Optimizing data search and retrieval in a semistructured database system

Amparo López Gaona and[1] and Andrés López Capistrán[2]

Facultad de Ciencias, Universidad Nacional Autonóma de México, México D.F. 04510

Abstract. The semistructured data model implemented in a database allows management data without any restriction to its structure. This freedom offers the possibility to store data (like XML) in a more natural way, instead of trying to adapt other models to handle this kind of data. However, data searching is a hard task because it has to examine a great amount of the database before being able to determine it has found the relevant data, this is due to the lack of knowledge about the data structure; this task may be too expensive. In this work we present a technique that is less expensive for the system, this technique is based in a dynamical scheme (called data summary), that reflects the structure without any data repetition and the scheme has a direct link to the data in such way that retrieving the relevant data is a simple and efficient task.

Keywords: semistructured data, query optimization, data summary.

1 Introduction

Today, there exists a vast amount of techniques aimed to ease query execution in DBMS. This systems are so complex that they are separated in several components, each having their own optimization techniques. In this work we describe the techniques used in the so called query processor, which its responsabiliy is to translate from a query statement to the sequence of basic operations of the internal storage for the database.

There are several types of optimization in the query processor, spanning from the logical level to physical level. Some algebras have been defined for handling semistructured data and XML [2, 1] and there are some other techniques at the logical level; this techniques are adapted from the well known "Push Selections Down" technique of the relational algebra [4].

At the *Lore* project [7, 9, 6] has been proposed several optimization techniques which are applied in the query processor; however, these optimizations are based on data access statistics, besides, their logical plan as well as the physical one are so related to the basic operations of the internal storage of its DBMS, that makes it difficult to adapt their techniques to a different DBMS, particularly those not using the same basic operations for the internal storage.

In this work we propose optimization techniques at the physical level, our techniques are directed towards data search and retrieve form a semistructured data (SSD) source like XML: We base our optimizations on the automata theory [5], specifically we use the transformation of a Nondeterministic finite automata (NDFA) to a deterministic finite automata (DFA), adapting that algorithm for the transformation to the SSD model.

2 Information search and retrieve in a semistructured data source

The SSD model has been defined [11] as a directed graph with labels on its edges and a special node, the root, which is used as an starting place for information searching. We use path expressions in our query language [3] as the mechanism that allows us to describe where the data of interest are, this data will be used in a query. Path expressions are defined as:

Definition 1. A *path expression* is a sequence of edge labels $l_1.l_2.l_3\dots l_n$ where each l_i with i=1...n is and edge label in the semistructured data graph. The result of a path expression $l_1.l_2.l_3\dots l_n$ applied on a data graph, is a collection of nodes v_n such that exists edges $(l_1, r), (r, l_2, v_2), (v_2, l_3, v_3), \dots, (v_{n-1}, l_n, v_n)$ where r is the root.

In order to make every graph's root content in a semistructured database, it is necessary that a labeled edge aiming towards the root exists. That edge's label will be the name of the graph it aims to. A graph with this characteristic is known as a *ssd-Table* in the definition of the query language *Ssquirel* [3], and a set of *SSD-Tablas* defines a semistructured database.

Secondary storage (disk) access are tended to be considered as the main unit to measure the database performance, because these actions are the ones who most slow down a query execution, something that is caused by the access time, search time, etc., the hard discs have. More than other mechanism used within a query in *Ssquirel*, path expressions are those who most use secondary storage access.

In order to show the excessive amount of disk access required to evaluate a path expression (without using any optimization technique), next it's shown a path expression evaluation: `Inventory.building.floor.equipment.desktop.mouse` over the database in figure 1, which was specifically designed to demonstrate clearly the big number of access needed to found path expression's result.

If it is considered that the *ssd-Table* "Inventory" has 150 buildings, each with an average of 3 floors and 10 computers per floor, there are 4500 desktops, where only 300 have mouse's information. We have to take in account that for each time children's retrieve is needed, a access to disk is taken. Therefore, to evaluate this path expression the following amount of access is needed:

- *1* access to retrieve the *SSD-Tabla* "Inventory" id.
- *1* accesses to retrieve the buildings.
- *150* accesses to retrieve the floors.
- *450* accesses to retrieve the equipment.

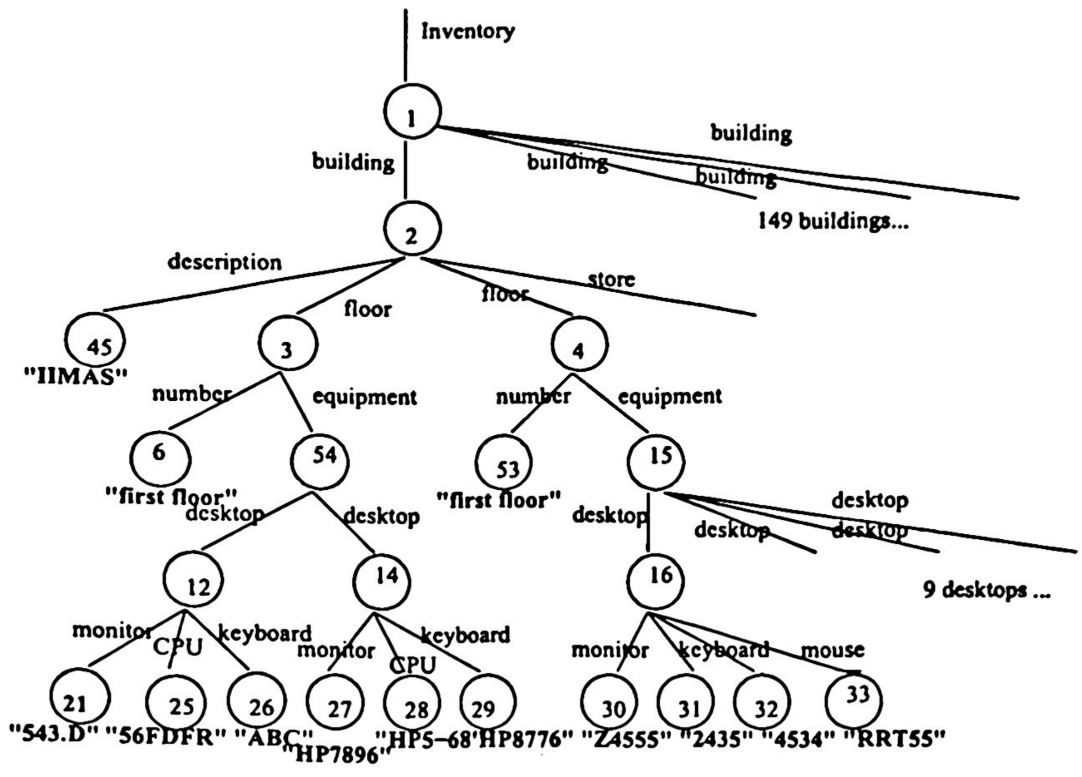

Fig. 1. Example semistructured database

- *450* accesses to retrieve the desktops.
- *4500* accesses to retrieve the mice of each desktop.

Total: *5552* accesses.

From the last 4500 accesses, only 300 will be really useful, in other words, there is a 93% of accesses wasted.The main reason of this to happend is that the DBMS can not determine which desktop doesn't have its mouse registered, know it is only possible after having tried to retrieve it.

Due to the above mentioned, a mechanism to search an retrieve information that could reduce the number of access in the process was needed.

3 Optimization techniques

In order to reduce the number of access to disk needed when searching data, we describe a technique that puts together all different paths in a data graph. Next, we'll see how to retrieve the collection of node that represent a path expression's result using this technique. First we will describe the main structured used by this technique.

Definition 2. A *path of labels* is a sequence of edge labels $l_1.l_2 \ldots l_n$ where each l_i with i=1... n is a edge label within the semistructured data graph, and for each couple $l_j.l_{j+1}$ with j=1... n-1 exists a connection between them with a node.

Definition 3. A *data summary* for a *ssd-Table* T is a directed graph G with labeled edge, where for each path of labels in T, exits an equivalent path in G, and for each path of labels in G, exists at least one in T.

For a *ssd-Table* sharing information with another, the *data summary* will consider every *path of labels* that can reach a node, starting in the *ssd-Table*'s root. We have to consider if the source *ssd-Table* where we are going to construct the *data summary* doesn't have any path with similar labels (this happens when the *ssd-Table* is already a DFA), this could made the *data summary* equal or bigger than the *ssd-Table* itself.

3.1 Expanded path expressions optimization

Definition 3 opens a possibility to check if a path expression will return data at compile-time. Furthermore, it is possible to expand those parts in a path expression that use wild-cards (character # in Ssquirel) or use closure operators (Kleene star "+", optional "?", repeatable "+"). This technique is based in the one described in [10]. It only considers paths that it is known will return data. The optimization is defined as follows:

Optimization I. Rewrite queries' path expressions replacing those parts that use close operators (*,+,?), at label level, with an equivalent that is found traversing the original path expression over the *data summary.*

For example, the path expression: `Inventory.#*.monitor` would be expanded to `Inventory.building.floor.equipment.desktop.monitor`

By doing this, the DBMS will not traverse those branches that will not reach the end of the path expression defined before. This optimization is made at compile-time.

3.2 Search data optimization

For path expression that do not use closure operator or wild-cards, we haven't defined yet a mechanism powerful enough to help the retrieve of information. The *data summary* is extended by adding more information about data.

Definition 4. A *key-valued data summary* for a *ssd-Table* T is a directed graph G with labeled edges, where for each label path in T exists one equal in G, and for each label path in G exists at least one in T; also, for each node in G it is bound a node collection of all SSD reachable from T's root through L, where L is the label path described from G's root to that node.

With a *key-valued data summary* is possible to get the node collection needed from a path expression. This is done traversing the *key-valued data summary* instead of the original *ssd-Table* graph. The optimization is defined as follows:

Optimization II. For all path expression $l_1.l_2 \dots l_n$ where l_1 is a *ssd-Table* name storage in the database, and it has, at the evaluation moment, a *key-valued data summary,* the node collection (if there is) that it returns is retrieved traversing the *key-valued data summary.*

There are to ways to bound the node collections reached by each node within the *key-valued data summary*:

- Using a special primitive data type which will hold all the above information, linking each of them with a labeled edge `Data-`. (`Data-` is a label reserved to this special link).
- Using edges with special labels `Data-` between original data and the *key-valued data summary*.

Both choices do work, but, the last one has a characteristic that makes it easier to keep update, because the original data and the *key-valued data summary* are connected. One example for this connection is shown in the figure 2.

A similar *key-valued data summary* is used as an index called "vpath" in [8]. Nevertheless, the connection between data (*ssd-Table* and *summary*) is not defined as the one presented here, something that makes the "vpath" difficult to keep updated.

Using this optimization technique, the node retrieval from the path expression before mentioned `Inventory.building.floor.equipment.desktop.mouse` is done as follows:

- *6* access to traverse the *key-value data summary* following the path describe by the path expression.
- *1* access to retrieve children connected with edges labeled as `Data-` from the node recovered from the last point.

The number of access was reduced from 5552 to only 7, which is very noticeable in this example due to the database's structure, which it was defined as it to help to visualize clarity the effect of this optimization.

3.3 Building a Data Summary

Building a *data summary* is similar to convert a NDFA to a DFA. There is a very well known algorithm [5] that performs this action. It can be adapted to the Semistructured model. To build it, each data graph's node is taken as a automata's state and every data graph's edge as a transition, taking edge's label as the transition character. To show this algorithm is beyond this paper and its implementation depends directly on how as the semistructured data model constructed.

It is important to point out one implementation detail about this algorithm. The NDFA to DFA conversion algorithm uses a search looking for a state set already built. This make the algorithm to be a exponential complexity one. This kind of algorithms are not desirable to be used in a DBMS, because a DBMS handle a high number of nodes, making the build of the *data summary* something very expensive. Thus, a good searching technique in this point is needed.

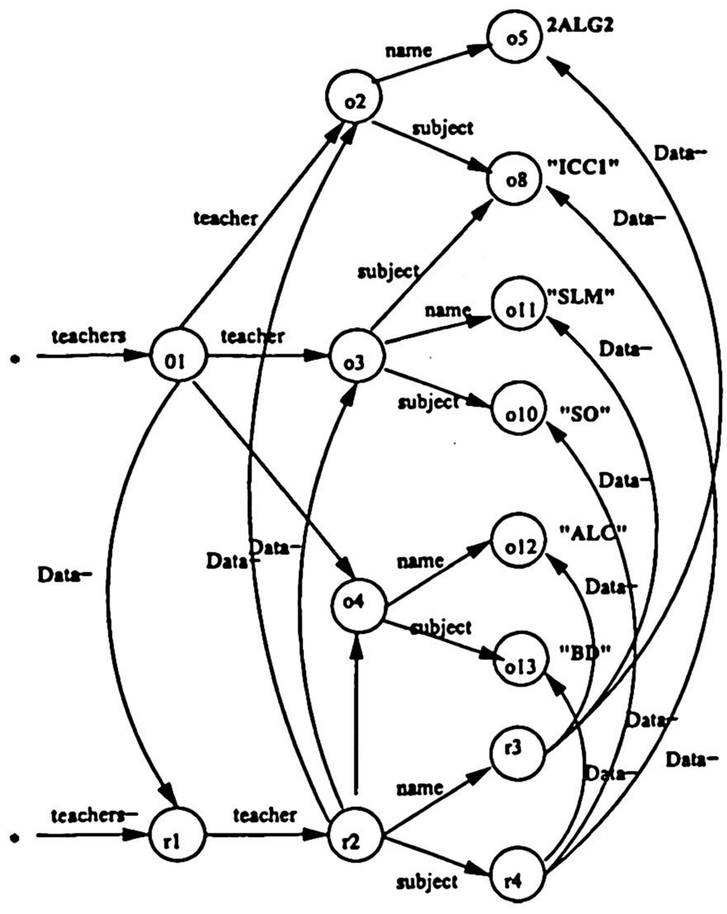

Fig. 2. Teachers *ssd-table* with its *key-valued data summary.*

4 Performance testing

All previously described concepts have been implemented inside of a query processor as part of a Semistructured Database Manager System developed at the Science School, UNAM, which has made possible to carry on some impact testing that the execution of the query experiments whether using a *data summary with identifiers.* We have the following conclusions:

- In the worst case, to use the *key-valued data summary* implies traverses one extra edge at execution time.
- In average, for big databases, there is a 25% execution time saved using the *key-valued data summary.*
- For small databases (between 1 and 100 nodes) there is not a clear difference using or not the *key-valued data summary* in the execution time.
- The required time to check which path expressions can start from the *data summary* does not cause a significant waste of time.

Thus, the use of a *key-valued data summary* in a big database is incomeproduced.

5 Conclusions

Through this paper, we have described how it is possible to reduce the number of access to disk needed to recover data from a semistructured data source. We focused directly in path expressions, because they provide to the queries the mechanism to access semistructured data. As we showed in the inventory's example, the DBMS needs to traverse all data graph branches to find the interest data. This happens because the DBMS doesn't have any knowledge about which branches will fulfill the complete path described by the path expression. With each branch needed to be traverse, the number of access to disk raise. Thus, a technique to optimize the traverse of semistructured data was needed.

In order to make the DBMS not to consider those branches that will not return any data, we proposed another structure that concentrates all different paths that exist within the original data graph, this structure is called *data summary*. At the beginning, this summary is used to discriminate path expressions that will not return data. Nevertheless, making a connection between the *data summary* and the data itself, it is possible to recover the interest data traversing the summary instead of the original data graph. By doing this, a greater number of access is saved, because the DBMS traverses a direct path without crossing any branch through it.

There are still other parts within a query to be explored that could get a benefit with the assistance of the *data summary* in the semistructured data environment. One of them would be to use the optimization II technique in a way in which will return the largest amount of data and out of these get those others related, something like reordering the join operators in relational algebra.

References

1. Beeri, C, Tzaban, Y. SAL: An Algebra for Semistructured Data and XML. In Proc. ACM SIGMOD Workshop on The Web and Databases (WebDB'99), pp.37-42. (1999)
2. Frasincar F., Houben G., Pau C. XAL: An Algebra for XML Query Optimization. In Database Technologies 2002, Thirteenth Australasian Database Conference, volume 5 of Conferences in research and Practice in Information Technology, pages 49-56. Australian Computer Society Inc. (2002)
3. Egar García-Cárdenas, Amparo López Gaona, Salvador López Mendoza. SSquirel: un lenguaje de consulta para bases de datos semiestructurados. 6° Workshop Iberoamericano de Ingeniera de Requisitos y Ambientes Software IDEAS'2003. pp 322-327. ISSN: 84-96023-05-2. Mayo 2003.
4. Garcia-Molina H., Ullman J.D., Widom J. Database System Implementation. Prentice Hall (2000)
5. Hopcroft, J.E., Motwani R., Ullman J.D. Introduction to automata languages, and computation. $2^{n}d$ Edition. Prentice-Hall. (2000)
6. McHugh J.G., Abiteboul S., Goldman R., Quass D. and Widom J. Lore: A database management system for semistructured data. SIGMOD Record, 26(3):54-66. (1997)
7. McHugh J.G. Data Management and Query Processing for Semistructured Data. Universidad de Stanford. (2002)

8. McHung J.G., Widom J.. Query Optimization for Semistructured Data. Technical Report, Stanford University. (1997)
9. McHung J.G., Query Optimization for Semistructured Data. Technical Report. Stanford University. (1997)
10. McHung J.G., Widom J. Compile-Time Path Expansion in Lore. In Proceedings of the Workshop in Query Processing for Semistructured Data and Non-Standard Data Formats, Jerusalem, Israel. (1999)
11. Suciu D. An overview of semistructured data. SIGACT New, 29(4):28-38. (1998)