MTA SZTAKI Közlemények 32/1985 13-25

AN EXPERIENCE WITH DYNAMIC DATA INDEPENDENCE

N. BUKOVSKI

Institute "Interprograma"
Sofia, Bulgaria

1. INTRODUCTION

Information handling in Data Dictionaries (DD) is maintained by program components, performing the data access, the data storing and organization. One way to implement such program components is by using Data Base Management Systems (DBMS): self-made or packages. In this paper we will discuss how a DBMS, as a part of the DD, can be used to satisfy the DD requirements for handling its information. For simplicity purposes by DBMS we will imply the program components, realizing exactly this DD function (although it would be more precise to denote it as DD DBMS). The DD has a fixed logical data structure, but nevertheless some changes could be made during the processes of DD installation and operation, e.g., adding user's information types, or physical data restructuring. This requires from the DBMS, upon which the DD is built, to support dynamic data independence. So the rest of the DD software is isolated, thus avoiding its recoding or recompilation. The way the dynamic data independence concept was incorporated into the DBMS of a Data Dictionary is presented in this paper. The DD concerned is an integrating tool in the PLUS complex - a program development environment [1], [2] and provides information for the numerous PLUS components. It, together with the DD extensibility feature, supporting user-defined information types, requires that all programs, accessing the DD, should be independent of its data structure and organization. We shall discuss how a self-made (autonomous) DBMS was designed to support dynamic data independence together with the advantages and drawbacks of the approach chosen.

II. INITIAL REQUIREMENTS

A DD consists of two software components: a DBMS, accessing the DD database, and functional software, implementing the DD functions - selective report generation, cross-referencing, etc. The information contained in the DD database comprises entities of different classes and the relationships among them. Each entity or relationship type consists of attributes: description, keywords, etc.

Considering the DD information and functions needed and its including in the PLUS complex, the following initial requirements for the DD were established:

(1) Data independence. The DBMS should allow changes in the data structure without recoding or recompilation of the application programs accessing them (by application program we imply here and later in the paper any program of the DD functional software or of the PLUS environment which accesses the DD information). These changes include:

- logical structure modifications, for example, the adding of entity attributes or classes;
- physical data restructuring intended to improve DBMS performance in compliance with the specific conditions of the DD usage. For example, another segmentation of the logical record can be chosen, depending on the specific usage frequency of its data elements, or the parameters of the hash address method can be adjusted to the prevalent data volume or changeability.

We will point out that these changes are required in the process of the DD operation which affects especially strongly the PLUS components accessing the DBMS of the DD.

(2) Interface simplicity. The simplicity of the interface, i.e. of the data manipulation statements, used to interface with the DBMS, requires that application program should not be concerned with the variety and complexity of the DD information. This has two aspects. The first one requires that application programs, accessing the DD database, should know only the data they are processing, not the DD information as a whole. The second objective requires a unified access method to be provided, irrespective of the logical type of the data needed: records or relationships. They should be processed in a uniform way, requiring a key - simple or composed, and a list of the necessary attributes. This requirement is influenced by the simplicity of the relational data model, dealing only with relations ("flat" files) and avoiding data structuring concepts, e.g., linked records.

III. PRELIMINARY DESIGN DECISION ON DATA INDEPENDENCE

We will make two preliminary decisions, allowing fulfillment of the first objective: data independence.

(1) Data independence requires the DBMS to perform transformation (mapping) of the data according to their schemes: external, logical and internal [3]. A choice has to be made about the moment data mapping is performed: at run time (dynamic), or before calling the DBMS (static). In [3] dynamic data independence is recommended when supporting unplanned (ad hoc) queries. The DD supports exactly such queries, providing reporting and interrogation facilities. The criteria, through which the DD data are selected, are so numerous, that preliminary planning of all queries and their structures is impossible, especially with the DD extensibility feature. The solution to derive the

queries data structure from the logical schema of the DD information burdens the programs processing these queries with the complexity of the whole logical scheme. So the only solution is to allow the definition of the data needed, i.e. the program view of the data it processes, to be done at run time.

(2) Another aspect of data independence concerns the PLUS components, which interface with the DBMS. Any change in the data structure or organization must not affect their programs. They have to be totally insulated, even their recompilation is not acceptable (though this could be permitted for the DD programs). So this requires that modification of the logical and physical schemas should be allowed at any time without revision of the programs.

These two considerations require that the DBMS should support dynamic data independence, both at logical and physical level.

IV. INTERFACE DEFINITION

The first step of the DBMS design will be defining of its data manipulation statements. The DBMS will be treated as a "black box", which provides the interface to the DD information. The second step will be designing of this black box, i.e. the DBMS.

The dynamic data independence we specified requires that programs, interfacing with the DBMS, should define in the data manipulation statements the structure of the data they process. On the other hand this definition has to be represented in a table-oriented way (as relations), thus satisfying the objective for interface simplicity. So the first step in the process of the interface definition will be specifying these relations.

There are two types of relations, corresponding to the DD information:

- entity relation. Each entity relation represents an entity class: every tuple of the relation corresponds to an entity, every domain - to an attribute. So a separate relation is maintained by the DBMS for each entity class. The entity relation is represented as
  $R(ID, A_1, A_2, \ldots)$

  where R - the type of the relation, i.e. the entity class;
  ID - the relation key, identifying its tuples (usually it is the entity name);
  $A_1, A_2, \ldots$ - the list of the entity attributes, composing this relation:

- relationship relation. A separate binary relation is maintained for each couple of entity classes, relationship between which is allowed. Each relation tuple corresponds to a relationship between two entities, every relation domain - to an attribute, describing this relationship (the so called intersection data). This type of relation is represented as

  $$R(ID_1 + ID_2, A_2, \ldots)$$

  where R - the relation type;
  ID_1+ID_2 - the composed key of this relation, identifying its tuples. ID_1, ID_2 are the keys of the two entity classes;
  $A_1, A_2, \ldots$ - the list of the relationship attributes (intersection data).

Now we shall define two data manipulation statements, using some notions of the relational data languages [4], [5]. We will point out that before using these operations, the DBMS user is provided with all relations, maintained by the DBMS and describing its information. For simplicity, only one type of data access will be described - data retrieval.

The two datamanipulation operators are as follows:

- FOR. This statement retrieves sequentially the tuples of a given entity relation. Each time the statement is performed, a tuple (i.e. an entity) is derived and only the attributes, specified in the statement, are moved. The statement has the following format:

$$FOR\ R(A_1, A_2, \ldots)$$

- PREDICATE. This statement seguentially processes only these entities or relationships of relations, which satisfay the condition specified in the statement. Each time the statement is performed, a tuple (i.e. an entity or a relationship) satisfying the condition is retrieved. When given an entity relation, the condition contains the name of the entity, which is to be retrieved. When given a relationship relation, the condition contains one or two names (keys) and all tuples, containing them in their composed key, are retrieved. If one entity name is specified, then all relationships of this entity are traced; if two entity names, then the relationship between these entities is processed. The statement has the following format:

$$PREDICATE\ \ R(A_1, A_2, \ldots): ID_1 = X[, ID_2 = Y]$$

where ID_1, ID_2 - specify the condition;

X, Y - names, used in the condition.

For example, if having a relationship "R_{12}" between the entity classes "C_1" and "C_2" with relationship attributes "$A_1, A_2, \ldots, A_{11}$", the relationship between two entities with keys "$NAME_1$" and "$NAME_2$" can be traced using the following statement:

$$PREDICATE\ R_{12}(A_1, A_3, A_4): ID_1 = NAME_1, ID = NAME_2$$

and as a result the relationship attributes "A_1, A_3, A_4" will be retrieved.

Finally, two features of the two statement will be mentioned:

- by means of the attribute list the programs are able to specify only the attributes they need, not all of the relation attributes;
- the list of the attributes can comprise only attributes, belonging to the corresponding relation. This restriction means that the list can be only a subset of the relation.

V. DD DBMS DESIGN

The main feature of the DBMS discussed is the dynamic data independence, so the DBMS design will be presented having in mind primarily this aspect.

Dynamic data independence means that a run time the DBMS performs a mapping between the three schemas: external, logical and integral. This requires that the three schemas should be *interpreted* at run time, when calling the DBMS. In this way they can be modified at any time without amending the programs using them. On the other hand interpretation of the schemas requires their storing in object format, as *coded tables*.

We already discussed the way the programs provide their view of data (i.e. the external schema) in the statements FOR and PREDICATE. This allows an easy transformation of these operators into coded tables to be done at compilation or at run time. Now we will discuss the way the logical and physical schemas are formed as coded tables in order to allow their interpretation.

The logical schema, describing the logical structure of the DD information, is built upon three types of tables:

(1) Entity Structure Table (EST). A separate table is maintained for each entity class. This table describes

its contents of attributes, their logical sequence, and the key used to identify the entities.

(2) Relationship Table (RT). A table is provided for every couple of entity classes a relationship between which is allowed. This table contains the keys of the classes (usually the entity names) and the contents and the logical sequence of the relationship attributes (intersection data).

These tables correspond to the relations, describing the program view of data (external schema) in the statements FOR and PREDICATE and allow specifying of the programmer's information needs in the data manipulation statements.

(3) Attributes Table (AT). This table describes each entity or relationship attribute: format-variable or fixed, length, type-numerical or coded, etc.

The internal schema is presented by means of the Physical Organization Table (POT). It describes the physical data structure and organization: storage allocation, block and record length, addressing parameters, etc.

The DBMS architecture and the algorithm of logical and physical mapping are shown in Fig. 1. The sequence of the algorithm actions, represented as circle numbers, is the following:

(1) The Application program calls the DBMS, providing the following information in the data manipulation statements:

- the type of the statement: FOR or PREDICATE;
- the type of the relation. We will note that from program point of view this type specifies an entity class or a relationship between two classes;

- the contents and the sequence of the attributes needed;
- the condition for the PREDICATE operator;
- the work area, in which the data are to be moved.

(2) The Logical Mapping Processor reads the table, corresponding to the specified relation. From this table it derives the required attributes within the logical sequence of the attributes, composing the table.

(3) The Logical Mapping Processor obtains from the attributes table the format and the length (if varied) of each attribute, specified by the application program.

(4) The Logical Mapping Processor calls the Physical Mapping Processor, providing it with the information thus obtained.

(5) The Physical Mapping Processor derives from the POT information about the physical location of the block, containing the required data, say volume, file relative block address, etc.

(6) The Physical Mapping Processor calls the I/O Processor, providing it with the physical address of the necessary block.

(7) The I/O Processor reads the block from the metadatabase into the buffer pool.

(8) The Physical Mapping Processor, using the information about the attributes format and length and the records blocking, performs the following:

- locates the position of the required attributes within the block;
- moves them into the work area of the application program in accordance with the sequence specified by it.

VI. RESULTS

The so designed DBMS allows dynamic performance of the following changes, concerning the data structure and organization, without revision of the application programs which relied on the previous structure:

(1) Changes of the logical structure. They include changes in the following:

- entity contents and structure. For example, adding of attributes or changing their order are allowed. This affects only the corresponding entity structure table;
- relationship between entities (e.g., the relationship between two classes can be deleted). This results in the modification of the corresponding relationship table;
- attribute format, e.g., changing of the attribute length or type (fixed or variable), etc.

The logical dynamic independence permits the insulation of the programs, accessing the DBMS, from changing information requirements (these changes can be transparent at the programming level). This flexibility greatly simplifies the program maintenance.

(2) Physical organization modification. This includes changes in addressing parameters, record and block length, storage allocation, etc. These changes allow tuning in accordance with the DBMS usage in order to improve its performance.

Dynamic data independence thus achieved provides to additional advantages:

- flexibility. The application programs access the DBMS in a way, independent of the data structure and organi-

zation, so their algorithm is not tied to the particular data representation. This reduces the efforts needed when changing the programs;

- simplicity. The interface to the DBMS represents the data in a table-oriented way, using some ideas of the relational data model. This simplifies application program development and maintenance.

Generally, dynamic data independence maintenance has two drawbacks: first, maintenance of tables with data description, and second, run time overhead due to the interpretation of these tables. In our case, the choice of dynamic data independence is justified. The first disadvantage can be compensated since the DD functional software needs such tables in order to process the DD information in a unified way, regardless of its type. The second drawback - the overhand, is not so significant because the volume of the DD information is not too great.

VII. CONCLUSION

This paper discusses the way a DBMS of a Data Dictionary was designed to support dynamic data independence. The experience is gathered during the design and implementation of the PLUS program development environment (though integrated with the PLUS complex, the DD can be used independently). The dynamic data independence, embedded inthe DBMS of the Data Dictionary, greatly reduces the efforts for integrating the DD with the PLUS components. The experience and the results achieved can be used in all Data Dictionaries having an extensibility feature or integrated with a program development environment. Generally, the approach is applicable not only to DBMS of Data Dictionaries, but also to any DBMS with dynamic data independence as a key requirement.

REFERENCES

1. Shoshlekov, I., PLUS language in the data base environment, Proceedings of the sixth international seminar on Data Base Management Systems, Hungary, 1983.

2. Bukowski, N., Metadata Handling in the PLUS Complex, Proceedings of the sixth international seminar on Data Base Management Systems, Hungary, 1983.

3. Martin, J., Computer Data Base Organization, Prentice-Hall, Inc., New Jersey, 1975.

4. Schmidth, J.W., Some high level language constructs for data of type relations, ACM Transactions on Data Base Systems, Vol. 2, No. 3, September 1977.

5. Beck, L.L., A Generalized Implementation Method for Relational Data Sublanguages, IEEE Transactions of Software Engineering, Vol. SE-6, No. 2, March 1980.

Dinamikus adat-függetlenséggel kapcsolatos tapasztalatok

N. BUKOVSKI

Összefoglaló

A cikkben a szerző bemutatja, hogyan lehet a dinamikus adat-függetlenség fogalmát felhasználni az Adat Szótárakkal kapcsolatos Adatbázis Kezelő Rendszerekben.

Опиты касающиеся динамической независимости данных.

Н. Буковски

Резюме

Показывается возможность исползования понятия независимости данных для систем обработки баз данных связанных с словарями данных.