

Towards Matrix Manipulation of Future Databases

Maurice Gittens <maurice at gittens dot nl>

1st March 2004

Abstract

Why would one manipulate database relations using only the unary and binary operators of relational algebra? The avenue of thought resulting from attempting to answer this question inspired the matrix based approach to the manipulation of database relations presented in this paper. Based on a formally defined ontology for a number of relational database concepts this paper introduces matrix based manipulation of database relations as defined in this paper.

The copyright of this document belongs to its authors. Making complete and unmodified copies of this document is allowed.

Contents

1 Database Values and Database Types	3
1.1 Introduction	3
1.2 Definition	3
1.3 Types, Type instances and Type templates for Type definitions	3
1.4 Database values	4
1.5 Equality of database values	4
1.6 Database value matrixes	5
2 Attributes, Tuples, Relations and Databases	6
2.1 Defining Database attributes	6
2.2 Defining Database tuples	6
2.3 Defining Database relations	7
2.3.1 Definition	7
2.4 Defining Databases	7
3 Operations on database relations as matrixes	8
3.1 Introduction	8
3.2 Scalar matrix operators	8
3.3 Additive matrix operators	9
3.4 Operator matrixes	9
3.5 Multiplicative matrix operators	9
4 Concluding and future work remarks	10
4.1 Looking back	10
4.2 Looking forward	10

Status: Preliminary draft...

Revision History

- March 1 2004; fixup inaccurcies; improve the wording; we are still not there yet...
- February 24 2004; complete the first draft of this paper

Chapter 1

Database Values and Database Types

1.1 Introduction

This section introduces formal definitions for a number of database concepts such as *database types*, *database type definitions* and *database values*. The mathematical vehicle used to provide a formal definition of these database concepts is generally known as Noam Chomsky's phrase structure grammar. However in this section the terminology used is adapted to the domain of databases.

1.2 Definition

A *database type definition* g is defined as a quadruple (N, Σ, S, P)

- N is a non empty finite set of nonterminal symbols
- Σ is a non empty set of terminal symbols.
- S element of N is a distinguished symbol called the start symbol
- P is a set of rewriting rules called production rules of the form:
$$\omega_1 \cdot A \omega_2 \rightarrow \omega_3 \cdot \omega_4 \quad A \in N$$

$\omega_1, \omega_2, \omega_3$, and ω_4 are elements $(N \cup T)^*$. The symbol A is called the subject of the production rule $\omega_1 \cdot A \omega_2 \rightarrow \omega_3 \cdot \omega_4$. $\omega_1 \cdot A \omega_2$ is called the left-hand side of the production rules while $\omega_3 \cdot \omega_4$ represents the right-hand side.

1.3 Types, Type instances and Type templates for Type definitions

An *instance* x for a type definition g is an element of Σ^* for which there exists a finite sequence $\omega_1, \omega_2, \dots, \omega_n$ ($n > 1$) of so-called *type templates* of g . A *type template* ω_i of a type definition g is an element of $(N \cup \Sigma)^* \cdot (N \cup \Sigma)^*$ where the following holds for ω_i :

$$\omega_1 = \cdot S$$

$$\omega_i = B_i C_i \cdot D_i E_i F_i$$

$$\omega_n = x \cdot$$

For ω_{i+1} the following holds:

1. $D_i \in N$ then $\omega_{i+1} = B_i I_i \cdot J_i F_i$ and $p_i = C_i \cdot D_i E_i \rightarrow I_i \cdot J_i$ is a production rule where $(1 \leq i \leq n-1)$.
2. $D_i \in T$ then $w_{i+1} = B_i C_i D_i \cdot E_i F_i$ where $(1 \leq i \leq n-1)$.

The set of instances which is can be derived from a type specification g , denoted $T(g)$, is called the *type* defined by g . $T(g)$ thus represents the set of all values of the type defined by G .

1.4 Database values

Let Σ represent a non empty set of symbols called the *alphabet* of a database. A *database value* is defined as:

1. any string over Σ^* .
2. any set of database values is also a database value.
3. when f, r are database values any pair (f, r) is also a database value.
4. when f, r, i are database values any triple (f, r, i) is also a database value.
5. when $td = (N, T, S, P)$ is a type definition, $T \subset \Sigma$, td is also a database value
6. nothing else is a database value.

Based on the definition of database values provided above, concepts such as database relations and database attributes will be defined in the following chapter.

1.5 Equality of database values

Let V represent the set of database values over Σ . The *equality function* eq for *database values* is defined as a function $V \times V$ to $\{true, false\}$ from with the following definition:

1. if v_1, v_2 are strings in Σ^* , $eq(v_1, v_2)$ is defined as $v_1 = v_2$ by the definition of equality for mathematical strings.
2. if v_1, v_2 are both sets of database values then $eq(v_1, v_2) = true$ if and only if $|v_1| = |v_2|$ and for each $x \in v_1$ there exists a $y \in v_2$ for which $eq(x, y) = true$.
3. if $v_1 = (x_1, y_1), v_2 = (x_2, y_2)$ are both pairs of database values then $eq(v_1, v_2)$ is defined as $eq(x_1, x_2) = true$ and $eq(y_1, y_2) = true$.
4. if $v_1 = (x_1, y_1, z_1), v_2 = (x_2, y_2, z_2)$ are both triples of database values then $eq(v_1, v_2)$ is defined as $eq(x_1, x_2) = true, eq(y_1, y_2) = true$ and $eq(z_1, z_2) = true$.
5. if v_1, v_2 are both type definitions then $eq(v_1, v_2)$ is defined as $T(v_1) = T(v_2)$, which is to say that v_1 and v_2 are equal if they generate the same set of instances.
6. in all other cases $eq(v_1, v_2) = false$.

1.6 Database value matrixes

A *database value matrix* m is defined as a rectangular array of database values.

$$m = \begin{pmatrix} v_{1,1} & v_{1,2} & \dots & v_{1,m} \\ v_{2,1} & v_{2,2} & \dots & v_{2,m} \\ \dots & \dots & \dots & \dots \\ v_{n,1} & v_{n,2} & \dots & v_{n,m} \end{pmatrix}$$

where each $v_{i,j}$ where $(1 \leq i \leq n)$ and $(1 \leq j \leq m)$ is a database value. Database value matrices have all the properties of mathematical matrices. When defining operations on database values such value matrixes will be instrumental.

Chapter 2

Attributes, Tuples, Relations and Databases

2.1 Defining Database attributes

A database attribute a is defined as a triple (g, a, v) where:

- g is a database type specification
- a is database value called the *attribute name* of a
- $v \in T(g)$, which is to say that v is an instance of the type defined by the type specification G

The *type* of the attribute a , denoted $type(a)$ is defined as $T(g)$. The *value* of the attribute a , denoted $value(a)$ is defined v . By virtue of being triples of database values, database attributes are also database values.

2.2 Defining Database tuples

A *database tuple* t is defined as a set of database attributes such that for each $a = (g_a, i_a, v_a) \in t$ the attribute name i_a is unique. The *type* or *heading* of a tuple t respectively denoted $type(t)$ and $heading(t)$ is defined as the set of pairs $\{(g_a, i_a) | (g_a, i_a, v_a) \in t\}$ which is the set of attribute types in t . By virtue of being sets of database values database tuples and database tuple types are also database values. Where G, V respectively represent the set of type specifications and the set of database values over an alphabet Σ . The *interpretation function* of a database tuple $t = \{(g_1, i_1, v_1), (g_2, i_2, v_2), \dots, (g_n, i_n, v_n)\}$ denoted $interpretationf(t)$ from $G \times V$ to V is defined as the set of pairs $\{(g_1, i_1), v_1), ((g_2, i_2), v_2), \dots, ((g_n, i_n), v_n)\}$. Thus $interpretationf(g_i, i_i) = v_i$ for each $(g_i, i_i, v_i) \in t$. By virtue of being a set of database values, interpretation functions are also database values.

2.3 Defining Database relations

2.3.1 Definition

A database relation r is defined as a triple (g_r, i_r, v_r) where:

- v_r is a set of tuples S such that for each tuple $t_1, t_2 \in S$ it holds that $type(t_1) = type(t_2)$. Additionally each tuple $t \in v_r$ can be partitioned into two sets I_i, C where:
 - I_i , denoted $id(t)$, represents a nonempty set of database attributes called the identity of t in v_r . For each $t_1, t_2 \in v_r$ it holds that $type(id(t_1)) = type(id(t_2))$. For each t_1, t_2 it holds that $id(t_1) = id(t_2)$ if and only if $t_1 = t_2$. Each $a \in I_i$ is called an *identity attribute*. The *identity type* of the relation r is defined as $type(id(t))$ where $t \in v_r$.
 - C , denoted $ca(t)$, represents a possibly empty set of database attributes called *characteristic attributes*. For each $t_1, t_2 \in v_r$ it holds that $type(ca(t_1)) = type(ca(t_2))$.
- i_r is a database value called the *relation name* of r .
- g_r is a type definition such that $v_r \in T(g_r)$.

The *type* of the database relation r , denoted $type(r)$ is defined g_r . The *value* of the database relation r , denoted $value(r)$ is defined as v_r . By virtue of being triples of database values, database relations are also database values. Where G, V respectively represent the set of database type specifications and the set of database values over an alphabet Σ . The *interpretation function* of a database relation $r = (g, i, v)$ denoted $interpretationf(r)$ from $G \times V$ to V is defined as: $interpretationf(r)_r = \{((type(id(t)), id(t)), t) | t \in v\}$.

2.4 Defining Databases

A database d is defined as a triple (C, i, R) where:

- R is a set of database relations denoted $value(d)$ such that for each $r = (g_r, i_r, v_r) \in R$ the relation name i_r is unique
- i is a database value called the *database name* of d .
- C is a set of *database integrity constraints*. A database integrity constraint c of a database d is defined as a triple (f, c, p) where $c = (r_c, g_c, k_c)$ is called the *child* and $p = (r_p, g_p, k_p)$ is called the *parent* of the constraint. Where V is the set of database values alphabet Σ , $f : V \times V \rightarrow V$. For the database integrity constraint c it holds that:
 - $r_c, r_p \in R$
 - For each $t_c \in r_c$ there must exist a $t_p \in r_p$ for which it holds that $f(g_c, k_c) = k_p$ while $T(k_p) = type(id(t_p))$ and $k_p = id(t_p)$.

Where G, V respectively represents the set of type specifications and the set of database values over an alphabet Σ . The *interpretation function* of a database $d = (C, i, \{(g_1, i_1, v_1), (g_2, i_2, v_2), \dots, (g_n, i_n, v_n)\})$ denoted $interpretationf(d)$ from $G \times V$ to V is defined as the set of pairs $\{((g_1, i_1), v_1), ((g_2, i_2), v_2), \dots, ((g_n, i_n), v_n)\}$. Thus $interpretationf(d) = \{((g, i), v) | (g, i, v) \in value(d)\}$.

Chapter 3

Operations on database relations as matrixes

3.1 Introduction

Given the definition of database values presented in the first chapter of this paper many different classes of operations can be defined on database value. However the specific focus of this chapter is on operations on matrixes of database values. Operations on database value matrixes can be partitioned into three classes corresponding to classes of operations defined on matrixes in general.

- scalar relation operators
- additive relation operations
- multiplicative relation operators

This classification is taken from a classification of operations on mathematical matrices.

3.2 Scalar matrix operators

Let v be a database value matrix while, while V represents the set of all database values and $f : V \rightarrow V$, be a function. Let

$$v = \begin{pmatrix} v_{1,1} & v_{1,2} & \dots & v_{1,m} \\ v_{2,1} & v_{2,2} & \dots & v_{2,m} \\ \dots & \dots & \dots & \dots \\ v_{n,1} & v_{n,2} & \dots & v_{n,m} \end{pmatrix}$$

A scalar operator $f(x)$ on the the value v is defined as :

$$f(x) \begin{pmatrix} v_{1,1} & v_{1,2} & \dots & v_{1,m} \\ v_{2,1} & v_{2,2} & \dots & v_{2,m} \\ \dots & \dots & \dots & \dots \\ v_{n,1} & v_{n,2} & \dots & v_{n,m} \end{pmatrix} = \begin{pmatrix} f(v_{1,1}) & f(v_{1,2}) & \dots & f(v_{1,m}) \\ f(v_{2,1}) & f(v_{2,2}) & \dots & f(v_{2,m}) \\ \dots & \dots & \dots & \dots \\ f(v_{n,1}) & f(v_{n,2}) & \dots & f(v_{n,m}) \end{pmatrix}$$

Viewing each $v_{i,j}$ in the matrix above as the value of a database relation allows relational algebra's *NOT* to be readily classified as a scalar matrix operation. ¹

¹This is not to say that there exist no other ways to classify the NOT operator

3.3 Additive matrix operators

Let v_1, v_2 be a database value matrixes, while V represents the set of all database values and $+$: $V \times V \rightarrow V$, be a function. Let

$$v_1 = \begin{pmatrix} v_{1,1} & v_{1,2} & \dots & v_{1,m} \\ v_{2,1} & v_{2,2} & \dots & v_{2,m} \\ \dots & \dots & \dots & \dots \\ v_{n,1} & v_{n,2} & \dots & v_{n,m} \end{pmatrix} \text{ and } v_2 = \begin{pmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,m} \\ w_{2,1} & w_{2,2} & \dots & w_{2,m} \\ \dots & \dots & \dots & \dots \\ w_{n,1} & w_{n,2} & \dots & w_{n,m} \end{pmatrix}.$$

The result of an additive relation operator $+$ on v_1 and v_2 denoted $value(r_1) + value(r_2)$ is defined as:

$$\begin{pmatrix} v_{1,1} & v_{1,2} & \dots & v_{1,m} \\ v_{2,1} & v_{2,2} & \dots & v_{2,m} \\ \dots & \dots & \dots & \dots \\ v_{n,1} & v_{n,2} & \dots & v_{n,m} \end{pmatrix} + \begin{pmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,m} \\ w_{2,1} & w_{2,2} & \dots & w_{2,m} \\ \dots & \dots & \dots & \dots \\ w_{n,1} & w_{n,2} & \dots & w_{n,m} \end{pmatrix} = \begin{pmatrix} v_{1,1} + w_{1,1} & v_{1,2} + w_{1,2} & \dots & v_{1,m} + w_{1,m} \\ v_{2,1} + w_{2,1} & v_{2,2} + w_{2,2} & \dots & v_{2,m} + w_{2,m} \\ \dots & \dots & \dots & \dots \\ v_{n,1} + w_{n,1} & v_{n,2} + w_{n,2} & \dots & v_{n,m} + w_{n,m} \end{pmatrix}.$$

When each $v_{i,j}, w_{k,l}$ in the matrixes above represent relation values, which are but the sets of database tuples, it is quickly recognized that binary operators in relational algebra can be classified as additive database matrix operators.²

3.4 Operator matrixes

An database operator matrix O is defined as a rectangular array of functions $f_{i,j}$ where $(1 \leq i \leq n)$ and $(1 \leq j \leq m)$ are functions $f_{i,j} : V \rightarrow V$ where V represents the set of all database values.

$$O = \begin{pmatrix} f_{1,1} & f_{1,2} & \dots & f_{1,m} \\ f_{2,1} & f_{2,2} & \dots & f_{2,m} \\ \dots & \dots & \dots & \dots \\ f_{n,1} & f_{n,2} & \dots & f_{n,m} \end{pmatrix}$$

3.5 Multiplicative matrix operators

Let V represent the set of all database values while $+$ represents a binary function from

$V \rightarrow V$ to V . Let $M = \begin{pmatrix} v_{1,1} & v_{1,2} & \dots & v_{1,p} \\ v_{2,1} & v_{2,2} & \dots & v_{2,p} \\ \dots & \dots & \dots & \dots \\ v_{m,1} & v_{m,2} & \dots & v_{m,p} \end{pmatrix}$ be a database value matrix. Let

$O = \begin{pmatrix} f_{1,1} & f_{1,2} & \dots & f_{1,n} \\ f_{2,1} & f_{2,2} & \dots & f_{2,n} \\ \dots & \dots & \dots & \dots \\ f_{p,1} & f_{p,2} & \dots & f_{p,n} \end{pmatrix}$ be an operator matrix. The multiplicative product of

M and O , denoted MO is defined as:

$$\begin{pmatrix} v_{1,1} & v_{1,2} & \dots & v_{1,m} \\ v_{2,1} & v_{2,2} & \dots & v_{2,m} \\ \dots & \dots & \dots & \dots \\ v_{n,1} & v_{n,2} & \dots & v_{n,m} \end{pmatrix} \begin{pmatrix} f_{1,1} & f_{1,2} & \dots & f_{1,n} \\ f_{2,1} & f_{2,2} & \dots & f_{2,n} \\ \dots & \dots & \dots & \dots \\ f_{p,1} & f_{p,2} & \dots & f_{p,n} \end{pmatrix} = \begin{pmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,n} \\ c_{2,1} & c_{2,2} & \dots & c_{2,n} \\ \dots & \dots & \dots & \dots \\ c_{m,1} & c_{m,2} & \dots & c_{m,n} \end{pmatrix}$$

where for each $c_{i,j}$, $(1 \leq i \leq m)$ and $(1 \leq j \leq n)$, can be written as: $c_{i,j} = f_{1,j}(v_{i,1}) + f_{2,j}(v_{i,2}) + \dots + f_{p,j}(v_{i,p})$. Alternatively we may write $c_{i,j} = \sum_{k=1}^p f_{k,j}(v_{i,k})$.

²It is not really relevant which particular definition of the relational operators the reader might prefer

Chapter 4

Concluding and future work remarks

4.1 Looking back

This paper introduced a formal a definition of a database ontology. Database matrices are suggested as a primary means for the effective manipulation of concepts such database base values and database tuples which were formally defined in the database ontology.

4.2 Looking forward

It is currently recognized that there exist different ways to map database concepts unto matrixes for manipulation. The actual way in which database based concepts are to be mapped unto matrices for manipulation is expected to be determined by the specifics of the transformation to be applied and the properties of the database values to be transformed. Expediently incorporating matrix manipulation techniques in future databases is deemed a valuable area of research.

Bibliography

- [1] C.J. Date, Hoge Darwen [2000] Foundation for Future Database Systems, Addison-Wesley Publishing Company.
- [2] J. Loeckx [1970] "Parsing for general phrase structure grammars". Information and control 16, 443-464.
- [3] J. van Eijck, E. Thijsse, [1989], Logica voor alfa's en informatici, Academic Service
- [4] Maurice Gittens, [2003] Generating efficient table driven parsers for non context free languages, <http://www.gittens.nl>
- [5] Seymour Lipschutz, [1976] Discrete Mathematics, McGraw-Hill book company