Algebra-based Approach for Incremental Data Warehouse Partitioning

Rima Bouchakri\textsuperscript{1,2}, Ladjel Bellatreche\textsuperscript{1}, and Zoé Faget\textsuperscript{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 important studies were concentrated on the static selection of a fragmentation schema, where they assume the knowledge of the workload. However, due to 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 with a set of operators defined on a flexible encoding of any fragmentation schema. This algebra is invoked when new queries change the content of the encoding. We conduct experiments to evaluate the efficiency and effectiveness of our finding.

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 well established phase in the database life cycle which is the deployment phase. It consists in selecting the best 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 a database/data warehouse on any platform: centralized [16], parallel [7], distributed [14], cloud [6], etc. The problem of PHDP has been largely studied in the literature in different database contexts: OLTP databases [13], data warehouses [1], scientific and statistical databases [15]. It consists in fragmenting a table, an index or a materialized view, into partitions (fragments), where each fragment contains a subset of tuples [16]. Many commercial (Oracle, DB2, SQLServer, Sybase) and academic DBMS (PostgreSQL and MySQL) implement it. Two main types of HDP are distinguished [3]: primary HDP and derived HDP. In the first partitioning, a given table is partitioned based on its own attributes. The primary HDP optimizes selection operations and is used in rewriting queries in distributed and parallel databases [14]. When the result of a fragmentation of a given table is propagated to another table, this partitioning is called derived HDP. This partitioning is feasible when a parent-child relationship exists between the two involved tables.
Several HDP modes exist to support both primary and derived HDP in centralized and parallel platforms (e.g., Teradata). For the primary HDP, we find simple mode (Range, List, Hash) and composite mode (Range-Range, List-Range, etc.). Derived partitioning has been recently supported by commercial DBMS under the name of referential partitioning (the Oracle11G case).

Historically, the formalization of the PHDP follows the evolution of database technology. In traditional databases, the PHDP has been formalized as follows: given a table \( T \) of a given database schema and a set of a priori known queries, the PHDP consists in fragmenting \( T \) into fragments such as the overall query processing cost is minimized. In the relational data warehouse (RDW), the PHDP got more attention, where algorithms were proposed to partition the whole schema. This is due to the characteristics of star join queries that involves simultaneously restrictions on the dimension tables and joins between the fact and dimension tables. To avoid a large number of final fragments, a constraint on this number has been added. As a consequence, a new formalization of the PHDP has been proposed [1]: given a data warehouse schema, a set of a priori known queries, and constraint that limits the number of the final fragments, PHDP consists in fragmenting the fact table based on the partitioning schemes of dimension tables such as the overall query processing cost is minimized and the final fragments do not exceed the 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 mainly involving the attributes used to partition either table or a schema. Note that 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 to support incremental algorithms. (iii) The question of deployment on the target platform of the resulting partition is ignored, especially for the dynamic context. Our vision is to propose an integrated solution for PHDP that satisfies these three objectives. Instead of spending time on developing algorithms, we claim that the presence of a flexible encoding for any HDP schema and the development of algebra whose operators are applied on that coding to capture any change of the partitioning schema when the workload evolves. To manage workload evolution, we introduce the notion of query profiling which studies of the impact of a new arrival query on the encoding (expanding it, keeping it as it is, or reducing it). These operations will be managed by the proposed algebra. Another advantage of our algebra is its contribution to deploy our solution. Each evolution of the encoding can be easily translated according to the target platform. For our study, we consider a RDW deployed on Oracle 11G.

This paper is organized as follows: Section 2 reviews the most important works

\(^1\) [http://www.darkenergysurvey.org](http://www.darkenergysurvey.org)
on static and incremental selection of \( \mathcal{HDP} \) schema. Section 3 describes our algebra and its properties for managing any fragmentation schema in the dynamic context. Section 4 describes the management of the arrival of new queries by the use of our encoding and algebra. In section 5, we present in details our algorithm that uses the notion of query profiles. In Section 6, we conduct experiments to show the efficiency and effectiveness of our proposal. Section 7 concludes the paper.

2 Related work

As we said before, \( \mathcal{HDP} \) got a lot of attention from academic and industrial communities, in developing algorithms. These algorithms may be classified into two categories according to the selection nature: static selection and dynamic selection. The static selection got the lion part of the works. It supposes that the inputs of the \( \mathcal{PHDP} \) (the schema of the database/warehouse and the workload) are known in advance and fixed. Four main approaches can be considered: (a) minterm generation algorithms [3, 14], (b) affinity-based algorithms [1, 11], (c) cost-model driven algorithms [4, 1, 9] and (d) data mining driven algorithms [10]. For more details, refer to [1].

To take into account the evolution of the inputs of the \( \mathcal{PHDP} \), dynamic selection has been proposed. The studies in this category may also be divided into two groups: dynamic selection when the database schema evolves and dynamic selection when the workload changes. Authors in [11] deal with distributed database redesign when queries change. They propose simple heuristics to manage the impact of these changes on fragments. Authors in [17] propose a dynamic design of distributed data warehouses. The main issue of this approach is that the fragmentation is triggered for each change. This approach may cause a high maintenance cost, and may be unnecessary in some cases where a change in the workload eventually leads to the same schema. Authors in [8] propose to re-fragment a relational centralized data warehouses when query change occurs. This approach is based on storing recent statistics. First, only the facts table is partitioned using only Range mode on one of its foreign keys. Second, histograms are built to observe the access behavior of queries to different fragments. At a given time and by the means of a cost model and the histograms, the data warehouse schema is re-partitioned. Then, the histograms are updated to store the occurred changes. The issues regarding the deployment of this solution are ignored. In our work [2], we propose an incremental selection of fragmentation schema based on Genetic Algorithms \( GA \) that is triggered after every changes on the workload. This approach may have the same limitation as for [17]. Couple of studies were proposed when the database schema evolves. In [12], the authors proposed a dynamic algorithm for partitioning continuously growing large databases. A heuristic is proposed to efficiently distribute new arriving data, based on its affinity with the different fragments in the application.

In this paper, we focus on handling query changes. The main contribution of our approach is that the re-fragmentation is launched if and only if the profile
of the new arrival query is different than the previous ones used to perform the partitioning. This profiling is based on our algebra, presented in the next section.

3 Fragmentation Algebra

Let us consider a $\mathcal{RDW}$ with a fact table $F$ and $d$ dimensions tables $D = \{D_1, D_2, \ldots, D_d\}$. A fragmentation schema is the result of the $\mathcal{RDW}$ partitioning process. It is defined on non-key dimension attributes $A = \{A_1, \ldots, A_n\}$. Each attribute $A_i$ has a Domain, called $\text{Dom}(A_i)$. $\text{Dom}(A_i)$ can be partitioned into $m_i$ sub-domains $\text{Dom}(A_i) = \{SD^i_1, SD^i_2, \ldots, SD^i_{m_i}\}$. For instance, if we consider the attribute $\text{City}$ of a given table $\text{Clients}$, the domain partitioning is given as follows: $\text{Dom}(\text{City}) = \{\text{‘Alger’}, \text{‘Oran’}, \text{‘Blida’}, \text{‘Kala’}, \text{‘Annaba’}, \text{‘Jijel’}\}$. Based on these notions, we define a Data Structure that represents the Maximal Fragmentation Schema $MFS$ of dimension tables (table 1). The number of fact table fragments is then the product of the numbers of dimensions fragments ($\prod_i m_i$).

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$SD^1_1$</th>
<th>$SD^1_2$</th>
<th>\ldots</th>
<th>$SD^1_{m_1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2$</td>
<td>$SD^2_1$</td>
<td>$SD^2_2$</td>
<td>\ldots</td>
<td>$SD^2_{m_2}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_n$</td>
<td>$SD^n_1$</td>
<td>$SD^n_2$</td>
<td>\ldots</td>
<td>$SD^n_{m_n}$</td>
</tr>
</tbody>
</table>

Table 1. Maximal Fragmentation Schema $MFS$

A similar encoding has been proposed in [2] where an incremental genetic algorithm used this encoding. The main drawback of this work is that the genetic algorithm is executed every time changes on the workload occur. Note that a new query may 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 should not trigger a new $\mathcal{HDP}$ selection. In order to determine the exact actions required after the arrival of a given query, we define a Fragmentation Algebra that contains all possible operations defined on a $\mathcal{HDP}$ schema.

3.1 Schema reduction and evolution

In a $\mathcal{RDW}$, the number of fragmentation attributes may be very large (hundreds of attributes). As a consequence, the number of fact table fragments in the Maximal Fragmentation Schema ($MFS$) is very large too. Suppose a $MFS$ with 30 attributes, where each attribute has 10 sub-domains. The number of fact table fragments is $\prod_i m_i = \prod_{i=1}^{30} 10 = 10^{30}$, 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, all other values of the attribute A, not specified in the sub-domains. For the fragmentation schema of the Table 2, the sets Else, are specified as follows:

- Else1 = \{SD_1, \ldots , SD_{m_1}\} \setminus \{SD_1, SD_2, SD_3, SD_5, SD_4, SD_6\}
- Else2 = \{SD_1, \ldots , SD_{m_2}\} \setminus \{SD_1, SD_2\}
- Else3 = \{SD_1, \ldots , SD_{m_3}\} \setminus \{SD_1, SD_3, SD_5, SD_7, SD_8, SD_9\}
- Else4 = \{SD_1, \ldots , SD_{m_4}\} \setminus \{SD_1, SD_4\}

<table>
<thead>
<tr>
<th>A_1</th>
<th>SD_1</th>
<th>SD_2, SD_3, SD_4</th>
<th>SD_5, SD_6</th>
<th>Else1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_2</td>
<td>SD_1</td>
<td>SD_2</td>
<td></td>
<td>Else2</td>
</tr>
<tr>
<td>A_3</td>
<td>SD_1, SD_2</td>
<td>SD_2, SD_3</td>
<td>SD_5, SD_6</td>
<td>Else3</td>
</tr>
<tr>
<td>A_4</td>
<td>SD_1</td>
<td>SD_2</td>
<td></td>
<td>Else4</td>
</tr>
</tbody>
</table>

Table 2. Reduction of the \(HDP\) schema \(MFS\) to \(FS\)

The transition from the \(MFS\) to the fragmentation schema \(FS\) is called the Reduction of the fragmentation Schema (\(RFS\)). It includes the following operations: (1) remove the attributes \(A_1, \ldots , A_n\), (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.

<table>
<thead>
<tr>
<th>A_1</th>
<th>SD_1</th>
<th>SD_2, SD_3, SD_4</th>
<th>SD_5, SD_6</th>
<th>Else1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_2</td>
<td>SD_1</td>
<td>SD_2</td>
<td></td>
<td>Else2</td>
</tr>
<tr>
<td>A_3</td>
<td>SD_1, SD_2</td>
<td>SD_2, SD_3</td>
<td>SD_5, SD_6</td>
<td>Else3</td>
</tr>
<tr>
<td>A_4</td>
<td>SD_1</td>
<td>SD_2</td>
<td></td>
<td>Else4</td>
</tr>
</tbody>
</table>

Table 3. Evolution of the fragmentation schema \(FS\) to \(FS'\)

The transition from the schema \(FS\) to the schema \(FS'\) is called Evolution of the \(HDP\) 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 \(Dom(A_5) = \{SD_1, \ldots , SD_{m_5}\}\), (2) merge the sub-domains of \(A_5\) into two sets \(\{SD_1, SD_2\}\) and \(Else_5 = \{SD_1, \ldots , SD_{m_5}\} \setminus \{SD_1, SD_2\}\), (3) split the set of sub-domains of \(A_5\) \(\{SD_1, SD_2\}\) into two sets, (4) merge the two sets \(\{SD_1\}\) and \(\{SD_2\}\) into \(SD_1, SD_2\) and (5) merge the two sets \(SD_1, SD_2\) and \(SD_3, SD_4\) into \(SD_1, SD_2, SD_3, SD_4\)
3.2 Operators description

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

- $Add(A_i, \{SD_{j_1}^i, \ldots, SD_{j_p}^i\})(FS)$: add the attribute $A_i$ to the fragmentation schema $FS$ including the set of sub-domains $\{SD_{j_1}^i, \ldots, SD_{j_p}^i\}$, which implies creating the set $Else_i = \{SD_{1}, \ldots, SD_{m_i}\} \setminus \{SD_{j_1}^i, \ldots, SD_{j_p}^i\}$.
- $Add_SD(A_i, \{SD_{j_1}^i, \ldots, SD_{j_p}^i\})(FS)$: add to the attribute $A_i$ a set of sub-domains $\{SD_{j_1}^i, \ldots, SD_{j_p}^i\}$ and delete it from the set $Else_i$.
- $Split_Dom(A_i, \{SD_{j_1}^i, \ldots, SD_{j_p}^i\}, \{SD_{k_1}^i, \ldots, SD_{k_r}^i\})(FS)$: split the set of sub-domains $\{SD_{j_1}^i, \ldots, SD_{j_p}^i\}$ of the attribute $A_i$ into two sets of sub-domains $\{SD_{j_1}^i, \ldots, SD_{j_p}^i\}$ and $\{SD_{k_1}^i, \ldots, SD_{k_r}^i\}$, where $\{k_1, \ldots, k_r\} \subseteq \{j_1, \ldots, j_p\}$.
- $Merge_Dom(A_i, \{SD_{j_1}^i, \ldots, SD_{j_p}^i\}, \{SD_{k_1}^i, \ldots, SD_{k_r}^i\})(FS)$: merge the two sets $\{SD_{j_1}^i, \ldots, SD_{j_p}^i\}$ and $\{SD_{k_1}^i, \ldots, SD_{k_r}^i\}$ into one, where $\{j_1, \ldots, j_p\} \subseteq [1, m_i]$ and $\{k_1, \ldots, k_r\} \subseteq [1, m_i]$.
- $Del_A(A_i)(FS)$: delete the attribute $A_i$ from the $HDP$ schema $FS$.
- $Del_SD(A_i, \{SD_{j_1}^i, \ldots, SD_{j_p}^i\})(FS)$: delete from the attribute $A_i$ the set containing the sub-domains $\{SD_{j_1}^i, \ldots, SD_{j_p}^i\}$ 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:

$$FS' = EFS(FS) =$$

$$Merge_Dom(A_3, \{SD_{2}^3\}, \{SD_{2}^3\}) \circ Merge_Dom(A_3, \{SD_{2}^3\}, \{SD_{1}^3\}) \circ Split_Dom(FS, A_3, \{SD_{2}^3, SD_{2}^3\}) \circ Add_A(A_5, \{SD_{1}^5, SD_{2}^5\})(FS).$$

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 ($Add_SD$, $Split_SD$, $Merge_Dom$, $Del_Dom$).

3.3 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.
\[ \text{Del}_A \circ \text{Add}_A = \text{Id} \]

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_{ij}\}) and Add_SD(A_i, \{SD_{ij}\}) are inverse operators.

**Equivalence rules**

1. Operators involving different attributes, or involving the same attribute but different subdomains, are commutative.

2. \[ \text{Merge}_\text{Dom}(A_i, \text{Set}_1, \text{Set}_2) \circ \text{Add}_\text{SD}(A_i, \text{Set}_1) \circ \text{Add}_\text{SD}(A_i, \text{Set}_2) \] is equivalent to \[ \text{Add}_\text{SD}(A_i, \text{Set}_1 \cup \text{Set}_2) \].

3. More generally, 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.

\[ \text{Del}_\text{SD}(A_i, \text{Set}_1) = \text{Merge}_\text{Dom}(A_i, \text{Set}_1, \text{Else}_i) \]

5. Deleting an attribute is equivalent to successively merging all subdomains of this attribute.

\[ \text{Del}_A(A_i) = \text{Merge}_\text{Dom}(A_i, \text{SD}_{i1}, \text{Else}_i) \cdots \circ \text{Merge}_\text{Dom}(A_i, \text{SD}_{im}, \text{Else}_i) \]

### 4 Queries Profiling

When new queries are executed on the \( \mathcal{RDW} \), 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:

```sql
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 Vn2 ... 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 \( A_i \ op \ V_{ij} \) \((1 < i < n \text{ and } 1 < j < k_i)\). Each attribute’s value \( V_{ij} \) can equal or be contained in a sub-domain \( SD_{ij} \). Therefore, the general expression of a selection predicate is \( A_i \ op \ \{SD_{ij1}, \ldots, SD_{ijp}\} \). Let’s consider a new query \( Q \) executed on a \( \mathcal{RDW} \) partitioned according to a fragmentation schema \( FS \). The execution of \( Q \) may
require adding new attributes, new sub-domains contained in $E_{s}\_i$, 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 $A_i$ on $\{SD_{j_1}^i, \ldots, SD_{j_p}^i\}$.

- $A_j$ does not appear in $FS$: this attribute is added to the fragmentation schema by the operation $Add\_A(A_i, \{SD_{j_1}^i, \ldots, SD_{j_p}^i\})(FS)$.

- $A_j$ appears in $FS$: We verify if the set of sub-domains $\{SD_{j_1}^i, \ldots, SD_{j_p}^i\}$ requires Algebra operations.

- All sub-domains contained in $\{SD_{j_1}^i, \ldots, SD_{j_p}^i\}$ appear as a set in $FS$: no operations.

- The set $\{SD_{j_1}^i, \ldots, SD_{j_p}^i\}$ is included in a set $\{SD_{k_1}^i, \ldots, SD_{k_m}^i\}$ in $FS$: this set is split using the operation $Split\_Dom(A_i, \{SD_{k_1}^i, \ldots, SD_{k_m}^i\}, \{SD_{j_1}^i, \ldots, SD_{j_p}^i\})(FS)$.

- All sub-domains contained in $\{SD_{j_1}^i, \ldots, SD_{j_p}^i\}$ appear in $t$ sub-sets $SubSet_1, \ldots, SubSet_t$ in $FS$, where $SubSet_1 \cup \ldots \cup SubSet_t = \{SD_{j_1}^i, \ldots, SD_{j_p}^i\}$: the $t$ sub-sets are merged by the operation $Merge\_Dom(A_i, SubSet_1) \circ Merge\_Dom(A_i, SubSet_2) \circ \ldots \circ Merge\_Dom(A_i, SubSet_t)(FS)$.

- A sub-set of sub-domains $\{SD_{k_1}^i, \ldots, SD_{k_p}^i\}$ doesn’t appear in $FS$, where $\{SD_{k_1}^i, \ldots, SD_{k_p}^i\} \subset \{SD_{j_1}^i, \ldots, SD_{j_p}^i\}$: the sub-domains contained in this sub-set are in $Else_i$. The sub-set is added to $FS$ using the operation $Add\_SD(A_i, \{SD_{k_1}^i, \ldots, SD_{k_p}^i\})(FS)$.

- There is no sub-domain of $\{SD_{j_1}^i, \ldots, SD_{j_p}^i\}$ that appears in $FS$: the sub-domains are all in $Else_i$. The set is added to $FS$ using the operation $Add\_SD(A_i, \{SD_{j_1}^i, \ldots, SD_{j_p}^i\})(FS)$.

- 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 than 20% of the workload, it’s deleted using the operation $Del\_A(A_j)(FS)$.

- A set of sub-domains $\{SD_{R_1}^i, \ldots, SD_{R_m}^i\}$ 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 than 20% of the workload, it’s removed using the operation $Del\_SD(A_i, \{SD_{j_1}^i, \ldots, SD_{j_p}^i\})(FS)$.

- 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 Operations induce by the query $Q_i$, we deduce if $Q_i$ 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 schema is needed. As consequence, the number of facts table fragments will increase.
2. **Reduction queries**: this profile describes queries that require the operations $\text{Del}_A()$, $\text{Del}_SD()$ and $\text{Merge}_Dom()$. When a Reduction query is executed on the $\mathcal{RDW}$, an $\mathcal{RFS}$ of the current schema is needed. Therefore, the number of facts table fragments will decrease.

3. **Mixed queries**: a Mixed query implies both Evolution and Reduction operations ($\text{Add}_A()$, $\text{Add}_SD()$, $\text{Split}_Dom()$, $\text{Del}_A()$, $\text{Del}_SD()$ and $\text{Merge}_Dom()$). The number of facts table fragments can either increase or decrease.

4. **Neutral queries**: a Neutral query does not affect the current $\mathcal{RDW}$ fragmentation schema. Let us consider a selection predicate $A_i \text{ op } \{SD_{i,j_1}^1, \ldots, SD_{i,j_p}^p\}$ in a query. This query is neutral if $A_i$ appears in $FS$ and all sub-domains, contained in $\{SD_{i,j_1}^1, \ldots, SD_{i,j_p}^p\}$, appear as a set in $FS$ or the operations defined leaves the $\mathcal{HDP}$ schema unchanged. For instance, if the query requires two inverse operators like $\text{Del}_A$ and $\text{Add}_A$, the identity operator $\text{Id}(FS)$ is obtained which has no effect on the fragmentation schema. Two operators $\text{Del}_A$ (resp. $\text{Del}_SD$) and $\text{Add}_A$ (resp. $\text{Add}_SD$) can be obtained, when executing one query, if the attribute $A$ (resp. the set $\{SD_{i,j_1}^1, \ldots, SD_{i,j_p}^p\}$) is no more frequently used by the workload which requires the operator $\text{Del}_A$ (resp. $\text{Del}_SD$) but a selection predicate is defined on the attribute $A$ (resp. the set $\{SD_{i,j_1}^1, \ldots, SD_{i,j_p}^p\}$) which generate the operator $\text{Add}_A$ (resp. $\text{Add}_SD$).

**Example 1.** Let us consider a data warehouse with a fact table Sales and two dimension tables: Client and Product. We give the current fragmentation schema $FS_1$ of the $\mathcal{RDW}$ 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 $FS_1$ to the changes given by each query (Table 5).

<table>
<thead>
<tr>
<th>Gender</th>
<th>M</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>City</td>
<td>Alger</td>
<td>Oran</td>
</tr>
</tbody>
</table>

**Table 4.** Example of a fragmentation schema $FS_1$

5 **Queries Profiling-based Incremental Algorithm**

We assume that a new query $Q_i$ is executed on a partitioned $\mathcal{RDW}$ represented by a current $\mathcal{HDP}$ Schema called $FS_i$. In order to adapt the current fragmentation schema of the $\mathcal{RDW}$, 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 $\mathcal{RDW}$ schema. We present an algorithm that summarizes the Incremental selection of $\mathcal{RDW}$ schema using query profiling. This algorithm uses the classic $\mathcal{RDW}$ schema selection based on Genetic Algorithms [2]. Some functions are needed to implement our algorithm:
<table>
<thead>
<tr>
<th>Query</th>
<th>Algebra Operations</th>
<th>Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>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'</td>
<td>Add_Dom(City, {Blida})(FS1) Add_A(PName, {P1})(FS1)</td>
<td>Evolution</td>
</tr>
<tr>
<td>SELECT * FROM Client C, Sales S WHERE S.IdC=C.Id And C.City='Alger' Or C.City='Oran'</td>
<td>Merge_Dom(City, {Alger}, {Oran})(FS1)</td>
<td>Reduction</td>
</tr>
<tr>
<td>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'</td>
<td>Merge_Dom(City, {Alger}, {Oran})(FS1) Add_A(FS1, PName, {P1})</td>
<td>Mixed</td>
</tr>
<tr>
<td>SELECT * FROM Client C, Sales S WHERE S.IdC=C.Id And C.City='Alger'</td>
<td>/</td>
<td>Neutral</td>
</tr>
</tbody>
</table>

Table 5. Queries Profiles

- AnalyseQProfile(Algebra, $Q_i$): returns the profile of the query $Q_i$ based on the Algebra.
- ComputeNewFS($FS_t, Q_i$): returns a new fragmentation Schema $FS_{t+1}$ using the current Schema $FS_t$ and the new query $Q_i$.
- FragmentationSelectionGA ($Q, FS, RDW, B$): Selects the best fragmentation Schema using Genetic Algorithm.
- NBfragments($FS$): returns the number of fact fragments of a given fragmentation Schema $FS$.

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 $RDW$.

Our algebra operators are directly deployed in Oracle11G. We choose Oracle11G, since it is the pioneer that supports the referential partitioning2. Due to lack of space, we cannot detail here how all operators are physically implemented under Oracle 11g. The interested reader can refer to our technical document3.

2 http://docs.oracle.com/cd/B10501_01/server.920/a96521/partiti.htm
### Incremental Selection of FS based on Queries Profiling

**Input:**
- \( \text{Algebra} \) : 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 \( NewFS \).

**Begin**

\[
\begin{align*}
\text{QueryProfil} & \leftarrow \text{AnalyseQProfile}(\text{Algebra}, Q_i); \\
\text{if} \quad \text{QueryProfil}= & \text{"Neutral"} \quad \text{then} \\
& \quad \text{Break; \{} \text{End Algorithm} \} \\
\text{end if} \\
\text{NewFS} & \leftarrow \text{ComputeNewFS}(FS, Q_i); \\
\text{if} \quad \text{QueryProfil}= & \text{"Evolution"} \quad \text{or} \quad \text{QueryProfil}= \text{"Mixed"} \quad \text{then} \\
& \quad \text{if} \quad \text{NBfragments}(\text{NewFS}) > B \quad \text{then} \\
& \quad \quad \text{NewFS} \leftarrow \text{FragmentationSelectionGA}(Q \cup \{Q_i\}, FS, RDW, B); \\
\text{end if} \\
\text{end if} \\
\text{End}
\end{align*}
\]

### 6 Experimentation under Oracle 11g

In order to evaluate our incremental selection based on Queries Profiling, we conduct experimental tests on a \( RDW \) schema of the APB1 benchmark [5] under the DBMS Oracle 11g. The \( RDW \) based on a star schema contains a facts table \( Actsvars \) (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 (jgap.sourceforge.net). In this study, we aim at evaluating the efficiency of the queries profiling performed using our 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.

#### 6.1 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 each selection, we write down the cost optimization rate of the executed queries illustrated in figure 2. We notice that the optimization of the workload cost given by both $GA$ and $GAQP$ is globally the same. This shows that profiling does not 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

---

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

<table>
<thead>
<tr>
<th>Query</th>
<th>Attributes</th>
<th>Profile</th>
<th>Queries</th>
<th>Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Group</td>
<td>Evolution</td>
<td>Q46, Q47, Q49, Q55, Q56, Q57, Q58, Q59, Q60, Q52, Q51, Q53, Q54, Q58</td>
<td>Neutral</td>
</tr>
<tr>
<td>Q2</td>
<td>Month, Quarter</td>
<td>Evolution</td>
<td>Q48, Q50, Q52, Q51, Q58, Q59</td>
<td>Mixed</td>
</tr>
<tr>
<td>Q3</td>
<td>Month, Class</td>
<td>Evolution</td>
<td>Q53, Q54</td>
<td>Reduction</td>
</tr>
<tr>
<td>Q4</td>
<td>City, Gender</td>
<td>Neutral</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q5</td>
<td>Month, Year, City, Class</td>
<td>Neutral</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q6</td>
<td>Class, Gender</td>
<td>Reduction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q7</td>
<td>City, Gender, Class</td>
<td>Mixed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q8</td>
<td>City, Gender, Group</td>
<td>Neutral</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
we write down the maintenance time. Results are given in figure 3. 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 do not 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. 2. Cost optimization rate (case 8 queries)  

Fig. 3. Maintenance Time under Oracle11g (case 8 queries)

6.2 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 RDW proposed in [17] 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 4) and the Maintenance Time under Oracle11g (figure 5). Profiles of the new queries are given in table 6, right. According to the result given by figure 4, 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 [1]. Second, the queries profiling does not affect the quality of any selected fragmentation schema. But, when analyzing the results of the figure 5, 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 $GAQP$. The queries with a Reduction or Neutral profiles do not 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 $DD$.

7 Conclusion

This work deals with incremental selection of a horizontal data partitioning schema in the context of data warehouse modelled 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 $RDW$.

We are currently working on the problem of incremental horizontal data partitioning by considering the evolution of the data warehouse schema and its instances using solutions issued from graph theory as in [6].
References


