Proceedings of the 5th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases, Madrid, Spain, February 15-17, 2006 (pp95-99)

The Change of Algorithmic Data Dependencies and their Properties in Relational Sets

BIRUTĖ PLIUSKUVIENĖ, PETRAS ADOMĖNAS
Information SystemsDepartment
Vilnius Gediminas Technical University
Saulėtekio al. 11, Vilnius
LITHUANIA

Abstract: In this article an extension of the relational model with a new method is presented; i.e., the realisation of algorithmic dependencies between attributes that are in initial data sets formed for solving a particular applied problem. The concept, classification, general expressions and change of algorithmic dependencies are defined.

Key-words: algorithmic dependencies, relational sets, attribute, identification model, tuple, domain.

1 Introduction

Problems arise while solving applied problems because as problem domains change initial data structures and their contents as well as algorithms of problems being solved often have to be changed. Since it is very difficult or impossible to plan for future modification beforehand, one is forced to design and program applied problems and to incorporate them into functioning systems anew. And that often changes algorithms for applied problems. Therefore, adaptive data processing systems are created whose purpose is to minimize difficulties in solving problems when data and the algorithms for their processing change. To create an adaptive system for data processing one must have data structures of large volume and various nature, in which both initial and processed data are provided in the same structures as expressed by relational sets (RS). These data structures can provide initial data and result data on real-world object, phenomena, processes, etc [1].

Since solving many applied problems requires that the data for processing be supplied in some order or sequence, for this purpose we provide the model for identifying relational sets that contains all the required identifiers. This model for identifying data structures provides an opportunity out of the totality of data expressed by relational sets to easily collect the data for designing and solving a particular problem in the order and quantity required. This allows one to control data completeness, decrease the time and resources for solving a problem, since the same model fits for any applied problem whose initial data and the data after processing are relational sets [2].

Having formed the set of initial data or the sets required for solving a particular problem, that problem can be solved as the realization of certain algorithmic dependencies between data because algorithmic dependencies appear between attributes. Therefore, while solving this problem the attributes are being associated, i.e., they depend on one another. As one of them changes other attributes can either change or not change. Also, the latter algorithmic dependencies may change. For one processing being designed they can be of some nature between the same attributes and for another processing they may need to be chosen in a similar manner.

2 The Classification of Algorithmic Dependencies

From a theoretical standpoint, the algorithmic data dependency is considered an algorithm that determines how data are processed while solving the problem required that is inside the same RS. If data are in a different RS, then it is necessary to perform the exclusive algorithmic dependency of data transformation. The algorithm realization of the latter dependency transforms out of any RS collection into one of the needed RS of initial data for a particular problem. The data in this data processing system are really attribute values c_{ij} of RS. For every attribute value or for any subset of RS values an ordered set of algorithmic dependencies is chosen whose programmable

modules for realizing algorithms of its elements perform data processing:

$$<f_{ij}> \rightarrow <c_{ij}> \rightarrow <c'_{ij}>,$$

where $<f_{ij}>$ is a set of algorithmic dependencies, $<c_{ij}>$ – a set of initial data and $<c'_{ij}>$ – the result of problem solution.

If, while solving a data processing problem, a need for new algorithmic dependencies or the models for their realization arises because their totality at hand cannot perform certain actions required for solving the problem, then a new algorithmic dependency and the model for its realization is created that are included into the sets of algorithmic dependencies and programmable models already present. The new algorithmic dependency together with the others also includes the realization of the solution algorithm for the new problem. We can thus claim that the set of all algorithmic dependency classes and the number of algorithmic dependencies in the class is open.

Algorithmic dependencies (AD) can be subdivided into six classes:

1) The AD of RS transformations – f^{t}. This class determines the operation that enables to transfer the attribute values required from the initial data sets for solving an already-formed particular problem to one common set. If needed, the structure and contents of data sets can be changed during transformation.

2) The computation-infological AD – f^{ci}. This class is divided into two closely interconnected but separate groups. The first group is used to describe various arithmetic operations also getting the results of these operations. The second AD group is used to compare the results of various arithmetic operations while checking the compatibility and correctness of various processes.

3) The computational AD in domains – f^{c}. Computation-infological ADs are applied to attribute values in RS tuples, so computational ADs are defined that are applied only to RS domains. If arithmetic operations have to be performed in domains, an additional domain of special tuple keys and the sums of various attribute values is introduced. In this case, the distribution of attribute values and the number of tuples can vary.

4) The internal AD of attribute values – f^{a}. This is a group of very varied algorithmic dependencies, rather different from one another, that are relatively permanent. It means that as a particular attribute value is used in solving different problems, that attribute value can differ. Even such algorithmic dependency of an attribute value as an attribute value – number or text – cannot be considered permanent. If we define an attribute value as text, this value could not be used in an arithmetic operation. While solving one problem one may need to consider attribute values to be numbers, and solving another – to be text.

5) The AD of program steps – f^{p}. Many algorithmic dependencies use data from various RS addresses so it is necessary to start a program step at the required address and continue in the required direction. Otherwise, the process of data processing becomes incorrect or impossible. All program steps in algorithmic RS (ARS) are divided into successive and non-successive. An ARS is a set composed of the identifiers for algorithmic dependencies of attribute values that unambiguously determines the use of attribute values in a certain operation of data processing.

6) The external AD – f^{e}. The external dependencies of algorithmic RS regulate the interrelation of the ARS of the data being processed with the adjacent ARS. These links cannot be separated from the RS of the data being processed.

Several of the main classes of algorithmic dependencies are described in more detail in other sections.

3 The Algorithmic Dependencies of RS Transformations

RS transformations is one kind of algorithmic transformations and is denoted by f^{t}. Set transformation are expressed by the formula that can change operation of relational algebra, and in many cases can exceed their capabilities. Since arithmetic or logical operation can be realized while performing transformations, the RS accepting data can be formed in part or fully out of completely new attribute values c_{ij} whose data has not been present at the source. If needed we can change both the structure and contents of data sets [3].

As noted before, an RS transformation is an operation that enables one to transfer attribute values from one set to another. The RS from which data is transformed is called a data source, and the set into which data is transformed is called an RS receiving data. If the data coordinates of the receiving RS are denoted by i and j, and the coordinates of a data source by k and l, then the data is transformed from addresses $< k / l >$ to addresses $< i / j >$. The RS addresses

from which data is taken and the RS addresses to which data is entered must be at the same distance from the start of their own RS. It is sufficient to denote RS addresses by l and j, i.e., to show only the distance c from the start of the tuple.

RS transformations can be subdivided according to semantics algorithms into unconditional, conditional, conditional continuous, conditional cyclical and result.

In all the transformations mentioned except for unconditional the conditions for comparing data are denoted by the following symbols: $=, \neq, >, <, \geq, \leq, \wedge, \vee, \neg$.

In conditional transformations the contents of addressed are compared, and the transformation is performed only if the condition is satisfied.

Transformation addresses in RS can be of several kinds:

- the addresses from which data is taken $< k / l >$;
- the addresses to which data is deposited $< i / j >$;
- the addresses whose contents are compared by the set condition $\{ k / l \}$, $\{ i / j \}$.

These addresses are merged into simple set not ordered ones because the addresses being compared can be very different. For example, the contents of one address can be compared with a large quantity of source addresses, and the transformation performed if the condition is satisfied. In this case the order of positioning source addresses has no significance whatsoever.

Hence, the address structure of the transformation formula is as follows:

$$\{ k / l \}^n < k / l >^n \xrightarrow{\Sigma} < i / j >^n \{ i / j \}^n.$$

Here the contents of addresses $\{ i / j \}$ and $\{ k / l \}$ are being compared; Σ is defined as a symbol for one of the comparison conditions mentioned above. When the condition is satisfied, data are transferred from address $< k / l >$ to addresses $< i / j >$. The arrow indicates the direction of data transformation. Index n means the number of set order. At the same time these indices can be all different or all the same. If n values are different, then the comparison of data to satisfy the condition Σ is performed in some groups and the transformation of data in the others. If n values are the same, then data are compared and transformed in the same RS; i.e., the change (data positioning and/or their quantity) of one RS structure is performed.

The data source can contain an unlimited number of RS and various schemas. Since such use of the data source is governed by one expression of the transformation formula, the formula must be capable of regulating the selection of data from the source and their inclusion into the receiving set. The symbol Δ denotes the data selection regulator, and the data inclusion regulator is denoted by ∇. So the RS transformation formula is as follows:

$$f^t(\Delta \{k / l\}^n <k / l>^n \xrightarrow{\Sigma} \nabla <i / j >^n \{i / j\}^n).$$

In case of unconditional transformation condition Σ is not in the formula and neither are the addresses in the parentheses. Hence, the formula of unconditional transformation:

$$f^t(\Delta < k / l >^n \longrightarrow \nabla < i / j >).$$

In expression $< i / j >$ index n has no meaning because the RS receiving data is always a single one.

In all transformation data can be transferred from one set to another in three ways:

- performing Cartesian transformation;
- performing transformation in tuples;
- performing transformation in domains.

We set the schema for writing down transformation:

[Data source – RS] ⟶ [RS receiving data] =
= [Transformation result – RS]

We provide an RS example of Cartesian unconditional transformation that is performed according to the formula:

$$f^t(< 1,2 > \rightarrow < 3,4 >).$$

K	L
a	b
c	d

⟶

M	N
e	f
g	h

=

M	N	K	L
e	f	a	b
e	f	c	d
g	h	a	b
g	h	c	d

Traditional capabilities of Cartesian product are extended by the formulas:

$$f^t(< 1,2 > \rightarrow < 1,3 >);$$
$$f^t(< 1,2 > \rightarrow < 2,4 >);$$

because the RS result changes the RS schema itself:

$$\begin{array}{|cc|}\hline K & L \\ \hline a & b \\ c & d \\ \hline\end{array} \longrightarrow \begin{array}{|cccc|}\hline M & K & N & L \\ \hline e & 0 & f & 0 \\ g & 0 & h & 0 \\ \hline\end{array} =$$

$$= \begin{array}{|cccc|}\hline M & K & N & L \\ \hline e & a & f & b \\ e & c & f & d \\ g & a & h & b \\ g & c & h & d \\ \hline\end{array}$$

The ability to change the RS schema is especially valuable because it extends the capabilities for manipulating data without increasing the quantity and complexity of programmable modules. An example of this can be the capability to get the following result by using address references in formula

$$f^{t}(<2> \rightarrow <3>):$$

$$\begin{array}{|cc|}\hline K & L \\ \hline a & b \\ c & d \\ \hline\end{array} \longrightarrow \begin{array}{|cc|}\hline M & N \\ \hline e & f \\ g & h \\ \hline\end{array} =$$

$$= \begin{array}{|ccc|}\hline M & N & L \\ \hline e & f & b \\ e & f & d \\ g & h & b \\ g & h & d \\ \hline\end{array}$$

Transformation in a tuple is performed by merging the tuples at the same distance from the start of an RS to one tuple:

$$\begin{array}{|ccc|}\hline D & E & F \\ \hline s_1 & r_1 & z_1 \\ t_1 & u_1 & v_1 \\ \hline\end{array} \longrightarrow \begin{array}{|ccc|}\hline A & D & B \\ \hline r & t & s \\ u & z & v \\ \hline\end{array} =$$

$$= \begin{array}{|cccc|}\hline A & D & B & F \\ \hline r & s_1 & s & z_1 \\ u & t_1 & v & v_1 \\ \hline\end{array}$$

Unconditional transformations in a tuple are performed by the formula:

$$f^{t}(<1,3> \rightarrow <2,4>).$$

If the data selection addresses in the data source and the data recording addresses in the receiving RS match, then unconditional data transformation in domains has to be performed, that is:

$$f^{t}(<1,2,3> \rightarrow <1,2,3>);$$

$$\begin{array}{|cccc|}\hline K & L & M & N \\ \hline a_1 & b_1 & e_1 & f_1 \\ c_1 & d_1 & g_1 & h_1 \\ \hline\end{array} \longrightarrow \begin{array}{|ccc|}\hline K & L & N \\ \hline a & b & e \\ c & d & g \\ \hline\end{array},$$

$$\begin{array}{|ccc|}\hline K & L & M \\ \hline a_2 & b_2 & e_2 \\ c_2 & d_2 & g_2 \\ \hline\end{array} = \begin{array}{|ccc|}\hline K & L & M \\ \hline a & b & e \\ c & d & g \\ a_1 & b_1 & e_1 \\ c_1 & d_1 & g_1 \\ a_2 & b_2 & e_2 \\ c_2 & d_2 & g_2 \\ \hline\end{array}$$

The examples of unconditional data transformation provided illustrate only in a general sense the capability to change the structure and contents of relational sets during transformation. Transformation capabilities can be very easily changed and extended.

4 Data Completeness

Guaranteeing data completeness is achieved in two ways, the use of which in problems is necessary if the problem algorithm requires such completeness at all. First, it must be guaranteed that for solving a problem all the required RS for that solution be provided. Second, every RS provided must have all the tuples required for solving the problem.

An RA is identified by *aBD,* where: *a* is a schema code (name); *B* – the RS filled with attribute values for a dependency subject, i.e. *B* – a subject code or name; *D* – a time factor, its interval or moment, defining RS data. In the identification model attributes are denoted by upper case letters and attribute values by the same lower case letters [3].

A compact representation of identifiers would be $a_{1,2}$ *B*–$b_{2\text{-}4}$ D–d_1 with the capability to denote each component with any indices required and changing their positions in the formula: *aBD, aDB, BaD*, *BDa*, *DaB*, *DBa*. This way, for solving a particular problem we can guarantee the provision of any number of RS required and in any order of RS required [3]. The formula shown here can be expanded into the identifier sequence for each RS:

$$a_{1,2}\ B\text{–}b_{2\text{-}4}\ \text{D–}d_1 =$$
$$= a_1b_2d_1\ ,\ a_1b_3d_1\ , a_1b_4d_1\ , a_2b_2d_1\ , a_2b_3d_1\ ,$$
$$a_2b_4d_1$$

So it is evident that if no RS are found in a data source whose identifiers are in the identifier sequence provided for a particular problem, the

missing RS for solving the problem are determined.

Checking data completeness in every RS provided for solution, it is determined if all tuples needed for a problem are in each of the RS. Therefore, at the RS schema it is necessary to indicate both key attribute and its value in a fixed order:

- schema – r (*K*, *L*, *M*, *N*);
- key attribute with keys –$K(c_{1j}, c_{2j}, c_{3j},..., c_{mj})$.

Hence during the transformation of attribute values the tuples not needed for solving a particular problem are not touched, but if in RS a tuple from sequence:

$$c_{1j}, c_{2j}, c_{3j},..., c_{mj},$$

is not found, then it is easy to determine that data is missing and precisely what tuples in a particular RS are not provided.

5 Computation-Infological Algorithmic Dependencies

Computation-infological ADs are algorithmic dependencies whose result of algorithm realisation is a number and a group of qualities that compute two numbers in a completely identical way and that relate them by one of the following symbols:

$$=, \neq, >, <, \geq, \leq.$$

Computation (arithmetic) operations can be:

$$+, -, :, \times \text{ and others.}$$

The first group of the algorithmic dependencies of this class is called the computation AD and is denoted by f^{+}, and the second – the infological AD and is denoted by $f^{=}$. The general expression for the first group of algorithmic dependencies is:

$$f^{+}(A \psi A \psi [c] \psi ... \rightarrow A),$$

where *A* is an address of an attribute value in a relational set; ψ – one of the symbols for computation operations; → – a reference to the address where the computation result is to be placed; [c] – a numerical constant (percentage calculation can serve as an example of using a constant).

In the general expression for infological algorithmic dependencies a symbol for comparing data can replace only once any symbol of computation operations:

$$f^{=}(A \psi A \psi A \psi ... A).$$

Since this class closely relates computation operations and information logic reflected in the real object and processes, the algorithmic dependencies described are called computation-infological dependencies.

Conclusions

The realisation of the data algorithmic dependencies defined not only indicates how we can process data while solving a certain applied problem but also enables to process them. It is possible because every algorithmic dependency for its concept has the realisation algorithm and the programmable module for realising that algorithm. Hence, we can use the same algorithms of algorithmic dependencies to solve various problems. It allows us to extend the list of solvable problems without writing new programmable tools, since the said algorithms can be realised by independent or autonomous programmable modules. If existing algorithmic dependencies are not sufficient for solving a particular problem, the system can be extended. That is performed not by creating programmable tools that would realise the algorithm of a new problem directly but by enhancing the existing system with new algorithmic dependencies that would enable, together with already existing ADs, to solve a new problem. This capability of enhancing the system often provides new capabilities not planned directly. And as the system is enhanced with new algorithmic dependencies, such additions are needed ever less frequently.

References:

[1] P.G.Adomėnas, Data Functional Feature Sets and their Adaptability, *Proc. V East–European Conference on Advances in Databases and Information Systems*, Vilnius, 2001, pp. 131-140.

[2] B.Pakalniškytė, P.Adomėnas, The Identification Model of Data Structuries, *Proc. IV Conference on Information Technology,* Alytus, 2005, pp. 157-162.

[3] P.G.Adomėnas, A.Čiučelis, Data Aggregation Sets in Adaptive Data Model, *Informatica*, Vol. 13, No 4, 2002, pp. 381-392.

[4] A.Binemann-Zdanowicz, Current Issues in Databases and Information Systems, *Proc. East-European Conference on Advances in Databases and Information Systems*, Prague, 2000, pp. 307-314.

[5] B.Thalheim, Dependencies in Relational Database, Teubner, 1991.

[6] J.D.Ullman. Principles of Databese and Knowledge-Base Systems. Computer Science Press, Rockville, 1988.