A new way of optimizing OLAP queries

Arnaud Giacometti* — Dominique Laurent** — Patrick Marcel* — Hassina Mouloudi*

*LI, Université François Rabelais de Blois-Tours-Chinon
IUP GEII, Antenne Universitaire, 3 place Jean Jaurès
41000 Blois
{arnaud.giacometti,patrick.marcel}@univ-tours.fr
Hassina.Mouloudi@etu.univ-tours.fr

**LICP, Université de Cergy-Pontoise
dominique.laurent@dept-info.u-cergy.fr

ABSTRACT. For around 10 years, the academic research in database has attempted to define a commonly agreed logical modeling for the multidimensional and hierarchical nature of data manipulated with OLAP treatments (called datacube, or cube for short). But only recently has the concept of representation of a cube on a screen, or the optimization of OLAP queries at a logical level, been taken into account in this study. As many others, we believe that these two concepts are essential for the definition of a multidimensional query language. In this article, we propose to consider representations of cubes as first class citizens for query optimisation at the logical level. To reach this goal, we formally define the concept of representation by using the model of complex values [ABI 95]. This allows to have a single model for manipulating both cubes and their representations through typical OLAP operations. These typical operations are studied to propose rewrite rules in order to optimize OLAP queries.

RÉSUMÉ. Depuis environ 10 ans, la définition d’un modèle concensuel englobant la nature multidimensionnelle et hiérarchisée des données manipulées par les traitements OLAP (appelées cube de données) est à l’étude. Mais c’est seulement récemment que les concepts de représentation d’un cube à l’écran et d’optimisation de requêtes OLAP à un niveau logique ont été pris en compte dans cette étude. Nous pensons, comme beaucoup, que ces concepts sont essentiels à la définition d’un langage de requêtes pour OLAP. Dans cet article, nous proposons de considérer les représentations de cubes de données comme une base pour l’optimisation de requêtes au niveau logique. Pour ce faire, nous définissons formellement le concept de représentation en utilisant le modèle des valeurs complexes [ABI 95]. Cela permet d’avoir un modèle unique pour manipuler un cube et ses représentations via les opérations OLAP usuelles. L’étude de ces opérations nous permet de donner des règles de réécriture pour optimiser les requêtes OLAP.

KEYWORDS: OLAP, query language, logical modeling, optimisation

MOTS-CLÉS: OLAP, langage de requêtes, modélisation logique, optimisation
1. Introduction

Even if OLAP (On-line Analytical Processing, [CHA 97]) logical modeling has been deeply studied in the database community [GYS 97, AGR 97, VAS 00, CAB 97, HAC 97, MEN 04], it is still too early for a commonly agreed logical model to arise. As noticed in [JAR 02], it seems that the practical value of such studies involves both efficient query execution and multidimensional visualization. To the best of our knowledge, there is no work that studies the possibilities of optimizing queries based on the representation of a cube on a screen.

In this paper we concentrate on:

– The formal definition of representation of a cube, i.e., the logical counterpart of what is displayed on a screen.
– The definition of a language for manipulating both cubes and representations.
– The optimization of OLAP queries at a logical level, i.e., based on the properties of the operators.

We propose to describe in a single logical model both datacubes and their representations. We show that the model of complex values [ABI 95] is a good candidate to achieve this goal. We translate into this model the most typical basic OLAP operators [MAR 99]. The language so defined allows the user to precisely:

– Describe what the output of a query is, in terms of representation on a screen.
– Know how the facts displayed have been computed, in particular aggregations are computed on the basis of the actual hierarchies the user sees.

This framework allows us to study the optimization of OLAP queries at a logical level in the following way:

– We give rewriting rules involving the OLAP operators that are used to generate the optimized form of the query.
– Based on the rewriting rules, we identify a canonical form for OLAP queries that can be obtained from any query.
– We rewrite the query so that evaluating this query only deals with the data that will be displayed on the screen as a result.

A lot of work on the optimisation of OLAP queries has been done by studying how to rewrite OLAP queries in the presence of precomputed aggregates ([PAR 01] is a recent work in this area and contains a short overview). This approach does not take into account how the answers to the queries are represented on the screen. Because we concentrate on determining the part of the answer that is to be displayed, our approach can be seen as orthogonal to this one.

To our knowledge, there is no study of the properties of OLAP operations, whereas this study exists for the traditional relational operations [ABI 95]. Only [VAS 00] proposes a logical model and uses it for the optimization of OLAP queries at a logical level. In this work, the authors adapt the classical view subsumption techniques to the
OLAP framework. In that case again our technique can be seen as orthogonal to this one.

In the context of OLAP, the concept of representation of a cube (what [KAR 03] calls the presentation level) attracted very little attention. Many of the logical models and languages proposed in the literature confuse logical level and presentation level. To our knowledge, only two proposals have been made:

– The MDX API proposed in [Mic 98]. This complex SQL-like language, where the intuitions behind the SELECT and WHERE clauses do not really correspond to those of SQL, mixes the presentation and logical level and lacks of theoretical foundations.

– The CPM model [MAN 03], that seems to be an attempt for giving MDX expressions a clear logical foundation. In this model, the logical and presentation levels are separated. Our work can be seen as an extension of this research, towards the use of the presentation level as a basis for query optimisation.

The rest of the paper is structured as follows: In Section 2, we propose a logical definition of a cube and a representation in terms of complex values. In Section 3, we define an algebra by translating the basic OLAP operators into the algebra for complex values. Section 4 deals with the optimization process, and Section 5 draws conclusions and future work.

2. Cubes and Representations

In this work, we make a clear distinction between a cube, a representation of a cube and the visualization of a representation. Intuitively, we propose to consider that logically:

– A cube is the set of all basic facts that can be presented to the user, including the most detailed data and the precomputed aggregates. A cube is composed of a fact table and a set of dimension tables, as in a star schema [JAR 02]. This corresponds to what [VAS 00] calls the detailed data set.

– The representation of a cube is what the user wants to see, under the form of a multidimensional cross-tab. A representation corresponds to the result of a query over a cube or a representation. It is composed of the set of facts the user wants to see and of the description of the axes of the cross-tab. If the representation the user wants to see has more than two axes, as it cannot be easily displayed on a two-dimensional screen, only the first two-dimensional slice of the representation is displayed. Note that in our formalism, this two-dimensional slice is also a representation of a cube.

To take the concept of representation into account, we propose to use the model of complex values [ABI 95] since this model is very well adapted for describing the nesting of attributes. Note that a first investigation of this idea occurs in [DEK 98].

1. For the sake of space we refer the reader to [ABI 95] for a presentation of this model.
In what follows, we distinguish two categories of attributes, as usual in the OLAP context: the member attributes, having names like $L_2$ or $F$, and the measure attribute, named $m$. Without loss of generality, we consider only one measure attribute. In our model, both cubes and representations are complex values $\langle I_A, I_F, I_P \rangle$ having sort the same generic form $\langle A : \text{sort}(A), F : \text{sort}(F), P : \text{sort}(P) \rangle$ where:

- $I_A$ is a complex value that describes:
  - In the case of a cube: The dimensions of the cube and their hierarchies$^2$.
  - In the case of a representation: The axes of the representation.

- $I_F$ is a complex value that describes:
  - In the case of a cube: The complete set of facts of the datacube.
  - In the case of a representation: The facts that the user wants to see.

- $I_P$ is a relation that describes:
  - In the case of a cube: a default position for each member of the cube on an axis of a representation built from this cube.
  - In the case of a representation: The position of each displayed member on an axis.

We now turn to the formal definitions.

2.1. Cube

An instance $I_C$ of an $N$-dimensional cube $C$ is a tuple $I_C = \langle I_{A\text{All}}, I_{F\text{All}}, I_{P\text{All}} \rangle$ that is a complex value having sort $\text{sort}(C) = \langle A_{\text{All}} : \text{sort}(A_{\text{All}}), F_{\text{All}} : \text{sort}(F_{\text{All}}), P_{\text{All}} : \text{sort}(P_{\text{All}}) \rangle$ where:

- $I_{A\text{All}}$ is a tuple $\langle D_1, \ldots, D_N \rangle$ that describes the dimensions of the cube. It is a complex value having sort $\text{sort}(A_{\text{All}}) = \langle D_1 : \text{sort}(D_1), \ldots, D_N : \text{sort}(D_N) \rangle$.

For all $i \in [1, N]$, $I_{D_i}$ is a complex value describing a particular dimension $i$ of the cube. The nested structure of this complex value allows to describe the hierarchy associated with the dimension in a straightforward manner. It has sort: $\text{sort}(D_i) = \{\{L_i^0 : \text{dom}(L_i^0), D_i^1 : \{L_i^1 : \text{dom}(L_i^1), \ldots, D_i^q : \{L_i^q : \text{dom}(L_i^q)\}\}\} \ldots\}$. For a dimension $i$, each attribute $L_i^0$ describes a level of the hierarchy, $j$ being the depth of this level in the dimension. In the following, we denote by $\mathcal{L}_i(C)$ the set of all attributes $L_i^j$ in dimension $i$ of cube $C$, i.e. $\mathcal{L}_i(C) = \{L_i^j \mid 0 \leq j \leq q_i\}$, and by $\mathcal{L}(C)$ the set of all attributes $L_i^j$ of cube $C$, i.e. $\mathcal{L}(C) = \bigcup_{i \in [1, N]} \mathcal{L}_i(C)$. Finally, let $v \in \text{dom}(L_i^j)$ and $k \in [0, j - 1]$. We define the classical notion of ancestors of $v$ at level $k$ by: $\text{anc}(v, L_i^k) = \pi_{L_i^k}(\sigma_{L_i^j = v}(\text{unnest}_{D_i^j}(\ldots(\text{unnest}_{D_i^1}(I_{D_i}))\ldots)))$.

- $I_{F\text{All}}$ is the set of all facts of the cube. It is a complex value having sort $\text{sort}(F_{\text{All}}) = \{(F_{\text{All}}^0 : \text{dom}(F_{\text{All}}^0), \ldots, F_{\text{All}}^q : \text{dom}(F_{\text{All}}^q), m : \text{dom}(m))\}$. $I_{F\text{All}}$

$^2$ For the sake of readability, the model is restricted to only one hierarchy per dimension.
represents the fact table of the cube. It describes every fact at every level of detail. Hence the following equalities hold: \( \forall i \in [1, N], \text{dom}(F_{\text{All}}^i) = \bigcup_{j \in [0,q_i]} \text{dom}(L_{ij}) \).

– \( I_{\text{All}} \) is an instance of relation of sort \( \{ \langle \text{member} : \bigcup_i (\bigcup_j \text{dom}(L_{ij})), \text{position} : \mathbb{N} \rangle \} \) that associates each member of the cube with a distinct integer. This integer corresponds to the default position of a member on an axis when a representation is built from a cube. This relation defines a total ordering over \( \text{dom}(L_{ij}) \), for all \( i \) and \( j \).

**Example 2.1** As an example, we consider a 5 dimensional cube named \( \text{sales} \), inspired by the example used in [Mic 98]. The dimensions are:

– Year: the different years.
– Quarter: the months grouped in quarters.
– Location: the cities grouped in regions and countries.
– Product: the items grouped in categories.
– Salesman: the different salespersons.

The sort of the cube \( \text{sales} \) is:

\[
\begin{align*}
\text{sort}(C_{\text{sales}}) &= \langle A_{\text{All}} : \text{sort}(A_{\text{All}}), F_{\text{All}} : \text{sort}(F_{\text{All}}), P_{\text{All}} : \text{sort}(P_{\text{All}}) \rangle, \\
\text{sort}(A_{\text{All}}) &= \langle \text{Year} : \text{sort}(\text{Year}), \text{Quarter} : \text{sort}(\text{Quarter}), \\
&\quad \text{Location} : \text{sort}(\text{Location}), \text{Product} : \text{sort}(\text{Product}), \\
&\quad \text{Salesman} : \text{sort}(\text{Salesman}) \rangle, \\
\text{sort}(F_{\text{All}}) &= \{ \langle F_{\text{All}}^{1 : \text{allyear}}, 1990, \ldots, 2004 \rangle, F_{\text{All}}^{2 : \text{allquarter}}, q_1, \ldots, \text{dec} \}, \\
&\quad \quad F_{\text{All}}^{3 : \text{alllocation}}, \text{france}, \ldots, \text{paris} \}, \\
&\quad \quad F_{\text{All}}^{4 : \text{allproduct}}, \text{drink}, \ldots, \text{wine} \}, \\
&\quad \quad F_{\text{All}}^{5 : \text{allsalesman}}, \text{john}, \ldots, \text{bill} \} \}, \text{sales} : \mathbb{N} \}, \\
\text{sort}(P_{\text{All}}) &= \{ \langle \text{member} : \{1990, \text{john}, \ldots \}, \text{position} : \mathbb{N} \rangle \}.
\end{align*}
\]

The dimension \( \text{Quarter} \), an instance of \( F_{\text{All}} \) and an instance of \( P_{\text{All}} \) are described in Figure 1. In what follows, we consider an instance \( I_{\text{sales}} \) of the cube \( \text{sales} \).

### 2.2. Constructing a representation of a cube

A representation of a cube is constructed by using the \textit{Navigate} operation. Intuitively, this operation is a selection in the fact table of the cube in order to present the data at a given level for each dimension\(^3\). For example, the query \( \text{Navigate}_{\text{year,quarter,city,item,name}}(I_{\text{sales}}) = I_1 \) allows to obtain the representation displayed in Figure 2 (a). Note that only one 2-dimensional slice of this representation is displayed (on the left-hand side of the figure), i.e., the facts concerning the cities

---

\(3\). This operator is close to the navigate operator proposed in [VAS 00]. It is formally defined in the Appendix.
An instance of the dimension \( \text{Quarter} \):

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Location</th>
<th>Product</th>
<th>Salesman</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1988</td>
<td>( q_1 )</td>
<td>paris</td>
<td>drink</td>
<td>john</td>
<td>90</td>
</tr>
<tr>
<td>1988</td>
<td>( q_1 )</td>
<td>paris</td>
<td>beer</td>
<td>john</td>
<td>30</td>
</tr>
<tr>
<td>1988</td>
<td>( q_1 )</td>
<td>paris</td>
<td>wine</td>
<td>john</td>
<td>20</td>
</tr>
<tr>
<td>1988</td>
<td>( q_1 )</td>
<td>paris</td>
<td>milk</td>
<td>john</td>
<td>20</td>
</tr>
<tr>
<td>1988</td>
<td>( q_1 )</td>
<td>paris</td>
<td>all(all_{product})</td>
<td>john</td>
<td>1000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>all(year)</td>
<td>all(quarter)</td>
<td>all(location)</td>
<td>all(product)</td>
<td>all(salesman)</td>
<td>500000</td>
</tr>
</tbody>
</table>

An instance of \( F_{All} \):

\[
\text{sort}(\text{Quarter}) = \\
\{ (\text{\(top_Q\)} : \{\text{all\(quarter\)}\}, \\
D_1 : \{\text{quarter} : \{\text{\(q_1\)}, \text{\(q_2\)}, \text{\(q_3\)}, \text{\(q_4\)}\}\}, \\
D_2 : \{\text{month} : \{\text{\(\text{jan}\)}, \ldots, \text{\(\text{dec}\)}\}\}\} \\
\]

The sort of the dimension \( \text{Quarter} \)

Figure 1. The dimension \( \text{Quarter} \), the facts and default positions of the cube \( \text{sales} \)
scribing a particular axis of the representation. \( I_{A_k} \) is of sort \( \text{sort}(A_k) = \{(L_{i_0}^k : \text{dom}(L_{i_0}^k), U_k^1 : \{(L_{i_1}^k : \text{dom}(L_{i_1}^k), \ldots, U_k^S : \{(L_{i_S}^k : \text{dom}(L_{i_S}^k)) \ldots))\})\}. \) We denote by \( A_k(R) \) the set of attributes of \( \mathcal{L}(C) \) that occur on \( A_k \), i.e. \( A_k(R) = \{L_{i_0}^k, L_{i_1}^k, \ldots, L_{i_S}^k\} \), and by \( \mathcal{A}(R) \) the set of all attributes of \( \mathcal{L}(C) \) that occur on an axis of \( R \), i.e. \( \mathcal{A}(R) = \bigcup_{k=1}^{K} A_k(R) \).

- \( I_F \) is a complex value describing the facts depicted by the representation. \( I_F \) is of sort \( \text{sort}(F) = \{(I_{d_1}^1 : \text{dom}(I_{d_1}^1), \ldots, I_{d_S}^N : \text{dom}(I_{d_N}^N), m : \text{dom}(m))\}. \) We denote by \( \mathcal{F}(R) \) the set of attributes of \( \mathcal{L}(C) \) that occur in \( \text{sort}(F) \), i.e. \( \mathcal{F}(R) = \{I_{d_1}, \ldots, I_{d_N}\} \).

- \( I_p \) is a complex value of sort \( \{\text{member} : \bigcup_i (\text{dom}(L_i)), \text{position} : \mathbb{N}\} \) that associates each member of \( R \) with an integer. This integer corresponds to the relative position of the member on an axis of the representation.

**Example 2.2** As an example, we consider the representation \( I_7 \) of Figure 3 (c). \( I_7 = \langle I_A, I_F, I_P \rangle \) of sort \( \text{sort}(I_7) = \langle A : \text{sort}(A), F : \text{sort}(F), P : \text{sort}(P) \rangle \) where \( I_A = \langle I_{A_5}, I_{A_2}, I_{A_4}, I_{A_1} \rangle \) and:

- \( \text{sort}(A) = \langle A_5 : \text{sort}(A_5), A_2 : \text{sort}(A_2), A_4 : \text{sort}(A_4), A_1 : \text{sort}(A_1) \rangle \), with, for instance:
  \[
  \text{sort}(A_5) = \{\text{name} : \text{dom(name)}, \text{region} : \text{dom(region)}, \text{city} : \text{dom(city)}\}\}
  \]
  \[
  I_{A_5} = \langle \text{john}, U_5^1 : \{(\text{north}, U_5^2 : \{(\text{paris}, \text{blais}, \text{lille})\}, \ldots)\}, \ldots, \rangle
  \]
  \[
  I_F = \{(2000, q_1, \text{paris}, \text{drink}, \text{john}, 10), (2000, q_1, \text{blais}, \text{drink}, \text{john}, 20),
  (2000, q_1, \text{lille}, \text{drink}, \text{john}, 70) \ldots\}
  \]
  \[
  \text{sort}(P) = \{\text{member} : \bigcup_i (\text{dom}(L_i)), \text{position} : \mathbb{N}\}\}
  \]
  \[
  I_P = \{\langle \text{john}, 1\rangle, \langle \text{north}, 1\rangle, \langle \text{paris}, 1\rangle, \ldots, \langle q_1, 1\rangle, \ldots, \langle \text{drink}, 1\rangle \ldots, (2000, 1)\}
  \]

3. **OLAP operations**

In this section, we describe how cubes and representations are manipulated using the most typical OLAP operators [MAR 99]. Basically, using the definition of cubes and representations above, we translate the OLAP operators into the algebra for complex values (as defined in [ABI 95]). We consider the following OLAP operators, that are classified according to 3 categories:

- Restructuring operators that change the viewpoint on data. Operators in this category are **Permute, Switch, Nest**.
Figure 2. Outputs of steps 1-4. On each figure, the left-hand side displays the first 2-dimensional slice of the representation, and the right-hand side displays the member at the lowest position for each hidden axis.

(a) $I_1 = \text{Navigate}_{\text{year,quarter,city,item,name}}(I_{\text{sales}})$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Quarter</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>10</td>
<td>50</td>
<td>70</td>
<td>50</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>40</td>
<td>10</td>
<td>80</td>
<td>10</td>
<td>80</td>
<td>50</td>
<td>30</td>
</tr>
<tr>
<td>City</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Item</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) $I_2 = \pi_{\text{year}}(\pi_{\text{quarter}}(\pi_{\text{region,city}}(\pi_{\text{category,item}}(\pi_{\text{name}}(I_1)))))$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Quarter</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>10</td>
<td>50</td>
<td>70</td>
<td>50</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>40</td>
<td>10</td>
<td>80</td>
<td>10</td>
<td>80</td>
<td>50</td>
<td>30</td>
</tr>
<tr>
<td>City</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Item</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(c) $I_3 = \sigma_{\text{member}}(\pi_{\text{item=beer, item=wine, category=food}}(I_2))$

(d) $I_4 = \text{Aggregate}_{\text{category=food}}(\text{sum}(\text{sales}))(I_3)$
Figure 3. Outputs of steps 5-7

- Operators that change the level of detail. We consider two different operators in this category:
  
  - The Navigate operator, introduced in the previous section, defines for each dimension of the cube at which level the data are to be represented.
the Aggregate operator groups the members of an axis according to the part of the hierarchy displayed in the representation, and then aggregates the measures accordingly.

– Filtering operators, that are the extension of classical selection and projection to representations.

Note that the classical roll-up, drill-down or slice&dice OLAP queries can be expressed in our framework by using this set of operators. For instance, as the Navigate operator selects from the fact table the data at a given level for each dimension, it can be used to express drill-down or roll-up queries.

We introduce the different operators in an example, the formal definitions are given in the Appendix. The example is based on the following query:

\[
\text{Switch}_{\text{year}}:1988,2000(\text{Nest}_{A_5}(A_3)(\sigma_{\text{member.region}=\text{north}}(\text{Permute}_{A_1,A_5}(\text{\textit{Aggregate}}_{\text{item.category};\sum(\text{sales})}(\sigma_{\text{item.beer;item.wine}\vee\text{category}=\text{food}}(\pi_{A_1}(\pi_{A_2}(\pi_{A_3}(\pi_{\text{category.item}}(\pi_{A_5}(\text{\textit{Navigate}}_{\text{year.quarter.city.item.name}(I_{\text{sales}})))))))))))))))
\]

This query aims at presenting the sales of food and the cumulated sales of beer and wine for year 2000 in the north, detailed by salespersons and quarters. The final output of this query is given in Figure 3 (c). We examine the different steps independently. The first step, which involves the Navigate operation, is illustrated in the previous section. We now illustrate the filtering operations.

3.1. Filtering

We illustrate step 2 and step 3 of the query:

– step 2: \(I_2 = \pi_{A_1}(\pi_{A_2}(\pi_{A_3}(\pi_{\text{category.item}}(\pi_{A_5}(\text{\textit{Navigate}}_{\text{year.quarter.city.item.name}(I_{\text{sales}})))))))) \)

– step 3: \(I_3 = \sigma_{\text{item.beer;item.wine}\vee\text{category}=\text{food}}(I_2) \)

The outputs are given in Figure 2 (b) and (c). These two steps reduce the amount of displayed facts (for selection) and displayed levels (for projection).

Note that after step 3, the displayed slice has changed. This is due to the fact that the item milk, that was at position 1 of axis \(A_4\) in \(I_2\) is no more selected in representation \(I_3\). Then among the selected items, the one at the lowest position, beer, is displayed first. Note also that in our formalism, a representation must always depict the hierarchies that have been used to aggregate the data. Thus, for a given dimension, selection on members can not be applied on a member attribute that identifies a level deeper than the level currently displayed. Otherwise the measures displayed would no more correspond to the displayed hierarchy. The same remark holds for the projection operation.
3.2. Changing the level of detail

Step 4, i.e., $I_4 = \text{Aggregate}_{\text{item} \rightarrow \text{category}}(\sum \text{sales})(I_3)$ illustrates a way of changing the level of detail of the representation. Note that we distinguish two different ways of changing the level of detail: by using the Navigate operation to present the data at a given level for each dimension, or by using the Aggregate operation. Intuitively, the Aggregate operation groups and aggregates the data according to the groupings depicted on a particular axis. This operation does not change the sort of the axes.

The output of step 4 is given in Figure 2 (d). Note that the facts that are displayed are no more the facts in the fact table of the cube sales. This is so because the Aggregate operator has been used to compute the sales of drink, but only for the drinks that were selected in step 3, i.e., beer and wine.

3.3. Restructuring

The last operations to be considered are the restructuring operations. These operations allow to exchange the positions of two axes (Permute), exchange the positions of two members (Switch) or nest two axes together (Nest). They are illustrated by the following steps:

- step 5: $I_5 = \text{Permute}_{A_5,A_1}(I_4)$
- step 6: $I_6 = \text{Nest}_{A_5}(A_3)(\sigma_{\text{region}=\text{north}}(I_5))$
- step 7: $I_7 = \text{Switch}_{\text{year}=1988,2000}(I_6)$

The outputs of these three steps are given in Figure 3. Note that these operations do not change the facts of the representation, but only the way they are displayed.

4. OLAP query optimisation

In this section, we propose an optimisation technique for OLAP queries. We first introduce this technique informally.

4.1. Intuitions

This optimisation technique consists in determining which part of the query output will be displayed on the screen. This part is the first 2-dimensional slice of the representation, which is also a representation. This slice can be computed by adding selection conditions to the initial query. Intuitively, these conditions are obtained by inspecting the parameters of the restructuring and filtering operations in the query. As the costly operation is Aggregate, the optimization consists in pushing the conditions
before the *Aggregate*. We first explain how selection conditions are computed, and then we present how these conditions are added to the query.

When statically inspecting a query, the *Nest* and projection operations are used to identify which attributes are displayed and on which axis. The *Permute* operation is used to identify the first 2 axes of the representation, i.e., the axes that are fully displayed. The remaining hidden axes contain attributes for which only values at the lowest position are visible. These values can be deduced from the *Switch* operation and the default positions of the members.

As an example, we consider the query presented in Section 3. It can be found by inspecting the parameters of the *Permute*, *Nest*, *Switch* and projection operations that:

- The final representation has 4 axes.
- The attributes on the two hidden axes are *year*, *item* and *category*.
- The values of these attributes at the lowest position are respectively 2000, *beer* and *drink*.

It can be found by inspecting the parameters of the *Aggregate* and selection operations that:

- The facts displayed concern levels *year*, *quarter*, *category*, *city* and *salesman*.
- The measures for *drink* are computed only from the measures for *beer* and *wine*.

Hence the selection condition that is generated is \( \text{year} = 2000 \land \text{category} = \text{drink} \).

### 4.2. Canonical form of an OLAP query

Our optimisation technique is based on a particular form of queries, which we call *canonical form*. We define the canonical form of an OLAP query, that complies with the standard intuition behind OLAP queries: A *Navigate* identifies the levels at which the facts are to be represented, some ad-hoc aggregations, i.e., sequences of selections and aggregations, can be performed, and finally the restructuring operations construct the viewpoint. The canonical form of a query is given below (\( \circ \) denotes the composition of operations):

\[
Q = \circ^{\mathcal{W}}_{k=1} \text{Switch}_{L_{x_{k}}^k:v_{k}^k:v_{k}^{\prime}} \circ^{\mathcal{P}}_{k=1} \text{Permute}_{A_{x_{k}}:A_{y_{k}}} \circ^{\mathcal{S}}_{k=1} \text{Nest}_{A_{x_{k}}(A_{v_{k}})} \\
\circ^{\mathcal{N}}_{i=1} \pi_{X_i} \circ^{\mathcal{C}}_{k=1} [\text{\sigma}^{\text{measure}}_{\phi_k} \circ \text{Aggregate}_{L_{x_{k}}^k \rightarrow L_{y_{k}}^k; f(m)} \circ \text{\sigma}^{\text{member}}_{\phi_k}]
\]
the following lemma, which proof can be found in [GIA 04], states that all queries have an equivalent canonical form.

**Lemma 4.1** Every OLAP query can be put under the canonical form.

Moreover, the rewriting rules presented in Figure 4 allow to transform any query into its canonical form. Note that the last rule is valid since in our formalism, selection cannot be applied on an attribute that identifies a level deeper than the displayed level. For example, on representation $I_7$ displayed in Figure 3 (c), selection on item, e.g., $item = beer$ is not allowed, because facts are displayed at the category level.

Among the algebraic operators, the *Aggregate* is the most costly since it is used to compute new facts. So the principle of our technique is to propagate the selection conditions before the aggregate operation. Getting back to our example, the rewritten query is:

$$
\text{Perm}
\text{ute}_{A_1, A_2}(\text{Nest}_{A_3}(\pi_{A_1} \pi_{A_2} \pi_{A_3} \pi_{\text{region}, \text{city}} \pi_{\text{category}, \text{item}} \pi_{\text{name}}
$$

$$
\text{Aggregate}_{\text{item} \rightarrow \text{category}}; \text{sum}(\text{sales}) (\sigma_{\text{member}} (\text{Navigate}_{\text{year}, \text{quarter}, \text{city}, \text{item}, \text{name}}(I_{\text{sales}}))))
$$

where $S = (\text{region} = \text{north} \land (\text{item} = \text{beer} \lor \text{item} = \text{wine} \lor \text{category} = \text{food}) \land (\text{year} = 2000 \land \text{category} = \text{drink})$. Moreover it is easy to see that condition $\text{category} = \text{drink}$ is redundant in $S$, thus we can consider $S = (\text{region} = \text{north} \land (\text{item} = \text{beer} \lor \text{item} = \text{wine}) \land \text{year} = 2000)$.

### 4.3. Optimisation algorithm

We are now ready to present the algorithm for optimizing OLAP queries. It consists of one function, named *Compute_First_Slice_Selection*, given Figure 5.

Let us examine each step of the function. Recall that $\text{Navigate}_{L_{d_1} \ldots L_{d_N}}$ outputs a representation having $N$ axes, with attributes of dimension $D_i$ on axis $A_i$.

1) This step only looks for the axes of the representation.

2) For every axis, this step performs the following four actions:

   a) It finds the depth at which the data will be presented. For axis $A_i$, this depth is either $d_i$ if $A_i$ is not one of the parameters of $\text{Aggregate}$ (recall that $d_i$ is the depth on axis $A_i$ according to which the facts of $I$ are depicted), or it is the smallest $l_k$ such that $\text{Aggregate}_{L_{d_i} \ldots L_{d_N}} \rightarrow L_{d_i} \ldots L_{d_N}; f(m) \in Q$ otherwise.
Figure 4. Rewriting rules for OLAP algebra. In this figure, \( \circ \) denotes the composition of operations, \( Op \in \{\text{Permute, Switch, Nest, } \pi\} \), \( \sigma^{mb} \) denotes a selection on members, and \( \sigma \) denotes a selection on measures or members.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>( \text{Nest}<em>{A_i}(\text{Permute}</em>{A_j}(I)) = \text{Permute}<em>{A_j}(\text{Nest}</em>{A_i}(I)) ) if ( j \notin [k,l] )</td>
</tr>
<tr>
<td>r2</td>
<td>( \text{Nest}<em>{A_i}(\text{Permute}</em>{A_j}(I)) = \sigma^{mb}<em>{\text{Permute}</em>{A_j}(\text{Nest}_{A_i}(I))} )</td>
</tr>
<tr>
<td>r3</td>
<td>( \text{Nest}<em>{A_i}(\text{Switch}</em>{\text{Nest}<em>{A_j}(I)}(I)) = \text{Switch}</em>{\text{Nest}_{A_j}(I)}(I) )</td>
</tr>
<tr>
<td>r4</td>
<td>( \text{Nest}<em>{A_i}(\text{Switch}</em>{\text{Permute}<em>{A_j}(\text{Nest}</em>{A_k}(I))}(I)) = \text{Switch}<em>{\text{Permute}</em>{A_j}(\text{Nest}_{A_k}(I))}(I) )</td>
</tr>
<tr>
<td>r5</td>
<td>( \text{Permute}<em>{A_i,A_j}(\text{Switch}</em>{\text{Nest}<em>{A_k}(I)}(I)) = \text{Switch}</em>{\text{Nest}<em>{A_k}(I)}(\text{Permute}</em>{A_i,A_j}(I)) )</td>
</tr>
<tr>
<td>r6</td>
<td>( \text{Switch}<em>{\text{Permute}</em>{A_i,A_j}(\text{Nest}<em>{A_k}(I))}(I) = \text{Permute}</em>{A_i,A_j}(\text{Switch}<em>{\text{Nest}</em>{A_k}(I)}(I)) )</td>
</tr>
<tr>
<td>r7</td>
<td>( \sigma_{\text{Op}(I)}(\text{Switch}<em>{\text{Nest}</em>{A_k}(I)}(I)) = \text{Op}(\sigma_{\text{Op}(I)}(I)) )</td>
</tr>
<tr>
<td>r8</td>
<td>( \sigma^{mb}<em>{\text{Op}(\text{Switch}</em>{\text{Nest}<em>{A_k}(I)}(I))} = \text{Op}(\sigma^{mb}</em>{\text{Switch}<em>{\text{Nest}</em>{A_k}(I)}(I)}) )</td>
</tr>
<tr>
<td>r9</td>
<td>( \text{Nest}<em>{A_i}(\text{Switch}</em>{\text{Permute}<em>{A_j}(\text{Nest}</em>{A_k}(I))}(I)) = \text{Switch}<em>{\text{Permute}</em>{A_j}(\text{Nest}<em>{A_k}(I))}(\text{Nest}</em>{A_i}(I)) )</td>
</tr>
<tr>
<td>r10</td>
<td>( \sigma^{mb}<em>{\text{Switch}</em>{\text{Permute}<em>{A_j}(\text{Nest}</em>{A_k}(I))}(I)} = \text{Switch}<em>{\text{Permute}</em>{A_j}(\text{Nest}<em>{A_k}(I))}(\sigma^{mb}</em>{\text{Switch}<em>{\text{Permute}</em>{A_j}(\text{Nest}_{A_k}(I))}(I)}) )</td>
</tr>
</tbody>
</table>

b) It assigns to the axis a location number in \([1,N]\), which is \( i \) for axis \( A_i \).

c) It finds the set of attributes appearing on the axis, by using the parameters of the projection operation. These attributes are simply \( X_i \) for axis \( A_i \).

d) It finds the initial position of every member appearing on the axis.

3) For each axis being the result of \( \text{Nest} \) operations, this step computes the set of attributes appearing on this axis. For each \( \text{Nest}_{A_{x_k}}(A_{y_k}) \) operation, the set of attributes on axis \( A_{x_k} \) is the union of the sets of attributes of \( A_{z_k} \) and \( A_{y_k} \). Note that when two axes of a representation are nested, the resulting representation has one axis less. This means that the location number of the axes that are located after \( A_{y_k} \) has to be updated. After this step, only \( N - S \) axes remain. The location number of each of these \( N - S \) axes remains in \([1,N-S]\). The set \( \mathcal{A}' \) contains these \( N-S \) axes.

4) This steps computes the new location number of two axes if these two axes are permuted.

5) This steps computes the new position of two members if these two members are switched. Note that the members that are not switched keep their original position, i.e., the ones that are given by the cube \( I \).

6) This step computes the selection conditions. Recall that the idea is to find the members at the lowest position on the axes that are hidden. These axes are the ones of \( \mathcal{A}' \) having a location number greater than 2. For each such axis \( i \), we consider each attribute appearing on this axis. For each such attribute \( L^k_i \), we identify the members that will belong to the resulting representation, i.e., the members that appear in the selection condition \( \psi^{L^k_i} \) (or all the members of these attributes if this condition is \( \text{true} \)). Then the member \( v \) having the lowest position is found, and the condition \( L^k_i = v \) is generated.
**Function** Compute First Slice Selection\([Q]\)

**Input:** A query under canonical form
\[
Q = \circ_{k=1}^{w} Switch_{\psi_{k=1}} \circ_{k=1}^{P} Permute_{\alpha_{k=1}} \circ_{k=1}^{S} Nest_{\nu_{k=1}} (A_{\nu_{k}})
\]
where for all \(k\), \(\sigma_{\nu_{k}}^\text{measure} \circ \Delta_{N}^{I_{k}} \circ \Delta_{m}^{I_{k}} \circ \Delta_{N}^{I_{k}} \circ \Delta_{m}^{I_{k}} \circ \Delta_{N}^{I_{k}}

**Output:** A condition \(\varphi\) of selection

**Local:** \(att(A_{i})\) is the set of attributes on axis \(A_{i}\), \(loc(A_{i})\) is the location of axis \(A_{i}\), \(pos(v_{k})\) is the position of a member \(v_{k}\)

1. Let \(A = \{A_{1}, \ldots, A_{N}\}\)
2. \# Aggregate and Projection operators
   for \(i = 1\) to \(N\) do begin
      \(k_{i} = \min\{\{I_{k} | Aggregate_{L_{k}^{i}} \circ F_{k} \in Q \land i_{k} = i\}\} \cup \{d_{i}\}\)
      \(loc(A_{i}) = i\)
      \(att(A_{i}) = X_{i}\)
      for every \(v \in \bigcup_{j} (\bigcup_{j} dom(L_{j}^{i}))\) such that \(L_{j}^{i} \in att(A_{i})\) do
         \(pos(v) = \pi_{\text{position}}(\sigma_{L_{j}^{i}=v}(I_{F_{All}}))\)
   end for
3. \# Nest operators
   for \(k = S\) to \(1\) do begin
      \(att(A_{\nu_{k}}) = att(A_{\nu_{k}}) \cup att(A_{\nu_{k}})\);
      for \(A_{i} \in A\) such that \(loc(A_{i}) > loc(A_{\nu_{k}})\) do \(loc(A_{i}) = loc(A_{i}) - 1\)
   end for
4. \# Permute operators
   for \(k = P\) to \(1\) do \(x = loc(A_{\nu_{k}})\); \(loc(A_{\nu_{k}}) = loc(A_{\nu_{k}})\); \(loc(A_{\nu_{k}}) = x\) end for
5. \# Switch operators
   for \(k = W\) to \(1\) do \(x = pos(v_{k})\); \(pos(v_{k}) = pos(v_{k}')\); \(pos(v_{k}') = x\) end for
6. \# Selection conditions on the hidden axes
   \(\varphi = \text{true}\)
   \(A' = \{A_{i} \in A | \exists Nest_{A_{i}}(A_{i}) \in Q\}\)
   for \(l = 3\) to \(|A'|\) do begin
      Let \(A_{i} \in A'\) such that \(loc(A_{i}) = l\);
      for every \(i\) such that \(L_{i}^{k_{i}} \in att(A_{i})\) do begin
         \(\varphi = \varphi \land (L_{i}^{k_{i}} = v)\) where \(v \in \text{dom}(L_{i}^{k_{i}})\) and
         \(pos(v) = \min\{\{pos(v') | v' \in \pi_{\text{position}}(\sigma_{\phi}(\text{unnest}_{D_{i}}(\ldots(\text{unnest}_{D_{i}}(I_{D_{i}}))))\})\}\)
      where \(\sigma_{\phi} = \circ_{k=1}^{G} \sigma_{\nu_{k}}\)
   end for
   end for
7. Return \(\varphi\)

**Figure 5.** The function Compute First Slice Selection
Example 4.1 Consider the OLAP query used throughout the paper:

\[ q = \text{Switch}_{year,1988,2000} \left( \text{Switch}_{\text{region}} \left( \text{Nest}_{A_5}(A_3) \text{Permute}_{A_1,A_5} \left( \begin{array}{c}
\pi_{\text{member}} \\
\sigma_{\text{region}=\text{north}} \\
\text{Aggregate}_{\text{item} \rightarrow \text{category} \rightarrow \text{sum}(\text{sales})} \\
\pi_{\text{year}}(\pi_{\text{quarter}}(\pi_{\text{region,city}}(\pi_{\text{category,item}}(\pi_{\text{name}}( I_{\text{sales}}))))))
\end{array} \right) \right) \right) \]

First, we illustrate how the rewriting rules are used to put this query under canonical form. The rules can be applied as follows:

- \[ q \xrightarrow{r_5} q_1 \] (selection can be performed before restructuring)
- \[ q_1 = \text{Switch}_{year,1988,2000} \left( \text{Nest}_{A_5}(A_3) \text{Permute}_{A_1,A_5} \left( \begin{array}{c}
\pi_{\text{member}} \\
\sigma_{\text{region}=\text{north}} \\
\text{Aggregate}_{\text{item} \rightarrow \text{category} \rightarrow \text{sum}(\text{sales})} \\
\pi_{\text{year}}(\pi_{\text{quarter}}(\pi_{\text{region,city}}(\pi_{\text{category,item}}(\pi_{\text{name}}( I_{\text{sales}}))))))
\end{array} \right) \right) \]
- \[ q_2 = \text{Switch}_{year,1988,2000} \left( \text{Switch}_{\text{region}} \left( \text{Nest}_{A_5}(A_3) \text{Permute}_{A_1,A_5} \left( \begin{array}{c}
\pi_{\text{member}} \\
\sigma_{\text{region}=\text{north}} \\
\text{Aggregate}_{\text{item} \rightarrow \text{category} \rightarrow \text{sum}(\text{sales})} \\
\pi_{\text{year}}(\pi_{\text{quarter}}(\pi_{\text{region,city}}(\pi_{\text{category,item}}(\pi_{\text{name}}( I_{\text{sales}}))))))
\end{array} \right) \right) \]
- \[ q_3 = \text{Switch}_{year,1988,2000} \left( \text{Switch}_{\text{region}} \left( \text{Nest}_{A_5}(A_3) \text{Permute}_{A_1,A_5} \left( \begin{array}{c}
\pi_{\text{member}} \\
\sigma_{\text{region}=\text{north}} \\
\text{Aggregate}_{\text{item} \rightarrow \text{category} \rightarrow \text{sum}(\text{sales})} \\
\pi_{\text{year}}(\pi_{\text{quarter}}(\pi_{\text{region,city}}(\pi_{\text{category,item}}(\pi_{\text{name}}( I_{\text{sales}}))))))
\end{array} \right) \right) \]
- \[ q_4 = \text{Switch}_{year,1988,2000} \left( \text{Switch}_{\text{region}} \left( \text{Nest}_{A_5}(A_3) \text{Permute}_{A_1,A_5} \left( \begin{array}{c}
\pi_{\text{member}} \\
\sigma_{\text{region}=\text{north}} \\
\text{Aggregate}_{\text{item} \rightarrow \text{category} \rightarrow \text{sum}(\text{sales})} \\
\pi_{\text{year}}(\pi_{\text{quarter}}(\pi_{\text{region,city}}(\pi_{\text{category,item}}(\pi_{\text{name}}( I_{\text{sales}}))))))
\end{array} \right) \right) \]
- \[ q_5 = \text{Switch}_{year,1988,2000} \left( \text{Switch}_{\text{region}} \left( \text{Nest}_{A_5}(A_3) \text{Permute}_{A_1,A_5} \left( \begin{array}{c}
\pi_{\text{member}} \\
\sigma_{\text{region}=\text{north}} \\
\text{Aggregate}_{\text{item} \rightarrow \text{category} \rightarrow \text{sum}(\text{sales})} \\
\pi_{\text{year}}(\pi_{\text{quarter}}(\pi_{\text{region,city}}(\pi_{\text{category,item}}(\pi_{\text{name}}( I_{\text{sales}}))))))
\end{array} \right) \right) \]
- \[ q_6 = \text{Switch}_{year,1988,2000} \left( \text{Switch}_{\text{region}} \left( \text{Nest}_{A_5}(A_3) \text{Permute}_{A_1,A_5} \left( \begin{array}{c}
\pi_{\text{member}} \\
\sigma_{\text{region}=\text{north}} \\
\text{Aggregate}_{\text{item} \rightarrow \text{category} \rightarrow \text{sum}(\text{sales})} \\
\pi_{\text{year}}(\pi_{\text{quarter}}(\pi_{\text{region,city}}(\pi_{\text{category,item}}(\pi_{\text{name}}( I_{\text{sales}}))))))
\end{array} \right) \right) \]

The query \( q_6 \) can now be used as input of the function Compute_First_Slice_Selection. Note that for notational purpose, the selection

\[ \sigma^{new}_{(\text{region}=\text{north}) \land (\text{item}=\text{beer} \lor \text{item}=\text{wine} \land \text{category} = \text{food})} \] is rewritten \( \sigma^{\text{new}} = \emptyset_1^0 \) with:
We illustrate each step of the algorithm:

1) \( A = \{A_1, A_2, A_3, A_4, A_5\} \)

2) For \( i = 1 \) to 5 we have:

<table>
<thead>
<tr>
<th>( i )</th>
<th>( k_i )</th>
<th>( \text{att}(A_i) )</th>
<th>( \text{loc}(A_i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>\text{year}</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>\text{quarter}</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>\text{region, city}</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>\text{category, item}</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>\text{name}</td>
<td>5</td>
</tr>
</tbody>
</table>

For instance, when \( i = 4, d_i = 2 \) since \( \text{item} \), the level at depth 2, appears in the parameter list of the \( \text{Navigate} \) operation. \( k_4 = 1 \), since \( \text{category} \), the level at depth 1, appears in the parameter list of the \( \text{Aggregate} \) operation, and thus \( k_4 = \min\{1\} = 1 \). \( \text{att}(A_4) \) is the list of parameter of the \( \pi_{A_4} \) operation, and \( \text{loc}(A_4) \) is set to 4.

3) The only Nest operation is \( \text{Nest}_{\{A_2, A_3\}} \). Hence \( \text{att}(A_5) = \{\text{name, region, city}\} \), and \( \text{loc}(A_4) = 3, \text{loc}(A_5) = 4 \).

4) The only Permute operation is \( \text{Permute}_{A_1, A_2} \). Hence \( \text{loc}(A_1) = 4 \) and \( \text{loc}(A_5) = 1 \).

5) The only Switch operation is \( \text{Switch}_{\text{year}, 1988, 2000} \). Hence \( \text{pos}(1988) = 13 \) and \( \text{pos}(2000) = 1 \).

6) As we have \( A' = \{A_1, A_2, A_4, A_5\}, l \) ranges from 3 to 4.

   - When \( l = 3, A_k = A_4 \) and \( \text{att}(A_4) = \{\text{category, item}\} \). Moreover, \( k_4 = 1 \), \( L_1 = \text{category} \in \text{att}(A_4) \) and \( \sigma_{A_4} = \sigma_{A_4} = (\text{item} = \text{beer} \lor \text{item} = \text{wine}) \lor (\text{category} = \text{food}) \). Thus:
     \[
     \pi_{\text{category}}(\sigma_{A_4}(\text{unnest}_{D_2}(\text{unnest}_{D_1}(\text{Product})))) = \{\text{drink, food}\}.
     \]
     As \( \text{pos}(	ext{drink}) = 1 < \text{pos}(	ext{food}) = 2 \) (see Figure 1 (b)), we have \( v = \text{drink} \) and \( \varphi = \text{true} \lor (\text{category} = \text{drink}) \).

   - When \( l = 4, A_k = A_1 \) and \( \text{att}(A_1) = \{\text{year}\} \). Moreover \( k_1 = 1 \), \( L_1 = \text{year} \in \text{att}(A_1) \) and \( \sigma_{A_1} = \sigma_{A_1} = \text{true} \). Thus:
     \[
     \pi_{\text{year}}(\sigma_{A_1}(\text{unnest}_{D_2}(\text{IYear}))) = \{1988, \ldots, 2004\}.
     \]
     As \( \text{pos}(2000) = 1 \), we have \( v = 2000 \) and \( \varphi = (\text{category} = \text{drink}) \land (\text{year} = 2000) \).

7) The function returns the selection condition \( \varphi = (\text{category} = \text{drink}) \land (\text{year} = 2000) \).

\[\text{Discussion}\]

It is important to note that our optimisation technique consists in a syntactical exploration of the query, and thus it does not require accessing the data of the cube from which the representation is computed. In that respect the cost of the optimisation can be seen as not significant compared to the global cost of processing the query.
Note also that, as mentioned in the introduction, our optimisation technique can be used in conjunction with other optimisation techniques, such as for instance techniques taking advantage of materialized aggregates. Optimising an OLAP query by combining the two techniques could be done, in a naive way, by first applying our technique to determine what is the first slice to be displayed, and then determine what are the materialized aggregates to be queried.

5. Conclusion

In this paper we have proposed a new way of optimizing OLAP queries, based on the output of a query on a screen. We have used the model of complex values to define a cube and a representation of a cube at a logical level. We defined an algebra for manipulating cubes and representations and we showed how the properties of the algebraic operators can be used to optimize queries.

We are currently working on the following two open issues. First, we are investigating a way for optimizing sequences of OLAP queries. For example, if a query \( I' = Q_1(I) \) has been optimized, we can distinguish two cases for optimizing \( Q_2(I') \):

- Take advantage of what has been done for optimizing \( Q_1 \), if it can be found that \( Q_2 \) can be computed only by looking at the first slice of \( I' \), or
- Use the rewriting rules for optimizing \( Q_2(Q_1(I)) \) otherwise.

Second, we are extending both the language and the definition of representations. In this respect, typical restructuring operators like Push, Pull, or Unnest are under consideration, and we are thinking of defining more sophisticated representations, e.g., with nested cells. Finally we are currently implementing our approach.

6. References


Appendix

In this appendix, we give the formal definitions of the algebraic operators of the language. Note that for this language, we consider only atomic operators, i.e., Permute only exchanges two axes, Switch only exchanges two members. In that sense, the Rotate operation cannot be considered as atomic and can be expressed as a combination of Permutates and Switches. All operators are defined on representations, except the Navigate operator which is defined on a cube, and allows to construct an initial representation. The operators defined on representations are applied on an instance $I_R = \langle I_A, I_F, I_P \rangle$ of a representation $R$ having sort $\text{sort}(R) = \langle A : \text{sort}(A), F : \text{sort}(F), P : \text{sort}(P) \rangle$, where $I_A = \langle I_{A_1}, \ldots, I_{A_K} \rangle$ and $\text{sort}(A) = \langle A_1 : \text{sort}(A_1), \ldots, A_K : \text{sort}(A_K) \rangle$.

Restructuring

**Permute:** The Permute operation consists of interchanging two axes of a representation. Let $I_R = \langle I_A, I_F, I_P \rangle$ be an instance of a representation $R$. Given two axes $A_i$ and $A_j$ of $R$, $\text{Permute}_{A_i, A_j}(I_R)$ is an instance $I_{R'} = \langle I_{A'}, I_F, I_P \rangle$ of a representation $R'$ having sort $\text{sort}(R') = \langle A' : \text{sort}(A'), F : \text{sort}(F), P : \text{sort}(P) \rangle$ where:

- $\text{sort}(A') = \langle A_1 : \text{sort}(A_1), \ldots, A_i : \text{sort}(A_i), A_{i+1} : \text{sort}(A_{i+1}), \ldots, A_{i-1} : \text{sort}(A_{i-1}), A_j : \text{sort}(A_j), A_{j+1} : \text{sort}(A_{j+1}), \ldots, A_K : \text{sort}(A_K) \rangle$,

- $I_{A'} = \langle I_{A_1}, \ldots, I_{A_{i-1}}, I_{A_j}, I_{A_{i+1}}, \ldots, I_{A_{i-1}}, I_{A_i}, I_{A_{i+1}}, \ldots, I_{A_K} \rangle$.

Note that $\text{Permute}_{A_i, A_j}(I_R) = I_R$ for every $k \in [1, K]$. 

References


Switch: Let $I_R = \langle I_A, I_F, I_P \rangle$ be an instance of a representation $R$. Let $A_i$ be an axis of $R$ with $\text{sort}(A_i) = \{\langle L^0_i : \text{dom}(L^0_i), U^1_i : \{\langle L^1_i : \text{dom}(L^1_i), \ldots, U^b_i : \{\langle \text{dom}(L^b_i) \rangle \} \} \ldots \} \}$. 

The Switch operation consists of interchanging the positions $p$ and $p'$ of two values $v$ and $v'$ of an attribute $L^b_i$ on axis $A_i$. It is defined if $k = 0$, or if $i_k \neq i_{k-1}$, or if $i_k = i_{k-1}$ and $\text{anc}(v, X) = \text{anc}(v', X)$ where $X = L^j_{ik}$. 

Switch $\text{u}_k^{\text{ij}, v, v'}(I_R)$ is an instance $I_{R'} = \langle I_A, I_F, I_P' \rangle$ of a representation $R'$ having sort $\text{sort}(R') = \text{sort}(R)$, where:

- $I_P' = (I_P - I) \cup (\sigma_{\text{member}\_\text{position}}(\rho_{\text{position}\_\text{position}} - \rho_{\text{member}\_\text{position}}(I)) - I)$, with:
  - $I = \sigma_{\text{member}\_\text{position}}(\rho_{\text{member}\_\text{position}}(I_P))$

Nest: Let $I_R = \langle I_A, I_F, I_P \rangle$ be an instance of a representation $R$. Let $A_i$ and $A_j$ be two axes of $R$ with:

- $\text{sort}(A_i) = \{\langle L^0_i : \text{dom}(L^0_i), U^1_i : \{\langle L^1_i : \text{dom}(L^1_i), \ldots, U^b_i : \{\langle \text{dom}(L^b_i) \rangle \} \ldots \} \} \ldots \} \}

- $\text{sort}(A_j) = \{\langle L^0_j : \text{dom}(L^0_j), U^1_j : \{\langle L^1_j : \text{dom}(L^1_j), \ldots, U^b_j : \{\langle \text{dom}(L^b_j) \rangle \} \ldots \} \} \ldots \} \}

The Nest operation allows to nest the attributes initially on axis $A_i$, with the attributes on axis $A_j$. Nest$_{A_i}(A_j)(I_R)$ is an instance $I_{R'} = \langle I_A, I_F, I_P' \rangle$ of a representation $R'$ having sort $\text{sort}(R') = \langle A' : \text{sort}(A'), F : \text{sort}(F), P : \text{sort}(P) \rangle$, where:

- $\text{sort}(A') = \{A_1 : \text{sort}(A_1), \ldots, A_i : \tau_{A_1}, \ldots, A_{j-1} : \text{sort}(A_{j-1}), A_{j+1} : \text{sort}(A_{j+1}), \ldots, A_K : \text{sort}(A_K) \}$, with $\tau_{A_i} = \{\langle L^0_{ij} : \{I_{ij} \} : \{\langle \text{dom}(L^0_{ij}) \rangle \} \ldots \} \} \}

- $I_A = \text{tup\_create}(A_1, \ldots, A_{j-1}, A_{j+1}, \ldots, A_K)(\pi_{A_1}(I_A), \ldots, I_{A_{j-1}}(I_A), \pi_{A_{j+1}}(I_A), \ldots, \pi_{A_K}(I_A))$, with $I_{A_i}$ being defined by:

\[
I_{A_i} = \text{nest}_{\text{u}_1^{\text{ij}, v, v'}}(\ldots (\text{nest}_{\text{u}_1^{\text{ij}, v, v'}}(I_{ij})(I_i)(I_j)) \ldots ), \text{where:}
\]

- $I_{ij} = \text{nest}_{\text{u}_1^{\text{ij}, v, v'}}(\ldots (\text{nest}_{\text{u}_1^{\text{ij}, v, v'}}(I_{ij})(I_i \times I_j)) \ldots ),$ and
- $I_i = \text{unnest}_{\text{u}_1^{\text{ij}, v, v'}}(\ldots (\text{unnest}_{\text{u}_1^{\text{ij}, v, v'}}(\pi_{A_i}(I_A)))$.

Filtering

Selection on measure: Let $I_R = \langle I_A, I_F, I_P \rangle$ be an instance of a representation $R$. Let $c$ be a constant in $\text{dom}(m)$. \(\sigma_{\text{measure}}(I_R)\) is an instance $I_{R'} = \langle I_A, I_F, I_P' \rangle$ of a representation $R'$ having sort $\text{sort}(R') = \text{sort}(R)$ and where $I_P' = \sigma_{\text{measure}}(I_P)$. 

Selection on members: Let $I_R = \langle I_A, I_F, I_P \rangle$ be an instance of a representation $R$ and $A_k$ be an axis of $R$ with $\text{sort}(A_k) = \{\langle L^0_k : \text{dom}(L^0_k), U^1_k : \{\langle L^1_k : \text{dom}(L^1_k), \ldots, U^b_k : \{\langle \text{dom}(L^b_k) \rangle \} \ldots \} \} \ldots \} \}$. 

4. This OLAP operation should not be mistaken with the nest operation of the algebra for complex values [ABI 95].
Titre abrégé de l'article (à définir par title[titre abrégé]{titre}) 21

\((\{L^S_i^j : dom(L^I_i^S)\};\ldots))\). Given an attribute \(L^k_{i_0}\) in \(A_k(R)\) and a constant \(v \in dom(L^k_{i_0})\), the selection \(\sigma^\text{members}_v(\text{IF}_R)\) is defined if there exists an attribute \(L_{i_k}^k \in \text{IF}(R)\) such that \(i_x = i_k\) and \(j_x \leq d_x\). In that case, \(\sigma^\text{members}_v(\text{IF}_R)\) is an instance \(I_{R'} = (I'_A, I'_F, I_F)\) of a representation \(R'\) having sort \(\text{sort}(R') = \text{sort}(R)\) where:

\[-I'_A = (I_{A_1}, \ldots, I_{A_R}, I_A),\]

\[-I'_{A_k} = \text{nest}_{V^k_{1}}(\text{sort}_{\text{IF}_R}(\ldots \text{nest}_{V^k_{1}}(\text{sort}_{\text{IF}_R}(\ldots (\text{nest}_{V^k_{1}}(I_A)) \ldots)),\ldots)),\]

\[-I'_F = \sigma_v(I_F)\]

\[-X = \{v' \in \text{dom}(L^k_{i_k}) \mid \text{anc}(v', L^k_{i_k}) = v\}.,\]

**Projection:** Let \(I_R\) be an instance of a representation \(R\) and \(A_i\) an axis of \(R\) with \(\text{sort}(A_i) = \{(L^0_{i_0} : dom(L^0_{i_0})), U^i_{1} : \{(L^1_{i_1} : dom(L^1_{i_1})), \ldots, U^i_{p} : \{(L^p_{i_p} : dom(L^p_{i_p}))\}}\}\). Given a subset \(\{(L^0_{i_0}, \ldots, L^p_{i_p})\}\) of \(A_i(R)\) such that for every \(i, j \in [1, p]\), if \(x_i = x_j\), then \(y_i \leq y_j\), the projection \(\pi^A_{L^0_{i_0}, \ldots, L^p_{i_p}}(I_R)\) is defined if we have:

\(\{(L^k_{i_k} \in A_k(R) \mid (\exists L^k_{i_k} \in \text{IF}(R))(i = i_k \land j_k \geq d_k))\} \subseteq \{(L^0_{i_0}, \ldots, L^p_{i_p})\}\). In that case, \(\pi^A_{L^0_{i_0}, \ldots, L^p_{i_p}}(I_R)\) is an instance \(I_{R'} = (I'_A, I'_F, I_F)\) of a representation \(R'\) having sort \(\text{sort}(R') = (A' : \text{sort}(A'), F : \text{sort}(F), P : \text{sort}(P))\) where:

\[-\text{sort}(A') = (A_1 : \text{sort}(A_1), \ldots, A_i : \tau_{A_i}, \ldots, A_R : \text{sort}(A_R)),\]

\[-\pi^A_{L^0_{i_0}, \ldots, L^p_{i_p}}(I_R) = \{(L^0_{i_0} : dom(L^0_{i_0})), U^1_{1} : \{(L^1_{i_1} : dom(L^1_{i_1})), \ldots, U^p_{p} : \{(L^p_{i_p} : dom(L^p_{i_p}))\}}\)}\].

**Changing the level of detail**

**Navigate:** Let \(I_C = (I_{A_{\text{Alt}}, I_{F_{\text{Alt}}}, I_F_{\text{Alt}}})\) be an instance of a \(N\)-dimensional cube \(C\). Let \(L^0_{i_0}, \ldots, L^N_{i_0}\) be such that for every \(i \in [1, N]\), \(L^i_{i_0} \in \text{LI}(R)\). Navigate \(L^0_{i_0}, \ldots, L^N_{i_0}(I_C)\) is an instance \(I_R = (I_A, I_F, I_F)\) of a representation \(R\) having sort \(\text{sort}(R) = (A : \text{sort}(A), F : \text{sort}(F), P : \text{sort}(P))\) where:

\[-\text{sort}(A) = (A_1 : \text{sort}(A_1), \ldots, A_N : \text{sort}(A_N)),\]

\[-\text{sort}(F) = \{(L^0_{i_0} : dom(L^0_{i_0})), \ldots, L^N_{i_0} : \text{dom}(L^N_{i_0}), m : \text{dom}(m))\}\],

\(-I_A = (I_{A_1}, \ldots, I_{A_N})\) where for all \(i \in [1, N]\), \(I_{A_i}\) is defined by:

\[-I_{A_i} = \text{nest}_{V^i_{1}}(\text{sort}_{\text{IF}_R}(\ldots \text{nest}_{V^i_{1}}(\text{sort}_{\text{IF}_R}(\ldots (\text{nest}_{V^i_{1}}(I_A)) \ldots)),\ldots)),\]

\[-I_F = \pi^A_{L^0_{i_0}, \ldots, L^N_{i_0}}(I_{F_{\text{Alt}}}) \times \pi^A_{L^0_{i_0}, \ldots, L^N_{i_0}}(\text{Level}_1 \ldots \times \pi^A_{L^0_{i_0}, \ldots, L^N_{i_0}}(\text{Level}_N)),\]

\[-\text{Level}_i = \pi^A_{L^0_{i_0}, \ldots, L^N_{i_0}}(\text{nest}_{\text{IF}_R}(\ldots (\text{nest}_{\text{IF}_R}(I_{D_i})))))\), for every \(i \in [1, N]\),]
– $I_P = I_{P_{\text{All}}}$.

**Aggregate:** The aggregate operator allows to apply an aggregate function on the facts of a representation. This operator relies on the extended projection operation as defined in [ABI 95].

Let $I_R = \langle I_A, I_F, I_P \rangle$ be an instance of a representation $R$ and $A_i$ an axis of $R$ with $\text{sort}(A_i) = \{\langle L_{i0}^j : \text{dom}(L_{i0}^j), U_{i1}^1 : \{\langle L_{i1}^j : \text{dom}(L_{i1}^j) \rangle \}, \ldots, U_{iS}^S : \{\langle L_{iS}^j : \text{dom}(L_{iS}^j) \rangle \} \rangle \}$. Let $f$ be an aggregate functions defined on $\text{dom}(m)$. Given two attributes $L_{ix}^{jx}$ and $L_{iy}^{jy}$ in $A_i(R)$ such that $L_{ix}^{jx} \in F(R)$, $k = i_x = i_y$ and $d_k = j_x > j_y$, the aggregation $\text{Aggregate}_{L_{ix}^{jx} \rightarrow L_{iy}^{jy}, f(m)}(I_R)$ is an instance $I_{R'} = \langle I_A, I_{F'}, I_P \rangle$ of a representation $R'$ having sort $\text{sort}(R') = \langle A : \text{sort}(A), F' : \text{sort}(F'), P : \text{sort}(P) \rangle$, where:

– $\text{sort}(F') = \{\langle L_{i1}^{d_{i1}}, \ldots, L_{i_{k-1}}^{d_{i_{k-1}}}, L_{i_y}^{j_y}, L_{i_{k+1}}^{d_{i_{k+1}}}, \ldots, L_{i_N}^{d_N} : \text{dom}(m) \rangle \}$,
– $I_{P'} = \pi_{L_{i1}^{d_{i1}}, \ldots, L_{i_{k-1}}^{d_{i_{k-1}}}, L_{i_y}^{j_y}, L_{i_{k+1}}^{d_{i_{k+1}}}, \ldots, L_{i_N}^{d_N} : f(m)}(I)$, with $I = I_F \Join_{L_{ix}^{jx}} \text{unnest}_{U_{i1}}(\ldots(\text{unnest}_{U_{iS}}(I_{A_i})) \ldots)$. 
