A Maintenance Approach of a BJI Index Configuration

Saïd Taktak
University of Sfax, FSEGS Faculty, P.O.Box 1088
Miracl Laboratory
Sfax, Tunisia
Said.taktak@gmail.com

Jamel Feki
University of Sfax, FSEGS Faculty, P.O.Box 1088
Miracl Laboratory
Sfax, Tunisia
Jamel.Feki@fsegs.rnu.tn

***Abstract*— In data warehousing domain, OLAP (On Line Analytical Processing) queries are complex since they use several tables with huge cardinalities. Several optimization techniques have been studied in the literature as materialized views and bitmap join indexes (BJI). BIJ indexes are useful to pre-calculate star joins in order to reduce the execution cost. Current approaches for the selection of BJI define a configuration that optimizes a beforehand definite workload of queries. However, this workload can evolve in time and is likely to make obsolete the configuration of index created. In order to take into account the evolution of a workload of queries, we propose, in this article, a maintenance approach for the recommendation of a new configuration of indexes. Our approach starts with an evaluation of the current configuration of indexes and then adapts it to the new workload of queries with an aim of guaranteeing the stability of performances. Queries of the new workload are directly extracted from log files. Furthermore, to validate our approach, we carried out a series of experimentations on a data warehouse created with the DWEB benchmark.**

Keywords-data warehouse; bitmap join indexes; tuning.

I. INTRODUCTION

Due to the exponential increase in the volume of data, enterprises focus on the decisional information and, therefore move from a simple processing of data to a logical analysis of data. The data warehouse formalized in the early 90s by *Inmon* is the appropriate solution [1]. In fact, it is particularly designed to respond to complex decisional queries.

At the conceptual level, the data warehouse is usually modeled by a *star schema* that highlights the topic analyzed as a central *fact* (i.e., the fact table) which is composed of numerical attributes and connected to *dimensions* (i.e., dimension tables) representing the axes of analyses. The large volume of data manipulated by analytical queries (i.e., decisional) and the high number of tables to be joined raise the problem of performance [2]. In order to optimize these queries, taking intensive execution time, the data warehouse administrator (or designer) has to successfully achieve the physical design step [3]. This is why, he or she should select a set of optimization techniques that they consider pertinent to respond to the decision makers needs (i.e., expressed as a workload of queries defined beforehand). Several optimization techniques have been studied for relational data warehouses, some of which are inherited from traditional databases. A Database management system (DBMS) offers techniques such as:

- Materialized views [4] improve the execution time of queries by pre-computing the most expensive operations such as joins and aggregations. Consequently, the execution of some queries OLAP requires only the access to one materialized view instead of its original data tables.
- Fragmentation [5] allows dividing the data of a DW into multiple partitions that can be accessed separately. It can be either vertical or horizontal (by projection or selection algebraic operators).
- Parallel query processing [6]. The query is divided into components that can be treated simultaneously. The results are combined and delivered to the customer as a single component.
- Advanced indexes [7], etc.

These optimization techniques can be used in an isolated manner (i.e., selected independently) or in a combined way (by exploiting the dependencies between them). The second way provides good results, because each technique can compensate the shortcomings of others.

In the data warehouse context, the indexing technique is an important issue due to the large volume of manipulated data and to the complexity of processed queries. The bitmap join indexes (BJI) pre-compute the joins between the fact and its dimension tables. A BJI index is defined on the fact table by using the values of one or several attributes belonging to dimension tables. This increases the number of possible indexes [8]. Several research studies have proposed optimal solutions to help the data warehouse administrator to select a BJI configuration that minimizes the execution cost of a given workload of queries.

However, the task of the administrator is more complicated than that. In fact it is not sufficient to establish the appropriate optimization technique, but it is more important to adjust the use of this technique in response to the occurring evolution on the data warehouse to avoid the performance degradation. This evolution may affect three things: (1) the schema and the content of the data warehouse tables, (2) the size of the memory space allocated to the optimization techniques selected, and (3) the workload of queries on which the selection of optimization technique has been established. In this paper, we propose an approach for the maintenance of a current BJI configuration (e.g., in use) in order to face the evolution of a workload of queries. This

 ISBN: 978-1-61208-165-6

evolution may concern many aspects as the frequency of access of the queries, the addition of new queries or even the deletion of existing ones, etc.

This article is organized into six sections: Section 2 recalls the definition of the BJI and illustrates it through an example. Section 3 presents a formalization of the selection problem of BJI and explores the existing approaches of indexing and their lacks. Section 4 is devoted to the presentation of our maintenance approach. Section 5 describes a set of experiments we have done. Section 6 concludes this article and enumerates some perspectives.

II. BINARY JOIN INDEX

A binary join index BJI ("bitmap join index") allows pre-joining the fact table with its dimension tables in a data warehouse. A BJI has the same number of tuples as its fact table and as many columns as the number of distinct values of the dimension attribute on which the BJI index is built [9]. The bit at row *i* and column *j* of the BJI index is set to 1 if the i^{th} tuple of the fact table can be joined with the tuple of the dimension table that has the value of the indexed attribute (i.e., in column *j*). Otherwise, this bit is set to zero.

Fig. 1 represents the *Product_Type* BJI built on the fact table *Sales* using the *Type* attribute of the dimension table *Product.*

Dimension Table: Product

RID	PID	Label	Type
1	124	L1	T1
2	137	L2	T1
3	154	L3	T2
4	166	L4	T3
5	181	L5	T2
6	209	L6	T1

Fact Table: Sales

RID	PID	....	Amount
1	124		1213
2	137		23232
3	166		23244
4	154		4544
5	154		4544
6	166		444534
7	166		33550
8	181		6777
9	209		6555
10	209		4544

Binary Join Index

RID	T1	T2	T3
1	1	0	0
2	1	0	0
3	0	0	1
4	0	1	0
5	0	1	0
6	0	0	1
7	0	0	1
8	0	0	1
9	1	0	0
10	1	0	0

Figure 1. Bitmap join index (BJI).

In these tables, each tuple is identified with a unique identifier denoted RID (Row IDentifier) generated by the DBMS. The index of Fig. 1 can be constructed by the following SQL statement:

```
CREATE   BITMAP INDEX  Product_Type
ON       Sales (Product.Type)
FROM     Sales  S,  Product  P
WHERE    S.PID = P.PID
```

The first tuple of the *Sales* table is joined with a tuple of the *Product* table corresponding to a *Type* product *T1*. Therefore, the bitmap corresponding to type *T1* of the first row of this index is set to 1 and then the remaining bits are set to zero.

Note that a BJI is particularly useful for star joins; like conventional binary indexes, it is very beneficial for *Count(*)* queries where the response to these requests requires only access to the binary index. No access to data tables is necessary; we just need to count the number of 1 in the bitmap array that results of the requested operations. For instance, to determine the number of sales for products of *Type T2*, we count the number of 1 in column T2 of the BJI. For more complex queries (i.e., using several indexed attributes, the logical operators (e.g., AND, OR) are useful.

III. INDEX SELECTION PROBLEM

Index selection is a crucial step in the physical design of the data warehouse. It consists in building an index configuration to optimize the execution cost of a workload (of queries). This optimization can be realized respecting certain constraints, such as the storage space allocated to the index configuration or the cost of maintenance.

Generally, the algorithms proposed for index selection include three steps:

(1) Identification of candidate attributes for indexing.
(2) Pruning.
(3) Construction of an index configuration.

During the first step, a set of candidate attributes can be built manually by the administrator according to his expertise, or automatically by using a queries parser. The step of pruning is necessary to reduce the number of candidate attributes for indexing; it is done by referring to certain criteria, for example, by eliminating high cardinality attributes or those belonging to small tables [10].

The third step builds progressively a final configuration of BIJ respecting the constraints of execution and the storage cost. It is often done by selection algorithms (Glouton algorithms) or directed by data mining techniques [11] [12], or even genetic algorithms [13]. The quality of the generated index configuration is measured by its cost which is - calculated using the optimizer of the DBMS or by a mathematical cost model. All these approaches of BJI selection are applied in a static context (i.e., a workload defined beforehand); however, this workload may evolve in time and might yield obsolete its associated index configuration. To the best of our knowledge, the work in [14] is the only attempt that has tackled the problem of BJI dynamic selection after a workload evolution. It represents an extension to the approach of selecting BJI [15] which is based on data mining technique for the detection of frequent itemset. In fact, the proposed approach is to increment the frequent itemset by referring to a knowledge base that stores information (frequent itemset) of anterior executions. New frequent itemsets are analyzed to generate new candidate indexes; declined (now infrequent) itemsets correspond to indexes to be dropped.

The limited number of algorithms interested in the problem of dynamic index selection led us to propose a new approach. Our approach helps the administrator to maintain the existing index configuration during the evolution of queries executed on the data warehouse.

IV. MAINTENANCE APPROACH BY RECONFIGURING BJI INDEXES

In this section, we present our approach of BJI maintenance which reduces the execution's cost of a new

 ISBN: 978-1-61208-165-6

workload. Our approach is placed in a general framework, which allows illustrating the interest of BJI maintenance. Since the BJI's selection is based on a fixed set of queries, any change in this set may affect the existing BJI's configuration. Fig. 2 outlines the general architecture of our approach. Unlike the proposed approaches of BJI selection, it is not only based on the optimization process on a workload, but it also takes into consideration the existing BJI.

The objective of our approach is to assist the administrator to maintain an initial configuration of BJI during the evolution of the initial workload Q to a new one Q', by proposing a new index configuration which ensures the reduction of Q' cost. The problem can be formalized as follows:

- A data warehouse composed of d dimension tables $D = \{D_1, D_2 ..., D_d\}$ and a fact table F.
- A set of n BJI $I_{current} = \{BJI_1, BJI_2 \ldots, BJI_n\}$ created referring to the initial workload Q.
- f_{ui} represents the frequency use of BJI_i for a given period P sufficiently significant, so that it covers a maximum of treatments.
- Q is a workload of m queries $Q = \{q_1, q_2 ..., q_m\}$, extracted from the *DBMS* log file.
- f_{qi} frequency execution of query q_i.

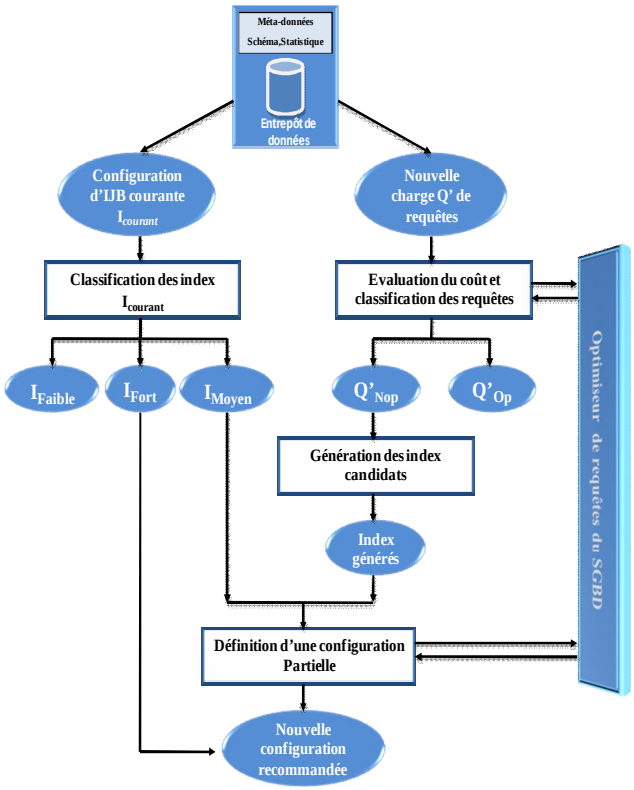

Figure 2. Maintenance approach architecture.

We detail in the following sections, the four steps illustrated in Fig. 2.

A. *Classification of existing indexes*

The current configuration of BJI was created specifically to optimize an initial workload Q of queries. The evolution of the workload Q to Q' can degrade the global performance of Q'. In fact, the new queries are not necessarily optimized compared to the current BJI configuration. In addition, some indexes may become unexploited if they are built to optimize queries actually infrequently executed.

To optimize the new workload Q', we proceed to analyze the utility of the current BJI indexes ($I_{current}$) based on the actual frequency of use (fu) of each index during a period of time and, secondly, on the average frequency fm of all $I_{current}$ indexes used during the same period. This average frequency is calculated by formula (1):

$$fm = \frac{\sum_{i=1}^{n} fu_i}{n} \quad (1)$$

This frequency will allow us to classify the $I_{current}$ set in three subsets:

- I_{high}: represents the subset of indexes heavily used, that means, those whose frequency of use is higher than $^{3*fm}/_{2}$. The BJI belonging to this group are considered important and will be retained to be present in the new configuration BIJ optimizing Q'.
- I_{medium}: represents the subset of indexes used moderately, those whose frequency of use is between $^{fm}/_{2}$ and $^{3*fm}/_{2}$. The relevance of these indexes will be reviewed during the index's selection step to decide whether to keep or reject.
- I_{low}: represents the subset of indexes slightly used, i.e., those whose frequency of use fu is less than $^{fm}/_{2}$. These indexes are rarely or completely unused when running queries from the initial workload. In addition, they occupy a memory space and require a maintenance cost without a justified usefulness. So it is better to remove these indexes and then recover their storage space.

Table I shows an $I_{current}$ set of 8 indexes to optimize a given initial workload Q. The total usage of all indexes is 80 and the average of their frequency of use fm is 10 (= 80/8).

TABLE I. EXAMPLE OF A SET OF BJI WITH THEIR FREQUENCY OF USE

Indexes	*frequency of use*
BJI_1	20
BJI_2	8
BJI_3	1
BJI_4	6
BJI_5	9
BJI_6	27
BJI_7	2
BJI_8	7
Total	80

According to our proposed classification, this set of indexes will be split into three subsets:

 ISBN: 978-1-61208-165-6

- $I_{high} = \{BJI_1, BJI_6\}$
- $I_{medium} = \{BJI_2, BJI_4, BJI_5, BJI_8\}$
- $I_{low} = \{BJI_3, BJI_7\}$

We find out that both indexes BJI_3 and BJI_7 are too little used by the initial workload and, therefore, can be dropped. The drop of indexes with a low-frequency usage optimizes the storage constraint, but it does not improve the global execution's performance.

In parallel, we study the new workload to examine whether it is optimized in respect to the current configuration otherwise we must change this configuration by adding new indexes. This requires the evaluation of the new workload (*Q'*) execution costs and the identification of its non optimized queries in order to consider them when defining the new configuration. The next section details how we examine this workload.

B. Costs evaluation and queries classification

First, this step consists in evaluating the *m* queries of the workload *Q'*. In order to get closer to reality, the study of the evolution of this workload considers the queries actually executed on the data warehouse; we extract these queries from the log file. We note $Cost(q_i)$ the execution cost of query q_i, and I_{qi} all old indexes used by the query optimizer for q_i. Based on these two factors of classification, we subdivide *Q'* into two subsets of queries:

- Q'_{unop} : represents non optimized queries from *Q'*; i.e., those who do not use any index during their execution or those who have a cost of performance higher than the average execution cost *CostAvg (Q')* of the new charge *Q'* with:

$$CostAvg\ (Q') = \frac{\sum_{i=1}^{m} fq_i \times Cost\ (q_i)}{\sum_{i=1}^{m} fq_i} \qquad (2)$$

  $Q'_{unop} = \{q_i' \in Q' / Cost\ (q_i') > CostAvg(Q')$ or $I_{qi}'=\emptyset\}$. Queries of this set will be analyzed in the step called *candidate index generation* in order to define a new index configuration.
- Optimized queries (Q'_{op}) is the set of remaining queries ($Q'_{op}=Q' - Q'_{unop}$). They don't interfere in the definition of a new index configuration.

C. Generating candidate indexes

It is to generate indexes that improve execution performance of the new workload *Q'*, by focusing on all the non-optimized queries (Q'_{unop}). We realize this task throughout the following three steps:

(1) Identification of indexable attributes for Q'_{unop}.
(2) Construction of a BJI configuration per query.
(3) Pruning of indexable attributes.

At the end of these three steps, which we detail below, we obtain a set of candidate indexes that will be evaluated (cf. Section *D*).

1) Identification of indexable attributes

Non optimized queries are handled by a syntactic analyzer to extract all attributes that may be carried for indexes. These attributes are those present in the WHERE, GROUP BY and ORDER BY clauses of queries. For example, the indexable attributes issued from the following query are: City, Month and Type.

```
SELECT   AVG (amount)
FROM     Sales S, Customer C, Product P, Time T
WHERE    S.CID = C.CID AND S.PID = P.PID
AND      S.TID = T.TID AND T.Month = 'MARCH'
AND      C.City IN ('SFAX', 'SOUSSE')
AND      P.TYPE = 'TOY';
```

2) Construction of a BJI configuration per query

Referring to the attributes extracted in the previous step, we construct a matrix query-attribute where it lines represent queries of Q' and columns represent candidate attributes for indexing. The existence of an indexed attribute in a query is represented by the integer 1 and its absence by zero.

Fig. 3 is an example of matrix built on a workload of six queries and six indexable attributes noted A, B, C, D, E, F.

	A	B	C	D	E	F
Q1	0	1	0	1	0	0
Q2	0	1	1	0	0	0
Q3	1	0	1	0	1	0
Q4	0	0	1	1	0	0
Q5	0	0	1	0	1	1
Q6	0	0	1	0	0	0

Figure 3. Matrix Query-Attribute.

3) Pruning

The input of this step is the configuration of candidate indexes from the previous step. This index configuration is beneficial for the whole set of queries because each index has been defined to optimize queries separately. However, this configuration can be very large: it may include some attributes which are not suitable for indexing (high cardinality or belonging to small tables) [16], therefore their removal is recommended to reduce the cost of storage space for indexes. This removal operation is the *Pruning process*; it is based on a *Fitness* parameter introduced by [17] in the context of frequent patterns. We were inspired by this work to define a simplified formula applicable to each individual attribute. This *Fitness* parameter takes into consideration both the frequency of occurrence of attributes in queries and the table size.

$$fitness = \sup_i \times \alpha_i \qquad (3)$$

where sup_i represents the frequency of occurrence of attribute *i* in all queries and $\alpha_i = |Di|/|F|$.

Attributes having a Fitness parameter less than the threshold *minsup* are not indexed; *minsup* in [0, 1] is defined as a parameter by the administrator.

The generated number of indexes is less than or equal to the number of the workload queries, firstly because some queries share the same indexes, secondly, it is unnecessary to generate indexes that already exist (belonging to an entire index of the current configuration I_{curent}). This step generates

 ISBN: 978-1-61208-165-6

a set of candidate indexes created referring to Q'_{unop}. This set will serve together with the moderated used indexes (I_{medium}) to recommend a fairly new configuration.

D. Definition of a configuration to recommend

The union of the candidate indexes generated from the previous step with those belonging to the set of indexes moderately used (I_{medium}) forms the set of all indexes to be selected for the final configuration (Fig. 2). During this stage, the indexes that do not ensure an important gain of cost are considered to be useless and therefore eliminated. We use a Glouton algorithm as in [17] to select the best indexes among the *n* candidate indexes. The selection of an index configuration is an iterative approach by selecting, at each iteration, the index that most reduces the cost of queries execution as the storage space constraint is respected. Finally, the union of the set of selected candidate indexes with the heavily used ones forms the final index configuration proposed to the data warehouse administrator (Fig. 2).

V. EXPERIMENTAL EVALUATION

In order to test our approach, we made an experimental study on a data warehouse, we have generated under Oracle 11g, with the benchmark DWEB1 (Data Warehouse Engineering Benchmark) [18]. This data warehouse is composed of a fact table with 2.043.271 rows, and four dimension tables with 4000, 100, 10000 and 1000 rows. We used DWEB1 to generate a workload of 50 star join queries. Several types of query were considered: *Count* queries, queries using aggregate functions (*Sum*, *Avg)*, queries with dimensional attributes in the SELECT clause, etc. We modified the cardinality of the attributes of the dimensions and certain clauses of query's restriction (WHERE clause attributes) to adapt them for indexing. Our approach of maintenance of BJI configuration is implemented in Visual Studio 2005 and a Core 2 Duo machine with 3 GB of RAM. We conducted a series of experiments that take place in three phases:

- Construction of an index configuration that optimizes our initial workload of 50 queries.
- Modification of this workload by following an increasing trend rate and computing the new cost before maintenance.
- Application of our approach of BJI configuration maintenance and calculation of the execution cost of the workload, after maintenance.
- Application of a second approach of maintenance (naïve) on the same workloads and comparison with the results of our proposed approach.

The execution of the initial 50 queries gave a total cost of input-output (I/O) equal to 173 918 without any index. After building an index configuration, the cost has been reduced to 138 231 thanks to the generation of 10 indexes (4 mono-attributes and 8 multi-attributes).

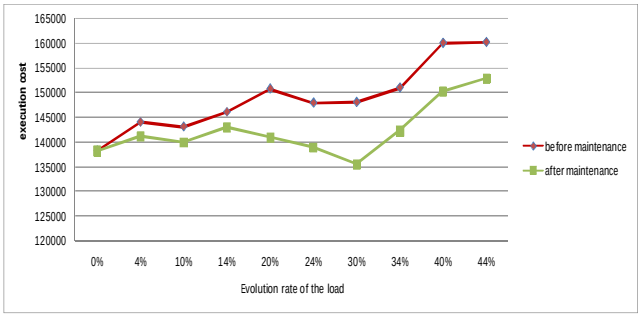

Figure 4. Evolution of the cost according to workload.

The next step is to study the effect of the initial query load evolution on overall performances. For this, we modified the workload several times, according to a variable rate of the charge evolution and keeping constant the number of queries to 50. This modification is to delete some queries and replace them with others of the same type (i.e., same tables and aggregate functions with different attributes of selection) to maintain constant the number of queries. The choice of query to remove is done randomly. The results (Fig. 4) show important performance degradation while the rate of the charge evolution increases. This degradation is due to the fact that newly added queries exploit, little or never, the initial index configuration (i.e., $I_{current}$). So, it's necessary to maintain this configuration. The curve of the execution cost before maintenance according to the evolution of the workload has generally an ascendant trend but we can see some times, zones of decline (Fig. 4) like for the evolution rate 24% and 30 %. This can be due to the fact that some existing indexes (i.e., present in the initial configuration) are used by the newly inserted queries. These used indexes are generally mono-attribute. The application of our maintenance approach, based on the re-calculation of the initial BJI configuration, allows the reduction of the execution cost of the new workload compared to its execution cost before maintenance. According to Fig. 4, the gap between costs before (upper curve) and after (lower curve) the maintenance of indexes becomes more and more important when the rate of the workload evolution increases, which is very interesting in practice because the evolution increases as time goes on.

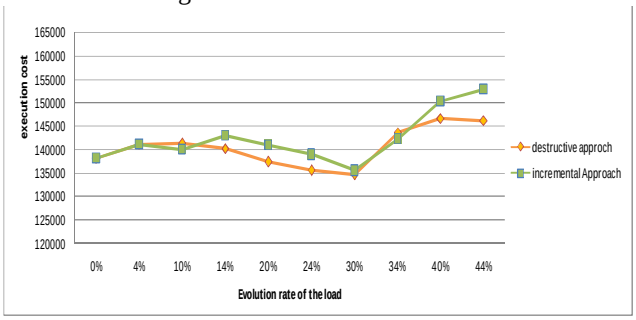

Figure 5. Incremental Approach Vs Destructive.

The experiments done until now approved the interest of applying our maintenance approach (incremental), by index reconfiguration, in order to reduce the execution cost of an

 ISBN: 978-1-61208-165-6

evolving query workload. We evaluate in this section another approach of maintenance (naive) whose principle is to remove all existing indexes and to create a new index configuration for each new workload. Certainly, this destructive approach may give good results in terms of cost of performance because it processes each workload independently from the existing one. Fig. 5 shows that the incremental maintenance approach gives results which are close to those obtained with the destructive approach, even equal in some cases, with the additional advantage of preserving the useful indexes and creating a very limited number of indexes; that constitutes a gain of time (creation of new indexes) and avoids fragmentation of disk space allocated to indexes.

In conclusion, our approach of incremental reconfiguration provides some stability in overall performance during the evolution of an initial workload.

VI. CONCLUSION

We proposed in this paper an approach to assist the administrator of the data warehouse to reconfigure BJI indexes, initially constructed on a workload of analytical queries, following its evolution. Our approach is incremental. It is characterized by the evaluation of the new workload compared according to the existing configuration of BJI in order to decide whether to recalculate a new configuration or to keep the old one. In practice, it proceeds to a classification of existing indexes in three categories: *low*, *medium* or *highly* used. Moreover, determining a new configuration eliminates the slightly used BJI and takes into consideration the old indexes (moderately and highly used) and the new workload to optimize. For the new query workload, we determine a set of candidate BJI indexes according to the conventional principle (extraction of indexable attributes, pruning, and construction of a BJI configuration per query). In order to test our approach, we have developed an iterative algorithm. It determines the cost of a query from its execution plan developed by the Oracle DBMS optimizer; it is to evaluate candidate indexes. The union of selected candidate indexes set and the highly used ones from the configuration to maintain forms the new index configuration proposed to the data warehouse administrator. We tested our approach on a data warehouse built with the benchmark DWEB by varying queries of the initial workload. The preliminary results are so encouraging. However, other experiments will be necessary for large scaling. Also, it would be interesting to study performance thresholds that trigger the recommendation process of reconfiguration.

REFERENCES

[1] W. H. Inmon and R. D. Hackathorn, “Using the Data Warehouse”, - USA : Wiley-QED Publishing, 1994.

[2] T. Stöhr, H. Märtens and E. Rahm, “Multidimensional database allocation for parallel”, Proceedings of the International Conference on Very Large Databases, 2000, pp. 273–284.

[3] S. Chaudhuri and V. Narasayya**,** “Selftuning database systems : A decade of progress”, Proceedings of the 33rd International Conference on Very Large Databases, 2007, pp. 3–14.

[4] M.Hung, M. Huang, D. Yang, and N. Hsueh, “Efficient approaches for materialized views selection in a data warehouse”, Inf. Sci., vol. 177, pp. 1333–1348, March 2007.

[5] S. Agrawal, V. Narasayya and B. Yang, “Integrating vertical and horizontal partitioning into automated physical database design”, Proceedings of the ACM SIGMOD International Conference on Management of Data, 2004, pp. 359–370.

[6] T. Stöhr, H. Märtens and E. Rahm, “Multidimensional database allocation for parallel”, Proceedings of the International Conference on Very Large Databases, 2000, pp. 273–284.

[7] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D, “Ullman. Index Selection for OLAP”, In Proceedings of the Thirteenth International Conference on Data Engineering (ICDE '97). IEEE Computer Society, Washington, DC, USA, 1997, pp. 208–219.

[8] P. O'Neil and G. Graefe, “Multi-table joins through bitmapped join indices”, SIGMOD Rec., vol. 24, pp. 8–11, September 1995.

[9] P. O'Neil and D. Quass, “Improved Query Performance with Variant Indexes”, in ACM SIGMOD International Conference on Management of Data (SIGMOD 1997). - Tucson , USA : 1997. - pp. 38–49.

[10] K. Boukhalfa, L. Bellatreche and Z. Benameur, “Index de jointure binaires : Stratégies de sélection et étude de performances ”, 6ème Journées Francophones sur les Entrepots de Données et Analyse en Ligne (EDA10). - Jerba-Tunisie : 2010. - pp. 175–190.

[11] K. Aouiche and J. Darmont, “Data mining-based materialized view and index selection in data warehouses”, J. Intell. Inf. Syst., vol. 33, pp. 65–93, August 2009.

[12] L. Bellatreche, R. Missaoui, H. Necir, and H. Drias, “A data mining approach for selectingbitmap join indices”, Journal of Computing Science and Engineering 2, n° 1 (2008): 206–223.

[13] J. Kratica, I. Ljubic et D. Tosic, “A Genetic Algorithm for the Index Selection Problem”, (EvoWorkshops'03) The 2003 International Conference on Applications of Evolutionary Computing, 2003, pp. 281–291.

[14] S. Azefack, K. Aouiche and J. Darmont, “Dynamic index selection in data warehouses”, 4th International Conference on Innovations in Information Technology (Innovations 07), 2007.

[15] K. Aouiche, J. Darmont, O. Boussaid, and F. Bentayeb, “Automatic Selection of Bitmap Join Indexes in Data Warehouses”, 7th International Conference on Data Warehousing and Knowledge Discovery (DAWAK 05), 2005.

[16] W. Dylan Bitmap Index – when to use it? [Online]// http://dylanwan.wordpress.com/2008/02/01/bitmap-index-when-to-use-it/. - 2008.

[17] A. H. Necir, L. Bellatreche and R. Missaoui, “DynaClose : Une approche de fouille de données pour la sélection des index de jointure binaires dans les entrepôts de données”, 3ème Journées Francophones sur les Entrepôts de Données et Analyse en Ligne (EDA’07), 2007.

[18] J. Darmont, F. Bentayeb and O. Boussaid, “Benchmarking data warehouses”, International Journal of Business Intelligence and Data Mining, Vol. 2, 2007.

 ISBN: 978-1-61208-165-6