Implementation and Selection of Derived Partitioning for Optimizing the Performances of Relational Data Warehouses

Michel Schneider, Hervé Lorinquer
LIMOS, Blaise Pascal University
63177 AUßIERE CEDEX (France)
Tel.: 33 4 73 40 50 09, Fax 33 4 73 40 50 01
michel.schneider@isima.fr, herve.lorinquer@isima.fr

1. INTRODUCTION

Three main techniques are available in RDBMS for optimizing the performances of databases in general and data warehouses in particular: indexes including join indexes, materialized views, partitioning.

Indexes can contribute largely to accelerate the execution time of requests, particularly when the number of values in the columns to be indexed is large. However, indexes need space to be installed; construction time and update time can be long. In the context of data warehouses, the traditional B-tree indexes are not well appropriate. Bitmap indexes offer better performances when the column of indexing has a low cardinality but their update is more expensive. Many variants of indexes were studied [18]. Most interesting are the join indexes [19]. Algorithms of index selection in a data warehouse context are proposed in [7], [8], [10], [13].

A materialized view constitutes an extremely effective way to minimize the execution of a query. If a query is completely materialized, its execution cost is strictly reduce to the reading of the view tuples. However a view presents several drawbacks. Like indexes, it needs space to be materialized. Construction time and update time can also be long. Relational DBMS generally authorize incremental update (also called fast refresh), but with very significant practical restrictions. Very often, the only possibility is to entirely recompute the view. Many algorithms were proposed for the selection of views to optimize a set of queries: algorithms without constraints [2], [21], algorithms directed by the space constraint [11], algorithms directed by a time maintenance constraint [12], [15], [22]. The simultaneous selection of a set of views and indexes is considered in various ways in [1], [3], [5], [9], [14], [20].

Partitioning is a well-known data base technique used for different purposes: to manage large sets of tuples or occurrences, to control distribution of data, to permit parallel execution, to improve the response time of requests. It was the subject of many studies, in particular in the context of object oriented data bases [4], [6], [16]. One distin-

2. ABSTRACT

Many works have been done to optimize the performances of relational data warehouses. Three main techniques are available in relational DBMS or in specific systems: indexes including join indexes, materialized views, partitioning. Each one can be used alone, or combined with the others. In this paper we will consider more precisely the partitioning technique. In general, only the direct partitioning of a relation (i.e., according to the attributes of this relation) is possible. However, in the context of a warehouse, it would be interesting to be able to implement the derived partitioning which consists in partitioning a relation according to the attributes of another relation referenced by the first. We show how to implement a derived partitioning with a relational DBMS and we establish its interest through several experiments.

In a data warehouse context, join indexes have almost the same objective as derived partitioning: accelerating requests with selection predicates involving columns of dimension tables. But surprisingly, these two techniques have not been compared systematically. We propose in this study a solution to implement derived horizontal partitioning and we show on a benchmark its interest relatively to join indexes. In this study we will consider the updates because they can influence significantly the behavior of the users.

In the context of relational warehouses, horizontal partitioning is more especially interesting. It consists in distributing the tuples of a relation in different physical zones on the disk. Editors of relational DBMS introduced it with the objective to facilitate the management of the fact table which is often very bulky. In particular, it makes it possible to conveniently manage the addition of new tuples in the fact table. Partitioning is then carried out according to the values of an attribute TIME of type DATE. Each partition corresponds to a range of values for TIME. The new tuples are inserted into a new partition (with the most recent dates). At the same time, to avoid a continual growth of the fact table, the oldest partition is destroyed. Horizontal partitioning can also be used for the optimization of requests. For example let us consider a request on the fact table involving a selection predicate with a coefficient of selection of 10% (10% of the tuples satisfy the predicate). By partitioning the table in two partitions P1 and P2 (P1 containing these 10% of tuples and P2 the 90% remaining), to solve the query it is enough to read the tuples of P1. The execution of the query is thus divided (approximately) by 10.

Commercial RDBMS offer varied facilities of horizontal partitioning. ORACLE, for example, allows partitioning of a table in RANGE mode and in HASH mode. The RANGE mode consists in defining the partitions by intervals of values on one or several columns of the table. HASH mode allows to place the tuples in a partition according to the result of a hash function (provided by the system) applied to the values of the partitioning columns. These two modes can be combined to benefit from the advantages of the one and of the other.

Derived horizontal partitioning consists in carrying out the horizontal partitioning of a relation by using the attributes of another relation to which the first is connected by referencing. Derived partitioning is particularly interesting in the context of data warehouses because most of the requests consist in aggregating attributes of the fact table with conditions of selection involving attributes of the dimension tables. Note that commercial RDBMS do not offer facilities to specify directly derived partitioning.

In our data warehouse context, join indexes have almost the same purpose as derived partitioning: accelerating requests with selection predicates involving columns of dimension tables. But surprisingly, these two techniques have not been compared systematically. We propose in this study a solution to implement derived horizontal partitioning and we show on a benchmark its interest relatively to join indexes. In this study we will consider the updates because they can influence significantly the behavior of the users.

Copyright © 2003, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.
The paper is organized as follows: in sections 2 and 3 we present respectively the benchmark and the requests we will use for the experiment, in section 4 we explicit our solution to install a derived partitioning, in section 5 we discuss the results of the experiments, in section 6 we conclude and we draw some perspectives.

2. THE BENCHMARK WAREHOUSE

The benchmark warehouse we use is generated from the generic APB1 benchmark [17]. The schema has a star configuration [19] and comprises the fact table Actvars and four dimension tables: Prodlevel, Custlevel, Timelevel, Chanlevel. This warehouse has been populated using the generation module of APB1. Attributes of each table, number of tuples, number of distinct values for each attribute are given in Figure 1. Each dimension table can be joined with the fact table through its first attribute.

This warehouse has been installed with ORACLE 9i on a Pentium IV 1.5 Ghz microcomputer (with a memory of 256 Mo and two 7200 rps 60 Go disks) running under Windows 2000 Pro.

3. THE QUERIES AND THE UPDATES

For our experiments, we consider five queries Q1 to Q5 with one join and one predicate, and three queries Q6 to Q8 with two joins and two predicates (Figure 2). A join is always between the fact table and one dimension table. Each predicate is associated to a join and involves an attribute of the corresponding dimension table. We have chosen the joins and the predicates in order to have very different levels of selectivity. Since an attribute in the predicates can take several different values, each of the queries Q1 to Q8 is parameterised. This means that a user can execute the query with any of these values. So a parameterised query defines a set of potential queries. There is a potential query for each value.

Figure 1: The star warehouse used for the experiments

<table>
<thead>
<tr>
<th>table</th>
<th>attribute</th>
<th>number of distinct values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actvars</td>
<td>Product_level</td>
<td>6500</td>
</tr>
<tr>
<td></td>
<td>Customer_level</td>
<td>640</td>
</tr>
<tr>
<td></td>
<td>Time_level</td>
<td>517</td>
</tr>
<tr>
<td></td>
<td>Channel_level</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>UnitsSold</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>DollarSales</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>DollarCost</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prodlevel</td>
<td>Code_level</td>
<td>9000</td>
</tr>
<tr>
<td></td>
<td>Class_level</td>
<td>605</td>
</tr>
<tr>
<td></td>
<td>Group_level</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>Family_level</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>Line_level</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Division_level</td>
<td>4</td>
</tr>
<tr>
<td>Custlevel</td>
<td>Store_level</td>
<td>900</td>
</tr>
<tr>
<td></td>
<td>Retailer_level</td>
<td>99</td>
</tr>
<tr>
<td>Timelevel</td>
<td>Tid</td>
<td>730</td>
</tr>
<tr>
<td></td>
<td>Year_level</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Quarter_level</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Month_level</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Week_level</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>Day_level</td>
<td>31</td>
</tr>
<tr>
<td>Chanlevel</td>
<td>Base_level</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>All_level</td>
<td>9</td>
</tr>
</tbody>
</table>

Figure 2: The queries used for the experiments

<table>
<thead>
<tr>
<th>SQL formulation</th>
<th>Predicate selectivity</th>
<th>Number of tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To have a complete view on the performances of the two techniques, it is important also to measure times for the updates. For this purpose we consider two situations UD (Update each Day) and UW (Update each Week) materializing the deletion followed by the insertion in the fact table of a number of tuples corresponding to the activity of one day (52 071 tuples) and one week (364 500 tuples). In each case we measure the time the system uses to make the operation.

4. IMPLEMENTING THE DERIVED HORIZONTAL PARTITIONING

Since the selection predicates in the queries Q1 to Q8 are expressed on attributes of dimension tables, it is not possible to used directly these attributes to specify the partitioning of the fact table. So, for each predicate pi we introduce in the fact table a column dp, whose integer value depends on the value of the corresponding attribute in the dimension table. To be more precise let us consider the case of query Q1. Its predicate p1 involves the attribute division_level which can take one of four values. So dp1 takes a value 1 to 4 depending on the value of division_level.

Using the dp, we can then specify the desired partitioning in one of the three modes: range or hash or hybrid. The queries must be manually rewritten in function of the dp in order to permit the system to take advantage of the partitioning.

Note that insertion operation of a tuple is slightly complicated. We must first determine through a join the value of each the dp in order to permit the system to place the tuple in the correct partition.

This implementation needs extra space for the dp. It uses also extra time for the updates.

5. EXPERIMENTAL RESULTS AND COMMENTS

In order to situate the interest of the derived partitioning we have made three series of experiments:
- one without optimization:
- one with a partitioning which depends of each query:
- one with join indexes.

Copyright © 2003, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.
Concerning our consolidated times, it appears that partitioning gives always the better results for the two situations a=25 and a=50. It is easy to see that join indexes are profitable only for small values of a (less than 5), i.e., when queries have about the same frequencies as the updates.

From these experiments we can deduce the following pragmatic rules:

Rule 1 : Select a partitioning relatively to a parameterised query if the selectivity for this query is low or if the frequency of the update is low compared to the one of the query.

Rule 2 : Select a join index relatively to a parameterised query if the selectivity for this query is high or if the frequency of the update is about the same as the one of the query.

6. CONCLUSION

The objective of this paper was to suggest an implementation and to explore the performances of horizontal derived partitioning.

Compared to join indexes, horizontal derived partitioning offers better performances for query time, especially when the selectivity of the selection predicates is low. With regard to the updates, they are less interesting, primarily when the number of partitions is high. When updating and querying interleave, a partitioning on an attribute A with different values is advantageous as soon as a parameterised query on A is executed more than 0.05*n times between two updates.

This work shows that the two techniques are rather complementary. There is thus interest to use them jointly as it had been already underlined through a theoretical model [6]. It would thus be necessary to design algorithms for combined selection. A realistic cost model would be necessary.

REFERENCES


[12] H. Gupta and I. S. Mumick, “Selection of views to materialize under a maintenance cost constraint,” in Proc. 8th Int. Conf. on Data-


Related Content

Massive Open Online Courses and Integrating Open Source Technology and Open Access Literature Into Technology-Based Degrees
www.irma-international.org/chapter/massive-open-online-courses-and-integrating-open-source-technology-and-open-access-literature-into-technology-based-degrees/184486

Project Control Using a Bayesian Approach
www.irma-international.org/chapter/project-control-using-a-bayesian-approach/184268

Particle Swarm Optimization from Theory to Applications
www.irma-international.org/article/particle-swarm-optimization-from-theory-to-applications/197378

A New Heuristic Function of Ant Colony System for Retinal Vessel Segmentation
Ahmed Hamza Asad, Ahmad Taher Azar and Aboul Ella Hassanien (2014). International Journal of Rough Sets and Data Analysis (pp. 15-30).
www.irma-international.org/article/a-new-heuristic-function-of-ant-colony-system-for-retinal-vessel-segmentation/116044

Online Information Retrieval Systems Trending From Evolutionary to Revolutionary Approach
www.irma-international.org/chapter/online-information-retrieval-systems-trending-from-evolutionary-to-revolutionary-approach/184161