc. 7514TS01, Washington DC, (Feb. 1975).
/BEL69/ Bellert. S. Topological analysis and synthesis of linear systems. J. Franklin Ins., 274, 1969, 425.
/BRE84/ Breitbart, M. ADDS : another multidatabase management system. 3-rd Int. Sem. on Distr. Data Sharing Syst., Parme, (March 1984), North Holland, 7-24.
/COD71/ Codd, E., F. A database sublanguage founded on the relational calculus. ACM SIGFIDET, (Nov. 1971), 35-68.
/COD79/ Codd, E., F. Extending the database relational model to capture more meaning. TODS. 4, 4, (June 1979).
/COD82/ Codd, E., F. Relational Databases : A Practical Foundation for Productivity. CACM, 25, 2 (Feb 1982).
/DAY84/ Dayal.U., Gouda, M., G. Using Semiouterjoins to Process Qeries in Multidatabase Systems. ACM-PODS, Waterloo (Canada), (April 1984), 153-162.
/DUF59/ Duffin, R. An analysis of the Wang algebra of networks. Trans. Amer. Math. Soc., 1959, 114-119.
/KUC82/ Kuck, S., M., Sagiv, Y. A Universal Relation Database System Implemented Via the Network Model. ACM-PODS, (March, 1982), 147-157.
/KUC83/ Kuck, S., M., Sagiv, Y. Designing Globally Consistent Network Schemas. ACM-SIGMOD, (May, 1983), 185-195.
/IMI84/ Imielinski, T., Rozenshtein, D. Towards a Flexible User Interface to Relational Database Systems : Processing Simple Second Order Queries. JCIT-4, Jerusalem, (June 1984), pub. IEEE, 350-367.
/KEN84/ Kent, W. The Universal Relation Revisited. TODS 8, 4, (Dec. 84), 644-648.
/KOR84/ Korth, H., F., Kuper, J., M., Feingenbaum, J., Gelder, van A., Ullman, J., D. System/U: A Database System Based on the Universal Relation Assumption. ACM-TODS, 9, 4, (Sep. 1984), 331-347.
/LIT72/ Litwin, W. Une methode d'analyse de graphes. Suppl. to Bull. of INRIA, 13, (Sep. 1972), 39.
/LIT81/ Litwin, W. Logical model of a distributed data base. 2-nd International Sem. on Distributed Data Sharing Systems. Amsterdam (June 1981). North-Holland. 173. 207.
/LIT82/ Litwin W. and all. SIRIUS Systems for Distributed Data Management. DISTRIBUTED DATA BASES. North-Holland, 1982, 311-366.
/LIT83/ Litwin, W., Kabbaj, K. Multidatabase Management Systems. ICS 83, Nurnberg, (March 83). B. G. Teubner, 482-505.
/LIT84/ Litwin, W. Jointures implicites dans le système MRDSM. Res. Rep. 8403191353-SESAME. INRIA, (March 1984), 12.
/LIT84a/ Litwin, W. MALPHA: a relational data manipulation language. IEEE-COMPDEC, Los Angeles, (May 1984).
/LIT84b/ Litwin, W. Concepts for multidatabase manipulation languages. JCIT-4, Jerusalem, (June 1984), pub. IEEE, 433,442.
/LIT85/ Litwin, W. An Overview of the multidatabase system MRDSM. INRIA Res. Rep. 374, (Jan. 85), 26.
/MAI83/ Maier, D., Ullman, J., D. Maximal Objects and the Semantics of Universal Relation Databases. ACM-TODS, 8, 1, (March 1983), 1-15.
/MAI84/ Maier. D., Ullman.J., D., Vardi, M. Y. On the foundations of the Universal Relation Model. ACM-TODS. 9, 2 (June 1984), 283-308.
/MUL82/ Multics Relational Data Store (MRDS) Reference Manual. Cii Honeywell Bull. Ref. 68 A2 AW53 REV4, (Jan. 1982).
/SAG81/ Sagiv, Y. Can We Use the Universal Instance Assumption Without Using Nulls? ACM-SIGMOD, (Apr. 1981), 108-120.

/SHI84/ Shili, B. Dépendances interbases et jointures implicites dans un système de gestion multibases (MRDSM). Th. Univ. de Tunis, ed. INRIA, (June 1984), 216.
/SUG84/ Suginara, K., Miyao, J., Kikuno, T., Yoshida, N. A semantic Approach to usability in relational database systems. IEEE-COMPDEC, Los Angeles, (May 1984), 203-210.
/STO76/ Stonebraker, M., Wong, E., Kreps, P., Held, G. The design and implementation of INGRES. ACM-TODS, 1, 3, 1976, 189-222.
/TEM84/ Templeton, M. An architecture for multidatabase systems. Position papers of 3-rd Int. Sem. on Distr. Data Sharing Syst., Parme, (March 1984), University of Parma, 40-42.
/ULL80/ Ullman, J., D. Principles of Database Systems. Comp. Sc. Press, 1980, 379.
/ULL84/ Ullman, J., D. On Kent's "Consequences of Assuming a Universal Relation". ACM-TODS 8, 4, (Dec., 84), 638-643.
/WAL84/ Wald, A., J., Sorenson, P. Resolving the Query Inference Problem Using Steiner Trees. ACM-TODS 9, 3, (Sep. 84), 348-368.
/WIE83/ Wiederhold, G. Database design. McGraw-hill Book Company, 1983.
/WON84a/ Wong, K. MRDSM : a relational multidatabase management system. 3-rd Int. Sem. on Distr. Data Sharing Syst., Parme, (March 1984), North Holland, 77-86.

ANNEX

The databases cinemas michelin and gault-m have the following schemas /LIT82/ :

```
db cinemas
   c ( c#*, cname, street, tel)
   m ( m#*, mname, kind)
   cm ( c#*, m#*, price)
db michelin
   r ( r#*, rname, street, type, stars, avprice,
tel)
   c ( c#*, cname)
   m (c#*, r#*, price)
db gault-m
   r ( r#*, rname, street, qual,  tel , type,
avprice, comment)
   c ( c#*, cname, ncal)
   m (c#*, r#*, price)
```

The character '*' indicates in DSL the key. The domain names are assumed the same as the ones of the corresponding attributes. The attribute values and in particular natural key values, are independent in different databases, even if attribute and domain names are the same. The attribute stars in michelin mesures the quality of a restaurant as a value ranging '   ' to '***'. qual of gault-m measures it as a fraction i/20 ; $0 \le i \le 20$. A restaurant may figure in one or both guides.

- The database s-p has the following schema :

```
s    ( s#*, name, tel, city)
p    (p#* ,  name, colour)
sp   (s#*, p#*, qty)
comp  (pmin*, pmaj*, tass)
```

The domains of s-p are assumed named as the corresponding attributes, except that the domain of pmin and of pmaj is the one of p#.

- the bank database schema and its hypergraph image at fig 3a, are from /MAI83/. For our purpose we considered, as by the way also in /MAI83/, that the predicates accat,.., lives define the corresponding relations. The meaning of predicates is straightforward (accat : account x is in bank y, lat : loan x is in bank y,..). The domains are assumed named as the corresponding attributes. Given the functional dependencies in /MAI83/, the relation schemas of bank are :

```
accat (acc*, bnk)
for ( (l*, amt)
has (acc*, bal)
hold (l*, c*)
lat (l*, bnk)
lives (c*, adr)
own (acc*, c*)
```

Note that banks have no natural key. That is why the bank graph (fig 3b) has no connection through bnk, in contrast to the hypergraph. Banks name in this schema, only properties of loans and of accounts.