Referential Partitioning Becomes a Reality: Deep Analysis and Selection Strategies

Ladjel Bellatreche[1] and Komla Yamavo Woameno[1]

LISI/ENSMA - Poitiers University
France
(bellatreche, woamenok)@ensma.fr

Abstract. Most of business intelligence applications use data warehousing solutions. The star schema modelling these applications is composed of hundreds of dimension tables and multiple huge fact tables. These applications require efficient access to data. Referential horizontal partitioning is one of physical design techniques fitting this requirement. It consists in splitting a fact table based on fragmentation schemes of set of dimension tables. Most of the existing works on referential partitioning are concentrated on how to split dimension tables, and not on how to select them. Selecting dimension table(s) among a set of candidates is a crucial performance issue that should be addressed. In this paper, we first show the complexity of choosing dimension table(s) to referential partition a fact table. Secondly, we present three strategies for selecting them. Finally, to validate of our proposal, we conduct intensive experimental studies using a mathematical cost model estimating the number of inputs outputs required for executing a set of queries on a partitioned warehouse. The obtained results are implemented using the APB1 benchmark data set on Oracle11G that supports referential partitioning.

1 Introduction

Enterprise wide data warehouses are becoming increasingly adopted as the main source and underlying infrastructure for business intelligence (BI) solutions. Star schemes or their variants are usually used to model these applications. They are composed of hundreds of dimension tables and multiple fact tables [10]. Queries running on such applications contain a large number of costly joins, selections and aggregations. They are called *mega queries* [18]. To ensure a high performance of mega queries, advanced optimization techniques are mandatory. By analyzing the main optimization techniques proposed in the literature and supported by commercial DBMSs, we realize that some are applied when creating *the schema of the data warehouse*. Horizontal partitioning and parallel processing are two examples of such techniques. Any technique having this characteristic is called an *upstream optimization technique*, since it is used before populating the warehouse. Other techniques are usually selected after the creation of data warehouse schema (when data warehouse is under exploitation). Examples of these techniques are materialized views and indexes, data compression, etc.

Any technique having this characteristic is called a *downstream optimization technique*. The use of upstream optimization techniques is more sensitive compared to the downstream techniques, since the decision of using them is made at the beginning stage of the warehouse development. For example, if database administrator (DBA) partitions his/her warehouse using a native data definition language (DDL), it will be very costly and time consuming to reconstitute the initial warehouse from the partitions. But, if he/she defines a downstream technique like a materialized view, he/she can easily drop or replace it by another one. Consequently, the use of upstream optimization techniques needs some carefulness. Note that upstream and downstream optimization techniques can be used conjointly (indexes are selected on a partitioned warehouse). Due to the sensitivity and the carefulness when using upstream techniques, this paper deals with one these techniques which is the *referential horizontal partitioning*.

Horizontal partitioning is one of important aspect of physical design [16]. It is a divide-and-conquer approach that improves query performance, operational scalability, and the management of ever-increasing amounts of data [20]. It divides any access method such as tables, indexes and materialized views into disjoint sets of rows that are physically stored and accessed separately. Unlike indexes and materialized views, it does not replicate data, thereby reducing storage requirement and minimizing maintenance overhead. Most of today's DBMSs (Oracle, DB2, SQL Server, PostgreSQL, Sybase and MySQL) offer native DDL support for creating (*Create Table ... Partition*) and manipulating horizontal partitions (*Merge* and *Split* partitions) [16, 17, 6]. Notice that when horizontal partitioning is applied on materialized views and indexes, it is considered as a downstream partitioning. Its classification depends on the nature of the used access method (table, materialized views and index).

Horizontal partitioning improves performance of queries by the mean of *pruning mechanism* that reduces the amount of data retrieved from the disk. When a partitioned table is queried, the database optimizer eliminates non relevant partitions by looking the partition information from the table definition, and maps that onto the *Where* clause predicates of the query. Two main pruning are possible: *partition pruning* and *partition-wise join pruning*. The first one prunes out partitions based on the partitioning key predicate. It is used when only one table is partitioned. The second one prunes partitions participating in a join operation. It is applicable when the partitioning key is same in the two tables.

A large spectrum of partitioning modes have been proposed and supported by the main DBMSs. We classify them into two main types, based on the number of participated tables in the partitioning process: *single table partitioning* and *table dependent partitioning*. In single table partitioning, a table is partitioned based only on its attributes. This partitioning is similar to *primary horizontal partitioning* [15]. To implement this partitioning, several modes exist: *Range*, *List*, *Hash*, *Round Robin* (supported by Sybase), *Composite* (*Range-Range, Range-List, List-List*, etc.), *Virtual Column partitioning*, etc. Single table partitioning is well adapted for optimizing selection operations, especially when partitioning key matches with selection attributes. In table dependent partitioning, a

table (usually called child) is partitioned based on the fragmentation schemes of other tables[1] of other tables (called parents). This partitioning is feasible if there is a parent-child relationship among these two tables [4, 6]. It is quite similar to the derived horizontal partitioning proposed in 80's [4]. To implement this partitioning, two main modes are available: *Native Referential Partitioning* and *User-driven Referential Partitioning*. In the native referential partitioning, a child table inherits the partitioning characteristics from its parent table. It is actually supported by Oracle11G, by the use of a native DDL (*Create Table ... Partition by Reference ...*). It optimizes selection and joins simultaneously. Therefore, it is well adapted for star join queries[2].

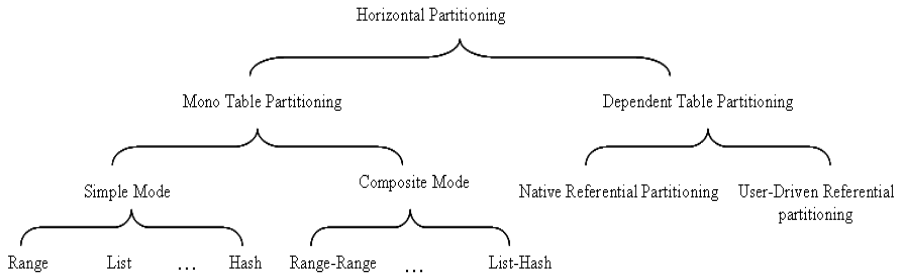

Fig. 1. A classification of the proposed partitioning modes

User-driven referential partitioning is a manual implementation of referential partitioning, i.e., it simulates it using the single table partitioning. More concretely, a parent table is first horizontally partitioned on its primary key then the child table is splitted using the foreign key referencing the parent table. This partitioning has been used for designing parallel data warehouses [7]. But, it is not well adapted for star join queries. Also it has some implementation restrictions [6]. Figure 1 classifies the proposed partitioning modes[3].

The use of referential horizontal partitioning for physical designing advanced applications such as data warehousing is not straightforward. To ensure a better utilization of this optimization technique, DBA should do the following steps: (i) selection of parent table(s) to be used to split the child table (which represents the fact table), (ii) identification of partitioning keys for each table, (iii) identification of borders of each partition for the case of *Range partitioning* and (iv) effective partitioning of child and parent tables.

[1] A fragmentation schema is the result of partitioning process.

[2] A star query is a join between a fact table and dimension tables, but the dimension tables are not joined to each other. Selection predicates are defined on each dimension table.

[3] By referential partitioning we mean the native one throughout the paper.

Most of the existing works on referential partitioning ignore the first step, since they suppose that dimension tables to be used to partition the fact table are given. For BI applications, where hundreds of dimension tables are used, it will be hard for DBA to select the relevant tables. In this paper, we address the problem of referential partitioning, where we propose a complete methodology for selecting dimension table(s) candidate to partition the fact table.

This paper is divided in six sections: Section 2 reviews the existing works on horizontal partitioning in traditional databases and data warehouses. Section 3 proposes three selection strategies of dimension table(s) to support referential horizontal partitioning. Section 4 gives the experimental results done on the data set of APB-1 benchmark using a mathematical cost model and on Oracle11. Section 5 concludes the paper by summarizing the main results and suggesting future work. Section 6 describes queries used in our experimentations.

2 Related Work

Many research works dealing with horizontal partitioning problem were proposed in traditional databases and data warehouses. In the traditional database environment, researchers concentrate their work on proposing algorithms to partition a given table by analyzing a set of selection predicates used by queries defined on that table. Three types of algorithms are distinguished: *mimterm generation-based approach* [15], *affinity-based approach* [9] and *cost-based approach* [2]. Most of them concern only single table partitioning.

Online analysis applications characterized by their complex queries motivate data warehouse community to propose methodology and efficient algorithms for horizontal partitioning. Noaman et al. [14] have proposed a methodology to partition a relational data warehouse in a distributed environment, where the fact table is derived partition based on queries defined on all dimension tables. Munneke et al. [13] proposed a fragmentation methodology for a multidimensional warehouse, where the global hypercube is divided into sub cubes, where each one contains a sub set of data. This process is defined by the *slice* and *dice* operations (similar to selection and projection in relational databases). This methodology chooses manually relevant dimensions to partition the hyper cube. In [1], a methodology and algorithms dealing with partitioning problem in relational warehouses are given. The methodology consists first in splitting dimension tables and using their fragmentation schemes to derive partition the fact table. It takes into account the characteristics of star join queries. Three algorithms selecting fragmentation schemes reducing query processing cost were proposed: *genetic, simulated annealing* and *hill climbing*. As in traditional database, these algorithms start with a set of predicates defined on all dimension tables. The main particularity of these algorithms is their control of generated fragments. [11] proposed an algorithm for fragmenting XML data warehouses based on k-means technique. Thanks to this k-means, the algorithms controls the number of fragments as in [1]. Three main steps characterize this approach: (i) extraction of selection predicates (simple predicates) from the query workload, (ii) predicate

clustering with the k-means method and (iii) fragment construction with respect to predicate clusters. This approach is quite similar to affinity based approach [9]. To summarize, most the proposed works proposed in traditional databases are mainly concentrated on a single table partitioning mode. Most of studies in data warehouses are mainly concentrated on dependent table partitioning, where the fact table is partitioned based on the fragmentation schemes of dimension tables, without addressing the problem of identification of dimension tables; except Munneke et al.'s work [13] that points out the idea of choosing and eliminating dimension to partition a hyper cube.

3 Selection of Dimension Tables

In this section, we propose a complete methodology to support referential horizontal partitioning in relational database (or data warehouse). This methodology offers to DBA the possibility to choose the dimension table(s) to partition the fact table and to split dimension and fact tables using his/her favourite partitioning algorithm.

Before proposing this methodology, we formalize the referential horizontal partitioning problem as follows:
Given a data warehouse with a set of d dimension tables $D = \{D_1, D_2, ..., D_d\}$ and a fact table F, a workload Q of queries $Q = \{Q_1, Q_2, ..., Q_m\}$ and a maintenance constraint W fixed by DBA that represents the maximal number of fact fragments that he/she can maintain. The referential horizontal partitioning problem consists in (i) identifying dimension table(s) candidate, (ii) splitting them using single partitioning mode and (iii) using their partitioning schemes to decompose the fact table into N fragments, such that: (a) the cost of evaluating all queries is minimized and (b) $N \leq W$.

Based on this formalization, we note that referential partitioning problem is a combination of two difficult sub problems: *identification of relevant dimension tables* and their *fragmentation schema selection* (which is known as a NP-complete problem [1]). In the next section, we study the complexity of the first sub problem (identification of relevant dimension tables).

3.1 Complexity of Selecting Dimension Tables

The number of possibilities that DBA shall consider to partition the fact table using referential partitioning is given by the following equation:

$$\binom{d}{1} + \binom{d}{2} + ... + \binom{d}{d} = 2^d - 1 \quad (1)$$

Example 1. Let us suppose a warehouse schema with three dimension tables *Customer*, *Time* and *Product* and one fact table *Sales*. Seven scenarios are possible to referential partition *Sales*: *Sales(Customer), Sales(Time), Sales(Product), Sales(Customer, Time), Sales(Customer, Product), Sales(Product, Time)* and *Sales(Customer, Time, Product)*. For a schema with a large number of dimension tables, it is hard for DBA to choose manually dimension table(s) to partition

his/her fact table. In the next section, we give strategies aiding DBA to select these tables.

3.2 Dimension Table Selection Strategies to Split the Fact Table

Before presenting dimension table selection strategies; some concepts are required.

The main contribution of referential partitioning is join optimization and data management. Therefore, any selection strategy should turn around join operation. In data warehousing, every join involves the fact table. The query graph of a such query contains a root table (representing the fact table) and peripheral tables (representing dimension tables) (see Figure 2 (a)). To optimize the execution of a star join query, an optimal sequence in which the tables are joined shall be identified[4]. This is problem is known as *join ordering*, which is a hard problem [19].

Intuitively, the join ordering problem is quite similar to dimension table selection problem. Join ordering is specific to one query (Figure 2 (b)), but dimension table selection problem is specific to all queries running on data warehouse, since data partitioning shall optimize performance of overall queries.

A simplest way to partition the fact table is to use the *largest dimension table(s)* since join operation is costly, especially when the size of involved tables is huge. The second naive solution is to choose the *most frequently used dimension table(s)*. The frequency of a dimension table D_i is calculated by counting its appearance in the workload.

The two naive strategies are simple to use. But they suffer from their ignorance of estimating the size of intermediate results of join operation. To consider this creterion, we propose the following strategy.

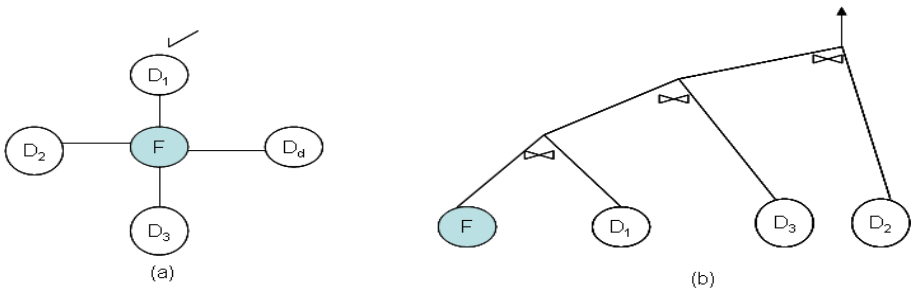

Fig. 2. (a) Star Query Graph, (b) Star query execution using left deep tree

[4] A join sequence is defined as any permutation of set of dimension tables.

Miminum Share Strategy Optimizing star join queries can be done by minimizing the size of intermediate result during query evaluation process [12]. Based on this result, we propose to referential partition the fact table based on dimension table(s) reducing the intermediate results.
Note that the size (in terms of number of tuples) of join between the fact table F and a dimension table D_i (which is a part of a star join query) , denoted by $||F \bowtie D_i||$ is estimated by:

$$||F \bowtie D_i|| = ||D_i|| \times Share(F.A_i) \quad (2)$$

where $||D_i||$ and $Share(F.A_i)$ represent respectively, the cardinality of dimension table D_i and the average number of tuples of the fact table F that refer to the same tuple of the dimension table D_i via the attribute A_i. To reduce the intermediate result of a star join query, we should choose the dimension table with a *minimum share*.

3.3 Referential Partitioning Methodology

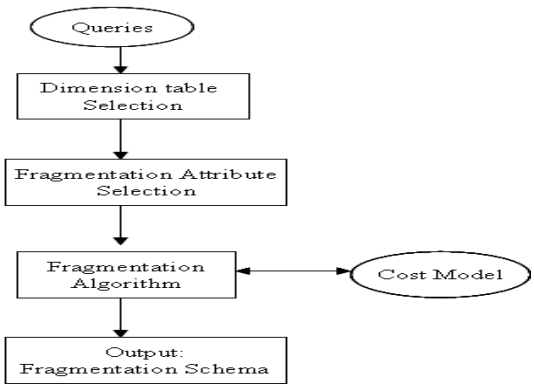

Fig. 3. A Methodology to Support Referential Partitioning

Now, we have all ingredients to propose our methodology offering a real utilization of data partitioning in data warehouse application design. It is subdivided into three steps described in Figure 3:

1. Selection of dimension table(s) used to decompose the fact table. This selection may be done using one of the previous criteria.
2. Extraction of fragmentation attributes (key partitioning) candidate. A fragmentation attribute is an attribute used by a selection predicate defined on a chosen dimension table.

3. Partition each dimension table using single table partitioning type. To generate partitioning schemes of chosen dimension tables, DBA may use his/her favourite algorithm. We propose to use one of our algorithms proposed in [1]. The main characteristic of these algorithms is that they generate a complete fragmentation schema of the warehouse (fact and dimension tables), where the number of generated fragments is less than a threshold fixed by DBA. The complete generation of fragmentation schemes (of fact and dimension tables) is ensured by using a particular coding. Each fragmentation schema is represented using a multidimensional array, where each cell represents a domain partition of a fragmentation schema. To illustrate this coding, let us suppose the following scenario: DBA choose dimension table *Customer* and Time to referential partition the fact table Sales. Three attributes (Age, Gender, Season), where Age and Gender belong to *Customer* dimension table, whereas Season belongs to Time. The domain of these attributes are: $Dom(Age) = [0, 120]$, $Dom(Gender) = \{$'M', 'F'$\}$, and $Dom(Season) = \{$"Summer", "Spring", "Autumn", "Winter"$\}$. DBA proposes an initial decomposition of domains of these three attributes as follows: $Dom(Age) = d_{11} \cup d_{12} \cup d_{13}$, with $d11 = [0, 18]$, $d_{12} =]18, 60[$, $d_{13} = [60, 120]$. $Dom(Gender) = d_{21} \cup d_{22}$, with $d_{21} = \{'M'\}$, $d_{22} = \{'F'\}$. $Dom(Season) = d_{31} \cup d_{32} \cup d_{33} \cup d_{34}$, where $d_{31} = \{$"Summer"$\}$, $d_32 = \{$"Spring"$\}$, $d_{33} = \{$"Autumn"$\}$, and $d_{34} = \{$"Winter"$\}$. Sub domains of all three fragmentation attributes are represented in Figure 4.

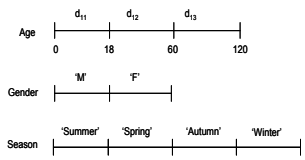

Fig. 4. Decomposition of Attribute Domains

Age	1	2	3	
Gender	1	2		
Season	1	2	3	4

Fig. 5. Coding of a Fragmentation Schema

Domain partitioning of different fragmentation attributes may be represented by multidimensional arrays, where each array represents the domain partitioning of a fragmentation attribute. The value of each cell of a given array representing an attribute belongs to $(1..n_i)$, where n_i represents the number of sub domain of the attribute (see Figure 5). Based on this representation, fragmentation schema of each dimension table D_j is generated as follows:

- all cells of a fragmentation attribute of D_j have different values: this means that all sub domains will be used to partition Dj. For instance, the cells of each fragmentation attribute in Figure 4 are different. Therefore, they all participate in fragmenting their corresponding tables (*Customer* and Time). The final fragmentation schema will generate 24 fragments of the fact table.

- all cells of a fragmentation attribute have the same value: this means that it will not participate in the fragmentation process. Table 1 gives an example of a fragmentation schema, where all sub domains of Season (of dimension table Time) have the same value; consequently, it will not participate in fragmenting the warehouse schema.
- some cells has the same value: their corresponding sub domains will be merged into one. In Table 1, the first ([0, 18]) and the second (]18, 60[) sub domains of Age will be merged to form only one sub domain which is the union of the merged sub domains ([0, 60[). The final fragmentation attributes are: Gender and Age of dimension table *Customer*.

4. Partition the fact table using referential partitioning based on the fragmentation schemes generated by our algorithm.

To illustrate this step, let us consider an example. Suppose that our algorithm generates a fragmentation schema represented by a coding described by Table 1. Based on this coding, DBA can easily generate scripts using DDL to create a

Table 1. Example of a Fragmentation Schema

Table	Attribute	SubDomain	SubDomain	SubDomain	SubDomain
Customer	Age	1	1	2	
Customer	Gender	1	2		
Time	Season	1	1	1	1

partitioned warehouse. This may be done as follows: *Customer* is partitioned using the composite mode (*Range* on attribute *Age* and *List* on attribute *Gender*) and the fact table using the native referential partitioning mode.

```
CREATE TABLE Customer
(CID NUMBER, Name Varchar2(20), Gender CHAR, Age Number)
PARTITION BY RANGE (Age)
SUBPARTITION BY LIST (Gender)
SUBPARTITION TEMPLATE (SUBPARTITION Female VALUES ('F'),
SUBPARTITION Male VALUES ('M'))
(PARTITION Cust_0_60 VALUES LESS THAN (61),
PARTITION Cust_60_120 VALUES LESS THAN (MAXVALUE));
```

The fact table is also partitioned into 4 fragments as follows:

```
CREATE TABLE Sales (customer_id NUMBER NOT NULL, product_id NUMBER NOT NULL,
time_id Number NOT NULL, price NUMBER, quantity NUMBER,
constraint Sales_customer_fk foreign key(customer_id) references CUSTOMER(CID))
PARTITION BY REFERENCE (Sales_customer_fk);
```

4 Experimental Results

We have conducted intensive experimental studies to evaluate our proposal. They are classified into two main types: (i) evaluation of the quality of each selection

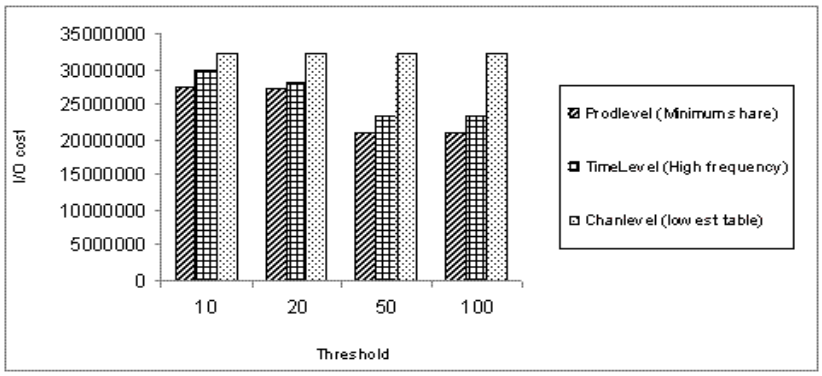

Fig. 6. Theoretical Results

strategy using a mathematical cost model and (ii) the obtained solutions obtained by this model are implemented on Oracle11G.

Dataset: We use the dataset from the APB1 benchmark [5]. The star schema of this benchmark has one fact table Actvars (33 323 400 tuples) and four dimension tables: Prodlevel (9 900 tuples), Custlevel (990 tuples), Timelevel (24 tuples) and Chanlevel (10 tuples).

Workload: We have considered a workload of 55 single block queries (i.e., no nested sub queries) with 38 selection predicates defined on 10 different attributes (lass_Level, Group_Level, Family_Level, Line_Level, Division_Level, Year_Level, Month_Level, Quarter_Level, Retailer_Level, All_Level). The domains of these attributes are split into: 4, 2, 5, 2, 4, 2, 12, 4, 4, 5 sub domains, respectively. We did not consider update queries (update and delete operations). Note that each selection predicate has a selectivity factor computed using SQL queries executed on the data set of APB1 benchmark. The set of queries are described in Appendix section.

Our algorithms have been implemented using Visual C++ performed under Intel Pentium 4 with a memory of 3 Go.

4.1 Theoretical Evaluation of Selection Strategies

The used cost model computes the inputs and outputs required for executing the set of 38 queries. It takes into account the size of intermediate results of joins and the size of buffer. For lack of space, this cost model is not included in this paper; for more details refer to [3].

We use hill climbing algorithm to select fragmentation schema of the APB1 warehouse [1]. It consists of the following two steps: (1) find an initial solution and (2) iteratively improve this solution by using hill climbing heuristics until no further reduction in total query processing cost. The initial solution is represented by multidimensional array, where its cells are filled in uniform way.

The first experiments evaluate the quality of each criterion: *minimum share*, *high frequency*, and *largest size*. The size and frequency of each dimension candidate to partition the fact table are easily obtained from the data set of APB1 benchmark and the 38 queries. The share criterion is more complicated to estimate, since it requires more advanced techniques such histograms [8]. For our case, we calculate the *share* using SQL queries executed on the data set of APB1 benchmark created on Oracle11G. For instance, the *share* of the dimension table *CustLevel* is computed by the following query:

```
Select avg(Number) as AvgShare
FROM (Select distinct customer_level, count(*) as Number
      From actvars
      Group By Customer_level);
```

For each criterion, we run our hill climbing algorithm with different values of the threshold (representing the number of generated fragments fixed by DBA) set to 10, 20, 50 and 100. For each obtained fragmentation schema, we estimate the cost of cost of evaluating the 38 queries. At the end, 12 fragmentation schemes are obtained and evaluated. Figure 6 summarizes the obtained results. The main lessons behind these results are: (1) the minimum share criterion outperforms other criteria (frequency, maximum share), (2) minimum share and largest dimension table criteria give the same performance. This is because, in our real data warehouse, the table having the *minimum share* is the largest one and (3) the threshold has a great impact on query performance. Note that the fact of increasing the threshold does not mean getting a better performance, since when it is equal to 50 and 100; we got practically the same results. Having a large number of fragments increases the number of union operations which can be costly, especially when the size of manipulating partition instances is huge.

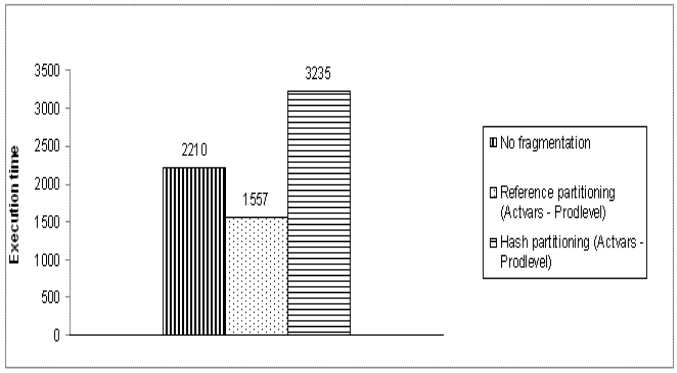

Fig. 7. Native and User Driven Referential Partitioning Comparaison

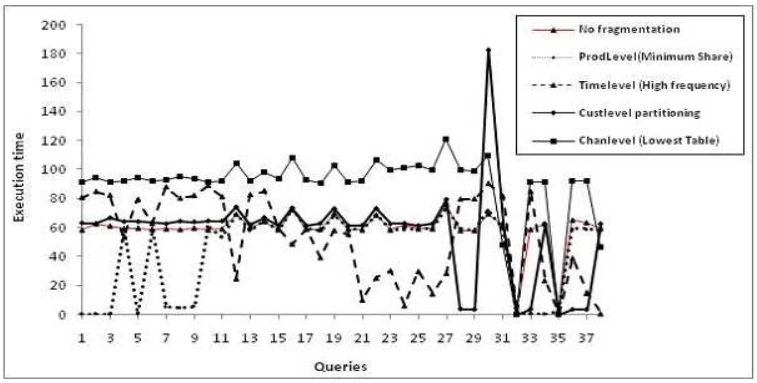

Fig. 8. Individual Evaluation of impact of Criterion on Query

4.2 ORACLE 11G Validations

To validate the theoretical results, we conduct experiments using Oracle11G. We choose this DBMS because it supports referential horizontal partitioning. The data set of ABP1 benchmark is created and populated using generator programs offered by ABP1. During this validation, we figure out that *Composite partitioning* with more than two modes is not directly supported by Oracle11G DDL [5]. To deal with this problem, *we propose the use of virtual partitioning column proposed by Oracle11G*. A virtual column is an expression based on one or more existing columns in the table. While a virtual column is only stored as metadata and does not consume physical space. To illustrate the use of this column, let D_i be a partitioned table in N_i fragments. Each instance of this table belongs to a particular fragment. The identification of the relevant fragment of a given instance is done by matching partitioning key values with instance values. To illustrate this mechanism, suppose that dimension table *ProdLevel* is partitioned into 8 partitions by the hill climbing algorithm using three attributes. A virtual column *PROD_COL* is added into *ProdLevel* in order to facilitate its partitioning using the *List* mode:

```
CREATE TABLE Prodlevel(
CODE_LEVEL CHAR(12) NOT NULL, CLASS_LEVEL CHAR(12) NOT NULL, GROUP_LEVEL CHAR(12) NOT NULL, FAMILY_LEVEL CHAR(12) NOT NULL,
LINE_LEVEL CHAR(12) NOT NULL, DIVISION_LEVEL CHAR(12) NOT NULL,
PROD_COL NUMBER(5) generated always as (case when Class_Level IN('ADX8MBFPWVIV','OC2WOOC8QIJ6','LB2RKOOZQCJD')
and  Group_Level='VTL9DOE3RSWQ' and Family_Level  in('JIHR5NBAZWGU','OX3BXTCVRRKU','M32G5M3AC4T5', 'Y45VKMTJDNYR') then 0
when Class_Level IN ('ADX8MBFPWVIV','OC2WOOC8QIJ6','LB2RKOOZQCJD') and  Group_Level='VTL9DOE3RSWQ' and Family_Level  not in
('JIHR5NBAZWGU','OX3BXTCVRRKU','M32G5M3AC4T5', 'Y45VKMTJDNYR') then 1 when Class_Level IN
('ADX8MBFPWVIV','OC2WOOC8QIJ6','LB2RKOOZQCJD') and  Group_Level not in ('VTL9DOE3RSWQ') and Family_Level in
('JIHR5NBAZWGU','OX3BXTCVRRKU','M32G5M3AC4T5','Y45VKMTJDNYR')
 then 2 when Class_Level IN ('ADX8MBFPWVIV','OC2WOOC8QIJ6','LB2RKOOZQCJD') and Group_Level not in ('VTL9DOE3RSWQ') and
Family_Level  not in('JIHR5NBAZWGU','OX3BXTCVRRKU','M32G5M3AC4T5','Y45VKMTJDNYR') then 3  when Class_Level NOT IN
('ADX8MBFPWVIV','OC2WOOC8QIJ6','LB2RKOOZQCJD') and  Group_Level='VTL9DOE3RSWQ' and Family_Level  in
('JIHR5NBAZWGU','OX3BXTCVRRKU','M32G5M3AC4T5','Y45VKMTJDNYR') then 4 when Class_Level NOT IN
('ADX8MBFPWVIV','OC2WOOC8QIJ6','LB2RKOOZQCJD') and  Group_Level='VTL9DOE3RSWQ' and Family_Level  not in
('JIHR5NBAZWGU','OX3BXTCVRRKU','M32G5M3AC4T5','Y45VKMTJDNYR') then 5
Else  7  end), PRIMARY KEY (CODE_LEVEL))
PARTITION BY LIST(PROD_COL)
(PARTITION PT1 VALUES(0) TABLESPACE HCTB,PARTITION PT2 VALUES(1) TABLESPACE HCTB,
PARTITION PT3 VALUES(2) TABLESPACE HCTB, PARTITION PT4 VALUES(3) TABLESPACE HCTB,
PARTITION PT5 VALUES(4) TABLESPACE HCTB, PARTITION PT6 VALUES(5) TABLESPACE HCTB,
PARTITION PT7 VALUES(6) TABLESPACE HCTB, PARTITION PT8 VALUES(7) TABLESPACE HCTB);
```

None materialized views and advanced indexes are created, except indexes on primary and foreign keys.

[5] For example, we can partition a table using three fragmentation attributes

Figure 7 compares the *native referential partitioning* and *user driven referential partitioning*. In both cases, the fact table is partitioned into 8 partitions based on *ProdLevel* table. In user driven referential partitioning, the *ProdLevel* is horizontally partitioned using the *hash mode* on the primary key (Code_Level) and the fact table is horizontally partitioned using also the *hash mode* on the foreign key (the number of partitions using hash must be a power of 2). Both partitioning are compared to the non partitioning case. The result shows that native horizontal partitioning outperforms largely the two other cases. *The user driven referential partitioning is not adapted to optimize star join queries and even more the non partitioned case outperforms it.*

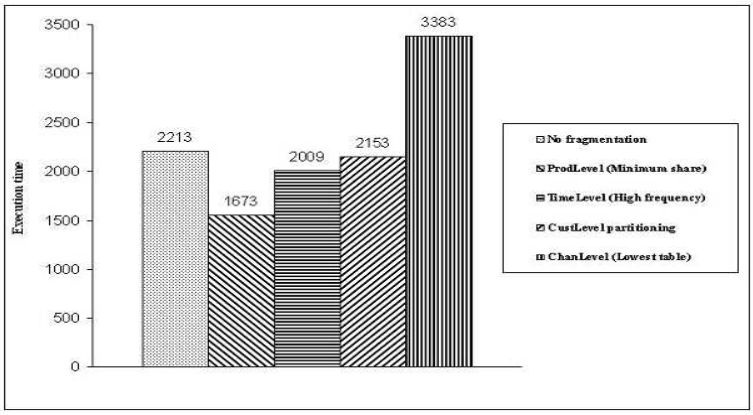

Fig. 9. Evaluation of all Strategies for overall Queries

Figure 8 compares the quality of solution generated by each criteria: *Minimum Share, Highest frequency* and *Lowest table*. These criteria are also compared with the non partitioned case. To conduct this experiment, we execute three times our climbing algorithm with a threshold equal to 10. Each query is executed individually on each generated fragmentation schema (one schema per selection criterion) and its execution time (given in second) is computed using *Enterprise manager of Oracle 11G*. Based on the obtained results, we propose to DBA the following recommendations:

- to improve performance of queries containing only one join and selection predicates involving only one dimension table, DBA has to partition the fact table based on that dimension table; without worrying about its characteristic (minimum share, highest frequency, etc.). More clearly, queries 1, 2, 5 optimized with the *minimum share strategy* and queries 21, 22 and 23 with high frequency, etc. This case is very limited in real data warehouse applications, where queries involve all dimension tables,

– to speed up queries involving several dimension tables, but only one of them has selection predicates, DBA has to partition his/her fact table based on the dimension table involving selection predicates,
– to accelerate queries involving several dimension tables and each one contains selection predicates (this scenario is most representative in the real life), DBA has to partition his/her fact table based on the dimension table having a *minimum share*. *Query 34* is an example of this scenario, where two dimension tables are used *ProdLevel* (with class_level='LB2RKO0ZQCJD) and *TimeLevel* (with two predicates Quarter_level= 'Q2' and Year_level='1996'). Another example is about the *query 38*, where two dimension tables are used, and each one has only one selection predicate. In this case, *TimeLevel* strategy (representing high frequency and also the table having a second minimum share) outperforms *ChanLevel* strategy representing the lowest table.
To complete our understanding, we use "*Set autotrace on*" utility to get execution plan of the query 34. First, we execute it on the non partitioned data warehouse and we generate its execution plan. We identify that Oracle optimizer starts by joining fact table with *ProdLevel* table (which has the minimum share criterion)[6]. Secondly, we execute the same query on a partitioned data warehouse obtained by fragmenting the fact table based on *ProdLevel*. The query optimizer starts by joining a partition of fact table with one of *ProdLevel* table and then does other operation. Based on these two execution plans, we can conclude that query optimizer uses the same criteria (minimum share) for ordering join. Referential partitioning gives a partial order of star join queries problem. Figure 9 summarizes the performance of each criterion for all queries.

5 Conclusion

Data partitioning is one of important aspect of physical design of advanced database systems. It is advocated by most commercial DBMS, especially Oracle11G. Two partitioning modes exist: single table partitioning and dependent table partitioning. The second mode is well adapted for optimizing selection and joins of mega queries of SAP applications. In this paper, we show the importance of the use of referential partitioning that consists in splitting a fact table based on partitioning schemes of dimension tables. A formalization of referential partitioning problem is presented and its complexity is given. In order to facilitate the task of DBA in choosing dimension tables candidate for referential partitioning his/her fact tables, we propose three strategies based on the share, frequency and cardinality of dimension table(s). Selecting the share criteria is similar to selecting the order of joins. It reduces the size of intermediate results of joins; therefore, it may be used in cascade over dimension tables. A complete methodology for using horizontal partitioning in data warehouse is proposed. It allows the use of single and dependent table partitioning modes. Two experimental studies were conducted: one using a mathematical cost model and another using Oracle11G

[6] The Oracle cost-based optimizer recognizes star queries.

with APB1 benchmark. We propose to support a composite partitioning with more than two attributes by Virtual Column partitioning mode. The obtained results are encouraging. Based on these results some recommendations are given to aid DBA to partition his/her data warehouse. Our methodology can be easily incorporated in any DBMS supporting referential partitioning. It will be interesting to consider two main extensions of this work: (i) extension/adaptation of the proposed methodology to the vertical partitioning and (ii) studying in deep the effect of the presence of partitioned tables to deal with the problem of join ordering.

References

1. L. Bellatreche, K. Boukhalfa, and Pascal Richard. Data partitioning in data warehouses: Hardness study, heuristics and oracle validation. In *International Conference on Data Warehousing and Knowledge Discovery (DaWaK'2008)*, pages 87–96, 2008.
2. L. Bellatreche, K. Karlapalem, and A. Simonet. Algorithms and support for horizontal class partitioning in object-oriented databases. *in the Distributed and Parallel Databases Journal*, 8(2):155–179, April 2000.
3. K. Boukhalfa, L. Bellatreche, and P. Richard. Fragmentation primaire et drive: tude de complexit, algorithmes de slection et validation sous oracle10g. Techreport http://www.lisi.ensma.fr/members/bellatreche, LISI/ENSMA, 2008.
4. S. Ceri, M. Negri, and G. Pelagatti. Horizontal data partitioning in database design. *Proceedings of the ACM SIGMOD International Conference on Management of Data. SIGPLAN Notices*, pages 128–136, 1982.
5. OLAP Council. Apb-1 olap benchmark, release ii. *http://www.olapcouncil.org/research/bmarkly.htm*, 1998.
6. G. Eadon, E. I. Chong, S. Shankar, A. Raghavan, J. Srinivasan, and S. Das. Supporting table partitioning by reference in oracle. *SIGMOD'08*, 2008.
7. P. Furtado. Experimental evidence on partitioning in parallel data warehouses. In *DOLAP*, pages 23–30, 2004.
8. P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. *ACM Trans. Database Syst.*, 27(3):261–298, 2002.
9. K. Karlapalem, S. B. Navathe, and M. Ammar. Optimal redesign policies to support dynamic processing of applications on a distributed database system. *Information Systems*, 21(4):353–367, 1996.
10. T. Legler, W. Lehner, and A. Ross. Query optimization for data warehouse system with different data distribution strategies. In *BTW*, pages 502–513, 2007.
11. H. Mahboubi and J. Darmont. Data mining-based fragmentation of xml data warehouses. In *ACM 11th International Workshop on Data Warehousing and OLAP (DOLAP'08)*, pages 9–16, 2008.
12. B. J. McMahan, G. Pan, P. Porter, and M. Y. Vardi. Projection pushing revisited. In *9th International Conference on Extending Database Technology (EDBT'04)*, pages 441–458, 2004.
13. D. Munneke, K. Wahlstrom, and Mohania M. K. Fragmentation of multidimensional databases. *in the 8th Australian Database Conference (ADC'99)*, pages 153–164, 1999.

14. A. Y. Noaman and K. Barker. A horizontal fragmentation algorithm for the fact relation in a distributed data warehouse. *in the 8th International Conference on Information and Knowledge Management (CIKM'99)*, pages 154–161, November 1999.
15. M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems : Second Edition*. Prentice Hall, 1999.
16. A. Sanjay, V. R. 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*, pages 359–370, June 2004.
17. Oracle Data Sheet. Oracle partitioning. *White Paper: http://www.oracle.com/technology/products/bi/db/11g/*, 2007.
18. E. Simon. Reality check: a case study of an eii research prototype encountering customer needs. In *11th International Conference on Extending Database Technology (EDBT'08)*, page 1, March 2008.
19. M. Steinbrunn, G. Moerkotte, and A. Kemper. Heuristic and randomized optimization for the join ordering problem. *VLDB Journal*, 6(3):191–208, 1997.
20. Sybase. *Sybase Adaptive Server Enterprise 15 Data Partitioning*. White paper, 2005.

6 Appendix

```
--Q1
select Time_level,count(*) from ACTVARS A, PRODLEVEL P
where A.product_level=P.code_level and P.Class_level='ADX8MBFPWVIV' group by Time_level;
--Q2
select line_level,sum(Dollarsales) from ACTVARS A,PRODLEVEL P
where A.product_level=P.code_level and P.Class_level='OC2WOOC8QIJ6' group by line_level;
--Q3
select count(*) from ACTVARS A,PRODLEVEL P
where A.product_level=P.code_level and P.Class_level='LB2RKO0ZQCJD';
--Q4
select Time_level,Avg(UNITSSOLD) from ACTVARS A,Timelevel T
where A.time_level=T.tid and T.Quarter_level='Q1' group by Time_level;
--Q5
select division_level,count(*) from ACTVARS A, PRODLEVEL P
where A.product_level=P.code_level and P.group_level='VTL9D0E3RSWQ' group by division_level;
--Q6
select Max(UNITSSOLD) from ACTVARS A,Timelevel T
where A.time_level=T.tid and T.Quarter_level='Q2';
--Q7
select year_level,sum(Dollarsales) from Actvars A, Prodlevel P, Timelevel T
where A.product_level=P.code_level and A.time_level=T.tid
and P.family_level='0X3BXTCVRRKU' group by year_level;
--Q8
select count(*) from ACTVARS A,PRODLEVEL P
where A.product_level=P.code_level and P.family_level='M32G5M3AC4T5';
--Q9
select Customer_level,Avg(Unitssold) from ACTVARS A,  PRODLEVEL P
where A.product_level=P.code_level and P.family_level='Y45VKMTJDNYR' group by Customer_level;
--Q10
select month_level,count(*) from ACTVARS A,Timelevel T
where A.time_level=T.tid and T.Quarter_level='Q3' group by month_level;
--Q11
select Sum(Dollarsales) from ACTVARS A,PRODLEVEL P
where A.product_level=P.code_level and P.line_level = 'KYNFXX46DWNI';
--Q12
select Product_level,count(*) from ACTVARS A,Timelevel T
where A.time_level=T.tid and T.Quarter_level='Q4' group by Product_level;
--Q13
select count(*) from ACTVARS A,PRODLEVEL P
where A.product_level=P.code_level and P.division_level = 'SIQV916OFEWC';
--Q14
select Customer_level,Sum(Dollarsales)from ACTVARS A,PRODLEVEL P
where A.product_level=P.code_level and P.division_level = 'BJE2TCMX5JDS' group by Customer_level;
--Q15
select count(*) from ACTVARS A,TIMELEVEL T
where A.time_level=T.tid and T.year_level = '1995';
--Q16
select Product_level,Sum(Dollarsales) from ACTVARS A,   TIMELEVEL T
```

```
where A.time_level=T.tid and T.year_level = '1996' group by Product_level;
--Q17
select division_level,Avg(Unitssold) from Actvars A,Timelevel T ,Prodlevel P
where A.time_level=T.tid  AND A.product_level=P.Code_level  and T.month_level = '1' group by division_level;
--Q18
select count(*) from ACTVARS A,TIMELEVEL T
where A.time_level=T.tid and T.month_level = '2';
--Q19
select Product_level,count(*) from Actvars A,  Timelevel T
where A.time_level=T.tid and T.month_level = '3'  group by Product_level;
--Q20
select division_level,Sum(Dollarsales) from Actvars A,Timelevel T,Prodlevel P
where A.time_level=T.tid  and A.product_level=P.Code_level  and  T.month_level = '4' group by division_level;
--Q21
select Avg(Unitssold) from ACTVARS A,TIMELEVEL T
where A.time_level=T.tid and T.month_level = '5';
--Q22
select Product_level,count(*) from ACTVARS A,  TIMELEVEL T
where A.time_level=T.tid and T.month_level = '6'  group by Product_level;
--Q23
select division_level,Sum(Dollarsales) from Actvars A,Timelevel T ,Prodlevel P
where A.time_level=T.tid and A.product_level=P.Code_level and T.month_LEVEL = '7' group by division_level;
--Q24
select Customer_level,count(*) from ACTVARS A,TIMELEVEL T
where A.time_level=T.tid and T.month_level = '9' group by Customer_level;
--Q25
select year_level,Sum(Dollarsales) from ACTVARS A,TIMELEVEL T
where A.time_level=T.tid and T.month_LEVEL = '10' group by year_level;
--Q26
select count(*) from ACTVARS A,TIMELEVEL T
where A.time_level=T.tid and T.month_level = '11';
--Q27
select Product_level,Time_level,Avg(unitssold) from Actvars A,Timelevel T
where A.time_level=T.tid and T.month_level = '12'
group by Product_level,Time_level;
--Q28
select year_level,month_level, Max(unitssold)
from Actvars A,Custlevel C ,Timelevel T
where A.customer_level=C.store_level and A.time_level=T.tid and
C.retailer_level = 'AB7D9LJB7BR9'  group by year_level,month_level;
--Q29
select count(*) from ACTVARS A,CUSTLEVEL C
where A.customer_level=C.store_level and C.retailer_level = 'F92YG4VUUA8A';
--Q30
select Product_level, SUM(unitssold) from Actvars A, Custlevel C
where A.customer_level=C.store_level and C.retailer_level = 'RVC9OK3MWOFA' group by Product_level;
--Q31
select count(*) from ACTVARS A,CHANLEVEL CH
where A.channel_level=CH.base_level and CH.all_level ='AVC9OK3MWOFA';
--Q32
select count(*) from ACTVARS A,CHANLEVEL CH
where A.channel_level=CH.base_level and  CH.all_level ='DEFGHIJKLMNO';
--Q33
select year_Level,month_level, sum(dollarsales)
from Actvars A,Custlevel C,Prodlevel P ,Timelevel T
where A.customer_level=C.store_level  and A.time_level=T.tid  and A.product_level=P.code_level
and P.class_level='OC2WOOC8QIJ6' and C.retailer_level='F92YG4VUUA8A'  group by year_level,month_level;
--Q34
select sum(dollarsales) from ACTVARS A, PRODLEVEL P,TIMELEVEL T
where A.product_level=P.code_level and A.time_level=T.tid and T.quarter_level='Q2'  and T.year_level='1996' and P.class_level='LB2RKOOZQCJD';
--Q35
select Customer_Level, Time_level, Avg(Unitssold)
from Actvars A, Chanlevel H, Prodlevel P, Timelevel T,Custlevel C
where A.product_level=P.code_level and A.time_level=T.tid and A.channel_level=H.base_level and P.class_level='LB2RKOOZQCJD'
and H.all_level ='ABCDEFGHIJKL' and T.quarter_level='Q1'
group by Customer_level, Time_level;
--Q36
select year_Level, division_level, Max(Unitssold)
from Actvars A, Custlevel C,Timelevel T ,Prodlevel P
where A.customer_level=C.store_level  and A.product_level=P.Code_level  and A.time_level=T.tid and T.month_level='1'
and  C.retailer_level='F92YG4VUUA8A'  group by year_level, division_level;
--Q37
select sum(dollarsales), Avg(Unitssold) from Actvars A, Custlevel C,Timelevel T
where A.customer_level=C.store_level and A.time_level=T.tid and
T.month_level='12' and C.retailer_level ='F92YG4VUUA8A';
--Q38
select Time_level,Min(unitssold)
from Actvars A, Chanlevel H,Timelevel T, Custlevel C
where A.channel_level=H.base_level and A.time_level=T.tid and T.year_level='1996' AND T.quarter_level='Q3'
and H.all_level='DEJ9OEA8F40J' group by Time_level;
```