Relational algebra reconstruction toward a relational engine

Ján Šturc

Computer Science Department of the Comenius University
Mlynská Dolina, 841 15 Bratislava

25.5.2000

Abstract

In this paper we discus a modification relational algebra introduced to database theory by E.F. Codd thirty years ago. The leading idea is extend the relational algebra to be complete for handling and optimization at least on the level of SQL engines. We discus possible extensions. The leading idea for the extensions is to preserve the liaison between algebra and logic. We discus an object oriented approach to an implementation of this extension.

1 Introduction

Despite the relatively long history of the relation algebra is treated as it was introduced by Codd [1] and conserved by C. H. Date, J. D. Ullman and D. Maier in the first textbooks. This is rather surprising because there is much older relation algebra developed by A. Tarski [13] in 1941, which is closely related to modeling of the first order predicate calcul. Tarski with Henkin and Monk [9] developed cylindric algebras which are suitable for modeling of the first order predicate calcul in general. From point of view of database theory the cylindric algebras seems not to be practical because they dwell on tables which set of attributes is infinite – the set of all variables.

In the recent time appeared few papers which try to generalize the relational algebra to deal with uncertain, probabilistic or fuzzy values. Although such generalizations seems to be easy, simply to add some columns, which carry the additional information. Practically, nothing else can be done. From the mathematical point of view a clean interpretation such modification may be quite complex. The problem is that the relational operations are closely liaised to logic. Generalizations of the logic need not preserve the algebraic interpretation. E.G.: Many valued (finite) logic is a Boolean algebra in the case the number of values is a power of two. Fuzzy and probabilistic logic are even more sophisticated. Really, in a probabilistic logic we need additional information (conditional probability) to compute an intersection of two sets.

The adherents of the object oriented databases talk about decline of relational model. We feel, relational and object oriented are different levels. While relational is really database model, not only database but universal approach to mathematical modeling for predicate calculus, the object oriented is approach to programming system implementation. We will try present an object oriented implementation of the relational model.

Before we start with generalizations, we consider it would be useful to put some effort into its reconstruction. One may wonder, why after thirty years of relational model, when there are several books on this topic, a reconstruction is needed. However, the relational algebra as it is presented in tutorials does not agree with the operations really used in the implementations of relational database systems. In particular relations and tables differ in using variables. While in practical use attribute renaming and aliasing is an often operation in theory it is neglected, despite the fact it corresponds to using variables in formulae and renaming corresponds to substitution.

The relational algebra is based on sets and first order predicate calculus. If neglect peculiarities the relational operations corresponds to logic connectives and (Cartesian product, join, intersection), nand (not and: difference, antijoin), and to existential quantifier (projection).

In real databases coexist the sets and multisets
(SELECT vers. SELECT DISTINCT and COUNT(x), SUM(x), AVG(x) vers. COUNT(DISTINCT x), SUM(DISTINCT x), AVG(DISTINCT x)). The classic database theory ignores aggregation functions, because they have no natural first order description and the multisets are considered as a feature to simplify implementation.

So far one may consider the relational algebra reconstruction as a play for pure mathematicians with little practical merit. But the distinction between finite and infinite tables between join and selection makes many optimizations more cumbersome than necessary, if does not prevent them at all.

Example 1 Let $R(x, y)$ and $S(z)$ be two tables. Consider the query
SELECT x, y, z FROM R, S WHERE $x + y = z$.
There is only one translation into classic relational algebra $\sigma_{x+y=Z}R(x, y) \times S(z)$. In our algebra with infinite table $add(x, y, z)$, where a triple $< x, y, z >$ in $add(x, y, x)$ iff $x + y = z$ the expression $R(x, y) \bowtie add(x, y, z) \bowtie S(z)$ is allowed, too. Evaluation of the second expression may be for some instances of the relations R and S considerably more efficient.

2 Concrete sets

2.1 Motivation

The core of the modifications of the relational algebra consist in concept of the set. We generalize the set concept so that the value of "membership" (characteristic function) may acquire also other values than true and false. On other hand the restriction on recursive characteristic functions narrows the concrete sets to subsets of an enumerable infinite set – universe of discourse. To be honest, we advocate to use the partial characteristic functions instead of sets. The generalization in this case becomes the change of range this function.

The implementation of a concrete set in a database is easy, we simply store it as a set of pairs $\langle element,\ value\ of\ the\ characteristic\ function \rangle$. This is a regular relation.

Definition 1 A concrete set S is a triple $S = \langle D, R, \chi_S \rangle$, where D is an ordinary set – the universe of discourse. R is also an ordinary set – the range of membership function and $\chi_S : D \to R$ called a characteristic function of the set S.

In addition to usual definition of an unary relation here the characteristic function is included into the definition. This allows the generalization to multisets or fuzzy sets.

Example 2 A concrete set A of real numbers. $A = \langle \Re, Boolean, \chi_A \rangle$, where $\chi_A(r) = true$ if only if $r \in A$ corresponds to a set A of real numbers.

At the first look we gain nothing except the fact, we can say that $insert(r)\ into(A)$ means $\chi_A(r) := true$ and $delete(r)\ from(A)$ means $\chi_A(r) := false$. But we get much more. We can speak about the set $A' = \langle \Re, \mathbf{N}, \chi' \rangle$, where $insert(r)\, into(A)$ means $\chi'_A(r) := \chi'_A(r) + 1$ and $delete(r)\, from(A)$ means $\chi'_A(r) := 0$. The set A and A' are indistinguishable as ordinary sets but are clearly different as concrete sets. However, the particular implementation of the function χ'_A remains still hidden. It may be implemented by a table or by count of occurrences in a list of elements, as well.

In generally we do no assumptions about range R of the function χ. It may be Boolean, natural, reals in scope $< 0, 1 >, \ldots$. May be it is worth to assume that R forms a complete lattice or a field. Some structure of R is necessary to enable us to define operations in term of operations on function χ.

The very common is to assume R to be a complete lattice $R = \langle L, \sqsubseteq, \sqcup, \sqcap, \top, \bot \rangle$, where $\sqcup$ is supremum (l.u.b.)and $\sqcap$ is infimum (g.l.b.) and to define for each concrete set $S = \langle D, R, \chi_S \rangle$ the carrier $car(S) = \{x : x \in D\ and\ \chi_S \sqsupset \bot\}$. The set of elements which surely in a concrete set S, we call kernel of S. It is defined as $ker(S) = \{x : x \in D\ and\ \chi_S = \top\}$.

Both the carrier and kernel may be considered as ordinary sets or as concrete sets as well. To describe them as concrete sets via manipulation with characteristic functions we generalize the sign function on a lattice.

Let L be a lattice and $x \in L$ then $sg(x) =$ **if** $(x \sqsupset \bot)$ **then** 1 **else** 0.
We introduce also the function $\overline{sg}(x) =$ **if** $(x = \top)$ **then** 1 **else** 0.

In terms of these functions $S = \langle D, R, \chi_S \rangle$ implies $car(S) = \langle D, GF(2), sg(\chi_S) \rangle$ and $ker(S) = \langle D, GF(2), \overline{sg}(\chi_S) \rangle$.

Note: The natural numbers $\mathbf{N}$ does not form a complete lattice, the top element does not exist. We define, the kernel of a multiset to be an empty set.

2.2 Disjunction and intersection

The semantics of objects is given trough operations. The set theoretic operations are natural for the ordinary sets but not for the concrete sets. We define these operations in terms of characteristic functions.

A concrete set $\emptyset$ is empty if its characteristic function $\chi_\emptyset \equiv \bot$.
Consequence $car(\emptyset) = \emptyset$.

disjunction:	$C = A \cup B$	if and only if	$\chi_C = \chi_A \sqcup \chi_B$
intersection:	$C = A \cap B$	if and only if	$\chi_C = \chi_A \sqcap \chi_B$

The definition above is satisfactory for ordinary sets, fuzzy sets and multisets but not for probabilistic sets. The reason is that the probability composition is govern by a bit different law – Bayesian formula. In this case $R = \langle < 0, 1 >, +, \times \rangle$ and

disjunction:	$C = A \cup B$	if and only if	$\chi_C = \chi_A + \chi_B - \chi_{A \cap B}$
intersection:	$C = A \cap B$	if and only if	$\chi_C = \chi_A \times p(B\|A)$,
	where $p(B\|A)$ is conditional probability of B assuming A. $p(B\|A) = \chi_B$ if and only if A and B are independent.		

From the last formulae we can conclude that the probabilistic approach requires additional knowledge of conditional probabilities of all possible intersections. We feel, this main reason why the fuzzy approach is preferred to probabilistic one.

2.3 Complementation and difference

The concept of membership in the set theory is rather general to be generalized. We see it on complement operation. It is easy to write $x \notin M$ but what it implies for the characteristic function. If R is Boolean it may mean negation but what is the carrier. For a concrete set the complement is clear because both the original set and its complement have the same carrier (domain of the characteristic function) so the complement is taken with respect to the universe of discourse. However, if the range of characteristic function is not an (Abelian) group with respect to addition it is not clear what the complement means.

An idea is to define set difference $C = A - B$ if and only if $\chi_C = \chi_A - \chi_B$.

In this case we have to accept negative membership what have no reasonable interpretation. (In some applications it may be realistic to say there scarce occurrences of an element x in a set M.) We prefer to define positive difference

$$a \dot{-} b = \begin{cases} a-b & \text{if } a > b \\ 0 & \text{otherwise} \end{cases}$$

set difference: $C = A \cup B$ if and only if $\chi_C = \chi_A \dot{-} \chi_B$

Because the universe of discourse is an ordinary set, it is isomorphic to the concrete set $D = \langle D, R, \chi_D \rangle$, where $\chi_D(x) = 1$ for all $x \in D$. In case the range R of the function χ is Boolean, we use the notation of the Galois field GF(2). The complement of a set A is the set $\overline{A} = D - A$ or

complement: $C = \overline{A}$ if and only if $\chi_C = 1 \dot{-} \chi_A$

Note, that despite the definition look very naturally, there are very strange concrete sets. E.g. there are concrete sets which are not disjoint with their complement or there is a concrete set equals to its complement.

One need not be satisfied with these strange properties and may try to define "complement" as a *"maximal"* disjoint set. In case of multiset with unrestricted occurrence such a definition is not correct. We define a *strong complement* of a set as the complement of its carrier and a *week complement* as the complement of its kernel. In terms of characteristic functions

strong complement: $C = \hat{A}$ if and only if $\chi_C = 1 - sg(\chi_A)$
week complement: $C = \underline{A}$ if and only if $\chi_C = 1 - \overline{sg}(\chi_A)$

Lemma 1 For a concrete set S it holds: $\underline{S} \supseteq \overline{S} \supseteq \hat{S}$. If S is an ordinary set then $\underline{S} = \overline{S} = \hat{S}$. Moreover $car(S) = \overline{\hat{S}}$ and $ker(S) = \overline{\underline{S}}$.

Lemma 2 Let $S = \langle D, R, \chi_S \rangle$ then $S \cup \hat{S} = D$ and $S \cap \hat{S} = \emptyset$.

The proof of these statements is obvious.

Note that all complements are isomorphic to ordinary sets so the second complementation are all equivalent in such case we prefer the complement notation.

According to lemma 2 one may prefer call a complement the strong one only. This is unnatural in term of characteristic function, which mirrors the logic used and the *tertium non datur* is generally not true in the other than binary logics.

2.4 Cardinality and count

Speaking about cardinality[1] of an ordinary set, it may mean, the number of elements, the sum of ones in the characteristic function. In this case we need assume that the range of characteristic can be embedded to a commutative ring.

Definition 2 Let $S = \langle D, R, \chi_S \rangle$ then $\#S = \sum_{x \in D} \chi_S(x)$. We shall call this sum count of S and denote it *count*(S), too.

[1] To be honest, we must say that definitions and arguments of this section works only in a countable universe of discourse D.

At another hand if one cares about number of different elements in a multiset or about the maximal number of different elements in a probabilistic or fuzzy set then the *cardinality* of the $car(S)$ is interesting. We shall call this number the *norm* of the set S and denote it by $|S| = \sum_{x \in D} sg(\chi_S(x))$. Sometimes only the number of "sure" elements of a set is interesting, we shall call it *inner norm* and denote by $||S|| = \sum_{x \in D} \overline{sg}(\chi_S(x))$.

The last two definitions are superfluous because the following holds $|S| = \#car(S)$ and $||S|| = \#ker(S)$.

2.5 Functions in numeric sets

Assume now that $A = \langle D, R, \chi_A \rangle$ is a *numeric set* that means D is set of numbers then one may be interesting in $sumA$ denoted by $\sum A$.
It holds $\sum A = \sum_{a \in D} a \times \chi_A(a)$.

Similarly to *count* the *sum* has its "set projection" $\sum A = \sum_{a \in D} a \times sg(\chi_A(a))$, too.

The average of a set S is defined simply as the quotient $avg(A) = sum(A)/count(A)$.

2.6 Extension and restriction

Definition 3 Let $S = \langle D, R, \chi_S \rangle$ and $car(S) \subseteq D_1 \subset D \subset D_2$.
Then the set $P = \langle D_1, R, \chi_P \rangle$ such that $\chi_P(x) = \chi_S(x)$ for all $x \in D_1$ is called a *restriction* of set S on the domain D_1.
And, $Q = \langle D_2, R, \chi_Q \rangle$ such that $\chi_Q(x) =$ **if** $x \in D$ **then** $\chi_S(x)$ **else** 0; is called an *extension* of the set S to the domain D_2.

Now the problem of consecutiveness of the set concept is clearly visible. An ordinary set describes *the class* of all possible restrictions and extensions of a concrete set.

2.7 Relations and Cartesian products

Definition 4 A concrete relation more precisely n-ary relation R is a quadruple $R = \langle n, D, L, \chi_R \rangle$, where D is an ordinary set – the universe of discourse. L is also an ordinary set – the range of membership function and $\chi_R : D^n \to R$ is the characteristic function. Where D^n means Cartesian power of D.

In spite the fact the Cartesian product is considered to be an elementary concept in the set theory, it is not:

1. There are two concepts of ordered pairs (couples), Kuratowski's pairs and functions from $\{0, 1\}$ to a set. Clearly, $\{a, \{a, b\}\} \neq \{\langle 0, a \rangle, \langle 1, b \rangle\}$. An isomorphism have to be constructed.
2. Strictly speaking, the Cartesian product of ordinary sets is not associative ($\langle \langle a, b \rangle, c \rangle \neq \langle a, \langle b, c \rangle \rangle$). So, the Cartesian power is not well defined. In turn an isomorphism have to be employed.

3. Because $\langle S, S\rangle \neq S$. The set and unary relation are not the same for ordinary set and relations.

The mathematicians knows these difficulties but they can live with them and do not feel the need to put effort into cleaning the fundamental concepts [6].

Concerning the ordinary set we accept the mathematical approach and define: $S^0 = \{\emptyset\} = true$ and $S^1 = S$ for arbitrary ordinary set.

According the definition above the concrete unary relation is the same as the concrete set. No additional assumption or isomorphism is needed except the arity forgetting morphism, which makes a set from a relation.

2.8 Cartesian product of the relations

Definition 5 Let $R = \langle m, D_1, L, \chi_R\rangle$ and $S = \langle n, D_2, L, \chi_S\rangle$ are two relations. Then the Cartesian product $T = R \times S$ is a relation $T = \langle m+n, D_1 \cup D_2, L, \chi_T\rangle$, where $\chi_T(ab) = \textbf{if } (a \notin D_1) \text{ or } (b \notin D_2) \textbf{ then } 0 \textbf{ else } \chi_R(a) \cdot \chi_S(b)$.

In the great majority of cases we can live with only one universe of discourse. In that case $\chi_T(ab) = \chi_R(a) \cdot \chi_S(b)$.

Note: Many sorted (different domains for attributes) seems to be from practical point of view more natural but from theoretical point of view it is inessential variation and it only complicates notation.

3 Tables

Basic concept of the database theory is table. Informally, a table consists of a heading and relation. Some mismatch into database theory has been brought by Codd [1] in his seminal paper, he did make difference between concepts table and relation. Surprisingly, the later papers and books accepted and disseminated this dichotomy.

Definition 6 A table T is a triple $T = \langle n, h, R\rangle$, h is a function (bijection[2]) from the set $\overline{n} = \{0, \ldots, n-1\}$ onto a finite set[3] of names of the columns, and R is an n-ary relation. The column names are called attributes, too.

Note: This definition is valid in classic (based on ordinary sets) database theory, too.

According to a naive interpretation the attributes (column names) are properties of objects. The objects (entities) itself are not recorded in the relation they implicitly corresponds to the rows of the table. In each row column crossing the value of the attribute for the object corresponding to the row is recorded.

From logical point of view, we can consider the column headings as the name of variables and the table as the set of all possible valuations of these variables which

[2]The assumption that h is bijection is only needed to say, T is an n-column table. We drop it later.
[3]ordinary set

takes values from the relation R. The relation in the logic is an interpretation of a predicate symbol. Of course the rows are the images of the all "elements" of R.

Example 3 The table $T = \langle 2, \{\langle 0, x\rangle, \langle 1, y\rangle\}, R\rangle$ corresponds to an atomic formula $R(x, y)$. The table $T = \{\langle x, y\rangle : \langle x, y\rangle \in R\}$ [4].

No of these views agree with concept of an attribute as a function which assign a value of the attribute to the objects, used in philosophy. This led Materna to the HIT model [7].

4 Algebra of the tables

The relational algebra uses quite a lot operations (set operations: union, difference; relational operations: projection, selection, cartesian product; joins: these operation can be expressed as a cartesian product followed by a selection and eventually a projection, therefore they are not independent. Despite the relative simplicity we cope with formal model of relational algebra since long time [12].

4.1 Operations and their liaison to the logic

We start with a rather technical operation *a renaming.*

Definition 7 Let $T = \langle n, h, R\rangle$ and u be a bijection of an n element set of variables to another n element set of variables not necessary disjoint, such that the range of u is the domain of h. Then the table $U = \langle n, u \circ h, R\rangle$ is called renaming of the table T on the attributes in the domain of u.

This is the operation, which is neglected in the majority of database publications e.g. [8, 14]. This neglection makes the relation algebra ambiguous and rather vague. The renaming has no meaning (It is algebraic counterpart of the renaming of variables in formulae.) but it makes other operations (specifically unions, differences and joins) to work.

The union operation is applicable on tables with equal number of column and headings so that rows of these tables are united. Formally, let $T_1 = \langle n, h, R\rangle$ and $T_2 = \langle m, h, S\rangle$. Then $T_1 \cup T_2 = \langle n, h, R \cup S\rangle$. If T_1 is a valuation of the formula $R(h(0), \ldots, h(n-1))$ and similarly T_2 evaluates the formula $S(h(0), \ldots, h(n-1))$. Then $T_1 \cup T_2$ evaluates the formula $R(h(0), \ldots, h(n-1)) \vee S(h(0), \ldots, h(n-1))$.

The difference operation requires the heading of the substrahend to be subset of heading of the minuend. Let $m < n$ and $g(\overline{m}) \subseteq h(\overline{n})$. Then $T_1 - T_2$ are those rows from T_1 which do not contain the "m - subrow" equal to some row of the table T_2. Formally, $T_1 - T_2 = \langle n, h, \sigma_{\neg S} R\rangle$. In the logic to the difference corresponds formula $R(h(0), \ldots, h(n-1)) \wedge \neg S(g(0), \ldots, g(m-1))$.

[4]We use the same symbol for a predicate and a relation interpreting this predicate.

The two operations above are counterparts of the set operations modified for the tables. Below we introduce two operations which correspond to relational operations.

Let as in the previous case $m < n$ and $g(\overline{m}) \subseteq h(\overline{n})$. The operation which select m-column table from the n-column one by column removing is called *projection.* The projection $\Pi_{h(\overline{m})}T$, where $T = \langle n, h, R\rangle$, is $T' = \langle n, h \circ g^{-1} \circ h, R\rangle$. In the logic this is expressed by the formula $\exists(h(\overline{n}) - g(\overline{m}))R(h(0), \dots, h(n-1))$.

Note: In many cases the projection is manipulation with the heading, and the real projected relation need not be produced. But, if needed the relation $P = \Pi_{g(0)\dots g(m-1)}R$ is the required relation.

The last operation, we introduce, is *join* (natural join). Let $T_1 = \langle m, f, R\rangle$ and $T_2 = \langle n, g, S\rangle$. The set of variables $f(\overline{m})$ and $\overline{n})$ need not be disjoint. If two values are mapped in the same variable, it is possible. only if these values are equal. In turn the real join need not be constructed. The join $T_1 \bowtie T_2$ is a table $T = \langle p, h, Q\rangle$, where $p = |f(\overline{m}) \cup g(\overline{n})|$,; the $h : p \to f(\overline{m}) \cup g(\overline{n})$ is such that $i < m$ implies $h(i) = f(i)$ and $h(i) = g(i)$ for $p - n < i < p$. And , of course, $Q = R \bowtie S$.

The formula of the logic corresponding to the join is
$Q(h(0), \dots, h(p-1)) = T(f(0), \dots, f(m-1)) \wedge Q(g(0), \dots, g(n-1))$.

Note: The join operation corresponds to Cartesian product if the attribute sets are disjoint and the intersection if the attribute sets are the same for both tables.

Because we have operations corresponding to the full set of logical operations: Relative complement (This enough because the universe of discourse is recursive.), and (join) and existential quantifier (projection). Moreover the renaming operation can simulate variable renaming (substitution).

5 Keys an pseudokeys

There are essential differences between relational algebra and our algebra of tables. In the algebra of tables there is no assumptions about finiteness of tables. Tables are assumed to be concrete sets of the rows, therefore they are recursive. In the relational algebra the relation are classified into finite database relations and infinite built-in relations. The built-in relations can only be used in the intersection with a finite relation and this operation is called selection.

We adopt a more general approach – the concept of an pseudokey, first mentioned by Hall et al. [3] as an input set.

Definition 8 Let $T = \langle n, h, R\rangle$ be a table. Then a subset I of $h(\overline{n})$ is called *pseudokey for the table* T if for arbitrarily chosen values of these attributes the number of rows in T with chosen values of these attributes *is finite* and, no proper subset of I preserves that property.

We remember:

Definition 9 Let $T = \langle n, h, R\rangle$ be a table. Then a subset I of $h(\overline{n})$ is called *key for the table* T if for arbitrarily chosen values of these attributes there is *at most* *one* row in T with chosen values of these attributes and, no proper subset of I preserves that property.

Example 4 For a database table the input set is the empty set. For the table $ADD(x, y, z)$ defined by the predicate $z = x + y$ on the set of natural numbers pseudokeys are $\{x, y\}$ and $\{z\}$.

By the definition the key is a pseudokey. Converse is not valid. The pseudokey $\{x, y\}$ is a key, too but $\{z\}$ is not.

Super sets of keys and pseudokeys are called super keys resp. super pseudokeys.

The pseudokey is a natural generalization of the concept of key. It has not too much meaning for ordinary database relations, because empty set is a pseudokey, and any set of attributes is a super pseudokey.

From the definition follows statement

Lemma 3 *Each table in regular sets has at least one key.*

Proof: Table is a set of rows. Therefore in a worst case the set of the all attributes is a key.

This need not to be valid in the extensions. We can imagine a strange table in multisets which has no key. Despite of this we can behave as the previous lemma were generally valid because the strange relations are implemented as their characteristic functions in regular set theory.

Note: In an implementation each input set must be supported by an corresponding algorithm computing rest of arguments. If such an algorithm is difficult (not known or not existing) one may decide support only some supersets of given input set.

Example 5 We want not support the partition of a number so support for the table ADD(x,y, z) only sets $\{x, y\}$,$\{x, z\}$ and $\{y, z\}$ by the algorithms for addition and subtraction.

One can include as many arithmetic function as needed. The only is needed to program necessary function and to encapsulate it in a table container.

Definition 10 The super pseudokeys supported in the real databases are called indices.

The tables are objects which provides an access method for each index. The usual database finite tables support at least one key called primary key and at least one index for the pseudokey empty set.

6 Equation solving

An algebraic expression in the algebra of tables define a new table via computation in the algebra. The simplest or **solved form** of an equation is equation

$R = \mathbf{E}(S_1, \dots, S_k)$, where S_i are known tables and $\mathbf{E}(S_1, \dots, S_k)$ is an expression in algebra of tables. The solution R can be compute by step by step computation of the algebraic operation. To simplify the computation one can use the algebraic laws[5] associativity a commutativity etc.

Having system of equations in the form[6] $R_i = \mathbf{E}_i(S_{i_1}, \dots, S_{i_k})$, for $1 \leq i \leq m$. We can draw a **dependency graph** which nodes are tables, and there is an arc from S_{i_l} to R_i iff $S_{i_l} \in \mathbf{E}_i$. If the dependency graph is acyclic the topological ordering determines the order of computation.

We have no general method to solve the equations with a cyclic dependency graph[7]. However, if the operator $\mathbf{E}$ is monotonous one may relay on Tarski's fixpoint theorem and iterate.

Lemma 4 *(Tarski)* The monotonous function $\mathbf{E}$ from a complete lattice to itself has the minimal fixpoint.

The algebra of tables is isomorphic to a Boolean ring [12]. Roughly speaking, tables can be considered as sets of rows and the lattice is the complete lattice of all subsets of a universal table as a top and empty set as bottom element.

The operator $\mathbf{E}$ is monotonous if it does not contain difference or the substrahend is constant (not appears in the recursion cycle).

In this case one can start with initial values of tables (tables having no initial value are set to be empty) and iterate until stabilize. (If no new tuple is added during one iteration, all the computation can be terminated.)

7 Beyond the first order

We have covered the relation algebra and it possible extensions and generalizations. According to [1] our algebras are complete. However, there are aggregation functions in the SQL. Usually the following aggregative function are considered: *min* – minimum, *max* – maximum, *sum*, *count* and *avg* – average. From the theory point of view the minimum and the maximum are superfluous, they can be expressed in relational algebra SELECT x, $min(y)$ FROM $R(x, y)$ GROUP BY x; is equivalent to $R(x, y) - \Pi_{xy}(R(x, y) \bowtie R(x, y_1) \bowtie (y_1 < y))$. Or more readable $\{\langle x, y\rangle : R(x, y) \wedge \neg\exists(y_1)(R(x, y_1) \wedge y_1 < y)\}$. Any straightforward implementation of the algebraic expression above requires at least quadratic time,the SQL do sorting for the **group by** operation and is more efficient. The same argument is valid for the maximum.

The problem of the others operations is more subtle. Of course $avg = \frac{sum}{count}$. The trivial function *one* defined as $one(x) = 1$ for all x can be introduced, and $count(x) = sum(one(x))$. So only the sum is really needed.

There have been some attempts to introduce aggregate functions [5, 11]. Both introduced extensions of algebra and calculus by aggregate functions and proved

[5]The safety restricts the use of algebraic laws on safe expressions. See later.

[6]This is not the most general form of equations, but it cover the datalog programmes.

[7]In the datalog such equation are called recursive.

their equivalence. One could express sum in relational algebra with minimal fix point (iteration) operator as fixpoint of a sequence of formulae:

$$\begin{array}{lll} sum(R,x,y) = y_1 & \text{iff} & R(x,y_1,t_1) \wedge \neg\exists(y,t)(R(x,y,t) \wedge \langle y,t\rangle \neq \langle y_1,t\rangle) \\ sum(R,x,y) = y_1 + y_2 & \text{iff} & R(x,y_1,t_1) \wedge R(x,y_2,t_2) \wedge \langle y_1,t_1\rangle \neq \langle y_2,t_2\rangle \wedge \\ & & \neg\exists(y,t)(R(x,y,t) \wedge \langle y,t\rangle \neq \langle y_1,t_1\rangle \wedge \langle y,t\rangle \neq \langle y_2,t_2\rangle) \\ & \vdots & \end{array}$$

In the broad terms, the n-th formula says, there is exactly n different couples of $\langle y_i, t_i\rangle$ that $R(x, y_i, t_i)$ for $1 \leq i \leq n$ and sum is equal to $\sum_{i=1}^{n} y_i$.

These formulae has only theoretical significance. They are so cumbersome that is no chance to evaluate their efficiently. We propose to introduce the second order function *subtotal* as the table:
$subtotal(R, x, z = \text{'sum'}y) \equiv \text{SELECT } x, sum(y) \text{ AS } z \text{ FROM } R \text{ GROUP BY } x;$

The *subtotal* requires R to be finite (safe) expression. This is (week) second order formula. The implementation of the subtotal function is easy. Simply, sort the relation R by x and when a value of x is exhausted produce the value of the aggregate. This strategy allows to produce several aggregates on one scan (sort) operation.

Although the other then sum aggregates are superfluous it is reasonable to implement them, because the computation of the aggregate function is more efficient than evaluation of complex expression.

Ullman [14] has introduce other two (week) second order functions *atov* – argument to variables and *vtoa* – variables to argument, which allow to use function symbols as data constructors in datalog. The datalog gains so the full power of prolog.

8 Evaluation and optimization

Our relational algebra is finitary version of the cylindric set algebra CS_ω. Contrary to mathematics we require of results to be finite. So a formula is "kosher" for database if its result is finite we shall call such formulae *safe*.

Rules of safety:

1. Stand alone atomic formula (table) is safe if at least one its pseudokey is available[8] (instantiated to constants).
2. All variables (attributes) which appear in a safe subformula become available.
3. A conjunctive formula (sequence of joins) is safe if it can be reordered and associated so that the first atom is safe and each next joined table has at least one pseudokey available.
4. Projection, renaming and union are safe if their operands are safe.
5. The difference is safe if the minuend is safe and the set of the attributes of the substrahend is subset of the attributes of the minuend.

[8]Ullmann uses the term bound, but though there is an analogy with quantifier bound variables, we try avoid the same term.

Example 6 Let the relation are those from example 4. Then the expressions $R(x,y) \bowtie ADD(x,y,z)$ and $R(x,y) \bowtie ADD(u,v,y)$ are safe. The stand alone $ADD(x,y,z)$ is not safe.

The safety rules determine margins for expression evaluation but does not fully determine the computation usually there remains enough freedom for optimization.

The difference between our algebra of tables and the relational algebra is there are no selections. The pushing selections down corresponds to proper order of joins. There are new laws for subtotal e.g: For aggregative function *sum* or *count*[9], if x is a key of $S(x,u)$, then

(1) $subtotal(R(x,y,z) \bowtie S(x,u), x, \text{`ag'}y) = subtotal(R(x,y,z), x, \text{`ag'}y)$

(2) $subtotal(R(x,y,z) \bowtie S(x,u), xu, \text{`ag'}y) = subtotal(R(x,y,z), x, \text{`ag'}y) \bowtie S(x,u)$

In the majority of cases the right hand side of this equation is considerably more efficient than the left one. In the case that x is no key of $S(x,u)$ the gain is even more dramatic.

(3) $subtotal(R(x,y,z) \bowtie S(x,u), x, \text{`ag'}y) =$
$subtotal(R(x,y,z), x, t = \text{`ag'}y) \bowtie subtotal(S(x,u), x, z = \text{`count'}u) \bowtie (y = t * z)$

The most of today's commercial database engines are not able to do this optimization.

The reasonable strategy of the computation is:

1. Create the expression tree where join operation joins as many operands as needed. (Joins are commutative and associative, we do not want to determine their order too early.)
2. Eliminate common subexpressions. (Standard technique from compilation theory may be used, only restriction is not to produce unsafe subexpression.)
3. Apply the rule for subtotal.
4. Determine order of joins.
5. Cascade the projections and push them down as possible.

To determine the order of joins it is useful to maintain pool of available attributes and assign weights to indices. From the all tables to be joined we choice: A table having available a key if such table does not exist a table with lowest weight. And put all attributes of the chosen table into pool available attributes. Repeat this process until all tables are exhausted.

9 Conclusions

We have been motivated by possible generalizations of the relational algebra. To recognize all the properties which have to be preserved by possible generalizations, we deal with formalization of existing relational algebra. A possible

[9]For the functions *min*, *max* and *avg* the rule (1) is valid in the each case.

formalization – table algebra reminds more manipulation with the tableaus than algebra of relation. This algebra has tight liaison to the logic. Similar result has been achieved by Imielinski and Lipski [4]. They related the relational algebra to the Tarski, Henkin and Monck cylindric algebra. However, cylindric algebras has been developed for construction of the mathematically interesting large models and did not care, too much, about finiteness and finite representation. Although the morphism between our table algebra and cylindric algebra exists, it does not preserve finite representation.

One may seems that, we introduce the cumbersome formalism to describe manipulation with attributes which has been used before in relational algebra informally. It is only partial true, separating the manipulation with the headings one can imagine an implementation which starts with selection of possible values each of the attributes as the intersection of columns of the databases relations and then backtracks and computes other values.

The last but not least our algebra is complete model of a database language it covers arithmetic and aggregate functions, too. We hope that the uniform approach with few operations make the optimization simpler. The last is not so clear. Probably, some optimizations are easier if we distinguish between selection, join and Cartezian product, and other ones are easier if we do not so.

The reconstructed model offer possibility of a very clear object-oriented implementation of a relational engine. To implement tables as class of objects, which may be finite or infinite. Each table provides methods which are associated with keys and indices (pseudokeys). The methods work as follows: giving them values of the key or index they return the set of all rows with this key or index. The engine consists of the implementation of relational operations: projection, join, difference, union, subtotal and operator of the minimal fixpoint. In principal such engine have full programming power (Every computable function can be described and computed.) No further extensions or embedding into an imperative programming language are needed.

References

[1] E. F. Codd: Relational model for large shared data banks. *Communications of the ACM 13*, 1970, pp. 367 – 387

[2] D. Day and S.Sarkar: A Probabilistic relational model and algebra. *ACM transactions on Database System* 21/3 (1996), pp. 339 – 369

[3] P. Hall, P. Hitchcock and S. Todd: An algebra for machine computations. Proc. of the second symposium on principles of programming languages, Palo Alto 1975, pp. 225 – 232

[4] T. Imielinski and W. Lipski: The relational model of data and cylindric algebras. ICS PAS Report 446, WarsaW, 1981.

[5] A. Klug: Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of the ACM 29/4*, 1982, pp. 986 – 997

[6] I. Korec: personal communication.

[7] P. Materna et al.: HIT – database model. Sofsem 81, 1981, pp. 195 –226 (In czech)

[8] D. Maier: The theory of relational databases. Computer Science Press, 1983.

[9] J. D. Monk: Mathematical logic. Springer-Verlag, 1976.

[10] D. A. Simovici and R. L. Tenney: Relational Database Systems. Academic Press, 1995.

[11] J. Steiner: Relational theory of queries. *Data & Knowledge Engineering 3*, 1988, pp. 181 – 196

[12] J. Šturc: A model for retrieval systems and some mathematical problems behind. MFCS 1977, LNCS 53, pp. 484 – 492

[13] A. Tarski: On calculus of relations. *Journal of symbolic logic 6*, 1941, pp. 73 – 89

[14] J. D. Ullmann: Principles of databases and knowledge bases. Computer Science Press, 1989.