JCIT-4 Jerusalem
June 1984

CONCEPTS FOR MULTIDATABASE MANIPULATION LANGUAGES

Witold LITWIN
INRIA - 78153, Le Chesnay, France.

The database approach is founded on the principle that data to be manipulated by a user constitute a database. In fact, many users need to manipulate data that constitute several databases. The database manipulation languages do not suffice then anymore. One needs multidatabase manipulation languages. We present some concepts for such languages, for relational databases.

1.INTRODUCTION.

The main principle of the database approach is that data to be managed together constitute a database (DB). A DB is defined by its conceptual schema /ANS75/. It consists of either a set of relations /COD71/, /COD79/ or of record types, /BAC77/ or of entity types /CHE76/ or of components /HAM79/ etc. Users manipulate either the DB or a view of the DB. The data management capabilities are provided by the database management system (DBMS). A DBMS provides a data definition and manipulation languages, view definition, concurrency control etc. Data may in particular be stored on several sites in which case the DB is distributed (DDB). The conceptual schema is then typically called global schema /STO77/, /ADI78/, /ROT78/, /LIN81/, /SMI81/,...

As we show in /LIT79/,...,/LIT82/, a probably unexpected consequence of the database approach is that if the conceptual data do not the constitute a single (D)DB, but they constitute several (D)DBs, then one lacks of systems and even of the principles for the corresponding management. Since once observed, this limitation appears clearly unacceptable (see the examples in the sequel), we have proposed to suitably generalize the database approach. We called the new approach mulitdatabase approach and we studied it in /LIT79/,..,/LIT83a/. Work on related problems is also described in /LAN82/, /DAY83/ and /LYN83/.

The main principle that our approach proposes instead of the one defining the database approach is that data to be managed together constitute one or several DBs. This means in particular that the ability to manage data defined through more than one conceptual or global schemas is, for us, one more basic (although forgotten) objective of database systems. We further consider that DBs to be managed together should constitute a multidatabase (MDB). An MDB is a set of DBs or of MDBs such that :

- there is a data manipulation language to express multidatabase manipulations (multidatabase manipulation language),
- there is a data definition language to define the MDB and, eventually, dependencies between DBs or MDBs (multidatabase definition language).

The idea in the concept of an MDB is that a collection of DBs is an MDB iff multidatabase manipulations may be formulated. For us, this idea is a natural generalization of the main one in the concept of a DB, with respect to earlier ideas on collections of files. In particular, an MDB may be a relational MDB that is an MDB constituted from relational (M)DBs. For obvious reasons relational MDBs are of particular interest.

Below, we describe a relational multidatabase manipulation language that we called MDSL (MultiDatabase SubLanguage). MDSL is the language of MRDSM system /LIT83/, /ABD83/, /VIG83/, /WON83/. It generalizes to the multidatabase environment the DSL language of the well known MRDS system, that is the DBMS of Multics system. It is also a practical application of the principles of MALPHA language /LIT83a/. Since DSL is to large extent SQL compatible, the proposed concepts should apply to other SQL based languages as well.

The properties of a relational MDB typically differ from the ones of a relational DB. The main reason for this is the presence of several conceptual schemas that, in addition, may be independently defined. In order to show these properties and the new needs they create we first present a case study. We then describe MDSL and show how it fulfills the needs.

Section 2 presents the case study. Section 3 presents MDSL. Section 4 is devoted to the conclusions. The discussion assumes DSL known, for instance through /MUL82/. Typographic differences, in particular with respect to the usage of upper/lover case letters, should be disregarded.

2.CASE STUDY.

We consider the following collection of DBs :

```
   DB CINEMAS
C (C# , CNAME, STREET, TEL)          Cinemas
M (M# , MNAME, KIND)                 Movies
P (C#,M#, HOUR, PRICE)               Projections
   ENDDB
```

```
   DB MICHELIN
 R (R#, RNAME, STREET, TYPE, STARS, AVPRICE, TEL)
 C (C#, CNAME)
 M (R#,C#, PRICE)
   ENDDB

   DB KLEBER
 REST (REST#, NAME, STREET, TYPE, FORKS, T#, OWNER,
MEANPRICE)
 C (C#, CNAME, NCAL)
 MENU (REST#, C#, PRICE)
   ENDDB

   DB GAULT-M
 R (R#, RNAME, STREET, QUAL, TEL, TYPE, AVPRICE,
REM)
 C (C#, CNAME, NCAL)
 M (R#, C#, PRICE)
   ENDDB

   DB MY-REST
 R (R#, RNAME, STREET, QUAL, TEL, TYPE, AVPRICE,
REM)
 C (C#, CNAME, NCAL)
 M (R#, C#, PRICE)
   ENDDB
```

The DB CINEMAS is assumed to be a public DB that describes (models) the on-going activities of cinemas in a city. The DBs MICHELIN, KLEBER, GAULT-M are also public DBs, created by the famous french restaurant guides of the same name (the full name for GAULT-M is Gault-Millau). Finally, the DB MY-REST is a personal restaurant guide. A user stores there the restaurants selected by himself, using as the model GAUL-M. In particular, he keeps in MY-REST the restaurants that are the best in GAUL-M ; this allows him to put his own remarks and to do not pay for the corresponding access.

The relations in the restaurant DBs model, respectively, restaurants, courses and menus. The following properties characterize these DBs, given the actual guides :

1. A restaurant may be recommended by more than one guide. However, not all restaurants are recommended by all the guides.

2. An object may be characterized by several keys with the same name, but different and even independent values. This is the case of R#, C# and M#.

3. The measures of a restaurant quality also differ from guide to guide. MICHELIN rates restaurants from zero to three stars (***). KLEBER - from zero to four "forks". GAULT-M rating is m/20 ; m = 0,1,..,20. There is no objective rule for the value of one measure given another one. Nevertheless, one is frequently sure that the guides disagree upon a restaurant.

4. By the same token, the guides may also disagree upon the average price of a meal, or upon a restaurant type. For instance, a restaurant may be chinese for one guide and vietnamese for another. The guides also may disagree upon the telephone number, although it is a candidate key within each DB.

5. In contrast, the guides always agree upon a restaurant name and the corresponding street name. This property allows thus users to recognize all the descriptions corresponding to the same restaurant. Similar properties exist with respect to courses and menus.

As long as any user is aware of only one of these DBs, no one sees a query to more than one DB. Otherwise this is no more the case. The examples below show many queries to several DBs that come then to mind. Especially, to a french one.

A query to only one DB is, in the sequel, called monodatabase query. In contrast, queries to several DBs, are called multidatabase queries. Database manipulation languages do not suffice for formulation of such queries. As one may see from the case study and from the examples below, the new needs that such queries create are at least as follows :

1. One may need to open simultaneously several, may be very many, DBs.

2. A manipulation may need to address a relation which name is no more its identifier, since it is shared by relations in other DBs that the manipulation concerns. This may also happen with respect to a database name.

3. A query may consist of a manipulation to be repeated for some DBs. The corresponding relations will typically share some attributes. However, a they will also have attributes on their own. They may also differ with respects to the choice of names for the same meaning, as well as with respect to the corresponding data types.

4. In particular, there may be several ways to present the result of a retrieval. A user may wish to separate the results corresponding to different DBs. Someone else may wish all the descriptions of the same real object to be merged to one tuple. Then, one may wish to homogenize some names or data values etc.

5. The multidatabase query may be a retrieval. However, it may also be an update, a deletion or an insertion. In particular data may need to flow not only from a DB to a temporary relation (workspace), but also between DBs.

6. Queries may lead to manipulations of database names themselves.

7. A multidatabase query should be expressible in possibly only one formal statement (command).

MDSL language is intended to be a response to at least these needs. Note incidentally that to respond to the need similar to (7) for the database queries, was one of the main goals of the relational data model /CODB2/.

3.MDSL LANGUAGE.

3.1 Basic relationship to DSL.

MDSL is by definition a generalization of DSL. As long as one deals with the database queries, any DSL rule is thus MDSL rule as well. The inverse is not true, i. e. incidentally MDSL brings new possibilities also for the database queries. These new possibilities will appear from what follows.

As for any database manipulation language, the data universe consists for DSL of (i) the the database (ii) one or more temporary relations called workspaces (WS). The purpose of the language is to define the data (information) flow between the DB and a WS (see fig 1 in /COD71/ illustrating the idea in a relational sublanguage). In contrast, for MDSL the data universe consists from several DBs. Data may flow between several (M)DBs and WSs, as well as between (M)DBs.

DSL objects, i. e. the data structures that one may designate in a query, are attributes or relations. MDSL objects are, in addition, DBs and MDBs. The examples in the sequel consider two MDBs :

NIGHTLIFE = {CINEMAS, MY-REST}
REST-G = {MICHELIN, KLEBER, REST-G}

The name of an MDB is called multidatabase name.

We call scope of a query the set of (M)DBs that it addresses. The scope of any DSL query is, by principle, a DB. MDSL query may involve any number of (M)DBs. In particular, its scope may be delimited only by the set of (M)DBs that are open while the query is issued.

The general form of DSL commands is :

C W T P

where :

- C is a command name,
- W is a workspace name (temporary relation name),
- T is a target list (select clause),
- P is a predicate (where clause).

T is a set of names that we call designators since they designates database objects or variables. In occurrence, T designates either a relation, or some attributes, or some tuple variables that are defined in an auxiliary RANGE statements. Together with P, T defines a relation which attributes are the ones resulting from T. The convention called inheritance rule /COD71/ renders the tuples and the names of the attributes of W the ones of T. W and P are sometimes optional.

The general form of MDSL query is syntactically the same. However, first, the following minor differences exist :
- one do not need to prefix the command name with `mrds_call` or `dsl_$`.
- The command name may follow the selection expression, like in LINUS (another language for queries to MRDS),
- W may be in commands other than define_temp_rel.
- database identification indexes called data_base_index in DSL are not mandatory.
- also like in LINUS, the clauses may be prefixed with numerical labels, since one may elaborate a query clause by clause, using a precompiler. The labels will be ignored in the examples that follow.

Next, and this is of course a major difference, the meaning of the above symbols is enlarged. First, T and P may refer to objects within different DBs. Next, they may define sets of relations. Consequently, W may designate a set of workspaces called in the sequel multiworkspace. Finally, W may also designate relations within an (M)DB. The sequel defines the corresponding details.

3.2 Designators.

DSL designator forms are X' or X'.X'', where X'' is an attribute name and X' is a relation name or a tuple variable name. The form X'.X'' means that one designates only X'' that are within X'. MDSL designator forms are :

(a) X1 , X2.X1 , X3.X2.X1 etc
(b) X3..X1 , X4.X3..X1 , X4...X1 , X4..X2.X1 etc
(c) Y. , Y.. , Y... etc
(d) .Y , .Y. , .Y.. etc

where :

- X1 is the name of an attribute, or of a relation or of an (M)DB or of a tuple variable. Or, it designates a so-called in the sequel semantic variable.

- X2 is the name of a relation, or of an (M)DB or of a corresponding variables.

- X3, X4,... designate MRDs or the corresponding variables.

- Y is a designator of a form within (a) or (b).

The designators (a) designate first all objects or variables named X1, then only X1 that are components of an X2 etc. The designators (b) designate all X1 that are components of a component of an X3 etc. The forms (c) mean first that Y designates at least a relation, then at least a DB etc. Finally, (d) means that one designates only attributes or the corresponding variables, or only relations etc.

It is obvious that any DSL designator is also MDSL designator. However, while in DSL any object designator denotes exactly one object, since relation names in a DB are all different, it is not the case of MDSL. For this reason MDSL distinguishes two types of object identifiers :
- unique identifiers that designate exactly one object,
- multiple identifiers that designate more than one object.

A unique identifier may become multiple when the scope is enlarged. For instance, 'R' is a unique identifier within the scope MY-REST, while it is a multiple identifier for the scope NIGHTLIFE.

Nevertheless, since the designator may be of any length, for any object and any scope one always dispose of at least one unique identifier.

3.3 Commands.
3.3.1 Main commands.

MDSL main commands are :

OPEN, CLOSE, LET, RETRIEVE, MODIFY, STORE, DELETE.

The form of OPEN command is :

OPEN D1, D2,...

where D are database or multidatabase designators (X1 is an (M)DB name). One multidatabase name opens all underlying DBs, saving thus some work to users. Vice versa, CLOSE D1, D2,... closes all the corresponding DBs. CLOSE without any D closes everything.

LET command acts like RETRIEVE except that it defines only the intension in the corresponding result. It may be useful as a name definition, to be applied afterwards in the extensional commands.

The other commands perform a retrieval or a modification of an (M)DB. The multidatabase usage of these commands is described further. The monodatabase case is DSL compatible.

3.3.2 Other commands.

The other commands of MDSL are :

(i) CREATE_MB, DESTROY_MB, DISPLAY_MB, DISPLAY_ODB, EXECUTE, HELP_M, CONSTRUCT, PROC, QUIT.

(ii) All commands of MRDS.

The commands DISPLAY display the composition of an MDB or names of DBs that are open. The command EXECUTE allows to invoke any command of the Multics system. The last three commands are for the precompiler. The others are self explanatory.

3.4 Multidatabase retrieval.
3.4.1. Elementary queries.

MDSL query is elementary if it is also DSL query or all designators are unique identifiers and the differences with respect to DSL query are between the following ones :
- some relation names are prefixed (with (M)DB names),
- W is a database relation,
- some designators consist of an attribute name only.
- the equi-joins among key attributes connected in an obvious unique way are implicit, i. e. are not specified.

An elementary query is multidatabase if, of course, some of the objects it designates are in different DBs. This may concern T, or P, or W with respect to T or P. In particular, if W designates database relations, then the inheritance rule leads to the creation of entirely new relations, regardless the schemas and tuples that could exist. Note that the clear-cut distinction between the notions of retrieval and of modification that exists in the database approach, becomes relative in the multidatabase one.

Example 1.

1. Retrieve from MY-REST and from CINEMAS the names of the restaurants and of cinemas that are on the same street.

```
OPEN NIGHTLIFE

RETRIEVE
-RANGE (X R) (Y CINEMA.C)
-SELECT X.RNAME Y.CNAME
-WHERE (X.STREET = Y.STREET)
```

2. Copy C of GAULT-M, into MY-REST.

```
OPEN GAULT-M MY-REST

RETRIEVE MY-REST.C
-SELECT GAULT-M.C
```

3.4.2. Multiple queries.

Although the concept of an elementary multidatabase query fulfills the need (2), it does not fulfill the need (3). For this purpose, MDSL proposes the concept of a multiple query. MDSL query is such a query if it involves multiple identifiers, or semantic variables or the choice of implicit joins is not unique. One multiple query acts like several, may be very many, elementary queries. These queries are called relevant subqueries. The result of a multiple query is typically a set of relations.

3.4.2.1. Multiple identifiers.

The first way to formulate multiple queries is to use multiple identifiers. If a query with such identifiers does not use also the semantic variables, then it is processed as follows :

(A1) - We call scope of an identifier the corresponding objects. We also call subquery a query that results from a multiple query when any multiple identifier is replaced with a unique identifier of an object within its scope and one choice of implicit joins is added. A relevant subquery is produced from a subquery through the following rules :

- Let T and P be respectively the target list and the predicate of a subquery. Let T' and P' be the respective parts of the relevant subquery. Then :

(i) - the designators of T' are all these of T that designate existing objects. If T has no such designators, then no relevant subquery results from. The subquery is then called irrelevant.
(ii)- let P'' be the largest predicate referring to existing objects and such that for some P''', that might be empty :

P = P'' \/ P'''

Then, if P'' is not empty, then P' = P''. Else the subquery is again irrelevant.

- The set of relevant subqueries is the one that results from all the choices of the unique identifiers and of the corresponding implicit joins, if any. This set may be processed in any order.

- The result of a subquery may be a workspace. W designates then the corresponding set that we call multiworkspace. The name of a workspace within W consists of the prefixes that were added to the multiple identifiers, in order to produce the subquery. The order of the prefixes may be important.

Example 2.

Retrieve the names, the telephone number and the remarks, if any, that characterize the restaurants that are chinese for MICHELIN or for GAULT-M.

```
OPEN REST-GUIDES

RETRIEVE -SELECT R.RNAME R.TEL R.REM
-WHERE (R.TYPE = `CHINESE`)
```

R is a multiple identifier. (A1) leads to two subqueries with, respectively, two sets of unique identifiers :

```
{MICHELIN.R.RNAME, MICHELIN.R.TEL}
{GAULT-M.R.RNAME, GAULT.M.TEL, GAULT-M.REM}
```

The result is the set of two relations. Note that one knows from this set not only the specified attribute values, but also which guide considers a restaurant as chinese (we recall that guides may disagree upon the characteristics of a restaurant). Note also that to the same restaurant may correspond two different telephone numbers.

The above query illustrates an important property of MDSL queries that we call scope independence. On the one hand, queries stay valid, i. e. correctly express the corresponding intension, when one shrinks a multiple identifier scope ((A1) automatically produces then less subqueries, until the identifier becomes unique). On the other hand, as long as the same object names are used, the scope may be extended to any size ((A1) produces then automatically more subqueries, while unique identifiers become gracefully multiple ones). Thus, the user may drop or add DBs without being forced to reformulate his query.

The concept of a multiple identifier fulfills the need (3), as long as the corresponding objects share names. Although not general, this case is quite frequent. People sharing a universe tender indeed also to share some common vocabulary.

3.4.2.2. Semantic variables.

We call semantic variable a designator which values are designators. This concept completes the one of the multiple identifier for the case when the names of the corresponding object differ. The values of the domain of a semantic variable are defined by a declaration that we call semantic RANGE. In contrast, the usual RANGE is in MDSL called tuple RANGE.

The forms of the semantic RANGE are :

```
(a) -RANGE-S {[Z] N1 .. Ni .. Nk}
(b) -RANGE-S ([Z] N1 .. Ni .. Nk)
```

where N are designators and Z is the name of the variable. Both N and Z may be compound. The corresponding components may then be shared by several semantic RANGEs. N and Z may also be tuples of designators. They have then all of the same length and each of them is within parentheses. The only difference between the forms (a) and (b) is that (b) requires the values of Z to be generated in the indicated order.

The brackets around Z are optional. Without them, any Ni selected to the result keeps its name, i. e. `Ni`. Else, all these names are replaced with `Z`.

Within (a), N may in particular be a semantic variable that we call implicit. The domain of an implicit variable results from its name. The reason for that are special characters that one may apply. These characters and the corresponding meanings are :

`*` <--> any character string,
`?` <--> any character at the corresponding position,
[D] <--> the set of the unique identifiers of the objects that are components of the object(s) designated by D.

RANGE with an implicit N is considered as the one where N is replaced with its domain.

A query with the semantic variables is considered as the set of elementary queries resulting from the substitutions of the unique identifiers to both the variables and the multiple identifiers (note that a semantic variable may in particular refer to a multiple identifier). The corresponding rules are similar to the ones of (A1) and may be found in /LIT83a/.

Example 3

1 Retrieve all the restaurants from REST-GUIDES that a guide considers chinese.

```
OPEN REST-GUIDES

RETRIEVE
-RANGE-S {X R REST}
-SELECT X -WHERE (X.TYPE = `CHINESE`)
```

X is a semantic variable which values are the names R and REST. Since REST is a unique identifier, the corresponding substitution produces an elementary query. In contrast, since R is a multiple identifier, the corresponding substitution produces two elementary queries. The final result is the set of three relations :

{MICHELIN (R#,...,TEL), KLEBER (REST#,..,MEANPRICE), GAULT-M (R#,..,REM)}

This set constitutes the multiworkspace W.

An alternative form for RANGE could also be :

-RANGE-S {X R*}

2. Retrieve from NIGHTLIFE first the restaurants then the cinemas where one can go for less than 25 ff.

```
OPEN NIGHTLIFE

RETRIEVE
RANGE ((Z  X X# Y)  (MY-REST R R# M)  (CINEMAS C C# P))
-SELECT X
-WHERE (Z.X.X# = Z.Y.X#) & (Z.Y.PRICE < 25)
```

3. Retrieve from GAULT-M into MY-REST all data relative to the restaurants rated more than 17/20.

```
OPEN GAULT-M, MY-REST

RETRIEVE MY-REST.X
-RANGE-S {X [GAULT-M]}
-SELECT GAULT-M.X
-WHERE (GAULT-M.QUAL > 17/20)
```

The values of X are the names of all relations of GAULT-M. The elementary queries that result from the substitution have no the clauses corresponding to the implicit joins. The execution of the query erases the previous schema and content of MY-REST, if any, given the inheritance rule. The new schema is the one given in the section 2. The new content consists of all the restaurants with QUAL > 17/20, of the courses corresponding to these restaurants through the implicit join, and of the related menus.

The queries using semantic variables are also scope independent as long as the additional DBs use the same names. In addition, they keep this property even if the new names differ, as long as they are within the domains. Otherwise, it may suffice to simply add a new name to a domain.

3.4.2.3 Target list options.

DSL target list T is assumed to refer to attributes that all exist. There is no such assumption in MDSL, basically since the substitution process may lead to a subquery which T and even P are larger than the scope (ex.2). This leads to the problem of the definition of the result of a (sub)query which asks for attributes that might, or might not, be in the scope. MDSL approach to this problem consists first from the already discussed rules for the relevant subqueries. Next, from the assumption that any attribute designator in MDSL T bears requirements called options. The options precise the composition of the result, when attributes may do not exist or, in contrast, may reveal redundant.

Let A be an attribute designator. The default (implicit) option on A, noted `A`, or `A)`, at the end of a list, means that even if A does not exist in the scope, the tuples without A may still enter the result (note that this option is DSL compatible as long as all A are in the scope). The other options are :

`A;` <--> the attribute to which A refers, directly or through substitutions must exist.
`A1|A2|..An|` <--> only one of the designated attributes is selected. If more than one of them exist in a scope, then the priority is given first to A1, then to A2 etc.
`A1:A2` <--> A1 is not selected without A2.

A subquery is relevant iff, in addition to the requirements discussed in the previous sub-sections, it respects the options (note that this requirement does not add anything if T bears only the implicit option). In particular, the options may be nested and applied each to sublists of designators, put therefore in parentheses. For instance, one may define the options :

(A1|A2; (A3, A4):A5)

which meaning is clear.

Note that the notion of options may also be applied to null values.

3.4.3. Dynamic attributes.

MDSL query may refer to to dynamic attributes that are names denoting values dynamically defined from attribute values for the purpose of the query. One may in this manner instantaneously and subjectively homogenize heterogeneous data types (need 4).

The declaration of a dynamic attribute has the following forms :

```
-ATTR-D N
-FROM T
-INTO T'
```

where :

- N designates the dynamic attribute.
- T represents the mapping from some existing attributes.
- T' represents the inverse mapping.

The clause INTO is necessary only when the dynamic attribute is used for a modification of a DB, a multiple one in particular, and T' cannot be deduced from T automatically (note the interesting research problem). One may avoid then FROM clause. T (T') may be a formula, a dynamically defined translation table, a standard function invocation or a nonstandard function invocation. A nonstandard function is declared like DSL nonstandard function. Details of the corresponding forms for T and T' may be found in /VIG84/.

Example 4.

Assuming (subjectively) that `***` rating of MICHELIN corresponds to GAULT-M.QUAL greater than 17/20, retrieve from these DBs the `***` restaurants. Present GAULT-M ratings in STAR scale.

```
OPEN REST-G
-ATTR-D QUAL
-FROM GAUL-M-STARS(QUAL)
-RANGE-S {[STARS] STARS QUAL}
-SELECT R -WHERE (R.STARS = `***`)
RETRIEVE
```

RANGE-S refers of course to the dynamic attribute that, in the case of name conflict, has always the priority over an existing one. GAUL-M-STARS is a nonstandard function, stored thus in a segment of the same name. It could be produced dynamically through EXECUTE command, invoking any Multics text editor. It expresses, in an arbitrary host language, the algorithm :

```
QUAL <-- if QUAL > 17/20 : QUAL = `***` endif;
```

Note that QUAL is defined only for the values of interest i. e. is undefined for `**` for instance.

3.4.4. Standard functions.

Any standard function of DSL is a standard function of MDSL, being, in addition, considered as generalized for the multidatabase usage. This means in particular that one may use it with semantic variables and multiple identifiers. On the other hand, the multidatabase environment calls for new specific functions. In particular, for the following ones, detailed for some in /LIT83a/ :

NAME. NAME(N) returns the names of the objects that N designates directly or after the substitutions. NAME(.N) returns the names of the containers of the object designated by N (or null), NAME(..N) = NAME(.NAME(.N)) etc.

EXTEND. EXTEND(R A1 .. Aj) extends the relation(s) designated by R with attributes which names are designated through A and values are constant and equal to the attribute names.

NORM. NORM(A1 .. Aj) applied to a set of relations merges into one tuple any number of tuples with the same values of key attributes designated through A.

NORM may thus in particular transform the result of a multiple query where a variable number of tuples may correspond to a real object, into a set of relations where each object is described through exactly one tuple (need 4 again).

UPTO. UPTO(n (A) [B]) applied to a multiple query provides at most n >= 1 tuples sharing the values of the attributes designated in the list A. Priorities correspond then to the order of the list B that designate DB names. A, n and B are optional. If A is not specified, the query processing stops after nonnull response from n DBs. The default value of n is 1. Finally, null B means that the user has no preference.

Example 5.

1. Names of the guides that recommend `Maxim's`.

```
OPEN REST-G

RETRIEVE
-RANGE-S {(X Y) (R RNAME) (REST NAME)}

-SELECT [EXTEND (X [NAME (.X)])].[NAME (.X)]
-WHERE (X.Y = `MAXIM'S`)
```

The clause select specifies the projections on the new attribute that EXTEND creates for each X. Brackets are used for the compatibility with DSL, although they are not absolutely necessary (LINUS for instance does not use them).

2. Apply NORM to the result of the ex. 3.1, on the basis of the property (5) stated in the section 2.

```
OPEN REST-G

RETRIEVE
-RANGE-S {(X [Y]) (R RNAME) (REST NAME)}

NORM (X.Y X.STREET)
-SELECT X -WHERE (X.TYPE = `CHINESE`)
```

NORM is before select since it is a so-called set function (LINUS terminology).

3. Retrieve 1 description of any `Maxim's`, possibly Michelin description.

```
OPEN REST-G

RETRIEVE
-RANGE-S ((X Y) (R RNAME) (REST NAME))

UPTO ( (X.Y X.STREET) [MICHELIN])
-SELECT X -WHERE (X.Y = 'MAXIM'S')
```

3.5 Multidatabase modifications.

MDSL queries that modify an (M)DB through an update, an insertion or a deletion, are either DSL queries or differ from such queries by the following properties :

1. One applies the new concepts : multiple identifiers, semantic variables etc.

2. W designates database relations or attributes within DBs or (multi)workspaces.

3. STORE is qualified with P which then filters the tuples to be inserted.

4. DELETE is characterized with W which then designates the objects that receive the corresponding projections of the deleted tuples, eventually extended with null values for attributes own to W.

5. The target attribute names differ from the source attribute names.

MDSL considers in general that the correspondence between the target and the source names results from the equality of names. The order is unimportant. In the case (5), one specifies the correspondence through semantic variables. The names of these

variables should then be bracketed and equal to the ones of the desired targets.

Using all these possibilities one may for instance modify a DB, using a multidatabase P. Or, through semantic variables or multiple identifiers, one may modify in one query several DBs. Next, through the usage of database object designators in W one may copy (STORE) or move (DELETE) data between (M)DBs. In particular, source data type may be transformed through dynamic attributes, etc. The queries that result are scope independent, with the limitations pointed out for the GET command.

Example 6.

1. Change to `456` the phone number `123` in all DBs of REST-GUIDES.

```
OPEN REST-GUIDES

-RANGE-S {TEL T#}
-SELECT TEL -WHERE (TEL = `123`)
MODIFY `456`
```

2. Assume that a multiworkspace W that is, in occurrence, Multics multisegment, contains two relations named MICHELIN and GAULT-M, constituted from tuples to be inserted to relations R in the corresponding DBs. Insert these tuples.

```
OPEN REST-G
STORE R -IF W
```

IF means "input file" in DSL terminology.

3. Put to MY-REST the restaurants rated more than 17/20 in GAULT-M, as well as the corresponding courses and menus.

```
OPEN GAULT-M, MY-REST

STORE MY-REST
-RANGE-S {X R C M}
-SELECT GAULT-M.X
-WHERE (GAULT-M.X.QUAL > 17/20)
```

Note the presence of implicit equi-joins, materialized when the command is processed for X =/= R. Note also that in contrast to GET from ex. 3.3, STORE does not erases the existing content of MY-REST (there is no inheritance rule).

4. Delete in REST-GUIDES all the restaurants with the phone number `123` (ex. 2, query 6).

```
OPEN REST-GUIDES

-RANGE-S (X.TEL R.TEL REST.T#)
-SELECT X -WHERE (X.TEL = `123`)
DELETE
```

Note that, like in (1) above, one may also formulate this query using a semantic variable on telephone number only.

5. Extract from MY-REST into MY-ONCE-A-YEAR-REST the restaurants that are more expensive than 700 ff per dinner, as well as the corresponding courses and menus. Assume that the attribute names are the same, except for TYPE and STREET that are shortened to T and ST in the target.

```
OPEN MY-REST, MY-ONCE-A-YEAR-REST
-RANGE-S {X R M}
-RANGE-S {(Y Z) (MY-ONCE-A-YEAR-REST MY-REST)}
-RANGE-S {[(ST T)] (STREET, TYPE)}

-SELECT Z.X -WHERE (Z.R.AVPRICE > 700)
DELETE Y.X

-SELECT Z.C -WHERE (Z.R.AVPRICE > 700)
STORE Y.C
```

Note that C could not be deleted (why ?).

4.CONCLUSIONS.

MDSL is an operational application of the multidatabase approach. While DSL allows to manipulate a database only, the proposed concepts allow to manipulate several databases as well. The corresponding manipulations may be retrievals, updates, insertions or deletions. In particular, data may flow not only between a database and a workspace, but also among databases themselves.

For this purpose MDSL applies new concepts like the ones of elementary or multiple query, of multiple identifier or semantic variable etc. These concepts respond especially to the needs identified through the case study. In particular they render MDSL easy to use, in the sense that the queries are typically expressed in only one command. In that, MDSL extends one of the main goals of the relational data model.

Most of the discussed concepts result also from our previous work on a theoretical ALPHA based multidatabase language that we called MALPHA. We conjectured in particular during this work that, since ALPHA is the theoretical root of the practical relational languages, the new concepts should apply to practical languages as well. MDSL is one proof of this applicability. We feel this proof important, given first that DSL is an SQL-like language, being thus very similar to the ones of most of the relational DBMSs. Next, given that it shows then rather clearly that QUEL based system may offer multidatabase facilities as well.

Even a glance on the state-of-the-art shows that the multidatabase manipulation languages are necessary. First, since they rather drastically increase the utility of the relational systems. Next, since, it should also be the case other data models, the functional ones in particular. Furthermore, since such languages will be necessary for users of videotex or documentary systems providing already the access to thousands of databases. Finally, they will be necessary for an easy usage of personal databases that the progress in networking and in microcomputers will bring massively.

REFERENCES.

/ABD83/ Abdellatif, A. Processing multiple queries in a multidatabase system (MRDSM). Mem. de Fin d'Etudes Fac. de Tunis. Ed. INRIA (June 1983), 83. In french.

/ADI78/ Adiba and all. Issues in distributed data base management systems : a technical overview. VLDB Berlin (Sep. 1978), 89-110.

/ANS75/ ANSI-SPARC interim report on data base management system. Doc. 7514TS01, Washington DC, (Feb. 1975).

/BAC77/ Bachman, Ch., Daya, M. The role concept in database models. VLDB, Tokyo, (Oct. 1977), 464-476.

/CHE76/ Chen, p. The entity-relationship model : Toward a unified view of data. TODS 1, 1, (March 1976), 9-36.

/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 Database: A Practical Foundation for Productivity. CACM, 25, 2 (Feb 1982).

/DAY83/ Processing Queries over Generalization Hierarchies in a Multidatabase System. VLDB 83, Florence, (Oct. 1983), 342-353.

/HAM79/ Hammer, M., McLeod, D. On database management system architecture. Lab. for Comp. Sc. MIT, MIT/LCS/TM-141, (Oct. 1979), 35.

/LAN82/ Landers, T., Rosenberg, R. L. An overview of MULTIBASE. DISTRIBUTED DATA BASES. North-Holland, 1982, 153-184.

/LIN81/ Lindsay, B. Object Naming and Catalog Management for a Distributed Database Manager. 2-nd Int. Conf. on Distributed Computing Systems. Paris, (Apr. 1981), 31-40.

/LIT79/ Litwin, W. Distributed data bases : a way of thinking about. International Seminar on Distributed Data Sharing Systems. Aix-en-Provence (May 1979), 19.

/LIT80/ Litwin, W. A model for a distributed data base. 2-nd ACM Comp. Exp. University of Laffayette, Luisiane (Feb. 1980).

/LIT81/ Litwin, W. Logical model of a distributed data base. Second 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 1983), 482-505.

/LIT83a/ Litwin, W. MDSL : a multidatabase manipulation language. EUTECO, Varese, (Oct. 1983). North-Holland 1983, 661-681.

/LYN83/ Lyngbaek, P., McLeod, D. An Approach to Object Sharing in Distributed Database Systems. VLDB 83, Florence, (Oct. 1983), 364-376.

/MOT81/ Motro, A., Buneman, P. Constructing Superviews. SIGMOD, Ann Arbor, (May 1981), 56-64.

/MUL82/ Multics Relational Data Store (MRDS) Reference Manual. Cii Honeywell Bull. Ref. 68 A2 AW53 REV4, (Jan. 1982).

/ROT78/ Rothnie, J. B. Distributed database management. VLDB, Berlin, (Sep. 1978).

/STO77/ Stonebraker, M. A distributed database version of Ingres. 2-nd Berkeley Workshop on DDM and Comp. Networks. Berkeley, Calif (May, 1977).

/TAN81/ Tanaka, K., Kambayashi, Y. Databases into a distributed database. VLDB 81 Cannes France (Sep. 1981). 131-143.

/VIG83/ Vigier C. Dynamic Attributes in MRDSM system. Tech. note, (Oct. 1983). In french.

/WIE83/ Wiederhold, G. Database design. McGraw-hill Book Company, 1983.

/WON83/ Wong, K. MRDSM-V2 architecture and functionalities Tech. note, (Sep. 1983). In french.