IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4528

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

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

## 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.*

## 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 distinguishes vertical partitioning, horizontal partitioning or mixed partitioning.

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 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.

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 | attribute | number of distinct values |
|---|---|---|
| Actvars (24 786 000) | Product_level | 6500 |
| | Customer_level | 640 |
| | Time_level | 517 |
| | Channel_level | 9 |
| | UnitsSold | - |
| | DollarSales | - |
| | DollarCost | - |
| Prodlevel (9 000) | Code_level | 9000 |
| | Class_level | 605 |
| | Group_level | 300 |
| | Family_level | 75 |
| | Line_level | 15 |
| | Division_level | 4 |
| Custlevel (900) | Store_level | 900 |
| | Retailer_level | 99 |
| Timelevel (24) | Tid | 730 |
| | Year_level | 2 |
| | Quarter_level | 4 |
| | Month_level | 12 |
| | Week_level | 52 |
| | Day_level | 31 |
| Chanlevel (9) | Base_level | 9 |
| | All_level | 9 |

*Figure 2 : The queries used for the experiments*

| | SQL formulation | Predicate selectivity | Number of tuples |
|---|---|---|---|
| Q1 | SELECT code_level, sum(dollarsales), sum(UnitsSold)<br>FROM actvars, prodlevel WHERE product_level = code_level<br>AND division_level = 'OZQBQEJCJ14V'<br>GROUP BY code_level; | 1/4 | 2 400 |
| Q2 | SELECT tid, sum(dollarsales)<br>FROM actvars, timelevel<br>WHERE time_level = tid<br>AND month_level = '01'<br>GROUP BY tid; | 1/12 | 62 |
| Q3 | SELECT code_level, sum(dollarsales), sum(UnitsSold)<br>FROM actvars, prodlevel WHERE product_level = code_level<br>AND line_level = 'IW8A44CP8JU3'<br>GROUP BY code_level; | 1/15 | 603 |
| Q4 | SELECT code_level, sum(dollarsales), sum(UnitsSold)<br>FROM actvars, prodlevel WHERE product_level = code_level<br>AND family_level = 'PASN4MAJI2OI'<br>GROUP BY code_level; | 1/75 | 121 |
| Q5 | SELECT code_level, sum(dollarsales), sum(UnitsSold)<br>FROM actvars, prodlevel WHERE product_level = code_level<br>AND group_level = 'P5FOJ5DXP9SF'<br>GROUP BY code_level; | 1/300 | 29 |
| Q6 | SELECT code_level, sum(dollarsales), sum(UnitsSold)<br>FROM actvars, timelevel,prodlevel<br>WHERE time_level = tid AND product_level = code_level<br>AND division_level = 'RLIOT0T4TER5'<br>AND month_level = '01'<br>GROUP BY code_level; | 1/4<br>1/12 | 1 621 |
| Q7 | SELECT code_level, sum(dollarsales), sum(UnitsSold)<br>FROM actvars, timelevel,prodlevel<br>WHERE time_level = tid AND product_level = code_level<br>AND month_level = '01'<br>AND line_level = 'HSLNCSTKB22Y'<br>GROUP BY code_level; | 1/12<br>1/15 | 430 |
| Q8 | SELECT code_level, sum(dollarsales), sum(UnitsSold)<br>FROM actvars, timelevel,prodlevel<br>WHERE time_level = tid AND product_level = code_level<br>AND month_level = '01'<br>AND family_level = 'B7VRJFFXA6D5'<br>GROUP BY code_level; | 1/12<br>1/75 | 117 |

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 $p_i$, we introduce in the fact table a column $dp_i$ 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 $p_1$ involves the attribute division_level which can take one of four values. So $dp_1$ takes a value 1 to 4 depending on the value of division_level.

Using the $dp_i$, 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 $dp_i$ 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 $dp_i$ in order to permit the system to place the tuple in the correct partition.

This implementation needs extra space for the $dp_i$. 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.

*Figure 3 : The results of the experiments and the calculated values of the consolidated times*
*(respectively without technique, with partitioning, with join indexes)*

| | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 |
|---|---|---|---|---|---|---|---|---|
| Query time (s) | 106 | 61 | 63 | 53 | 54 | 61 | 59 | 56 |
| UD (s) | 68 | 68 | 68 | 68 | 68 | 68 | 68 | 68 |
| CUD(25%) (s) | 174 | 251 | 320 | 1075 | 4118 | 800 | 2192 | 12668 |
| CUD(50%) (s) | 280 | 434 | 572 | 2082 | 8168 | 1532 | 4316 | 25268 |
| UW (s) | 75 | 75 | 75 | 75 | 75 | 75 | 75 | 75 |
| CUW(25%) (s) | 181 | 258 | 327 | 1082 | 4125 | 807 | 2199 | 12675 |
| CUW(50%) (s) | 287 | 441 | 579 | 2089 | 8175 | 1539 | 4323 | 25275 |

| | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 |
|---|---|---|---|---|---|---|---|---|
| Number of partitions | 4 (4R) | 12 (12R) | 15 (15R) | 75 (75H) | 300 (300H) | 48 (4R*12H) | 144 (12R*12H) | 900 (12R*75H) |
| Extra space (Mo) | 161 | 358 | 424 | 449 | 476 | 408 | 442 | 641 |
| Query time (s) | 53 | 9 | 13 | 4 | 1 | 11 | 2 | 1 |
| UD (s) | 69 | 70 | 70 | 88 | 112 | 86 | 105 | 135 |
| CUD(25%) (s) | 122 | 97 | 122 | 164 | 187 | 218 | 177 | 360 |
| CUD(50%) (s) | 175 | 124 | 174 | 240 | 262 | 350 | 249 | 585 |
| UW (s) | 105 | 92 | 121 | 154 | 199 | 144 | 164 | 220 |
| CUW(25%) (s) | 158 | 119 | 173 | 230 | 274 | 276 | 236 | 445 |
| CUW(50%) (s) | 211 | 146 | 225 | 306 | 349 | 408 | 308 | 670 |

| | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 |
|---|---|---|---|---|---|---|---|---|
| Join indexes | IJ1 | IJ2 | IJ3 | IJ4 | IJ5 | IJ1+IJ2 | IJ2+IJ3 | IJ2+IJ4 |
| Space for indexes (Mo) | 11 | 8 | 20 | 35 | 59 | 19 | 28 | 43 |
| Query time (s) | 152 | 24 | 57 | 26 | 8 | 19 | 9 | 3 |
| UD (s) | 69 | 69 | 71 | 69 | 68 | 68 | 71 | 69 |
| CUD(25%) (s) | 221 | 141 | 299 | 563 | 668 | 296 | 395 | 744 |
| CUD(50%) (s) | 373 | 213 | 527 | 1057 | 1268 | 524 | 719 | 1419 |
| UW (s) | 98 | 99 | 101 | 101 | 104 | 109 | 110 | 110 |
| CUW(25%) (s) | 250 | 171 | 329 | 595 | 704 | 337 | 434 | 785 |
| CUW(50%) (s) | 402 | 243 | 557 | 1089 | 1304 | 565 | 758 | 1460 |

The results are reported in the three tables (one for each series) of figure 3.

Concerning the series with partitioning, we have installed for each query a number of partitions equal to number of different values for the corresponding selection attribute. We use the range mode (R) when this number is small, otherwise the hash mode (H). Note that for the queries with two joins and two predicates, we use the hybrid mode.

Concerning the series with joins indexes, we install one index for the queries with one join and two indexes for the queries with two joins.

For each query we report the extra space used by the partitioning or the indexes, the query time (in seconds), the update time for UD and UW.

In order to situate the influence of queries frequencies, we have also calculated two consolidated times CUD(a%) (resp. CUW(a%)) for two values of a (25 and 50) and for the two cases UD and UW. Here a is the percentage of potential queries which can be posed between two updates. Recall from section 3 that there is a potential query for each value of the parameter in a predicate selection. Then CUD(a%) = 0.01*a*n*E+UD where E is the execution time for a query Q, n is the number of potential queries associated to Q, UD is the time of the update (Update each Day). In other words CUD(a%) gives the total time which is needed for executing a% of the potential queries plus the time of the update. The same holds for CUW. It is assumed that each potential query takes the same time for its execution. These consolidation times must be viewed as simple models to simulate real situations where queries and updates interleave.

We are now able to give some comments about the experiments. Concerning the query time, we observe that partitioning gives a profit even for a low selectivity. The profit is very important with a high selectivity (when the number of different values for the selection attribute is greater than 50). Note that there is an inversion for Q2 and Q3 which is explained by the size of the query result (62 tuples for Q2 and 603 tuples for Q3). Join indexes give also a profit as soon as the selectivity is sufficiently high (more than 10 different values for the selection attribute). But partitioning gives better results compared to join indexes.

Concerning the update time, join indexes are in general much more efficient than partitioning. Partitioning performs as well as indexes only for low selectivity and for daily updates.

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 n 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

[1] S. Agrawal, S. Chaudhuri, V.R. Narasayya, “Automated selection of materialized views and indexes in SQL databases”, in Proc. 26th Int. Conf. on Very Large Data Bases (VLDB), 2000, pp. 496-505.

[2] E. Baralis, S. Paraboschi, and E. Teniente, “Materialized view selection in a multidimensional database,” in Proc. 23rd Int. Conf. on Very Large Data Base (VLDB), 1997, pp. 156-165.

[3] L. Bellatreche, K. Karlapalem, and Q. Li, “Evaluation of indexing materialized views in data warehousing environments”, in Proc. Int. Conf. on Data Warehousing and Knowledge Discovery (DAWAK), 2000, pp. 57-66.

[4] L. Bellatreche, K. Karlapalem, M. Schneider and M. Mohania, “What can partitioning do for your data warehouses and data marts”, in Proc. Int. Database Engineering and Application Symposium (IDEAS), 2000, pp. 437-445.

[5] L. Bellatreche, K. Karlapalem, and M. Schneider, “On efficient storage space distribution among materialized views and indices in data warehousing environments”, in Proc. Int. Conf. on Information and Knowledge Management (ACM CIKM), 2000.

[6] L. Bellatreche, M. Schneider, M. Mohania, B. Bhargava, “Partjoin : an efficient storage and query execution design strategy for data warehousing”, Proc. Int. Conf. on Data Warehousing and Knowledge Discovery (DAWAK), 2002.

[7] S. Chaudhuri and V. Narasayya., “An efficient cost-driven index selection tool for microsoft sql server”, in Proc. Int. Conf. on Very Large Databases (VLDB), 1997, pp. 146-155.

[8] C. Chee-Yong, “Indexing techniques in decision support Systems”, Ph.D. Thesis, University of Wisconsin, Madison, 1999.

[9] S. Finkelstein, M. Schkolnick, and P. Tiberio, “Physical database design for relational databases”, ACM Transactions on Database Systems, Vol. 13, N° 1, pp. 91-128.

[10] H. Gupta et al., “Index selection for olap,” in Proc. Int. Conf. on Data Engineering (ICDE), 1997, pp. 208-219.

[11] H. Gupta, “Selection of views to materialize in a data warehouse”, in Proc. 6th Int. Conf. on Database Theory (ICDT), 1997, pp. 98-112, 1997.

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

base Theory (ICDT), 1999, pp. 453-470.

[13] Informix Corporation, "Informix-online extended parallel server and informix-universal server: A new generation of decision-support indexing for enterprise data warehouses", White Paper, 1997.

[14] W. J. Labio, D. Quass, and B. Adelberg, "Physical database design for data warehouses," in Proc. Int. Conf. on Data Engineering (ICDE), 1997, pp. 277-288.

[15] H. Mistry and al., "Materialized view selection and maintenance using multi-query optimisation", in Proc. ACM SIGMOD 2001, pp. 307-318.

[16] A. Y. Noaman and K. Barker, "A horizontal fragmentation algorithm for the fact relation in a distributed data warehouse", in Proc. 8th Int. Conf. on Information and Knowledge Management (CIKM), 1999, pp. 154-161.

[17] OLAP Council, "APB-1 olap benchmark, release II", http://www.olapcouncil.org/research/bmarkly.htm

[18] P. O'Neil and D. Quass., "Improved query performance with variant indexes", in Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 38-49.

[19] Red Brick Systems, "Star schema processing for complex queries", White Paper, July 1997.

[20] A. Sanjay, G. Surajit, and V. R. Narasayya, "Automated selection of materialized views and indexes in microsoft sql server", in Proc. Int. Conf. on Very Large Databases (VLDB), 2000, pp. 496-505.

[21] J. Yang, K. Karlapalem, and Q. Li, "Algorithm for materialized view design in data warehousing environment," in Proc. 23th Int. Conf. on Very Large Data Bases (VLDB), 1997, pp. 136-145.

[22] C. Zhang, X. Yao and J. Yang, "An evolutionary approach to materialized views selection in a data warehouse environment", IEEE Trans. on Systems, Man, and Cybernetics, Vol. 31, N°. 3, pp. 282-294.