VLDB Journal, 2, 153–171 (1993), Fred J. Maryanski, Editor
©VLDB

Query Languages for Relational Multidatabases

John Grant, Witold Litwin, Nick Roussopoulos, and Timos Sellis

Received December 5, 1990; revised version received June 18, 1992; accepted November 2, 1992.

Abstract. With the existence of many autonomous databases widely accessible through computer networks, users will require the capability to jointly manipulate data in different databases. A multidatabase system provides such a capability through a multidatabase manipulation language, such as MSQL. We propose a theoretical foundation for such languages by presenting a multirelational algebra and calculus based on the relational algebra and calculus. The proposal is illustrated by various queries on an example multidatabase. It is shown that properties of the multirelational algebra may be used for optimization and that every multirelational algebra query can be expressed as a multirelational calculus query. The connection between the multirelational languages and MSQL, the multidatabase version of SQL, is also investigated.

Key Words. Multidatabase, multirelational algebra, multirelational calculus, query optimization.

1. Introduction

The great diversity of computing systems and software accessible through computer networks has created a major problem in the interoperability and integration of heterogeneous systems and components. A special case in the area of database systems, and the topic of this article, is the logical grouping and cohesive access of relational databases.

A large mainframe with a database system or a database server like Compuserve may support dozens of databases. A local network can have several database servers and a database for hundreds of workstations (e.g., the ADMS system, Roussopoulos and Kang, 1986). A public database access system, such as the videotex system Teletel in France, may provide access to thousands of databases for millions of people. With the introduction of INGRES in Ultrix 4.0, the OS2/DB presentation

John Grant, Ph.D., is Professor, Dept. of Computer and Information Sciences, Towson State University, Towson, MD 21204; Witold Litwin, Ph.D., is Professor, Universite Paris Dauphine; Pl. du Mal. de Lattre de Tassigny, 75775 Paris Cedex 16, France; Nick Roussopoulos, Ph.D., is Professor, and Timos Sellis, Ph.D., is Associate Professor, Dept. of Computer Science, University of Maryland; College Park, MD 20742, USA.

manager and other inexpensive relational systems, SQL databases are spreading out on workstations and PCs.

A major consequence of this proliferation of databases is that users will usually have data in many databases and will frequently need to jointly manipulate data in different databases. A Teletel user who wishes to go out to a movie and dinner may wish to find the cinemas and restaurants that are on the same street in a cinema guide and restaurant guide database. The manager of a company may need to see the account balances that the company has at different bank branches. This manager may further wish to query all the bank branches (and their databases) simultaneously. A traveler looking for the cheapest route may need to query several airline, rail, and bus databases. Furthermore, an address change may require the updating of the corresponding databases.

Such manipulation requires functions that do not exist in standard manipulation languages which are designed to manipulate data in a single database. Classical data manipulation languages deal with single relations as objects of manipulation, while the need is to allow sets of relations as the objects. Additionally, data in different databases may be presented in a different format so that corresponding relations may not be union-compatible. The differences between similar data result from different perceptions of the same real universe—local needs, competition for customers, political incompatibility, etc. It would be impossible to provide integration via a global schema because there would be no agreement on the global administrator. The example databases given in the next section, based on actual databases, show some of the differences that may occur.

A system for the manipulation of data in autonomous databases is called a *multidatabase system* (MDBS) and the corresponding language is called a *multidatabase manipulation language* (MML) (Litwin et al., 1982). Databases that may be manipulated together without global integration are called interoperable (Litwin and Abdellatif, 1986). Litwin et al., 1989 present the multidatabase extension of SQL, called MSQL. All the functions of SQL are by definition functions of MSQL. MSQL also contains new functions designed for a nonprocedural manipulation of data in different and nonintegrated SQL databases. The rationale is that a user's query over these databases should be expressible as a single statement. For this purpose, MSQL manipulates sets of tables, called *multitables* or *multirelations.* In particular, the result of a query may be a multitable.

One way to deal with a multitable query is to decompose it into the set of relational queries it expresses and then to transform those queries to the relational algebra for optimization. Clearly, this is not necessarily the most efficient method. In particular, the techniques of Sellis (1988) for common subexpression processing are applicable. In addition to the detection of common subexpressions, the processing of useless queries should be avoided. A promising technique for dealing with multitable queries is to define an extension of the relational algebra and calculus for multitables. The system may then optimize a query both at a high level multitable formulation (first in the calculus and then in the algebra) and then perform optimization at the single relational level. This is the approach we introduce here.

Research related to multidatabase systems has been reported extensively in the past. In some sense, all the work in Distributed (homogeneous) Database Systems (Ceri and Pelagatti, 1984) is related. However, the work on heterogeneous databases is of more interest here; a comprehensive survey on this subject appears in Elmagarmid and Pu (1990). Most of the systems and approaches described address issues related to the interconnection of database systems that use different query languages and methods to allow exchange of data among such databases. The question of which model should be used to express queries in such systems is not addressed, however. Our research attempts to define such a model in the form of an algebra and a calculus that generalizes the standard relational algebra respectively for relational multidatabases.

The structure of this article is as follows. Section 2 introduces the multidatabase model, presents the basic definitions and the example databases. Section 3 defines the multirelational algebra and uses it to solve queries on the example databases. Section 4 presents the multirelational calculus, uses it to solve some of the example queries, and reduces the multirelational algebra to the multirelational calculus. Section 5 shows the connections of the multirelational algebra and calculus with MSQL, the extension of SQL to multidatabases. Section 6 provides properties of the multirelational algebra that can be used for query optimization. Section 7 concludes this article.

2. Model and Example for Multidatabases

We begin with a few definitions from relational databases. A *relation schema* is a relation name together with a set of attributes identified by names and drawing values from some domains. An element of a relation defined by the schema is called a *tuple*. A *relation* is a collection of tuples defined on the same schema. A *database schema* is a database name together with a set of relation schemas. A *database* is a set of relations satisfying the database schema. A *query* is a dynamic definition of a relation, including schema and instance, on a database.

A *multidatabase* is a collection of databases; it may be explicitly or implicitly named. A *multirelation* is a set of relations dynamically defined by a multirelational query. The relations of a multirelation are usually in different databases of a multidatabase; they may have different arities, attribute names, and domain types for models of the same elements of the real universe. Database and multidatabase names may be used as prefixes to clarify the naming. A *multirelational query* defines a (target) multirelation from a (source) multirelation through multirelational operations (to be defined later). A query contains some relation names, attribute names, and possibly domain value specifications (e.g., value expressions or unit and precision definitions, that are bound by the system to the actual names and value types). Some of the issues involving name equality, different units, and precision will be considered in Section 6.

A *multirelational(data)model* extends relational queries to multirelational queries. The two axioms given below relate our multirelational data model to the relational model. Axiom 1 refers to the naming of operations. Axiom 2 refers to compatibility.

Axiom 1: A multirelational operation, O, transforms one or more multirelations into a multirelation.

Axiom 2: For any relation, R, and relational operation on R, there is a multirelational operation with the same or appropriately modified name that provides the same result when applied to the multirelation $M = \{R\}$.

By compatibility, the relational model is a special case of the multirelational model with the cardinality of each multirelation equal to 1.

An operation, O, is evaluated first through binding to the actual relation names to find the multirelation(s) it should operate on, i.e. its argument(s). If the binding is empty, no multirelation is produced by the query. This is the equivalent of an error output in today's relational systems. Once the arguments of O are identified, O is further evaluated through the application of relational operations to each element of its arguments. It may happen that an operation cannot be evaluated. In that case no corresponding relation is produced by the query. This is different from the case in which the evaluation of a query results in the empty relation. Thus, in our model a query (with proper binding) always produces a multirelation that nevertheless might be empty. Notice that POSTGRES (Stonebraker and Rowe, 1986) has a similar behavior when accessing fields with query expressions that evaluate to relations. In some sense, such fields simulate sets of relations (multirelations in our context) but the model has never been defined formally and the semantics of such fields are not clear. Our proposal attempts to define precisely the semantics of sets of relations and the operations available on them.

When a multirelational operation is evaluated, it may happen that it is bound (refers) to an attribute not in the given relation. For example, a projection may refer to an attribute not in the relation to which the projection is applied. In the basic multirelational model that is used throughout this article, such an operation will be considered as not evaluable. It is also possible to define an extended multirelational model where the attributes not in the given relation are ignored. In this case, for a projection any attribute not in the relation is omitted from consideration. Any tuple or relation obtained by using the basic model is also obtained by using the extended model, but not vice versa. Both models have intuitive appeal. The extended model requires extensions to the current definitions of the relational operations. In particular, the extended relational projection should have at least one attribute so that no relation with an empty set of attributes can be obtained.

The databases used in the next three sections are described below. These databases are patterned after existing databases of the French banks BNP, Societe Generale, and CIC (*bnp*, *sg*, and *cic*, respectively). The other three, *etoile*, *nation*, and *opera*, are databases of BNP branches named for their location in Paris and used for local processing. The latter three contain only data about their customers.

Banks is a multidatabase, *Banks* = $\{bnp, sg, cic\}$.

Database *bnp:*
branch (br#,brname,street,street#,city,zip,tel)
account (acc#,cl#,balance,br#)
client (cl#,clname,cltel,cltype,street,street#,city,zip)

Database *sg:*
branch (br#,brname,street,street#,city,zip,tel,class)
account (acc#,br#,cl#,balance)
client (cl#,clname,cltel,cltype,street,street#,city,zip)

Database *cic:*
branch (br#,brname,street,street#,city,zip,tel)
account (acc#,br#,cl#,balance,open_date)
client (cl#,clname,cltel,cltype,street,street#,city,zip)

Database *etoile:*
account (acc#,cl#,balance,open_date)
client (cl#,clname,cltel,street,street#,city,zip)

Database *nation:*
account (acc#,cl#,balance,open_date)
client (cl#,clname,cltel,street,street#,city,zip)
Database *opera:*
account (acc#,cl#,balance,open_date)
client (cl#,clname,cltel,street,street#,city,zip,cltype)

3. Multirelational Algebra

The operations of the multirelational algebra proposed below extend the operations of the relational algebra on the model of multirelations presented in the previous section. Each operation is based on the corresponding operation of the relational algebra and yields the same result when applied to a multirelation that consists of a single relation. We do not claim minimality for the set of operations that we define. The multirelational algebra is then used for writing 12 queries on the multidatabase example of the previous section. To clarify the distinction between the relational and multirelational operations, we precede each of the latter with the letter "M."

We start by considering the unary operators MPROJECT and MSELECT. $\{R_1,...,R_n\}$ is an arbitrary multirelation. The first operation is projection

$$\text{MPROJECT}(\{R_1,...,R_n\}; A_1,...,A_m) = \\ \{\text{PROJECT}(R_1; A_1,...,A_m),..., \text{PROJECT}(R_n; A_1,...,A_m)\}$$

where PROJECT(R_i; $A_1,\ldots,A_m$) is the usual projection of R_i on the attributes $A_1,\ldots,A_m$ if all A_j appear in R_i; if some A_j do not appear in R_i, then PROJECT(R_i; $A_1,\ldots,A_m$) does not exist. Thus the multirelational projection has as its result the multirelation consisting of the projections on the individual relations of the given multirelation. Note that the result may have fewer than n relations.

The next operation is a multirelational selection,

$$\text{MSELECT}(\{R_1,\ldots,R_n\}\text{: C}) = \{\text{SELECT}(R_1\text{: C}),\ldots, \text{SELECT}(R_n\text{: C})\}$$

where C is a condition involving the attributes of R_i, constants, relational and logical operations. If C contains an attribute not in R_i, then SELECT(R_i: C) does not exist. Thus the multirelational selection has as its result the multirelation consisting of the selections on the individual relations of the given multirelation. Again, the result may have fewer than n relations.

Binary operations are applied to two multirelations. The standard definition (cross product) applies the binary operation to every applicable pair of relations from the two multirelations. In particular, the multirelational set operations MUNION, MINTERSECT, and MDIFFERENCE apply to union-compatible relations only. In practice it is useful to allow a version of binary operations where these are applied only within a database. A notation will be introduced for this purpose. The only operation formally defined here is MJOIN, but MPRODUCT, as well as the set operations, also will be illustrated in the examples. The join operation is defined as

$$\begin{aligned}&\text{MJOIN}(\{R_{11},\ldots,R_{1n}\},\{R_{21},\ldots,R_{2m}\}\text{: C}) = \{\text{JOIN}(R_{11}, R_{21}\text{: C}),\ldots,\\ &\quad \text{JOIN}(R_{11}, R_{2m}\text{: C}),\text{JOIN}(R_{12}, R_{21}\text{: C}),\ldots, \text{JOIN}(R_{1n}, R_{2m}\text{: C})\}\end{aligned}$$

where C is the join condition: a conjunction of equalities appropriate for the multirelations. Duplicate columns are eliminated as for the standard natural join. If condition C is inapplicable to the pair R_{1i}, R_{2j}, then JOIN(R_{1i}, R_{2j}: C) does not exist. Thus the multirelational join has as its result the multirelation consisting of all possible joins between the relations of the two multirelations. The result may have up to $m \times n$ relations.

Next we give multirelational algebra queries for 12 queries on the multidatabase and databases of the previous section.

Q1. For all clients in the *Banks* multidatabase find their client number, client type, and zip code.

MPROJECT({*Banks*:client};cl#,cltype,zip)

In this case *Banks*:client is the source multirelation; note the use of the colon as a qualifier for the multidatabase. The result is a multirelation that consists of three relations:

{PROJECT(*bnp*.client;cl#,cltype,zip),
PROJECT(*sg*.client;cl#,cltype,zip),
PROJECT(*cic*.client;cl#,cltype,zip)}

Q2. For all accounts in the *sg*, *cic*, *etoile*, and *nation* databases, find the account number and opening date.

MPROJECT({*sg*, *cic*, *etoile*, *nation*:account};acc#,open_date)

Because the designator open_date is inapplicable to the relation *sg*.account, the result is a multirelation that consists of three relations and not four (as would be the case otherwise):

{PROJECT(*cic*.account;acc#,open_date),
PROJECT(*etoile*.account;acc#,open_date),
PROJECT(*nation*.account;acc#,open_date)}

Q3. In the *Banks* multidatabase and *opera* database, find the accounts for which the balance is greater than 1,000.

MSELECT({*Banks*,*opera*:account}: balance > 1,000)

The result is a multirelation that consists of four relations:

{SELECT(*bnp*.account: balance > 1,000),
SELECT(*sg*.account: balance > 1,000),
SELECT(*cic*.account: balance > 1,000),
SELECT(*opera*.account: balance > 1,000)}

Q4. In the *Banks* multidatabase and *nation* database, find the branch number, branch name, and telephone number for all branches on the Av Champs Elysees.

MPROJECT({MSELECT({*Banks*,*nation*:branch}: street =
'Av Champs Elysees')};br#,brname,tel)

In this case the MSELECT operation does not apply to *nation*. The result is a multirelation that consists of three relations:

{PROJECT(SELECT(*bnp*.branch: street = 'Av Champs Elysees');br#,brname,tel),
PROJECT(SELECT(*sg*.branch: street = 'Av Champs Elysees');br#,brname,tel),
PROJECT(SELECT(*cic*.branch: street = 'Av Champs Elysees');br#,brname,tel)}

Q5. Combine (Join) the branch information in the *Banks* multidatabase with the account information in the *sg* and *cic* databases.

MJOIN({*Banks*: x.branch},{*sg*,*cic*: y.account}: x.branch.br# = y.account.br#)

Note the use of the variables x and y as implicit database names. The result is a multirelation that consists of the following six relations:

{JOIN(*bnp*.branch,*sg*.account: *bnp*.branch.br# = *sg*.account.br#),
JOIN(*bnp*.branch,*cic*.account: *bnp*.branch.br# = *cic*.account.br#),
JOIN(*sg*.branch,*sg*.account: *sg*.branch.br# = *sg*.account.br#),
JOIN(*sg*.branch,*cic*.account: *sg*.branch.br# = *cic*.account.br#),
JOIN(*cic*.branch,*sg*.account: *cic*.branch.br# = *sg*.account.br#),
JOIN(*cic*.branch,*cic*.account: *cic*.branch.br# = *cic*.account.br#)}

Q6. Find the corresponding branch number, classification, account number, and opening date for all branches and accounts in the *Banks* multidatabase.

MPROJECT(MJOIN({*Banks*:x.branch},{*Banks*:y.account}:
x.branch.br#=y.account.br#);br#,class,acc#,open_date)

The result is a multirelation that consists of a single relation because only *sg*.branch contains class and only *cic*.account contains open_date:
{PROJECT(JOIN(*sg*.branch,*cic*.account: *sg*.branch.br# = *cic*.account.br#);
br#,class,acc#,open_date)}

Q7. In the *Banks* multidatabase, join the branch information with the account information, but only within a single database.

MJOIN({*Banks*: x.branch},{*Banks*: y.account}: x.branch.br# =
y.account.br# ∧ x = y)

The requirement that the join be performed only for relations within one database is expressed by equating the databases represented by x and y. The result is a multirelation with three relations (rather than the nine relations that would be obtained by the standard join):
{JOIN(*bnp*.branch,*bnp*.account: *bnp*.branch.br# = *bnp*.account.br#),
JOIN(*sg*.branch,*sg*.account: *sg*.branch.br# = *sg*.account.br#),
JOIN(*cic*.branch,*cic*.account: *cic*.branch.br# = *cic*.account.br#)}

Q8. Combine (Multiply) the branch information from the *bnp* and *sg* databases with the account information from the *etoile*, *opera*, and *bnp* databases.

MPRODUCT({*bnp*,*sg*: branch},{*etoile*,*opera*,*bnp*: account})

The result is a multirelation with 6 Cartesian products:
{PRODUCT(*bnp*.branch,*etoile*.account),
PRODUCT(*bnp*.branch,*opera*.account),
PRODUCT(*bnp*.branch,*bnp*.account),
PRODUCT(*sg*.branch,*etoile*.account),
PRODUCT(*sg*.branch,*opera*.account),
PRODUCT(*sg*.branch,*bnp*.account)}

Q9. Same as Q8 but combine branches and accounts only within a single database.

MPRODUCT({*bnp*,*sg*: x.branch},{*etoile*,*opera*,*bnp*: y.account}: x = y)

The result contains one relation:
{PRODUCT(*bnp*.branch,*bnp*.account)}

Q10. Find the union of the *etoile* and *nation* databases.

MUNION({*etoile*},{*nation*})

The account and client relations are not union-compatible, hence the result is a multirelation that consists of two relations:

$\{$UNION(*etoile*.account,*nation*.account),
UNION(*etoile*.client,*nation*.client)$\}$

Q11. Find the intersection of the *etoile* and *nation* databases.

MINTERSECT($\{$*etoile*$\}$, $\{$*nation*$\}$)

As in the answer to Q10, the result contains two relations, even if the intersections are empty, in which case the empty relations could be distinguished in some system-defined manner (such as 1.empty, 2.empty), thereby indicating the number of pairs of tables on which the operation was performed:

$\{$INTERSECT(*etoile*.account,*nation*.account),
INTERSECT(*etoile*.client,*nation*.client)$\}$

Q12. Find the difference of the *etoile* and *nation* databases.

MDIFFERENCE($\{$*etoile*$\}$, $\{$*nation*$\}$)

The result is the multirelation:

$\{$DIFFERENCE(*etoile*.account,*nation*.account),
DIFFERENCE(*etoile*.client,*nation*.client)$\}$

4. Multirelational Calculus

This section defines the expressions of the multirelational calculus. This calculus extends the relational calculus to the model of multirelations in analogy to the way that the multirelational algebra extends the relational algebra. In particular, the multirelational calculus reduces to the relational (tuple) calculus in case the multirelation consists of a single relation. The multirelational calculus is then used for writing three of the 12 queries written in the previous section in the multirelational algebra. At the end of this section we show that the multirelational calculus is at least as powerful as the multirelational algebra.

Expressions (queries) in the multirelational calculus are of the form $\{t|\psi(t)\}$ where t is a tuple variable and ψ is a formula (multirelational version) of first-order logic, defined below. One important deviation from the relational calculus is that a tuple variable in the multirelational calculus denotes a tuple of indeterminate, but bounded, length (i.e., the number of elements in the tuple). This needs to be done in the multirelational case because a multirelation may contain similar relations of different arity, yet a tuple variable must apply to all the relations.

First we define the atoms (atomic formulas) allowed in ψ. There are five types.

1. $\{R_1,...,R_n\}(u)$
   where $\{R_1,...,R_n\}$ is a multirelation and u is a tuple variable. Note how first-order logic is extended to cover multirelations. This atom stands for the assertion that u is a tuple of the multirelation $\{R_1,...,R_n\}$.

2. $u.A\ \theta\ v.B$
   where u and v are tuple variables, A is an attribute appropriate to u, B is an attribute appropriate to v, and θ is a comparison operator. This atom stands for the assertion that the A component of u stands in the relation θ to the B component of v.

3. $u.A\ \theta\ \alpha$ $(\alpha\ \theta\ u.A)$
   where u and θ are as in formula 2 and α is a constant. The first atom stands for the assertion that the A component of u stands in the relation θ to the constant α. The atom in parentheses stands for the assertion that the constant α stands in the relation θ to the A component u.

4. $s = u * v$
   where s, u, and v are tuple variables. This atom stands for the assertion that the tuple s is the concatenation of the tuples u and v. (If u and v have common attributes, the system must distinguish those attributes for s in some way, such as by an appropriate qualification.)

5. $t = u$ [attribute list]
   where t and u are tuple variables. This atom stands for the assertion that the tuple t is the projection of the tuple u to the attributes in the attribute list, if the attribute list contains (positive) attributes only. Negated attributes are also allowed. A negated attribute means the omission of that attribute.

Formulas are defined in the usual recursive manner, starting with the atoms by applying the connectives $\wedge, \vee, \neg, \rightarrow$, and the quantifiers $\forall, \exists$. It is assumed that the multirelational calculus expression $\psi(t)$ has exactly one free tuple variable, namely t. Note also that the tuple variables are qualified by attributes, not positions. Consequently, the order of the columns of a relation in a multirelation is not specified.

We illustrate the multirelational calculus by showing how to express three of the 12 queries written in the previous section in the multirelational algebra. For each query we show the equivalent set of relational calculus queries using subscripted tuple variables to indicate tuple length, i.e., $t(i)$ for a tuple t of length i.

Q1. $\{t \mid \exists u\ (\{\textit{Banks}\text{:client}\}(\text{u}) \wedge \text{t} = \text{u[cl\#,cltype,zip]})\}$

The answer contains the three relations that answer the following queries in the relational calculus:

$$\{t(3) \mid \exists u\ (\textit{bnp}.\text{client}(u) \wedge t.\text{cl\#} = u.\text{cl\#} \wedge t.\text{cltype} = u.\text{cltype} \wedge t.\text{zip} = u.\text{zip})\},$$
$$\{t(3) \mid \exists u\ (\textit{sg}.\text{client}(u) \wedge t.\text{cl\#} = u.\text{cl\#} \wedge t.\text{cltype} = u.\text{cltype} \wedge t.\text{zip} = u.\text{zip})\},$$
$$\{t(3) \mid \exists u\ (\textit{cic}.\text{client}(u) \wedge t.\text{cl\#} = u.\text{cl\#} \wedge t.\text{cltype} = u.\text{cltype} \wedge t.\text{zip} = u.\text{zip})\}$$

Q2. $\{t \mid \exists u\ (\{\textit{sg, cic,etoile,nation}\text{:account}\}(u) \wedge t = u[\text{acc\#,open\_date}])\}$

The resulting multirelation has three relations because open_date is inapplicable to *sg*.acc; the three relations answer the following relational calculus queries:

$\{t(2)\ |\exists u\ (cic.\text{account}(u) \land t.\text{acc\#} = u.\text{acc\#} \land t.\text{open\_date} = u.\text{open\_date})\}$,
$\{t(2)\ |\exists u\ (etoile.\text{account}(u) \land t.\text{acc\#} = u.\text{acc\#} \land t.\text{open\_date} = u.\text{open\_date})\}$,
$\{t(2)\ |\exists u\ (nation.\text{account}(u) \land t.\text{acc\#} = u.\text{acc\#} \land t.\text{open\_date} = u.\text{open\_date})\}$

Q7. $\{t\ |\exists u \exists v \exists w\ (\{Banks\text{: x.branch}\}(u) \land \{Banks\text{: x.account}\}(v) \land u.\text{br\#} = \text{v.br\#} \land \text{w} = u * v \land t = \text{w}[\neg v.\text{br\#}])\}$

The qualifying x for both multirelations indicates that u and v must be tuples in relations of the same database. The resulting multirelation consists of the answers to the following three queries:

$\{t(10)\ |\ \exists u \exists v\ (bnp.\text{branch}(u) \land bnp.\text{account}(v) \land u.\text{br\#} = v.\text{br\#} \land t.\text{br\#} = u.\text{br\#} \land t.\text{brname} = u.\text{brname} \land t.\text{street} = u.\text{street} \land t.\text{street\#} = u.\text{street\#} \land t.\text{city} = u.\text{city} \land t.\text{zip} = u.\text{zip} \land t.\text{tel} = u.\text{tel} \land t.\text{acc\#} = v.\text{acc\#} \land t.\text{cl\#} = v.\text{cl\#} \land t.\text{balance} = v.\text{balance})\}$

$\{t(11)\ |\ \exists u \exists v\ (sg.\text{branch}(u) \land sg.\text{account}(v) \land u.\text{br\#} = v.\text{br\#} \land t.\text{br\#} = u.\text{br\#} \land t.\text{brname} = u.\text{brname} \land t.\text{street} = u.\text{street} \land t.\text{street\#} = u.\text{street\#} \land t.\text{city} = u.\text{city} \land t.\text{zip} = u.\text{zip} \land t.\text{tel} = u.\text{tel} \land t.\text{class} = u.\text{class} \land t.\text{acc\#} = v.\text{acc\#} \land t.\text{cl\#} = v.\text{cl\#} \land t.\text{balance} = v.\text{balance})\}$,

$\{t(11)\ |\ \exists u \exists v\ (cic.\text{branch}(u) \land cic.\text{account}(v) \land u.\text{br\#} = v.\text{br\#} \land t.\text{br\#} = u.\text{br\#} \land t.\text{brname} = u.\text{brname} \land t.\text{street} = u.\text{street} \land t.\text{street\#} = u.\text{street\#} \land t.\text{city} = u.\text{city} \land t.\text{zip} = u.\text{zip} \land t.\text{tel} = u.\text{tel} \land t.\text{acc\#} = v.\text{acc\#} \land t.\text{cl\#} = v.\text{cl\#} \land t.\text{balance} = v.\text{balance} \land t.\text{open\_date} = v.\text{open\_date})\}$

Note that the length of variable t is not the same for all three queries. (The negated attribute is used to avoid repeating the common attribute br#.)

We end this section by showing that the multirelational calculus is at least as powerful as the multirelational algebra.

Theorem: Every multirelational algebra query can be expressed as a multirelational calculus query.

Proof: The proof is by induction on the number of occurrences of multirelational algebraic operators in the multirelational algebra query E.

Base Case: 0 operators. In this case, E is a constant multirelation, $E = \{R_1,...,R_n\}$. This is expressed in the multirelational calculus as $\{t\ |\ \{R_1,...,R_n\}(t)\}$.

Induction: Assume that E has n operators and is constructed from E_1 and E_2 where E_1 is expressed by $\{t\ |\psi_1(t)\}$ and E_2 is expressed by $\{t\ |\psi_2(t)\}$.

1. E = MPROJECT(E_1;A_1,...,A_n) is expressed by $\{t\ |\exists u\ \psi_1(u) \land t = u[A_1,...,A_n]\}$.

2. E = MSELECT(E_1: C) is expressed by $\{t\ |\psi_1(t) \land C'\}$ where C′ is obtained from C by replacing each attribute A_i with $t.A_i$.

3. E = MJOIN(E_1,E_2: C) where E_1 contains an x qualification, E_2 contains an x or y qualification, and C contains appropriate equalities (including qualifications) is expressed by $\{t \mid \exists u \exists v \exists w \, (\psi_1(u) \wedge \psi_2(v) \wedge w = u * v \wedge C' \wedge t = w[\neg v.A_1, \neg v.A_2, ..., \neg v.A_k]\}$ where C′ is obtained by changing the qualification of each attribute to u and v in an appropriate manner and the join attributes are $A_1,...,A_k$.

4. E = MPRODUCT(E_1,E_2) where E_1 and E_2 may contain an x qualification each is expressed by $\{t \mid \exists u \exists v \; \psi_1(u) \wedge \psi_2(v) \wedge t = u * v\}$ where ψ_1 and ψ_2 contain any x qualification from E_1 and E_2.

5. E = MUNION(E_1,E_2) is expressed by $\{t \mid \psi_1(t) \vee \psi_2(t)\}$.

6. E = MINTERSECT(E_1,E_2) is expressed by $\{t \mid \psi_1(t) \wedge \psi_2(t)\}$.

7. E = MDIFFERENCE(E_1,E_2) is expressed by $\{t \mid \psi_1(t) \wedge \neg \psi_2(t)\}$.

The converse of this theorem is an interesting research problem.

5. An Example Multirelational Language

Litwin et al. (1989) presented a multidatabase extension of SQL, termed MSQL (Multidatabase SQL) intended for multidatabase systems. Any function of SQL is by definition a function of MSQL. New functions are designed for nonprocedural manipulation of data in different and basically mutually non-integrated SQL databases. This means that the user's wish (informal multidatabase query) should become a single MSQL statement. The definition of SQL is mainly that of ISO (ISO/DIS, 1986) and of the DB2 dialect (Date, 1983). The new possibilities that MSQL statements provide are as follows:

- single-statement table creation or alteration in any number of databases;
- retrieval or modification involving the joining of data in different databases;
- broadcasting of a retrieval or of a modification over any number of databases where data with similar meanings have the same or different naming rules, decompositions into relations, or value types;
- dynamic transformation of actual attribute meanings, units of measure etc., into user-defined value types that may be retrieved or updated;
- interdatabase queries for data flow between databases;
- dynamic aggregation of data from different databases using various new standard (built-in, aggregate) functions;

- creation of multidatabase views and of virtual databases over such views;
- creation of auxiliary objects like triggers, stored queries and transactions (procedures).

Because MSQL manipulates sets of tables, the language is a natural extension of SQL to handle multirelations. For example, consider the database of Section 2. The query performed by the branch manager of the *opera* branch:

Find in all other branches the accounts of customers who are millionaires in my branch

is expressed in MSQL as:

```
LET x BE sg cic etoile nation
SELECT * FROM x.account
WHERE x.account.cl# = opera.account.cl#
AND opera.account.balance > 1,000,000
```

The query is a multirelational query, as it expresses in MSQL several relational queries. The tables in the result need not be union compatible.

The algebra and calculus presented in the two previous sections are hard to use in a real-life system. Looking at the *Banks* example, we can observe that real-life databases are less uniform than in this example. They are usually semantically heterogeneous with respect to names, structure, and value types even though all are relational databases. This is because they are autonomous as banks and branches compete for clients. Nevertheless, the banks and branches also cooperate, the cooperation being naturally stronger between branches of the same bank. That is why data in different databases have also some similarities which are stronger within the same bank. In most real-life bank databases the following are true:

1. Banks partly disagree upon attribute names that model the same concepts. For example, one database may use *client-name,* while another may use *cl-name* for the same attribute.
2. The same client may be represented in several databases.
3. Primary key values in databases of different banks are independent, despite sometimes having the same column names.

In order to handle such heterogeneity, MSQL provides some basic language constructs to (a) express queries ranging over sets of databases, and (b) address attributes by specifying a prefix and not a complete attribute name. For example, the following MSQL query:

```
USE etoile
SELECT * FROM client
```

allows the formation of a simple SQL query on the *etoile* database. Similarly,

```
    USE *Banks*
    SELECT *bnp*.brname
    FROM *bnp*.branch, branch
    WHERE *bnp*.branch.street# = branch.street#
    AND *bnp*.branch.br# != branch.br#
```

finds all *bnp* branches that are located on the same street with other banks' branches (and other *bnp* branches as well). Notice in the last query, the multi-dot notation that allows the isolation of specific relations within the *bnp* database. The following query

```
    SELECT *
    FROM br%
    WHERE street# = 'Av. Champs Elysees'
```

shows the use of the wild character % to abbreviate attribute names. This is especially useful if, say, one database stores branch information in a *branch* relation, while another database stores it in a relation called *brch*. This feature allows performing queries on databases that are heterogeneous to some extent. Its syntax is similar to that of the SQL "LIKE" clause, but it is applied to data names instead of values.

Finally, if one wants to restrict the set of databases taking part in the evaluation of a query, MSQL allows the definition of *semantic variables*. For example, the following query checks if a client of the *opera* branch has an account in the *etoile* or *nation* branch:

```
    LET x BE *etoile* nation
    SELECT * FROM x.client
    WHERE x.client.clname = *opera*.client.clname
```

The query runs only on two databases, excluding the use of *bnp*, *sg*, *sic*, and *opera*.

We end this section by showing Q7 from the previous section in MSQL.

```
Q7. USE *Banks*
    LET x y BE *Banks*.*
    SELECT br# class acc# open_date
    FROM x.branch y.account
    WHERE x.branch.br# = y.account.br# AND x = y
```

6. Query Optimization Using Properties of the Multirelational Algebra

In analogy to the properties of the relational algebra in Ullman (1982), we investigate the algebraic properties of the multirelational algebra and show how we can use them to process queries efficiently. Note that we consider a relation to be a set of tuples. Therefore, neither MJOIN nor MPRODUCT is commutative, because the order of element tuples is not the same.

Properties of Multirelational Algebra:

1. MPRODUCT is always associative, but MJOIN need not be associative. MPRODUCT can always be taken; hence it is associative. The problem with MJOIN is that a condition may be applicable or not, depending on the order in which the MJOINs are performed. For example, suppose that $R(A,B)$, $S(C,D)$, $T(E,F)$ are relations and consider
   MJOIN($\{$MJOIN($\{R\},\{S\}$: $A = F\}$,$\{T\}$: $C = E$)
   MJOIN($\{R\}$,$\{$MJOIN($\{S\},\{T\}$: $C = E\}$: $A = F\}$
   The first version does not provide an answer because the inside MJOIN is inapplicable; the second version provides an answer.

2. MSELECTs cascade, but some MPROJECTs do not cascade.
   Selection conditions can be done in any order because a condition that is not applicable in one order is not applicable in another. However, for projection, consider $R(A,B,C)$, now MPROJECT($\{R\}$;A,B) exists, but MPROJECT($\{$MPROJECT($\{R\}$;A,B,D)$\}$;A,B) does not, even though all the attributes in the outside projection are contained in the inside projection.

3. If all the attributes in the MSELECT condition are in MPROJECT, then MSELECT and MPROJECT commute.
   MPROJECT($\{$MSELECT($\{R_1,...,R_n\}$:C)$\}$;$A_1,...,A_n$) =
   MSELECT($\{$MPROJECT($\{R_1,...,R_n\}$:$A_1,...,A_n\}$:C)
   if condition C contains at most the attributes $A_1,...,A_n$. For each R_i, if MPROJECT followed by MSELECT exists, then so does MSELECT followed by MPROJECT, and vice versa. Without that condition, it may be the case that a relation for the left-hand side does not appear in the right-hand side.

4. If all the attributes in the MSELECT condition are in every relation of the first multirelation, then MSELECT and MPRODUCT commute.
   MSELECT($\{$MPRODUCT($\{R_{11},...,R_{1n}\}$, $\{R_{21},...,R_{2m}\}$)$\}$:C) =
   MPRODUCT($\{$MSELECT($\{R_{11},...,R_{1n}\}$: C)$\}$, $\{R_{21},...,R_{2m}\}$)
   if the attributes in C appear in all of $R_{11},...,R_{1n}$. Now both the left-hand side and the right-hand side have $n \times m$ relations. Without this proviso, the right-hand side may have fewer relations than the left-hand side.

5. MSELECT commutes with the set operations MUNION, MINTERSECT, and MDIFFERENCE. This result holds because the set operations require union-compatible relations.

6. With the MPROJECT attributes appropriately chosen (see below) for the multirelations, MPROJECT and MPRODUCT commute. That is
   MPROJECT($\{$MPRODUCT($\{R_{11},...,R_{1n}\}$, $\{R_{21},...,R_{2m}\}$)$\}$; $A_1,...,A_n$) =
   MPRODUCT($\{$MPROJECT($\{R_{11},...,R_{1n}\}$;$B_1,...,B_m\}$,
   $\{$MPROJECT($\{R_{21},...,R_{2m}\}$;$C_1,...,C_k\}$)

if $\{A_1,...,A_n\} = \{B_1,...,B_m, C_1,...,C_k\}$ and $B_1,...,B_m$ appear in all of $R_{11},...,R_{1n}$, and $C_1,...,C_k$ appear in all of $R_{21},...,R_{2m}$. Now both the left-hand side and the right-hand side have $n \times m$ relations. Without the proviso, the right-hand side and the left-hand side may have a different number of relations.

Given the above, the multirelational algebra should be useful for the optimization of multirelational queries. Below, we illustrate the use of the multirelational algebra through a straightforward example using MSQL.

As mentioned earlier, multirelational queries are at a higher level than relational ones in the sense of being less procedural. To illustrate this point, and one application of the multirelational algebra, consider the following example. A bank has the database *Archives* and a number of autonomous databases, one per branch, constituting a multidatabase called *Branches,* which is similar to the database that was defined in Section 2. The difference is that each branch database has a relation, clientinfo(cl#,clname,balance,acc#,open_date), that combines the attributes of the "client" and "account" relations we have been using in our examples. The *Archives* database has a relation, cust-89(cl#,clname,balance), that sums up information about the banking of each customer in 1989. Suppose that a manager wishes to know the names and balances of all the customers that opened an account on 1/1/89. The MSQL query would be as follows:

```
(M1) USE Archives Branches
     SELECT X.clname Y.balance
     FROM Archives.cust-89 X, clientinfo Y
     WHERE Y.open_date = '1/1/89' AND X.cl# = Y.cl#
```

In query (M1), clientinfo is a multiple identifier of any relation in any database in *Branches.* The semantics of (M1) is the multirelation defined by all the elementary queries resulting from the substitutions of clientinfo by the name of a relation in a database in *Branches.* For example, suppose that query (M1) is translated to the following set of queries (a transaction)

```
(M2) BEGIN TRANSACTION

     SELECT X.clname Y.balance
     FROM Archives.cust-89 X, etoile.clientinfo Y
     WHERE Y.open_date = '1/1/89' AND X.cl# = Y.cl#

     SELECT X.clname Y.balance
     FROM Archives.cust-89 X, nation.clientinfo Y
     WHERE Y.open_date = '1/1/89' AND X.cl# = Y.cl#

     SELECT X.clname Y.balance
     FROM Archives.cust-89 X, opera.clientinfo Y
     WHERE Y.open_date = '1/1/89' AND X.cl# = Y.cl#

     END TRANSACTION
```

These queries would then be translated into the relational algebra for an optimized execution. To observe the limits of this type of processing, suppose that there are 100 branches. If query (M1) is processed as the multiple query (M2), then the system would execute the same selection 100 times over *Archives*. To make the processing of (M2) more efficient, the system has to discover that there is a common expression in this multiple query that can be factored out. This is based on Property 4 (above) extended to MJOINs, which allows selections to be pulled out. For example, consider that query (M1) is translated first into its multirelational form:

```
(M3) MPROJECT(
        MJOIN({Branches:clientinfo},
        MSELECT({Archives.cust-89} :
          open_date='1/1/89') : cl#=cl#) ; clname, clientinfo.balance)
```

The next natural step for the query processor is to examine the cardinalities of the multirelations involved. It will find that MSELECT operates over a single relation. The easy optimization rule is then to factor out the selection clause:

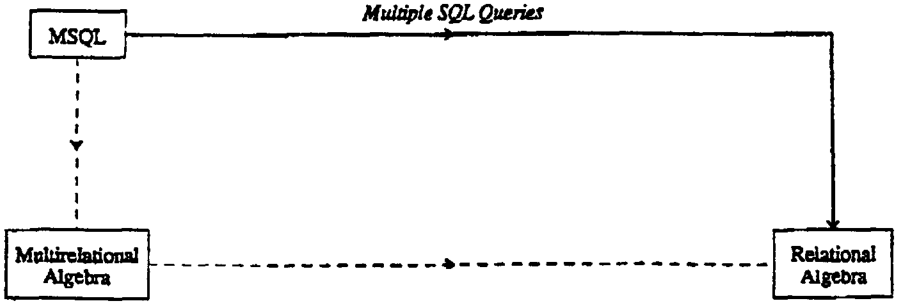
```
(M4) W:= MSELECT({Archives.cust-89} : open_date = '1/1/89')
        MPROJECT(MJOIN({Branches:clientinfo}, W : cl#=cl#) ; clname).
```

Query (M4) can then be transformed easily to the following transaction:

```
(M5) BEGIN TRANSACTION

        USE Archives
        INSERT W

        SELECT *
        FROM cust-89 X
        WHERE X.open_date = '1/1/89'

        SELECT clname Y.balance
        FROM W, etoile.clientinfo Y
        WHERE Y.open_date = '1/1/89' AND W.cl# = Y.cl#

        SELECT W.clname Y.balance
        FROM W, nation.clientinfo Y
        WHERE Y.open_date = '1/1/89' AND W.cl# = Y.cl#

        SELECT W.clname Y.balance
        FROM W, opera.clientinfo Y
        WHERE Y.open_date = '1/1/89' AND W.cl# = Y.cl#

        END TRANSACTION
```

This decomposition is more efficient than (M2), because it avoids 99 useless repetitions of the selection. Using straightforward extensions of the traditional optimization

Figure 1. Alternative query transformations and executions



rules, one may also define decompositions where projections are factored out before join operations are performed (Rule 6 in the beginning of the section). In conclusion, because the multirelational algebraic expressions are at a higher level than the relational ones, the lower path in the diagram of Figure 1 should be highly preferable to the upper one. This is due to the fact that multirelational queries need to be translated to a form that includes basic algebra or calculus operations where transformation rules are to be applied. Applying such rules on SQL statements (upper path) is much more difficult than applying them on (multirelational) algebra statements.

7. Conclusions

In this article, the relational algebra and calculus were extended to a multirelational algebra and calculus, where multitables are semantically heterogeneous. The multirelational algebra will be needed in the multidatabase environment that will become prevalent in the next few years. Our multirelational operations are defined based on the semantics of the multirelational model. They are compatible with the relational ones, but some of the properties of the relational operations do not hold for the multitables. There does not seem to be any practical multirelational algebra that could preserve them. We showed, however, how some properties of the algebra can be used in optimizing MSQL queries.

One further step is to analyze the extended model. For example, the properties of the multirelational algebra can be studied where nonexistent attributes for a projection are ignored. Or, we may allow the user to define through a function the pairs of relations that are to be examined in cases of the MJOIN and MPRODUCT operations. Generally, one can devise different semantics for the multirelational operations we have suggested, and studying such alternative semantics is an interesting topic. Finally, another interesting future direction would be to define optimization rules extending standard optimization techniques used in relational database systems to the case of multidatabases.

Acknowledgments

This research was sponsored by the National Science Foundation under Grant IRI-8921951, the University of Maryland Institute for Advanced Computer Studies (UMIACS), the Air Force Office for Scientific Research under Grant AFOSR-89-0303, and NASA under Grant NAS5-31351.

References

ACM Computing Surveys Special Issue. Heterogeneous Databases, Elmargamid, A. and Pu, C., eds., *ACM Computing Surveys,* (22)3, 1990.

Ceri, S. and Pelagatti, G. *Distributed Databases: Principles and Systems,* New York: McGraw Hill, 1984.

Date, C.J. *A Guide to DB2,* Reading, MA: Addison-Wesley, 1983.

Information Processing Systems: Database Language SQL, *Draft International Standard,* ISO/DIS 9075, 1986.

Litwin, W. and Abdellatif, A. Multidatabase Interoperability, *IEEE Computer Journal,* (19)12:10-18, 1986.

Litwin, W., et al. SIRIUS Systems for Distributed Data Management. In: Schneider, H.J., ed., *Distributed Databases,* New York: North-Holland, 1982.

Litwin, W., Abdellatif, A., Zeroual, A., Nicolas, B., and Vigier, Ph. MSQL: A multidatabase language, *Information Sciences,* (49), 1989.

Roussopoulos, N. and Kang, H. Principles and techniques in the design of ADMS, *IEEE Computer Journal,* (19)12:19-25, 1986.

Sellis, T. Multiple-query optimization, *ACM Transactions on Database Systems,* (13)1: 23-52, 1988.

Stonebraker, M. and Rowe, L. The design of POSTGRES, *Proceedings of the 1986 ACM-SIGMOD Conference,* Washington, DC, 1986.

Ullman, J.D. *Principles of Database Systems,* Second Edition, Rockville, MD: Computer Science Press, 1982.