Algebra-based Approach for Incremental Data Warehouse Partitioning

Rima Bouchakri[1,2], Ladjel Bellatreche[1], and Zoé Faget[1]

LIAS/ENSMA – Poitiers University, Futuroscope, France
LCSI/ESI – High School of Computer Science – Algiers, Algeria
(rima.bouchakri, bellatreche, zoe.faget)@ensma.fr

Abstract. Horizontal Data Partitioning is an optimization technique well suited to optimize star-join queries in Relational Data Warehouses. Most works focus on a static selection of a fragmentation schema. However, due to the evolution of data warehouses and the ad hoc nature of queries, the development of incremental algorithms for fragmentation schema selection has become a necessity. In this work, we present a Fragmentation Algebra containing all operators needed to update a schema when a new query arrives. To identify queries which should trigger a schema update, we introduce the notion of query profiling.

1 Introduction

In the era of big data, there is a need to develop new models, data structures, algorithms, and tools for processing, analyzing, mining and understanding this huge amount of data using High Performance Computing (HPC). The diversity of HPC platforms contributes in developing a new issue related to the deployment of data on relevant platform to satisfy the requirements of end users in terms of query processing and data manageability. Horizontal Data Partitioning (HDP) is a pre-condition for deploying data on many platform: centralized [17], parallel [9], distributed [16], cloud [8], etc. The problem of horizontal data partitioning ($\mathcal{PHDP}$) has been largely studied in the literature in different database contexts: OLTP databases [15], data warehouses [3], scientific and statistical databases []. Horizontal data partitioning consists in fragmenting a table, an index or a materialized view, into partitions (fragments) [17]. Besides, many Database Management Systems DBMS implements it (Oracle, DB2, SQLServer, Sybase, PostgreSQL and MySQL). Two main types of horizontal partitioning are distinguished [5]: primary partitioning and derived partitioning. In the primary table partitioning, a table is partitioned according to its attributes. It optimizes selection operations and used in rewriting queries in distributed and parallel databases [16]. The referential partitioning allows partitioning a table according to the primary partitioning of another table if a relationship parent-child exists.

Several partitioning modes exist to support primary horizontal partitioning in centralized and parallel platforms: Range, List and Hash. These modes are combined to form the composite modes. The derived partitioning is also supported by commercial DBMS such as Oracle.

The $\mathcal{PHDP}$ is formalized as follows: given a database/data warehouse schema, a set of a priori known queries, and a partitioning constraints that limits the number of final fragments, the $\mathcal{PHDP}$ consists in fragmenting the tables of the given schema into sets of fragments such as the overall query processing cost is minimized and the number of the final fragments does not exceed the partitioning constraint. The obtained fragments are disjoint and the union of all fragments belonging to a set of fragments is equal to the schema of its corresponding table. By examining the literature, we get three main observations: (i) Most partitioning algorithms consider a well-known set of queries to perform a static selection. However, the ad-hoc nature of the OLAP and scientific queries calls for the development of incremental data partitioning algorithms. (ii) Partitioning algorithms use simple data structures. Business or scientific projects (such as Dark Energy Survey[1]) manage extremely large databases with hundreds of attributes candidates for partitioning process and need more sophisticated data structures. (iii) Direct deployment of the partitioning results is time consuming. The partitioning result needs to be rapidly deployed on the target platforms. To satisfy these objectives, some naive solutions may exist like adapting the existing partitioning solutions by incorporating a threshold indicating the time where the repartitioning is needed as in []. Our vision differs from the traditional approaches since we are driven by the previously mentioned observations. As a consequence, we propose a flexible coding that adapts to any partitioning schema. Using this coding, we develop an algebra where operators capture any change of the partitioning schema due to workload evolution. We also introduce the notion of query profiling used together with our algebra to derive a new fragmentation schema from the current one. The operators of this algebra are easily implemented in the target platform. For our study, we consider a data warehouse schema implemented in Oracle 11G.

2 Related work

Horizontal Data Partitioning is a non-redundant technique used in *physical design* of data warehouses. It is based on reorganization of data in order to optimize *star join queries*, which contain multiple complex joins and selection operations performed on a facts table and several dimension tables of a star-schema data warehouse. The horizontal data partitioning was first announced as a logical design technique of relational and object databases [14]. Due to the efficiency of this technique in optimizing data processing, it is used in physical design of data warehouse. Besides, many Database Management Systems DBMS implement it (Oracle, DB2, SQLServer, Sybase, PostgreSQL and MySQL). Commercial DBMS use referential partitioning to optimize star-join queries [12].

In order to optimize star join queries, which involve restrictions and joins, using *HDP*, authors in [3] show that the best partitioning scenario of a relational data warehouse is performed as follow : a primary partitioning of the dimension tables is performed, followed by a reference partitioning of the fact

[1] `http://www.darkenergysurvey.org`

table according to the dimension tables' fragmentation schema. Therefore, the horizontal data partitioning got a lot of attention from academic and industrial communities. We can classify these works into two categories according to the selection nature.

Static selection: This category includes the majority of HDP works. These works define a static selection that can't deal with changes occurring on the RDW, specially the execution of new queries that don't exist in the current workload. Four main approaches can be considered (a) minterm generation algorithms [5, 16], (b) affinity-based algorithms [3, 14] (c) cost-model driven algorithms [1, 6, 2, 11] and (d) data mining driven algorithms [13].

Incremental selection: Due to the evolution of data warehouse and the ad hoc nature of the queries entered online, the development of incremental algorithms for fragmentation schema selection becomes a necessity. Authors in [14] deal with distributed database redesign. They propose simple heuristics to deal with merging and splitting fragments. Authors in [18] propose a dynamic design of distributed data warehouses. This approach is composed of three concepts: (a) Dynamic process of extraction, transformation and load activated each time there is a change in data sources. (b) Bases of knowledge to store history of the data warehouse. (c) Dynamic process of fragmentation and replication activated each time there is a change in the workload. The main issue of this approach is that the fragmentation is triggered for each change which causes a high maintenance cost, and may be unnecessary some cases where a change in the workload eventually leads to the same schema. Authors in [10] propose to deal with workload evolution by defining a refragmentation approach of relational centralized data warehouses. This approach is based on storing recent statistical information. First, only the facts table is partitioned using the 'By Range' Oracle mode on one of its foreign keys. Second, histograms are built observing queries' access to different fragments. At a given time, using a cost model and the histograms, a refragmentation is performed if the evaluation of criterion representing the queries cost is satisfied. Then, the histograms are updated to store the occurred changes. In this work, the fragmentation of the fact table is performed By Range according to only one foreign key of one dimension table. But the OLAP queries contain complex star joins operations between facts table and many dimension tables. Therefore, a beneficial fragmentation should be performed according to many dimensions tables. Also, the refragmentation in based on the same fragmentation attribute. A refragmentation on another attribute could be more beneficial to deal with the workload evolution. In our work [4], we propose an incremental selection of fragmentation schema based on Genetic Algorithms GA that is triggered after every changes on the workload. As stated before, this causes a high maintenance cost and may be unnecessary in some cases. As a consequence, we propose a new incremental fragmentation approach that analyze first the query profile to determine if a refragmentation of the RDW is needed. Our approach is based on a Fragmentation Algebra that models all the operations required to update a fragmentation schema.

3 Fragmentation Algebra FA

In [4], we introduced an incremental selection of a Fragmentation Schema based on a flexible encoding. A current FS is represented by an array that contains the fragmentation attributes and their sets of sub-domains. An attribute associated to one of its sub-domain set make a selection predicate that is part of the fragmentation schema of a dimension table. We encode the schema by affecting a number to each sub-domain. The sub-domains with the same number will be merged into one sub-domain. This encoding is used in a selection algorithm based on Genetic Algorithm.

The main problem of the incremental approach that we presented in [4] is the computation of a new fragmentation schema using GA after *every* change that occurs on the workload. Since the HDP selection is NP-Hard, computing a new selection of FS costs time and resources. Each new query executed on the RDW can cause an extension or a reduction of the fragmentation schema by adding/deleting new attributes, adding/deleting new sub-domains or splitting/merging existing sub-domains. Also, new queries may have the same definition than existing ones. This means that some queries shouldn't trigger a new HDP selection. In order to determine the exact actions required after the execution of a given query, we define a Fragmentation Algebra that contains all possible operations on a fragmentation schema. We first define a flexible encoding of the fragmentation schema on which we can perform a Reduction or an Evolution. Then, we present the Algebra operators and their properties.

3.1 Flexible encoding

Let's consider a data warehouse RDW with a facts table F and d dimensions tables $D = \{D_1, D_2, \cdots, D_d\}$. A fragmentation schema is defined on non-key dimension attributes $A = \{A_1, \cdots, A_n\}$. Each attribute A_i has a set of possible values called Attribute Domain Dom(A_i). Dom(A_i) can be partitioned into m_i sub-domains $Dom(A_i) = \{SD_1^i, SD_2^i, \cdots, SD_{m_i}^i\}$. For instance, if we consider the attribute City of a given table Clients, the domain partitioning is given as follows:

- Dom(City)= {'Alger', 'Oran', 'Blida', 'Kala', 'Annaba', 'Jijel'}

Thus, an attribute and a sub-domain make a selection predicate used to specify the fragmentation schema of a dimension table in a RDW. According to this, we define a *Data Structure* that represents the Maximal Fragmentation Schema MFS of dimension tables (table 1). The number of fact table fragments is the product of dimensions fragments numbers ($\prod_1^n m_i$).

3.2 Schema reduction and evolution

In Data Warehousing context, the number of fragmentation attributes is very large (tens or hundreds of attributes). Moreover, the amount of data stored in a RDW is huge. As a consequence, the number of facts table fragments in the MFS is very large too. Suppose a MFS defined on 30 attributes, where

A_1	SD_1^1	SD_2^1	$\cdots$	$SD_{m_1}^1$
A_2	SD_1^2	SD_2^2	$\cdots$	$SD_{m_2}^2$
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
A_n	SD_1^n	SD_2^n	$\cdots$	$SD_{m_n}^n$

Table 1. Maximal Fragmentation Schema MFS

each attribute has 10 sub-domains. The number of facts table fragments is $\prod_1^n m_i = \prod_1^{30} 10 = 10^{30}$, which is impossible to manage. Therefore, a fragmentation schema can be not maximal by merging sub-domains or excluding some attributes from the fragmentation process. We can obtain a reduced fragmentation schema as illustrated in table 2. We denote by $Else_i$ all other values of the attribute A_i not specified in the sub-domains. For the fragmentation schema of the table 2, the sets $Else_i$ are specified as follows:

- $Else_1 = \{SD_1^1, \cdots, SD_{m_1}^1\} \setminus \{SD_1^1, SD_2^1, SD_3^1, SD_5^1, SD_4^1, SD_6^1\}$
- $Else_2 = \{SD_1^2, \cdots, SD_{m_2}^2\} \setminus \{SD_1^2, SD_2^2\}$
- $Else_3 = \{SD_1^3, \cdots, SD_{m_3}^3\} \setminus \{SD_1^3, SD_3^3, SD_5^3, SD_7^3, SD_9^3, SD_8^3\}$
- $Else_4 = \{SD_1^4, \cdots, SD_{m_4}^4\} \setminus \{SD_1^4, SD_2^4\}$

A_1	SD_1^1	SD_2^1, SD_3^1, SD_5^1	SD_4^1, SD_6^1	$Else_1$	
A_2	SD_1^2	SD_2^2	$Else_2$		
A_3	SD_1^3, SD_3^3	SD_5^3	SD_7^3, SD_9^3	SD_8^3	$Else_3$
A_4	SD_1^4	SD_2^4	$Else_4$		

Table 2. Reduction of the fragmentation schema MFS to FS

The transition from the MFS schema to the FS schema is called Reduction of the Fragmentation Schema (RFS). It includes the following operations: (1) remove the attributes $A_5, \cdots, A_n$, (2) merge the sub-domains of the attributes A_1, A_2, A_3 and A_4. On the other hand, the fragmentation schema can evolve by adding new fragmentation attributes or splitting the different sets of sub-attributes. We present in the table 3 the evolution of the fragmentation schema FS given in the table 2.

The transition from the schema FS to the schema FS' is called Evolution of the Fragmentation Schema (EFS). EFS is a dual operation of the reduction operation that involves the following operations: (1) add the attribute A_5 and the sub-domains $Dom(A_5) = \{SD_1^5, \cdots, SD_{m_5}^5\}$, (2) merge the sub-domains of A_5 into two sets $\{SD_1^5, SD_2^5\}$ and $Else_5 = \{SD_1^5, \cdots, SD_{m_5}^5\} \setminus \{SD_1^5, SD_2^5\}$, (3) split the set of sub-domains of A_3 $\{SD_7^3, SD_9^3\}$ into two sets, (4) merge the two sets $\{SD_5^3\}$ and $\{SD_7^3\}$ into $\{SD_5^3, SD_7^3\}$ and (5) merge the two sets $\{SD_8^3\}$ and $\{SD_9^3\}$ into $\{SD_8^3, SD_9^3\}$

A_1	SD_1^1	SD_2^1, SD_3^1, SD_5^1	SD_4^1, SD_6^1	$Else_1$
A_2	SD_1^2	SD_2^2	$Else_2$	
A_3	SD_1^3, SD_3^3	SD_5^3, SD_7^3	SD_9^3, SD_8^3	$Else_3$
A_4	SD_1^4	SD_2^4	$Else_4$	
A_5	SD_1^5, SD_2^5	$Else_5$		

Table 3. Evolution of the fragmentation schema FS to FS'

3.3 Operators description

Let FS a fragmentation schema. In order to perform an evolution EFS or a reduction RFS, we define a set of operators that represent *an Algebra of Fragmentation AF*. We consider the attribute A_i where $1 < i < n$, n the number of different attributes, and m_i the number of sub-domains of A_i. Each operator takes a fragmentations schema FS as input and produces a fragmentation schema FS'

- $Add\_A(A_i, \{SD^i_{j_1}, \cdots, SD^i_{j_p}\})(FS)$: add the attribute A_i to the fragmentation schema FS including the set of sub-domains $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$, which implies creating the set $Else_i = \{SD^i_1, \cdots, SD^i_{m_i}\} \setminus \{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$.
- $Add\_SD(A_i, \{SD^i_{j_1}, \cdots, SD^i_{j_p}\})(FS)$: add to the attribute A_i a set of sub-domains $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$ and delete it from the set $Else_i$.
- $Split\_Dom(A_i, \{SD^i_{j_1}, \cdots, SD^i_{j_p}\}, \{SD^i_{k_1}, \cdots, SD^i_{k_s}\})(FS)$: split the set of sub-domains $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$ of the attribute A_i into two sets of sub-domains $\{SD^i_{k_1}, \cdots, SD^i_{k_s}\}$ and $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\} \setminus \{SD^i_{k_1}, \cdots, SD^i_{k_s}\}$, where $\{k_1, \cdots, k_s\} \subset \{j_1, \cdots, j_p\}$.
- $Merge\_Dom(A_i, \{SD^i_{j_1}, \cdots, SD^i_{j_p}\}, \{SD^i_{k_1}, \cdots, SD^i_{k_s}\})(FS)$: merge the two sets $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$ and $\{SD^i_{k_1}, \cdots, SD^i_{k_s}\}$ into one, where $\{j_1, \cdots, j_p\} \subset [1, m_i]$ and $\{k_1, \cdots, k_s\} \subset [1, m_i]$.
- $Del\_A(A_i)(FS)$: delete the attribute A_i from the fragmentation schema FS.
- $Del\_SD(A_i, \{SD^i_{j_1}, \cdots, SD^i_{j_p}\})(FS)$: delete from the attribute A_i the set containing the sub-domains $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$ and include it in the set $Else_i$.

Using this Algebra, we can express the schema evolution EFS of the schema FS (table 2) into the schema FS' (table 3) as follows:

$FS' = EFS(FS) =$
$Merge\_Dom(A_3, \{SD^3_8\}, \{SD^3_9\}) \circ Merge\_Dom(A_3, \{SD^3_5\}, \{SD^3_7\}) \circ$
$Split\_Dom(FS, A_3, \{SD^3_7, SD^3_9\}) \circ Add\_A(A_5, \{SD^5_1, SD^5_3\})(FS)$.

We can classify these algebra's operations into two categories:
1. Evolution operations: the operations required to perform an EFS are $Add\_A()$, $Add\_SD()$ and $Split\_Dom()$.
2. Reduction operations: the operations required to perform a RFS are $Del\_A()$, $Del\_SD()$ and $Merge\_Dom()$.

Another classification would be between vertical operations (Add_A, Del_A) and horizontal operations ($Add\_SD, Split\_SD, Merge\_Dom, Del\_Dom$).

3.4 Operators properties

We now give notable properties of the previously introduced operators. Those properties will be useful for optimization purposes such as rewriting of operations, query scheduling or discarding mutually canceling operations.

In all that follows, we assume the operators make sense on a given current schema.

Inverse operators We introduce the identity operator $Id(FS)$ (which leaves the fragmentation schema unchanged), as the identity element of our algebra.

1. Del_A (resp. Add_A) is the left (resp. right) inverse of Add_A (resp. Del_A). The two operators are not commutative in the general case.
   $Del\_A \circ Add\_A = Id$.
2. $Split\_Dom(A_i, Set_1, Set_2)$ and $Merge\_Dom(A_i, Set_1, Set_2)$ are inverse operators.
3. $Del\_SD(A_i, \{SD^i_j\})$ and $Add\_SD(A_i, \{SD^i_j\})$ are inverse operators.

Equivalence rules

1. Operators involving different attributes, or involving the same attribute but different subdomains, are commutative.
2. $Merge\_Dom(A_i, Set_1, Set_2) \circ Add\_SD(A_i, Set_2) \circ Add\_SD(A_i, Set_1)$ is equivalent to $Add\_SD(A_i, Set_1 \cup Set_2)$.
3. More generaly, a sequence of Add_SD operations ending with the corresponding $Merge\_Dom$ operation is equivalent to adding the union of subdomains.
4. Deleting a set of subdomains of A_i is equivalent to merging the set with the current $Else_i$.
   $Del\_SD(A_i, Set_1) = Merge\_Dom(A_i, Set_1, Else_i)$
5. Deleting an attribute is equivalent to successively merging all subdomains of this attribute.
   $Del\_A(A_i) = Merge\_Dom(A_i, SD^i_1, Else_i) \circ \ldots \circ Merge\_Dom(A_i, SD^i_{m_i}, Else_i)$

4 Queries Profiling

When new queries are executed on the RDW, the fragmentation schema may be updated in order to take into account the workload changes. According to the executed queries, the fragmentation schema can be updated using a reduction RFS, an Evolution EFS or both. If the definition of the executed queries is similar to the current workload, no changes are required. In order to determine the required operations to adapt a fragmentation schema to the workload evolution, we analyze the new executed query to determine all the Algebra operations required. We give the general description of a star join query as follows:

```
SELECT *
FROM F, D1, D2, ..., Dd
WHERE F.ID1=D1.ID1 AND F.ID2=D2.ID2 ... AND F.IDd=Dd.IDd
AND (A1 op V11 OR A1 op V12 ... OR A1 op V1k1)
AND (A2 op V21 OR A2 op V22 ... OR A2 op V1k2) ...
```

```
AND (An op Vn1 OR An op Vn1 ... OR An op Vnkn)
[ORDER BY ... ]
[GROUP BY ... ]
[HAVING ... ]
```

The fragmentation schema is defined on the fragmentation attributes and their sub-domains appearing in the selection predicates of the WHERE clause. When executing a new query, changes are defined by the selection predicates A_i op V_{ij}. Each attribute's value V_{ij} can equal or be contained in a sub-domain SD^i_j. Therefore, the general expression of a selection predicate is A_i op $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$. Let's consider a new query Q executed on a data warehouse partitioned according to a fragmentation schema FS. The execution of Q may require adding new attributes, new sub-domains contained in $Else_i$, merging or splitting sub-domains' sets and/or deleting infrequent attributes or sub-domains. We study the possible Algebra Operations induced by the selection predicates A_i op $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$.

- A_i doesn't appears in FS : this attribute is added to the fragmentation schema by the operation $Add\_A(A_i, \{SD^i_{j_1}, \cdots, SD^i_{j_p}\})(FS)$.
- A_i appears in FS: We verify if the set of sub-domains $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$ requires Algebra operations.
- All sub-domains contained in $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$ appear as a set in FS: no operations.
- The set $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$ is included in a set $\{SD^i_{L_1} \cdots, SD^i_{L_m}\}$ in FS: This set is split using the operation $Split\_Dom(A_i, \{SD^i_{L_1}, \cdots, SD^i_{L_m}\}, \{SD^i_{j_1}, \cdots, SD^i_{j_p}\})(FS)$
- All sub-domains contained in $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$ appear in t sub-sets $SubSet_1, \cdots, SubSet_t$ in FS, where $SubSet_1 \cup SubSet_2 \cup \cdots \cup SubSet_t = \{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$: the t seb-sets are merged by the operation $Merge\_Dom(A_i, SubSet_1) \circ Merge\_Dom(A_i, SubSet_2) \circ$ $... \circ Merge\_Dom(A_i, SubSet_t)(FS)$.
- A sub-set of sub-domains $\{SD^i_{k_1}, \cdots, SD^i_{k_s}\}$ doesn't appear in FS, where $\{SD^i_{k_1}, \cdots, SD^i_{k_s}\} \subset \{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$: the sub-domains contained in this sub-set are all in $Else_i$. The sub-set is added to FS using the operation $Add\_SD(A_i, \{SD^i_{k_1}, \cdots, SD^i_{k_s}\})(FS)$
- There in no sub-domain of $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$ that appears in FS: the sub-domains are all in $Else_i$. The set is added to FS using the operation $Add\_SD(A_i, \{SD^i_{j_1}, \cdots, SD^i_{j_p}\})(FS)$.
- An attribute A_j is no more frequently used by the workload: for each attribute, we calculate the use rate by the workload. If the attribute is used by less then 20% of the workload, it's deleted using the operation $Del\_A(A_j)(FS)$.
- A set of sub-domains $\{SD^i_{R_1}, \cdots, SD^i_{R_h}\}$ of the attribute A_i is no more frequently used by the workload: for each set, we calculate the use rate by the workload. If the set is used by less then 20% of the workload, it's removed using the operation $Del\_SD(A_i, \{SD^i_{j_1}, \cdots, SD^i_{j_p}\})(FS)$.
- All the sub-domains of A_i are merged into one set to form the set $Else_i$: this may happen after operations $Merge\_Dom()$ and/or $Del\_SD()$ were applied

In this case no fragmentation is defined on A_i. This attribute is removed using the operation $Del\_A(A_j)(FS)$.

Once we know all possible Algebra Operationsinduce by the query Q, we deduce if Q causes a RFS, an EFS, both or none. As a consequence, we elaborate four query profiles:

1. **Evolution queries:** this profile describes queries that require the operations $Add\_A()$, $Add\_SD()$ and/or $Split\_Dom()$. When an Evolution query is executed on the RDW, an EFS of the current RDW schema is needed. As consequence, the number of facts table fragments will increase.
2. **Reduction queries:** this profile describes queries that require the operations $Del\_A()$, $Del\_SD()$ and $Merge\_Dom()$. When a Reduction query is executed on the RDW, an RFS of the current RDW schema is needed. Therefore, the number of facts table fragments will decrease.
3. **Mixed queries:** a Mixed query implies both Evolution and Reduction operations ($Add\_A()$, $Add\_SD()$, $Split\_Dom()$, $Del\_A()$, $Del\_SD()$ and $Merge\_Dom()$). The number of facts table fragments can either increase or decrease.
4. **Neutral queries:** a Neutral query doesn't affect the current RDW fragmentation schema. Let's consider a selection predicate A_i op $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$ in a query. This query is neutral if A_i appears in FS and all sub-domains, contained in $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$, appear as a set in FS or the operations defined leaves the fragmentation schema unchanged. For instance, if the query requires two inverse operators like Del_A and Add_A, the identity operator $Id(FS)$ is obtained which has no effect on the fragmentation schema. Two operators Del_A (resp. Del_SD) and Add_A (resp. Add_SD) can be obtained, when executing one query, if the attribute A (resp. the set $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$) is no more frequently used by the workload which requires the operator Del_A (resp. Del_SD) but a selection predicate is defined on the attribute A (resp. the set $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$) which generate the operator Add_A (resp. Add_SD).

Example 1. Let's consider a data warehouse with a facts table Sales and two dimension tables: Client and Product. We give the current fragmentation schema $FS1$ of the RDW in table 4. We consider four queries. For each query we give its description, its profile and the Algebra's operations required to adapt the current fragmentation schema $FS1$ to the changes given by each query (table 5).

Gender	M	F		
City	Alger	Oran	$Else_2$	

Table 4. Example of a fragmentation schema $FS1$

Query	Algebra Operations	Profile
SELECT * FROM Client C, Product P, Sales S WHERE S.IdC=C.Id And S.IdP=P.Id And C.City='Blida' And P.PName='P1'	Add_Dom(City, {Blida})(FS1) Add_A(PName, {P1})(FS1)	Evolution
SELECT * FROM Client C, Sales S WHERE S.IdC=C.Id And C.City='Alger' Or C.City='Oran'	Merge_Dom(City, {Alger}, {Oran})(FS1)	Reduction
SELECT * FROM Client C, Product P, Sales S WHERE S.IdC=C.Id And S.IdP=P.Id And C.City='Alger' Or C.City='Oran' And P.PName='P1'	Merge_Dom(City, {Alger}, {Oran})(FS1) Add_A(FS1, PName, {P1})	Mixed
SELECT * FROM Client C, Sales S WHERE S.IdC=C.Id And C.City='Alger'	/	Neutral

Table 5. Queries Profiles

5 Incremental selection of Fragmentation Schema based on Queries Profiling $GAQP$

We assume that a new query Q_i is executed on a partitioned RDW. In order to adapt the current fragmentation schema of the data warehouse, we analyze Q_i using our Algebra in order to determine the query profile. According to the profile, we decide of the physical operations to perform in order to update the fragmentation schema. We present an algorithm that summarizes the Incremental selection of fragmentation schema using query profiling. This algorithm uses the classic fragmentation schema selection based on Genetic Algorithms that we presented in our previous work [4].

In order to illustrate our Incremental selection based on query profiling, we present an architecture that summarizes the steps to perform when the workload evolves (figure 1). When a new query Q_i is executed on the RDW, its profile is determined based on the Algebra. If the query profile is "Neutral" or "Reduction", no selection and no implementation on the RDW is required, since cost gain will be marginal compared to the time needed to select and implement a new schema. However, if the query profile is "Evolution" or "Mixed", a new fragmentation schema $NewFS$ is computed. Then, if the $NewFS$ has a number of fact fragments that violates the constraint B, a new selection of fragmentation schema based on genetic algorithm is performed. Finally, the obtained fragmentation schema is implemented on the RDW.

Incremental Selection of FS based on Queries Profiling
Input:
$Algebra$: a set of the Algebra operators
Q : the workload containing m queries
Q_i : the new executed query
FS : the current fragmentation schema of the RDW
RDW : data that compose the Cost Model used in the Genetic Algorithms
B : maximum number of fact fragments
Output: Fragmentation schema of the dimensions tables $NewFS$.
Notations:
AnalyseQProfile: returns the query profile according to the Algebra of Fragmentation
ComputeNewFS: returns a new FS using the current FS and the new query Q_i
FragmentationSelectionGA: Select an Optimal FS using Genetic Algorithms
NBfragments: Compute the number of facts fragments generated by the entered FS
Begin
QueryProfil ← AnalyseQProfile(Algebra, Q_i);
if QueryProfil="Neutral" **then**
Break; {*End Algorithm*}
end if

NewFS ← ComputeNewFS(FS, Q_i);
if QueryProfil="Evolution" or QueryProfil="Mixed" **then**
if NBfragments($NewFS$)> B **then**
$NewFS$ ← FragmentationSelectionGA($Q \cup \{Q_i\}$, FS, RDW, B);
end if
end if
End

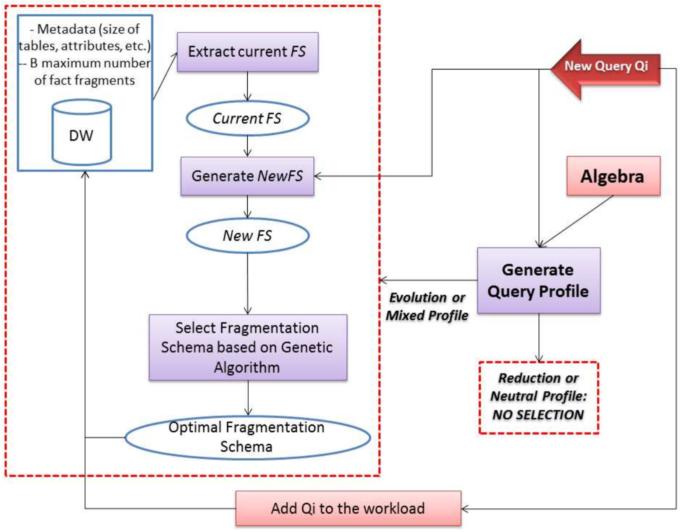

Fig. 1. Architecture of Incremental Fragmentation Selection based on Queries Profiling

6 Portability of the Fragmentation Algebra under Oracle 11g

We consider a data warehouse partitioned according to a fragmentation schema FS. When updating a schema FS into a new schema FS' by an EFS or a RFS, it's important to know the physical operations to perform in order to physically implement the schema FS' on the RDW. Since the physical operations depend on the Database Management System, we consider the $DBMS$ Oracle 11g[2]

Given a fragment P, the physical operations that could be performed are given as follows:

- **Split(P, Ct)**: horizontal fragmentation of P into two disjoint fragments $P1$ and $P2$ according to the criteria Ct. Let's consider P a fragment of the table Client. Split(P, City='Algiers') generates $P1$ that contain clients of the city Algiers and $P2$ that contains the other clients.
- **Merge(P1, P2)**: merge $P1$ and $P2$ into one fragment.
- **Move(P, TBS)**: move P to the table-space TBS, if there is no more space for P in the current tablespace. This operation is performed after a Merge fragment, or if new tuples are inserted into P.
- **Extend(P)**: extend the fragment P.
- **Create(P)**: create a new fragment to store new inserted instances that don't belong to any existing fragment.

[2] Managing Partitioned Tables and Indexes `http://docs.oracle.com/cd/B10501_01/server.920/a96521/partiti.htm`

In the general case, evolution in the data warehouse could be a workload evolution or a data evolution. Each evolution requires specific operations.

– **Data Evolution** means adding new tuples into the RDW. As a consequence, fragments could be extended, moved into another table-space or created. The physical operations required are Move, Extend and Create.
– **Workload Evolution** causes adding new fragmentation attributes and domains or changing the use frequency of attributes by queries. As a result, implementing a new fragmentation schema on a partitioned RDW required merging/splitting fragments and moving fragments because of merging operations (operations Split, Merge and Move).

In this work we focused on Workload evolution. We need the three operations Split, Merge and Move. For each operation we give the corresponding syntax in Oracle 11g. In what follows, P is a fragment or a table.

– **Split(P, Ct)**: Suppose a partitioned table Client to split. The SQL syntax for this operation is:

```
ALTER TABLE Client
   SPLIT PARTITION City VALUES ('Alger')
   INTO
    ( PARTITION Client1, PARTITION Client2 );
```

– **Merge(P1, P2)**: The SQL syntax for this operation is given as follows:

```
ALTER TABLE Client
    MERGE PARTITIONS Client1, Client2 INTO PARTITION Client3;
```

– **Move(P, TBS)**: The SQL syntax for this operation is given as follows:

```
ALTER TABLE Client
       MOVE PARTITION Client1 TABLESPACE tbs_2;
```

According to this, we interpret the Algebra of Fragmentation in the physical level. For each operation on the Algebra, we give the physical operations needed to physically implement the Algebra operation on the RDW. For this purpose, we define two new operations $Identify\_Part(Ct)$ and $Mergeable(P1, P2, A_i)$ given as follows:

– $Identify\_Part(Ct)$: Identify the fragment characterized by the predicates specified in Ct. Assuming the following operation $Identify\_Part("City = Algiers")$. This operation returns a set of fragments P1, ..., PL which satisfy the criterion $"City = Algiers"$.
– $Mergeable(P1, P2, A_i)$: boolean function that returns "true" if the two fragments P1 and P2 can be merged on the attribute A_i. It returns 'false' otherwise. Two fragments are called mergeable if they are identified by the same combination of selection predicates except for a single predicate. This predicate is defined on the merge attribute A_i.

Example 2. Considering a partitioned dimension Client according to the fragmentation schema shown in figure 2. The fragmentation attributes and domains are given as follows:
- Dom(Gender)= 'F','M'
- Dom(City)= 'Alger', 'Oran', 'Blida', 'Kala', 'Annaba', Jijel'
We apply the operations $Identify\_Part(Ct)$ and $Mergeable(P1, P2, A_i)$ on

(a)

Gender	M	F
City	Alger, Oran, Blida	$Reste_2$

(b)

Client1				
Id	...	City	Gender	Job
10		Alger	M	M1
4		Alger	M	M3
6		Oran	M	M3
5		Blida	M	M2

Client2				
Id	...	City	Gender	Job
1		Oran	F	M1
12		Oran	F	M2
2		Blida	F	M1
8		Blida	F	M3

Client3				
Id	...	City	Gender	Job
9		Kala	M	M4
3		Kala	M	M3
7		Jijel	M	M5

Client4				
Id	...	City	Gender	Job
11		Annaba	F	M4
13		Kala	F	M5

Fig. 2. (a) Fragmentation schema FSc, (b) Partitioned dimension Client

the Client schema of figure 2.(b). The results are given as follows:
- $Identify\_Part(City = Blida)$ = Client1, Client2.
- $Identify\_Part(City = Alger)$ = Client1.
- $Identify\_Part(Gender = F)$ = Client2, Client4.
- $Mergeable(Client1, Client2, Gender)$ = True.
- $Mergeable(Client1, Client2, City)$ = False.
- $Mergeable(Client1, Client3, City)$ = True.
- $Mergeable(Client1, Client4, Gender)$ = False.
- $Mergeable(Client1, Client4, City)$ = False.

We present the physical implementation of the Algebra's operations.

1. $Add\_A(FS, A_i, \{SD^i_{j_1}, \cdots, SD^i_{j_p}\})$: add an attribute A_i requires identifying the fragment(s) containing one or more sub-domains from the set $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$, then partition these fragments into two sub-fragments each, where the first sub-fragment is identified by the selection predicate A_i in $(SD^i_{j_1}, \cdots, SD^i_{j_p})$ and the second one in is defined by the predicate A_i in $(Else_i)$. We give the corresponding algorithm.

   $Ens\_Part = Identify\_Part(A_i \text{ in } (SD^i_{j_1}, \cdots, SD^i_{j_p}))$
   for each P in Ens_Part **do**
       Split(P, A_i in $(SD^i_{j_1}, \cdots, SD^i_{j_p})$)
   end for

Example 3. Considering the fragmentation schema of the table Client FSc presented in figure 2.(a). We perform the Evolution of FSc into $FSc1$. $FSc1 = EFS(FSc) = Add\_A(FSc, Job, \{M1, M2\})$. Where Dom(Job)={'M1', 'M2', 'M3', 'M4', 'M5'}. The new scheme $FSc1$ is given in figure 3. We give the algorithm execution result :

$Ens\_Part = Identify\_Part(Job \text{ in } (M1, M2)) = \{Client1, Client2\}$
$Split(Client1, Job \text{ in } (M1, M2))$
$Split(Client2, Job \text{ in } (M1, M2))$

(a)

Gender	M	F
City	Alger, Oran, Blida	$Reste_2$
Job	M1, M2	$Reste_3$

(b)

Client11				
Id	...	City	Gender	Job
10		Alger	M	M1
5		Blida	M	M2

Client21				
Id	...	City	Gender	Job
1		Oran	F	M1
12		Oran	F	M2
2		Blida	F	M1

Client12				
Id	...	City	Gender	Job
4		Alger	M	M3
6		Oran	M	M3

Client22				
Id	...	City	Gender	Job
8		Blida	F	M3

Client3				
Id	...	City	Gender	Job
9		Kala	M	M4
3		Kala	M	M3
7		Jijel	M	M5

Client4				
Id	...	City	Gender	Job
11		Annaba	F	M4
13		Kala	F	M5

Fig. 3. (a) $FSc1$: EFS on FSc, (b)Partitioned table Client according to $FSc1$

2. $Add\_SD(FS, A_i, SD^i_j)$: this operation requires partitioning all fragments identified by the predicate $A_i = SD^i_j$ into two fragments. The corresponding algorithm is given as follows:

   $Ens\_Part = Identify\_Part(A_i = SD^i_j)$
   for each P in Ens_Part **do**
       Split(P, $A_i = SD^i_j$)
   end for

   Example 4. Consider the following schema evolution: $FSc2 = EFS(SFc1) = Add\_SD(FSc1, City, \{Kala\})$. The fragmentation schema update is given in the figure 4. The corresponding physical operation are given in the algorithm execution result.

   $Ens\_Part = Identify\_Part(City = Kala) = \{Client3, Client4\}$
   $Split(Client3, City = Kala)$
   $Split(Client4, City = Kala)$

3. $Split\_Dom(FS, A_i, \{SD^i_{j_1}, \cdots, SD^i_{j_p}\}, \{SD^i_{k_1}, \cdots, SD^i_{k_s}\})$: split the set of subdomains $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$ of the attribute A_i into two sets of sub-domains

(a)

Gender	M	F	
City	Alger, Oran, Blida	Kala	Reste$_2$
Job	M1, M2	Reste$_3$	

(b)

Client11

Id	…	City	Gender	Job
10		Alger	M	M1
5		Blida	M	M2

Client21

Id	…	City	Gender	Job
1		Oran	F	M1
12		Oran	F	M2
2		Blida	F	M1

Client12

Id	…	City	Gender	Job
4		Alger	M	M3
6		Oran	M	M3

Client22

Id	…	City	Gender	Job
8		Blida	F	M3

Client31

Id	…	City	Gender	Job
9		Kala	M	M4
3		Kala	M	M3

Client41

Id	…	City	Gender	Job
13		Kala	F	M5

Client32

Id	…	City	Gender	Job
7		Jijel	M	M5

Client42

Id	…	City	Gender	Job
11		Annaba	F	M4

Fig. 4. (a) $FSc2$: EFS on $FSc1$, (b) Partitioned table Client according to $FSc2$

$\{SD^i_{k_1}, \cdots, SD^i_{k_s}\}$ and $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\} \backslash \{SD^i_{k_1}, \cdots, SD^i_{k_s}\}$, where $\{k_1, \cdots, k_s\} \subset \{j_1, \cdots, j_p\}$. Physically, this operation is performed by splitting the fragment(s) identified by the predicate (A_i in $(SD^i_{k_1}, \cdots, SD^i_{k_s})$)

$Ens\_Part = Identify\_Part(A_i$ in $(SD^i_{k_1}, \cdots, SD^i_{k_s}))$
for each P in Ens_Part **do**
  Split(P, A_i in $(SD^i_{k_1}, \cdots, SD^i_{k_s})$)
end for

Example 5. Consider the following schema evolution:
$FSc3 = EFS(SFc2) = Split\_Dom(FSc2, \{Alger, Oran, Blida\}, \{Blida\})$. The fragmentation schema $FSc3$ is given in the figure 5. The corresponding physical operations are given in the algorithm execution result.

$Ens\_Part = Identify\_Part(City = Blida) = \{Client11, Client21, Client22\}$
$Split(Client11, City = Blida)$
$Split(Client21, City = Blida)$
$Split(Client22, City = Blida)$

4. $Merge\_Dom(FS, A_i, \{SD^i_{j_1}, \cdots, SD^i_{j_p}\}, \{SD^i_{k_1}, \cdots, SD^i_{k_s}\})$: Merge the two sets $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$ and $\{SD^i_{k_1}, \cdots, SD^i_{k_s}\}$ into one, where $\{j_1, \cdots, j_p\} \subset [1, m_i]$ and $\{k_1, \cdots, k_s\} \subset [1, m_i]$. This operation required merging the fragment identified by (A_i in $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$) with those identified by (A_i in $\{SD^i_{k_1}, \cdots, SD^i_{k_s}\}$) if they are mergeable. Non-mergeable fragments are left unchanged.

$Ens\_Part1 = Identify\_Part(A_i$ in $(SD^i_{j_1}, \cdots, SD^i_{j_p}))$
$Ens\_Part2 = Identify\_Part(A_i$ in $(SD^i_{k_1}, \cdots, SD^i_{k_s}))$

(a)

Gender	M	F		
City	Alger, Oran	Blida	Kala	$Reste_2$
Job	M1, M2	$Reste_3$		

(b)

Client111				
Id	..	City	Gender	Job
10		Alger	M	M1

Client211				
Id	...	City	Gender	Job
1		Oran	F	M1
12		Oran	F	M2

Client112				
Id	..	City	Gender	Job
5		Blida	M	M2

Client212				
Id	...	City	Gender	Job
2		Blida	F	M1

Client12				
Id	..	City	Gender	Job
4		Alger	M	M3
6		Oran	M	M3

Client22				
Id	...	City	Gender	Job
8		Blida	F	M3

Client31				
Id	..	City	Gender	Job
9		Kala	M	M4
3		Kala	M	M3

Client41				
Id	...	City	Gender	Job
13		Kala	F	M5

Client32				
Id	..	City	Gender	Job
7		Jijel	M	M5

Client42				
Id	...	City	Gender	Job
11		Annaba	F	M4

Fig. 5. (a) $FSc3$: EFS on $FSc2$, (b) Partitioned table Client according to $FSc3$

for each P1 in Ens_Part1 **do**
  for each P2 in Ens_Part2 **do**
    if $((Mergeable(P1, P2, A_i)$ and $(Ens\_Part1 \neq \oslash)$ and $(Ens\_Part2 \neq \oslash)$ **then**
      Merge(P1,P2)
      Ens_Part1 = Ens_Part1 - {P1}
      Ens_Part2 = Ens_Part2 - {P2}
    end if
  end for
end for

Example 6. Consider the following schema Reduction:
$FSc4 = RFS(SFc3) = Merge\_Dom(FSc3, \{Alger, Oran\}, \{Blida\})$. The fragmentation schema $FSc4$ is given in the figure 6. The corresponding physical operations are given in the algorithm execution result.

$Ens\_Part1 = Identify\_Part(City$ in $(Alger, Oran)) = \{Client111, Client211, Client12\}$,
$Ens\_Part2 = Identify\_Part(City = Blida) = \{Client112, Client212, Client22\}$,
$Mergeable(Client111, Client112, City)$ = True ,
$Merge(Client111, Client112)$,
$Ens\_Part2 = \{Client211, Client12\}, Ens\_Part2 = \{Client212, Client22\}$,
$Mergeable(Client211, Client22, City)$ = False,
$Mergeable(Client12, Client212, City)$ = False,
$Mergeable(Client12, Client22, City)$ = False,
$Mergeable(Client211, Client212, City)$ = True,

$Merge(Client211, Client212)$,
$Ens\_Part1 = \{Client12\}$, $Ens\_Part2 = \{Client22\}$,
$Mergeable(Client12, Client22, City) =$ False

(a)

Gender	M	F	
City	Alger, Oran, Blida	Kala	Reste$_2$
Job	M1, M2	Reste$_3$	

(b)

Client11

Id	…	City	Gender	Job
10		Alger	M	M1
5		Blida	M	M2

Client21

Id	…	City	Gender	Job
1		Oran	F	M1
12		Oran	F	M2
2		Blida	F	M1

Client12

Id	…	City	Gender	Job
4		Alger	M	M3
6		Oran	M	M3

Client22

Id	…	City	Gender	Job
8		Blida	F	M3

Client31

Id	…	City	Gender	Job
9		Kala	M	M4
3		Kala	M	M3

Client41

Id	…	City	Gender	Job
13		Kala	F	M5

Client32

Id	…	City	Gender	Job
7		Jijel	M	M5

Client42

Id	…	City	Gender	Job
11		Annaba	F	M4

Fig. 6. (a) $FSc4 : RFS$ on $FSc3$, (b) Partitioned table Client according to $FSc4$

5. $Del\_A(FS, A_i)$: to remove the attribute A_i from a FS, we need to create a new fragmentation schema where all the sub-domains of A_i are merged into a single set. We express this by using the logical operation $Merge\_Dom$. The algorithm for deleting an attribute of fragmentation is given as follows:

   for each SD_j^i in Dom(A_i) **do**
       Merge_Dom(A_i, SD_j^i, $SD_{(j+1)}^i$)
   end for

   Example 7. Consider the following schema Reduction:
   $FSc5 = EFS(SFc4) = Del\_A(FSc4, Job)$. The fragmentation schema $FSc5$ is given in the figure 7. The corresponding physical operation are given as follows:

   $Merge\_Dom(Job, \{M1, M2\}, Else3)$:

   $Ens\_Part1 = Identify\_Part(Job \text{ in } (M1, M2))) = \{Client11, Client21\}$,
   $Ens\_Part2 = Identify\_Part(Job \text{ in } Else_3) = \{Client12, Client22, Client31,$
   $Client41, Client32, Client42\}$,
   $Mergeable(Client11, Client12, Job) =$ True,
   $Merge(Client11, Client12)$,
   $Ens\_Part1 = \{Client21\}$,
   $Ens\_Part2 = \{Client22, Client31, Client41, Client32, Client42\}$,

$Mergeable(Client21, Client22, Job) =$ True,
$Merge(Client21, Client22)$,
$Ens\_Part1 = \{\ \}$,
$Ens\_Part2 = \{Client31, Client41, Client32, Client42\}$,

(a)

Gender	M	F	
City	Alger, Oran, Blida	Kala	$Reste_2$

(b)

Client1				
Id	...	City	Gender	Job
10		Alger	M	M1
5		Blida	M	M2
4		Alger	M	M3
6		Oran	M	M3

Client2				
Id	...	City	Gender	Job
1		Oran	F	M1
12		Oran	F	M2
2		Blida	F	M1
8		Blida	F	M3

Client31				
Id	...	City	Gender	Job
9		Kala	M	M4
3		Kala	M	M3

Client41				
Id	...	City	Gender	Job
13		Kala	F	M5

Client32				
Id	...	City	Gender	Job
7		Jijel	M	M5

Client42				
Id	...	City	Gender	Job
11		Annaba	F	M4

Fig. 7. (a) $FSc5 : RFS$ on $FSc4$, (b) Partitioned table Client according to $FSc5$

6. $Del\_SD(FS, A_i, \{SD^i_{j_1}, \cdots, SD^i_{j_p}\})$: delete from A_i the set containing the sub-domains $\{SD^i_{j_1}, \cdots, SD^i_{j_p}\}$ and include it in the set $Else_i$. This can be expressed by the operation $Merge\_Dom(A_i, \{SD^i_{j_1}, \cdots, SD^i_{j_p}\}, Else_i)$.

Example 8. Consider the following schema Reduction:
$FSc6 = EFS(SFc5) = Del\_SD(FSc5, Kala) = Merge\_Dom(City, \{Kala\}, Else_2)$.
The fragmentation schema $FSc6$ is given in the figure 8. The corresponding physical operation are given as follows:

$Ens\_Part1 = Identify\_Part(City = Kala) = \{Client31, Client41\}$,
$Ens\_Part2 = Identify\_Part(City$ in $Else_3) = \{Client32, Client42\}$,
$Mergeable(Client31, Client32, City) =$ True,
$Merge(Client31, Client32)$,
$Ens\_Part1 = \{Client41\}$,
$Ens\_Part2 = \{Client42\}$,
$Mergeable(Client41, Client42, City) =$ True,
$Merge(Client41, Client42)$,
$Ens\_Part1 = \{\ \}$,
$Ens\_Part2 = \{\ \}$,

(a)

Gender	M	F
City	Alger, Oran, Blida	$Reste_2$

(b)

Client1				
Id	…	City	Gender	Job
10		Alger	M	M1
4		Alger	M	M3
6		Oran	M	M3
5		Blida	M	M2

Client2				
Id	…	City	Gender	Job
1		Oran	F	M1
12		Oran	F	M2
2		Blida	F	M1
8		Blida	F	M3

Client3				
Id	…	City	Gender	Job
9		Kala	M	M4
3		Kala	M	M3
7		Jijel	M	M5

Client4				
Id	…	City	Gender	Job
11		Annaba	F	M4
13		Kala	F	M5

Fig. 8. (a) $FSc6$: RFS on $FSc5$, (b) Partitioned table Client according to $FSc6$

7 Experimentation under Oracle 11g

In order to evaluate our incremental selection based on Queries Profiling, we conduct experimental tests on a real data warehouse from the APB1 benchmark [7] under the DBMS Oracle 11g. The data warehouse based on a star schema contains a facts table *Actvars* (24 786 000 tuples) and 4 dimension tables *Prodlevel* (9000 tuples), *Custlevel* (900 tuples), *Timelevel* (24 tuples) and *Chanlevel* (9 tuples). The genetic algorithm is implemented using the JAVA API *JGAP*. In this study, we aim to evaluate the efficiency of the queries profiling performed using the fragmentation algebra. To well analyze the queries, we first conduct small-scale tests on a workload of 8 queries, then we realize larger-scale tests on a workload of 60 queries. The 60 queries generate **18 indexable attributes** *(Line, Day, Week, Country, Depart, Type, Sort, Class, Group, Family, Division, Year, Month, Quarter, Retailer, City, Gender and All)* that respectively have the following cardinalities : 15, 31, 52, 11, 25, 25, 4, 605, 300, 75, 4, 2, 12, 4, 99, 4, 2, 3.

7.1 Small-scale tests

In this experiment, we first consider an empty workload. Then, we suppose that eight new queries are successively executed on the RDW. We give the attributes and the profile of each query (table 6, left). Under a constraint $B = 40$, the three first queries Q_1, Q_2 and Q_3 triggers an incremental selection. These queries have an *Evolution* profile, since the constraint B has not yet been violated. The Queries Q_4 and Q_7 are *Mixed* and requires an incremental selection where the queries Q_5, Q_6 and Q_8 have respectively a *Neutral*, *Reduction* and *Neutral* profiles and don't require a new RDW partitioning. We compare the Incremental Selection based on Queries Profiling $GAQP$ to the classic incremental selection GA (both selections use the same genetic algorithm). For each new query and

Query	Attributes	Profile
Q1	Group	Evolution
Q2	Month, Quarter	Evolution
Q3	Month, Class	Evolution
Q4	City, Gender	Mixed
Q5	Month, Year, City, Class	Neutral
Q6	Class, Gender	Reduction
Q7	City, Gender, Class	Mixed
Q8	City, Gender, Group	Neutral

Queries	Profile
Q46, Q47, Q49, Q55, Q56, Q57, Q60	Neutral
Q48, Q50, Q52, Q51, Q58, Q59	Mixed
Q53, Q54	Reduction

Table 6. Workloads descriptions and queries profiles

each selection, we note the cost optimization rate of the executed queries illustrated in figure 9. We notice that the optimization of the workload cost given by both *GA* and *GAQP* is globally the same. This shows that profiling doesn't influence the quality of the solution selected by the incremental selection process. Next, we compare the two selections according to *the Maintenance Time*. The maintenance time is the time required to effectively implement a fragmentation schema on the data warehouse under Oracle 11g. After the execution of each query and for each selection (*GA* and *GAQP*), we implement under Oracle 11g the new fragmentation schema on the *RDW* and we note the maintenance time. Results are given in figure 10. For the *GA* selection, each query requires a selection and implementation of a new fragmentation schema, the global maintenance time after the execution of the ten queries is 193.6 minutes which correspond to 3 hours and 13 minutes. For the *GAQP* selection, the queries with the profiles *Reduction* and *Neutral* don't trigger a new incremental selection, so no changes occur on the *RDW*. The global maintenance time is 109.3 minutes (1 hours and 49 minutes). As a result, the *GAQP* selection reduces the global maintenance time by 43.5% compared to *GA* selection.

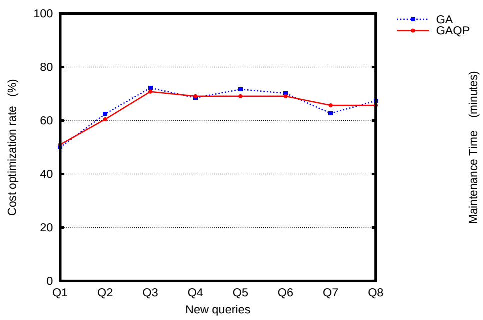

Fig. 9. Cost optimization rate (case 8 queries)

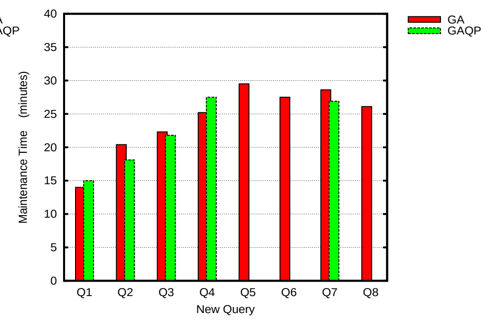

Fig. 10. Maintenance Time under Oracle11g (case 8 queries)

7.2 Larger-scale tests

We consider a workload of 45 queries executed on a partitioned RDW. The current fragmentation schema of the RDW is obtained by a static selection using the 45 queries with a constraint $B = 100$. After that, we suppose that 15 new queries are successively executed on the RDW. We perform the two selections (GA and $GAQP$). We also implement an existing approach: the incremental fragmentation selection named DD based on the dynamic design of data warehouses proposed in [18] that we adapt in a centralized context. For each selection and each new query, we store the cost optimization rate of the executed queries (figure 11) and the Maintenance Time under Oracle11g (figure 12). Profiles of the new queries are given in table 6, right. According to the result given by

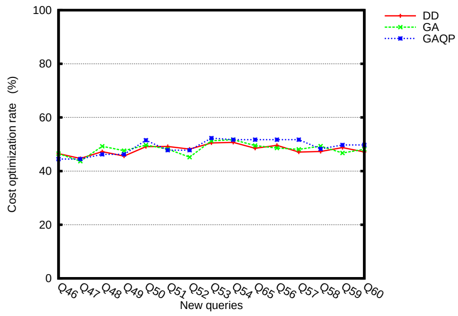

Fig. 11. Cost optimization rate (case 60 queries)

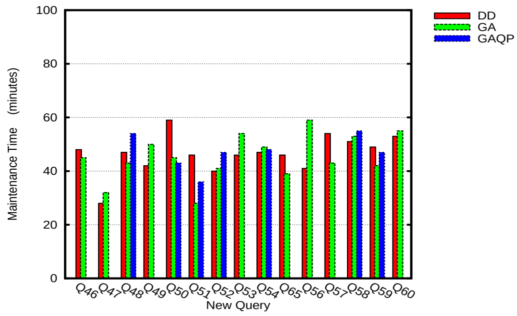

Fig. 12. Maintenance Time under Oracle11g (case 60 queries)

figure 11, the workload costs obtained by the three selections are similar. First the incremental selection of fragmentation schema in DD, GA and $GAQP$ are all based on the horizontal fragmentation selection approach proposed in [3]. Second, the queries profiling doesn't affect the quality of any selected fragmentation schema. But, when analyzing the results of the figure 12, we notice that the $GAQP$ selection gives a better maintenance time then the GA and DD selection. The global maintenance time of GA and DD selections is respectively 678 minutes (11 hours and 18 minutes) and 697 minutes (11 hours and 37 minutes) when the global maintenance time of $GAQP$ selection is 331 minutes (5 hours and 31 minutes) which reduces the global maintenance time by 52%. This is due to the fact that GA and DD perform a selection of a new fragmentation schema after the execution of each new query. On the other hand, among the 15 queries only 6 queries trigger a new incremental selection (Mixed profile) for the selection $GAPQ$. The queries with a Reduction or Neutral profiles don't require any changes on the RDW.

Therefore, according to the important parameter namely the maintenance time required to implement a new fragmentation schema on a partitioned RDW, the $GAQP$ approach is better than the classic GA incremental selection and the existing approach DD.

8 Conclusion

This work deals with incremental selection of a horizontal data partitioning schema in the context of data warehouse modeling by a star schema. We propose a Fragmentation Algebra containing all possible operations that can be performed on a fragmentation schema in order to take into account workload evolution. Using our Algebra, we define Queries Profiling. According to the profile of a new executed query, we determine if a selection of a new fragmentation schema is required. We give the architecture of the incremental selection of fragmentation schema based on queries profiling and the Fragmentation Algebra. Then, we give an insight of the physical operations required to implement the Algebra operations under Oracle 11g. Finally, we conduct an experimental study under the DBMS Oracle 11g to show the efficiency of the queries profiling. We showed that using queries profiling reduces by more than 50% the global maintenance time required to implement a new selection fragmentation schema on a partitioned RDW.

In this work, we deal with workload evolution. One perspective is to consider other changes occurred on the data warehouse such as data evolution.

References

1. L. Bellatreche, K. Boukhalfa, and M. Mohania. Pruning search space of physical database design. In *18th International Conference On Database and Expert Systems Applications (DEXA'07)*, pages 479–488, 2007.
2. L. Bellatreche, K. Boukhalfa, and P. 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.
3. L. Bellatreche, K. Boukhalfa, and P. Richard. Referential horizontal partitioning selection problem in data warehouses: Hardness study and selection algorithms. *International Journal of Data Warehousing and Mining*, 5(4):1–23, March 2009.
4. R. Bouchakri, L. Bellatreche, Z. Faget, and S. Breß. A coding template for handling static and incremental horizontal partitioning in data warehouses. *To appear in Journal of Decision Systems (JDS) Vol. 22*, 2013.
5. 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.
6. C.-H. Cheng, W.-K. Lee, and K.-F. Wong. A genetic algorithm-based clustering approach for database partitioning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 32(3):215–230, 2002.
7. O. Council. Apb-1 olap benchmark, release ii. *http://www.olapcouncil.org/research/bmarkly.htm*, 1998.

8. C. Curino, E. P. C. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational cloud: a database service for the cloud. In *CIDR*, pages 235–240, 2011.
9. C. Curino, Y. Zhang, E. P. C. Jones, and S. Madden. Schism: a workload-driven approach to database replication and partitioning. *PVLDB*, 3(1):48–57, 2010.
10. H. Derrar, M. Ahmed-Nacer, and O. Boussaid. Dynamic distributed data warehouse design. *Journal of Intelligent Information and Database Systems, Vol.7, No.1*, 2013.
11. A. Dimovski, G. Velinov, and D. Sahpaski. Horizontal partitioning by predicate abstraction and its application to data warehouse design. In *Advances in Databases and Information Systems*, pages 164–175. Springer, 2011.
12. C. A. Galindo-Legaria, T. Grabs, S. Gukal, S. Herbert, A. Surna, S. Wang, W. Yu, P. Zabback, and S. Zhang. Optimizing star join queries for data warehousing in microsoft sql server. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 1190–1199. IEEE, 2008.
13. 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.
14. 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.
15. M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems : Second Edition*. Prentice Hall, 1999.
16. M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems: Second Edition*. Prentice Hall International, Inc., 1999.
17. A. Sanjay, V. R. Narasayya, and B. Yang. Integrating vertical and horizontal partitioning into automated physical database design. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 359–370, June 2004.
18. K. Tekaya. Dynamic distributed data warehouse design. *IRMA International Conference*, 2007.