A Multi-Set Extended Relational Algebra
A Formal Approach to a Practical Issue

Paul W.P.J. Grefen Rolf A. de By
{grefen,deby}@cs.utwente.nl
University of Twente
The Netherlands

Abstract

The relational data model is based on sets of tuples, i.e. it does not allow duplicate tuples in a relation. Many database languages and systems do require multi-set semantics though, either because of functional requirements or because of the high costs of duplicate removal in database operations. Several proposals have been presented that discuss multi-set semantics. As these proposals tend to be either rather practical, lacking the formal background, or rather formal, lacking the connection to database practice, the gap between theory and practice has not been spanned yet. This paper proposes a complete extended relational algebra with multi-set semantics, having a clear formal background and a close connection to the standard relational algebra. It includes constructs that extend the algebra to a complete sequential database manipulation language that can either be used as a formal background to other multi-set languages like SQL, or as a database manipulation language on its own. The practical usability of the latter option has been demonstrated in the PRISMA/DB database project, where a variant of the language has been used as the primary database language.

1 Introduction

The relational data model is nowadays in widespread use, both in database research and practice. In its formal definition, originally proposed in [7], the model is based on sets of tuples, i.e. it does not allow duplicate tuples in a relation. As such, its definition can be based on standard set theory. Many database languages and systems do require a relational data model with multi-set semantics though. There are two major reasons for this. In the first place, relations allowing duplicate tuples are useful in many application domains where duplicate entities can exist. In the second place, the high costs of duplicate removal in database operations is often prohibitive for the use of a data model that does not allow duplicates.

Several proposals have been presented that discuss multi-set semantics for the relational model. Some of these proposols tend to be rather practical, lacking the formal mathematical background, as is the case in many SQL-based approaches. Other approaches are rather formal, e.g. [1], lacking the connection to database practice. Further, approaches exist that try to capture multi-set semantics within a set-based relational theory. An example is the work in [12], where multi-sets are represented in subsets of columns of set-based relations. Concluding, we can state that the gap between theory and practice for multi-set relational approach has not been spanned yet.

This paper proposes a complete extended relational algebra with multi-set semantics. The algebra has both a clear formal background and a close connection to the standard relational algebra. The expression equivalences used in the set-oriented relational context for query optimization also hold in the proposed multi-set context. The proposal further includes constructs that extend the algebra to a complete sequential database manipulation language. This language can either be used as a formal background for other multi-set languages like SQL [6, 2], or as a database manipulation language on its own. The use of the relational algebra as background for SQL has been discussed for example in [5]. The practical usability of the extended relational algebra as a database manipulation language has been demonstrated in the PRISMA/DB database project [3], where a variant of the language, called XRA [10], has been used as the primary database language.

1063-6382/94 $3.00

1.1 Structure of this paper

This paper covers the complete spectrum from elementary multi-set relational database structures to complete multi-statement transactions. Section 2 starts off with the discussion the structures of the relational data model with multi-set relations. The multi-set extended relational algebra expressions are discussed in Section 3. This section also pays attention to expression equivalence in the multi-set algebra. In Section 4 the algebra is extended to include statements, programs, and transactions to obtain a complete database manipulation language. The paper ends with a number of conclusions in Section 5.

2 Multi-set relational structures

The relational data model consists of *structures* and *operations*. This section presents the structures; operations are discussed in the sections to follow. The structures represent the static properties of the model. They have been defined originally in [7], and described thereafter in many textbooks like [15, 14, 13, 8]. In this section, the structures are redefined to capture the notion of multi-sets of tuples. Note that integrity constraints are not discussed in this paper, although they are sometimes considered part of the relational data model [7, 14]. Interested readers are referred to [11], where this topic is discussed in detail.

The definition of multi-set relational databases is constructed below in several steps. The first basic notion is that of a *domain*.

Definition 2.1 A *domain* $\mathcal{A}$ is a set of atomic values. The term *atomic* refers to the fact that each value in the domain is indivisible as far as operators of the relational data model are concerned. □

Common types of domains are the basic data types of integers, reals, booleans, and strings. More specialized types as time, date, or money are possible too; note that they are also atomic in the sense of the definition above. In the definition below, domains are combined to form *relation schemas*.

Definition 2.2 A *relation schema* $\mathcal{R}$ consists of a relation name and a list of attributes $\langle A_1, \cdots, A_n \rangle$. Each attribute A_i is defined on a domain $dom(A_i)$. The type of $\mathcal{R}$ is defined as $dom(\mathcal{R}) = dom(A_1) \times \cdots \times dom(A_n)$. A *relation* or *relation instance* R of relation schema $\mathcal{R}$ is a multi-set of elements in $dom(\mathcal{R})$, i.e. a function $R : dom(\mathcal{R}) \rightarrow I\!N$, where $I\!N$ denotes the domain of the natural numbers. The value of $R(x)$ is called the *multiplicity of x in R*. □

Two comparison operators are defined on multi-set relations: the equality and multi-subset operators. Both operators have intuitive semantics. They are defined formally below.

Definition 2.3 Given are two multi-set relations R_1 and R_2 defined on schema $\mathcal{R}$. The *equality* operator $=$ and *multi-subset* operator $\subseteq$ are defined as follows:

$$R_1 = R_2 \quad \equiv \quad (\forall x)(x \in dom(\mathcal{R}) \Rightarrow R_1(x) = R_2(x))$$

$$R_1 \subseteq R_2 \quad \equiv \quad (\forall x)(x \in dom(\mathcal{R}) \Rightarrow R_1(x) \leq R_2(x))$$

□

A relation instance consists of *tuples* as defined below.

Definition 2.4 A *tuple* r of schema $\mathcal{R}$ is an element in $dom(\mathcal{R})$. A tuple r is an element of relation R if its multiplicity in R is greater than zero: $r \in R \Leftrightarrow R(r) > 0$. The value of the ith attribute of tuple r is denoted as $r.i$. The number of attributes of r is denoted as $\#r$. The projection $\pi_\alpha(r)$ is obtained by concatenating the attributes from r as specified by the attribute list α into a new tuple. In this, α is a list of prefixed integers $\langle \%i_1, \ldots, \%i_n \rangle$ with $n \geq 1$ and $1 \leq i_j \leq \#r$ for $1 \leq j \leq n$. The *concatenation* of two tuples $r_1 \oplus r_2$ is defined as the concatenation of the attributes of r_1 and r_2 in the specified order. The equality of two tuples $r_1 = r_2$ is defined for tuples having the same schema; $r_1 = r_2$ holds if all corresponding attributes of r_1 and r_2 have equal values. □

The operators π_α and $\oplus$ defined here on tuples are used in the sequel of this paper for relation schemas as well with obvious semantics. For reasons of brevity, their formal definition is omitted.

As stated above, relations are defined as multi-sets of tuples; this means that duplicate tuples are allowed in a relation. Multi-sets can be denoted as a collection of individual tuples r, possibly containing duplicates, or as a set of pairs $(r, R(r))$ without duplicates, where $R(r)$ denotes the number of occurrences of r in R. Further, attributes in a relation schema are ordered to enable attribute addressing by index, rather than by name. This is a notational convention that implies no restrictions with respect to the situation with explicit attribute names, but enables addressing the

attributes of anonymous relations. Attribute numbers are prefixed in attribute lists to avoid ambiguity with normal integer constants.

Definition 2.5 A *database schema* $\mathcal{D}$ is a set of relation schemas $\{\mathcal{R}_1, \cdots, \mathcal{R}_n\}$. A *database* or *database instance* D of database schema $\mathcal{D}$ is a set of relation instances $\{R_1, \cdots, R_n\}$. The set of all possible database instances of schema $\mathcal{D}$ is called the *database universe* $U_\mathcal{D}$, so $U_\mathcal{D} = dom(\mathcal{R}_1) \times \cdots \times dom(\mathcal{R}_n)$. □

Note that a database schema is a set of relation schemas; consequently, relations in a database are always addressed by name. A database instance is also commonly referred to as *database state*, this to clearly distinguish the concept from a database transition, as defined below.

Definition 2.6 A *database transition* of database schema $\mathcal{D}$ is an ordered pair of database states $\langle D^{t_1}, D^{t_2} \rangle$ of schema $\mathcal{D}$, with $t_1, t_2 \in I\!N$ and $t_1 < t_2$. The values t_1 and t_2 are called the *logical time* of the database states. □

Usually, a database transition describes two successive states of the database, so $t_2 = t_1 + 1$ in the definition above. This type of transition is called a *single-step transition*. If not stated otherwise, the term transition is used for single-step transitions in this paper.

3 Multi-set relational expressions

This section introduces multi-set expressions on relational databases. First, the standard relational algebra is discussed. The constructs in this algebra are based on the standard relational algebra operators as they can be found in many textbooks on database systems, e.g. [15, 13, 8]. Note, however, that they are modified to deal with multi-sets of tuples. Next, the algebra is extended with additional operators enhancing the expressiveness of the language with arithmetic operations, aggregate functions, and duplicate elimination. After the relational algebra expressions have been defined, attention is paid to expression equivalence. Expression equivalences specify possibilities for expression rewriting, a topic that is very important for query optimization.

3.1 Standard relational algebra

Below, the basic relational algebra is defined. The basic algebra contains a minimal set of operators to form relational expressions. The algebra is then extended with additional operators that do not enhance the expressiveness of the algebra, but make life somewhat easier. Similar to the notation for multi-set relations, the multiplicity of a tuple x in a multi-set expression E is denoted as $E(x)$.

Definition 3.1 The *basic relational algebra* defines basic relational expressions [13]. A database relation is a basic relational expression. Let E_1, E_2, and E_3 denote basic relational expressions; E_1 and E_2 are defined on schema $\mathcal{E}$, E_3 is defined on schema $\mathcal{E}'$. Then the following constructs are basic relational expressions:

- The *union*[1] $E_1 \uplus E_2$ collects the elements of E_1 and E_2 into a multi-set with schema $\mathcal{E}$:

$$E_1 \uplus E_2 = \{\ (x, E_1(x) + E_2(x)) \mid x \in dom(\mathcal{E})\ \}$$

- The *difference* $E_1 - E2$ "subtracts" the contents of E_2 from the contents of E_1 resulting a multi-set with schema $\mathcal{E}$:

$$E_1 - E_2 = \{\ (x, max(0, E_1(x) - E_2(x))) \mid x \in dom(\mathcal{E})\ \}$$

- The *product* $E_1 \times E_3$ forms the cartesian product of the elements of E_1 and E_3 resulting a multi-set with schema $\mathcal{E} \oplus \mathcal{E}'$:

$$E_1 \times E_3 = \{\ (x \oplus y, E_1(x) \cdot E_3(y)) \mid x \in dom(\mathcal{E}) \wedge y \in dom(\mathcal{E}')\ \}$$

- The *selection* $\sigma_\varphi E_1$ selects elements from a multi-set that meet a condition φ defined on individual tuples in $dom(\mathcal{E})$, resulting a multi-set with schema $\mathcal{E}$:

$$\sigma_\varphi E_1 = \{\ (x, E_1(x)) \mid x \in dom(\mathcal{E}) \wedge \varphi(x)\ \} \cup \{\ (x, 0) \mid x \in dom(\mathcal{E}) \wedge \neg\varphi(x)\ \}$$

  In this definition, φ can be seen as a function from $dom(\mathcal{E})$ into the boolean domain.

[1] The $\uplus$ symbol is used here for the multi-set union to avoid confusion with the set union, denoted by the usual symbol $\cup$.

- The *projection* $\pi_\alpha E_1$ projects a multi-set E_1 on the attributes in attribute list α[2], resulting a multi-set with schema $\pi_\alpha \mathcal{E}$:

$$\pi_\alpha E_1 = \{ (\pi_\alpha x, \Sigma_{\varphi(y)} E_1(y)) \mid x \in dom(\mathcal{E}) \}$$
where
$$\varphi(y) \equiv y \in dom(\mathcal{E}) \wedge \pi_\alpha y = \pi_\alpha x$$

□

Note that attribute numbers in selection conditions and projection attribute lists are prefixed to avoid ambiguity with normal integer constants.

The basic algebra is extended below with additional commonly used operators. The choice of operators is closely related to the normal set-based operators. Multiple variants of the same set-based operator, like the union operators proposed in [1], are avoided.

Definition 3.2 The *standard relational algebra* is defined here as the basic relational algebra extended with two additional constructs. Any basic relational expression is a standard relational expression. Let E_1, E_2, and E_3 denote standard relational expressions; E_1 and E_2 are defined on schema $\mathcal{E}$, E_3 is defined on schema $\mathcal{E}'$. Then the following constructs are standard relational expressions:

- The *intersection* $E_1 \cap E_2$ produces a multi-set consisting of the elements that are both in E_1 and E_2, having schema $\mathcal{E}$:

$$E_1 \cap E_2 = \{ (x, min(E_1(x), E_2(x))) \mid x \in dom(\mathcal{E}) \}$$

- The *join* $E_1 \bowtie_\varphi E_2$ produces a selection on the product of E_1 and E_2, having schema $\mathcal{E} \oplus \mathcal{E}'$:

$$E_1 \bowtie_\varphi E_3 = \{ (x \oplus y, E_1(x) \cdot E_3(x)) \mid x \in dom(\mathcal{E}) \wedge y \in dom(\mathcal{E}') \wedge \varphi(x \oplus y) \} \cup \{ (x \oplus y, 0) \mid x \in dom(\mathcal{E}) \wedge y \in dom(\mathcal{E}') \wedge \neg\varphi(x \oplus y) \}$$

□

As stated above, the intersection and join operators are not necessary from a purely functional point of view. This is shown by expressing them in the other operators in the theorem below.

[2]Here the summation $\Sigma_{\varphi(x)} f(x)$ is to be interpreted as the sum of $f(x)$ for all x satisfying $\varphi(x)$.

Theorem 3.1 The intersect and join operators can be expressed in the difference, respectively selection and product, operators introduced before as follows:

$$E_1 \cap E_2 = E_1 - (E_1 - E_2)$$
$$E_1 \bowtie_\varphi E_2 = \sigma_\varphi(E_1 \times E_2)$$

The proof of the first equivalence is given below. The rather trivial proof of the second equivalence is omitted for reasons of brevity.

Proof. The equivalence is proven by simply substituting the operators by their definitions as presented above:

$$E_1 - (E_1 - E_2) =$$
$$E_1 - \{ (x, max(0, E_1(x) - E_2(x)) \mid x \in dom(\mathcal{E}) \} =$$
$$\{ (x, max(0, E_1(x) - max(0, E_1(x) - E_2(x)))) \mid x \in dom(\mathcal{E}) \}$$

Given the definition of the difference operator, all we have to prove now, is that the following equality holds:

$$max(0, E_1(x) - max(0, E_1(x) - E_2(x))) = min(E_1(x), E_2(x))$$

This can easily be done by distinguishing the following two cases:

$E_1(x) \leq E_2(x)$:
$$max(0, E_1(x) - max(0, E_1(x) - E_2(x))) = max(0, E_1(x) - 0) = E_1(x) = min(E_1(x), E_2(x))$$
$E_1(x) > E_2(x)$:
$$max(0, E_1(x) - max(0, E_1(x) - E_2(x))) = max(0, E_1(x) - (E_1(x) - E_2(x))) = E_2(x) = min(E_1(x), E_2(x))$$

□

An example of a standard relational expression is given below. This and the following examples are based on a simple beer database consisting of two relations relations *beer* and *brewery*. The former describes per beer its name, brewer, and alcohol percentage. The latter describes per brewery its name, and city and country of its location. The schemas are shown below:

beer $(name, brewery, alcperc)$
brewery $(name, city, country)$

Example 3.1 The multi-set of all names of beers brewn in the Netherlands can be calculated as follows:

$$\pi_{(beer.name)} \sigma_{country='NL'} (beer \bowtie_{beer.brewery=brewery.name} brewery)$$

If several Dutch brewers brew beers with the same name, the result of this expression will contain duplicates. □

3.2 Extended relational algebra

The standard relational algebra above can be used for the specification of standard relational algebra expressions. The algebra lacks some important expressive power however: arithmetic expressions on attributes are not possible, duplicates cannot be removed, and aggregates over multi-sets are not included. The definition of the extended relational algebra expressions includes these features.

Before the expressions can be defined, aggregate functions are introduced in a multi-set relational context below.

Definition 3.3 The *multi-set aggregate functions* compute an aggregate value on a specified attribute of a multi-set expression. Let E be a multi-set defined on schema $\mathcal{E}$, and β an attribute of $\mathcal{E}$. The multi-set aggregate functions are defined as follows:

- The *count* function :

  $$CNT_\beta E = \sum_{x \in dom(\mathcal{E})} E(x)$$

  where parameter β is a dummy parameter, included only for reasons of syntactical uniformity.

- The *sum* function:

  $$SUM_\beta E = \sum_{x \in dom(\mathcal{E})} x.\beta \cdot E(x)$$

  where β must have a numeric domain.

- The *average* function:

  $$AVG_\beta E = SUM_\beta E / CNT_\beta E$$

  where β must have a numeric domain.

- The *minimum* function:

  $$MIN_\beta E = \\ min\{\ x.\beta \mid x \in dom(\mathcal{E}) \wedge E(x) > 0\ \}$$

- The *maximum* function:

  $$MAX_\beta E = \\ max\{\ x.\beta \mid x \in dom(\mathcal{E}) \wedge E(x) > 0\ \}$$

□

Note that the set of aggregate functions defined above is rather arbitrary; other choices can be made, including statistical aggregate functions for example. Note further that the average, minimum and maximum functions are in fact partial functions, since they are not defined on empty multi-sets.

Definition 3.4 The *extended relational algebra expressions* are defined as the standard relational expressions extended with three additional constructs. Any standard relational expression is an extended relational expression. Let E be an extended relational expression defined on schema $\mathcal{E}$. Then the following constructs are extended relational expressions:

- The *extended projection* $\pi_\alpha E$ is similar to the normal projection defined above, but α contains arithmetic expressions defined on the attributes of E, rather than attributes of E only. These arithmetic expressions can be seen as functions from $dom(\mathcal{E})$ into a basic domain. Given $\alpha = \langle e_1, \ldots, e_n \rangle$ with $n \geq 1$, the extended projection on a tuple x is defined as[3]:

  $$\pi_\alpha x = [e_1(x), \ldots, e_n(x)]$$

  Here, the square brackets denote tuple construction. Given this redefinition of the tuple projection, the definition of the extended projection operator on multi-sets is the same as the definition of the normal projection operator given before in 3.1. The normal projection operator can be seen as a special case of the extended operator. The extended projection is denoted with the same symbol as the normal projection for reasons of readability; in the sequel of this paper, the π symbol denotes the extended projection.

- The *unique* expression δE calculates the multi-set obtained by duplicate removal on E, having schema $\mathcal{E}$:

  $$\delta E = \\ \{\ (x,1) \mid x \in dom(E) \wedge E(x) > 0\ \} \cup \\ \{\ (x,0) \mid x \in dom(E) \wedge E(x) = 0\ \}$$

- The *groupby* expression $\Gamma_{\alpha,f,\beta} E$ on an expression E with schema $\langle A_1, \ldots, A_n \rangle$ calculates a multi-set aggregate function f on an attribute β producing a value in domain $\mathcal{F}$ per group of tuples, where the grouping is defined by equality of the attributes in the (duplicate-free) attribute

[3] As for the normal projection, the extended projection operator is used for relation schemas as well.

list $\alpha = (\%a_1, \ldots, \%a_k)$:

$$\Gamma_{\alpha,f,\beta}E = \\ \{\ (x,1) \mid x \in G\ \} \cup \\ \{\ (x,0) \mid x \in D' \wedge x \notin G\ \} \\ \text{where} \\ G = \{\ x \in D' \mid (\exists y \in E)(x = \pi_\alpha y \oplus \\ [f(\sigma_{\%a_1 = x.1 \wedge \cdots \wedge \%a_k = x.k}(E), \beta)])\ \} \\ \text{and} \\ D' = dom(A_{a_1}) \times \cdots \times dom(A_{a_k}) \times \mathcal{F}$$

The resulting multi-set has schema $\pi_\alpha \mathcal{E} \oplus ran(f(\pi_\beta \mathcal{E}))$, i.e. the schema of the grouping attributes extended with the type of the range of the aggregate function. If the attribute list α is empty, the groupby expression calculates an aggregate function over the attributes of all tuples in a multi-set; in this case, the result is one single-attribute tuple:

$$\Gamma_{\langle\rangle,f,\beta}E = [f(E,\beta)]$$

□

Example 3.2 The average alcohol percentage of all beers per country can be computed by the following expression:

$$\Gamma_{\langle country\rangle, AVG, alcperc} \\ (beer \bowtie_{beer.brewery = brewery.name} brewery))$$

To reduce the size of intermediate results of this expression, a projection operator may be inserted as shown below:

$$\Gamma_{\langle country\rangle, AVG, alcperc}{}^{\pi_{\langle alcperc, country\rangle}} \\ (beer \bowtie_{beer.brewery = brewery.name} brewery))$$

This example shows the importance of multi-set semantics in a practical context. If multi-set semantics are used, both expressions above yield the same result, as one would expect. If set-semantics are used, however, the second expression produces a different (and incorrect) result! Note that the use of set semantics requires that the projection operator removes duplicates (which are very likely to exist in this example), thereby causing incorrect aggregate values. The equivalent expression in SQL is the following [2]:

```
SELECT country, AVG(alcperc)
FROM beer,brewery
WHERE beer.brewery = brewery.name
GROUP BY country
```

□

3.3 Expression equivalence

Expression equivalence is important for query optimization (see e.g. [4]). The equivalences in the normal set relational algebra generally hold for the multi-set relational algebra as well. A number of important cases is discussed below. A complete list is omitted for reasons of brevity.

Theorem 3.2 The selection and projection operators have the distribution property over the union operator. Given two multisets E_1 and E_2 with schema $\mathcal{E}$, we have:

$$\begin{aligned} \sigma_\varphi(E_1 \uplus E_2) &= \sigma_\varphi E_1 \uplus \sigma_\varphi E_2 \\ \pi_\alpha(E_1 \uplus E_2) &= \pi_\alpha E_1 \uplus \pi_\alpha E_2 \end{aligned}$$

Proof. The proof of the first equivalence is given here; the second proof is omitted. To proof the first equivalence, we start with expanding the right hand size expression; this form is then restructured to the left hand side expression:

$$\sigma_\varphi E_1 \uplus \sigma_\varphi E_2 = \\ (\{\ (x, E_1(x)) \mid x \in dom(\mathcal{E}) \wedge \varphi(x)\ \} \cup \\ \{\ (x,0) \mid x \in dom(\mathcal{E}) \wedge \neg\varphi(x))\ \}) \uplus \\ (\{\ (x, E_2(x)) \mid x \in dom(\mathcal{E}) \wedge \varphi(x)\ \} \cup \\ \{\ (x,0) \mid x \in dom(\mathcal{E}) \wedge \neg\varphi(x))\ \}) = \\ \{\ (x, E_1(x) + E_2(x)) \mid x \in dom(\mathcal{E}) \wedge \varphi(x)\ \} \cup \\ \{\ (x, 0+0) \mid x \in dom(\mathcal{E}) \wedge \neg\varphi(x))\ \} = \\ \sigma_\varphi\{\ (x, E_1(x) + E_2(x)) \mid x \in dom(\mathcal{E})\ \} = \\ \sigma_\varphi(E_1(x) \uplus E_2(x))$$

□

Note that the distribution property does *not* hold for the *unique* operator δ over the *union* $\uplus$. Here we have the following relation:

$$\delta(E_1 \uplus E_2) \subseteq \delta E_1 \uplus \delta E_2$$

Theorem 3.3 The product, join, union, and intersection operators have the associative property. Given three multisets E_1, E_2, and E_3 with schemas $\mathcal{E}_1$, $\mathcal{E}_2$, respectively $\mathcal{E}_3$, we have the following equivalences:

$$\begin{aligned} (E_1 \times E_2) \times E_3 &= E_1 \times (E_2 \times E_3) \\ (E_1 \bowtie_\varphi E_2) \bowtie_{\varphi'} E_3 &= E_1 \bowtie_\varphi (E_2 \bowtie_{\varphi'} E_3) \\ (E_1 \uplus E_2) \uplus E_3 &= E_1 \uplus (E_2 \uplus E_3) \\ (E_1 \cap E_2) \cap E_3 &= E_1 \cap (E_2 \cap E_3) \end{aligned}$$

Proof. The proof of the first equivalence is given; the other proofs are omitted for reasons of brevity.

$$
\begin{array}{l}
(E_1 \times E_2) \times E_3 = \\
\{\ (x \oplus y, E_1(x) \cdot E_2(y)) \mid \\
\quad x \in dom(\mathcal{E}_1) \wedge y \in dom(\mathcal{E}_2)\ \} \times E_3 = \\
\{\ ((x \oplus y) \oplus z, (E_1(x) \cdot E_2(y)) \cdot E_3(z)) \mid \\
\quad (x \in dom(\mathcal{E}_1) \wedge y \in dom(\mathcal{E}_2)) \wedge z \in dom(\mathcal{E}_3)\ \} = \\
\{\ (x \oplus (y \oplus z), E_1(x) \cdot (E_2(y) \cdot E_3(z))) \mid \\
\quad x \in dom(\mathcal{E}_1) \wedge (y \in dom(\mathcal{E}_2) \wedge z \in dom(\mathcal{E}_3))\ \} = \\
E_1 \times \{\ (y \oplus z, E_2(y) \cdot E_3(z)) \mid \\
\quad y \in dom(\mathcal{E}_2) \wedge z \in dom(\mathcal{E}_3)\ \} = \\
E_1 \times (E_2 \times E_3)
\end{array}
$$

□

4 Multi-set relational programs

The previous section has discussed the expressions of the extended relational algebra. In this section we add constructs that build a complete sequential data manipulation language on this basis. Note that the language including these constructs is still called an extended relational algebra, but that it is not an algebra in the mathematical meaning of the word.

First, the basic *statements* are introduced. The statements define constructs to be used for querying and updating a database. Statements can be grouped into *programs* to specify more complex operations against a database. Finally, programs can be given certain execution characteristics to form database *transactions*. Given these properties, transactions are the best level for database access in practice.

4.1 Basic statements

The statements of the extended relational algebra include constructs to query and update a multi-set relational database. They are defined below.

Definition 4.1 The *extended relational algebra statements* are defined as follows. Let R be a database relation, and E an extended relational expression of the same schema. Then the following constructs are extended relational algebra statements:

- The *insert* statement $insert(R, E)$ adds the elements of E to relation R:

  $$
  \begin{array}{l}
  insert(R, E) \equiv \\
  R \leftarrow R \uplus E
  \end{array}
  $$

- The *delete* statement $delete(R, E)$ removes the elements of E from relation R:

  $$
  \begin{array}{l}
  delete(R, E) \equiv \\
  R \leftarrow R - E
  \end{array}
  $$

- The *update* statement $update(R, E, \alpha)$ modifies the elements in the intersection between R and E according to the attribute expression list α with the same schema as E:

  $$
  \begin{array}{l}
  update(R, E, \alpha) \equiv \\
  R \leftarrow (R - E) \uplus \pi_\alpha(R \cap E)
  \end{array}
  $$

  Note that π_α is a structure-preserving extended projection operator here, i.e. it results a multi-set of the same schema as its operand.

- The *assignment* $R = E$ assigns the multi-set E to a new and implicitly defined relational variable R:

  $$
  \begin{array}{l}
  (R = E) \equiv \\
  R \leftarrow E
  \end{array}
  $$

- The *query statement* $?E$ sends the result of expression E as output to the user of the database system; the statement has no effect on the database.

In this definition, the symbol $\leftarrow$ denotes replacement. □

Example 4.1 If brewery Guineken decides to increase the alcohol percentage of its beers by 10%, this can be reflected in the example database by the following update statement:

$$
\begin{array}{l}
update(beer, \sigma_{brewery='Guineken'}beer, \\
\quad \langle name, brewery, alcpec * 1.1 \rangle)
\end{array}
$$

In standard SQL this is the following statement [2]:

UPDATE *beer*
SET $alcperc = alcperc * 1.1$
WHERE $brewery =$ *'Guineken'*

□

4.2 Programs

Extended relational algebra statements can be grouped into *programs* as defined below to specify more complex operations on a database.

Definition 4.2 The *extended relational algebra programs* are defined as follows. Let a be an extended

relational algebra statement and p an extended relational algebra program. Then the following constructs are extended relational algebra programs:

- The *single-statement program* a.
- The *multi-statement program* $p; a$.

□

4.3 Transactions

Operations executed against a database are grouped into transactions to form database programs with certain characteristics.

Definition 4.3 A *transaction* consists of an extended relational algebra program $a_1; \cdots; a_n$ enclosed in *transaction brackets*, to be executed against a database D:

$$T = (a_1; a_2; \cdots; a_n)$$

The parentheses denote the transaction brackets, respectively *begin* and *end*. During the execution of the actions a_i, the database is in a number of *intermediate states*. These states are not normal database states as they may contain temporary relations defined by assignment statements. If the logical time of D is t, then the state after the execution of action a_i is denoted as $D^{t.i}$; $D^{t.0} \equiv D^t$ denotes the state before the execution of a_1. The *end* bracket takes care of the transition from $D^{t.n}$ to a normal database state: if the transaction can *commit*, temporary relations are removed from $D^{t.n}$ and the result $D^{t.n}\downarrow$ is installed as D^{t+1}; if the transaction must *abort*, D^t is installed as D^{t+1}. The states $D^{t.1}, \cdots, D^{t.n}$ have no semantics beyond the execution of T. The pre-transaction state D^t and post-transaction state D^{t+1} are visible to other transactions as well. This means that T is executed in isolation [4]. □

Informally, a transaction is a unit of work executed against a database state. Speaking more formally, a transaction T can be seen as an operator that transforms a database state D into another state $T(D)$ [9], and can thus be associated with a single-step transition of a database:

$$D \xrightarrow{T} T(D)$$

According to the basic transaction model, the execution of a transaction T must satisfy the atomicity, correctness, isolation, and durability properties [9, 4]. In the context of this paper, the atomicity property clearly describes the difference between an extended relational algebra program and a transaction as follows. The execution of T must always satisfy the atomicity property; this means that the effect of any execution of T on the initial database state D must be such that either the effects of T are completed fully, or D remains unchanged. So, if $T = (a_1, \ldots, a_n)$, the following must hold:

$$(T(D) = D^{t.n}\downarrow) \vee (T(D) = D)$$

5 Conclusions

This paper proposes a multi-set extended relational algebra language with three important properties. In the first place, the language has a simple and mathematically defined semantics. In the second place, the language has similar properties as the standard relational algebra. This implies that most research results that hold for the standard algebra also hold for the extended algebra. Examples are equivalence transformations used for query optimization. In the third place, the language has complete expressiveness, necessary for the description of database applications. As such, it can be used as a well-defined database manipulation language on its own, or as a background for existing languages like SQL. The well-defined multi-set semantics provides a solid basis for the functional requirements of real-world applications.

The semantics of the language proposed in this paper is based on set theory. This is the strength of the approach, but also causes a few limitations. As sets do not impose any order on their elements, sort operators and cursor manipulation cannnot be expressed in this formalism, and can thus not be part of the language discussed in this paper. The design of the language is open to extensions to improve its expressiveness, however. The addition of a transitive closure operator allowing expressions with a recursive nature is discussed in [11], for example.

The extended relational algebra has been put into practice in the PRISMA project as the primary data manipulation language of the PRISMA/DB parallel database system. For this purpose, the language has been extended with special operators to support parallel data processing. This demonstrates that extensions are well possible, without violating the well-structuredness of the language.

Acknowledgements

Thanks go to Peter Apers from the University of Twente and Jennifer Widom from IBM Almaden Research Center for their comments and suggestions with respect to earlier versions of this work.

References

[1] J. Albert; *Algebraic Properties of Bag Data Types*; Proceedings 17th International Conference on Very Large Data Bases; Barcelona, Spain, 1991.

[2] American National Standards Institute; *The Database Language SQL*; Document ANSI X3.135, 1986.

[3] P.M.G. Apers, C.A. v.d. Berg, J. Flokstra, P.W.P.J. Grefen, M.L. Kersten, A.N. Wilschut; *PRISMA/DB: A Parallel, Main-Memory Relational DBMS*; IEEE Transactions on Knowledge and Data Engineering, Vol.4, No.6, 1992.

[4] S. Ceri, G. Pelagatti; *Distributed Databases, Principles and Systems*; McGraw-Hill, New York, USA, 1984.

[5] S. Ceri, G. Gottlob; *Translating SQL into Relational Algebra: Optimization, Semantics, and Equivalence of SQL Queries*; IEEE Transactions on Software Engineering, Vol.11, No.4, 1985.

[6] D. Chamberlin et al.; *SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control*; IBM Journal of Research and Development, Vol.20, No.6, 1976.

[7] E.F. Codd; *A Relational Model for Large Shared Data Banks*; Communications of the ACM, Vol.13, No.6, 1970.

[8] R. Elmasri, S.B. Navathe; *Fundamentals of Database Systems*; Benjamin/Cummings, Redwood, USA, 1989.

[9] J. Gray; *The Transaction Concept: Virtues and Limitations*; Proceedings 7th International Conference on Very Large Data Bases; Cannes, France, 1981.

[10] P.W.P.J. Grefen, A.N. Wilschut, J. Flokstra; *PRISMA/DB 1.0 User Manual*; Memorandum INF91-06; University of Twente, The Netherlands, 1991.

[11] P.W.P.J. Grefen; *Integrity Control in Parallel Database Systems*; Ph.D. Thesis; University of Twente, 1992.

[12] A. Klausner, N. Goodman; *Multirelations - Semantics and Languages*; Proceedings 11th International Conference on Very Large Data Bases; Stockholm, Sweden, 1985.

[13] H.F. Korth, A. Silberschatz; *Database System Concepts*; McGraw-Hill, New York, USA, 1986.

[14] D.C. Tsichritzis, F.H. Lochovsky; *Data Models*; Prentice-Hall, Englewood Cliffs, USA, 1982.

[15] J.D. Ullman; *Principles of Database Systems, Second Edition*; Computer Science Press, Rockville, USA, 1982.