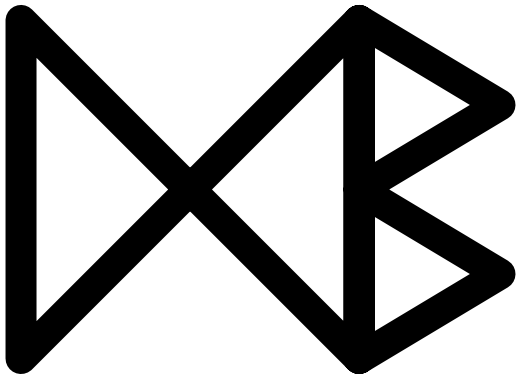


Algebra-Based Optimization of XML-Extended OLAP Queries

Xuepeng Yin and Torben Bach Pedersen

November 1, 2006

TR-17

A DB Technical Report

Title	Algebra-Based Optimization of XML-Extended OLAP Queries
Author(s)	Xuepeng Yin and Torben Bach Pedersen
Publication History	October 2006. A DB Technical Report

For additional information, see the DB TECH REPORTS homepage: ⟨www.cs.aau.dk/DBTR⟩.

The DB TECH REPORTS icon is made from two letters in an early version of the Rune alphabet, which was used by the Vikings, among others. Runes have angular shapes and lack horizontal lines because the primary storage medium was wood, although they may also be found on jewelry, tools, and weapons. Runes were perceived as having magic, hidden powers. The first letter in the logo is "Dagaz," the rune for day or daylight and the phonetic equivalent of "d." Its meanings include happiness, activity, and satisfaction. The second letter is "Berkano," which is associated with the birch tree. Its divinatory meanings include health, new beginnings, growth, plenty, and clearance. It is associated with Idun, goddess of Spring, and with fertility. It is the phonetic equivalent of "b."

Abstract

In today's OLAP systems, integrating fast changing data, e.g., stock quotes, physically into a cube is complex and time-consuming. The widespread use of XML makes it very possible that this data is available in XML format on the WWW; thus, making XML data logically federated with OLAP systems is desirable. This report presents a complete foundation for such OLAP-XML federations. This includes a prototypical query engine, a simplified query semantics based on previous work, and a complete physical algebra which enables precise modeling of the execution tasks of an OLAP-XML query. Effective *algebra-based* and *cost-based* query optimization and implementation are also proposed, as well as the execution techniques. Finally, experiments with the prototypical query engine w.r.t. federation performance, optimization effectiveness, and feasibility suggest that our approach, unlike the physical integration, is a practical solution for integrating fast changing data into OLAP systems.

1 Introduction

On-line Analytical Processing (OLAP) technology enables data warehouses to be used effectively for on-line analysis, providing rapid responses to iterative complex analytical queries. Usually an OLAP system contains a large amount of data, but *dynamic data*, e.g., stock prices, is not handled well in current OLAP systems. To an OLAP system, a well designed dimensional hierarchy and a large quantity of pre-aggregated data are the keys. However, trying to maintain these two factors when integrating fast changing data physically into a cube is complex and time-consuming, or even impossible. However, the advent of XML makes it very possible that this data is available in XML format on the WWW; thus, making XML data accessible to OLAP systems is greatly needed.

Our overall solution is to logically federate the OLAP and XML data sources. This approach *decorates* the OLAP cube with "virtual" dimensions using XML data, allowing *selections* and *aggregations* to be performed over the decorated cube. In this report, we describe a robust federation query engine in detail with query plan generation, optimization and evaluation techniques. First, a query semantics that simplifies earlier definitions [32] is proposed. Here, redundant and repeated logical operators are removed and a concise and compact logical query plan can be generated after a federation query is analyzed. Next, a *physical query algebra* that, unlike the previous *logical algebra*, is able to model the real execution tasks of a federation query, is described. Here, all concrete data retrieval and manipulation operations in the federation are integrated, meaning that we obtain a much more precise foundation for performing efficient query optimization and cost estimation. Third, the process that transforms logical plans to physical plans is introduced. During this process, the novel *logical-to-physical conversion rules* are used to turn the plans in the logical algebraic form into the executable forms expressed in the physical algebra. The *query evaluator* then retrieves and manipulates data from/in the federation components, according to tasks scheduled by the physical plans. Fourth, to improve the performance of the query engine, the full set of *rule-based* and *cost-based* query optimization techniques is introduced. This includes a query rewriter that generates equivalent logical plans, as opposed to physical plans in relational systems [36], a set of transformation rules specialized for OLAP-XML federations, a predicate rewriting technique, inlining, a cost model and cost functions specialized for the federation components, and a functioning query engine implemented with all above techniques. Finally, experimental results are introduced with respect to federation performance, optimization effectiveness and federation feasibility, suggesting that the logical OLAP-XML federation is comparable to a physical integration of OLAP and XML data in terms of performance, and therefore can be the practical solution to gaining flexible access to fast changing data in XML format from OLAP systems.

We believe that we are the first to propose a practical solution for logical OLAP-XML federations. Also, we believe to be the first to implement a robust query engine with the proposed optimization and evaluation techniques to enable OLAP-XML federation queries. More specifically, the novel contributions are as following. First, a simplified query semantics (compared to [32]) is proposed. Second, a physical query

algebra is defined. Third, novel algebra-based query optimization and evaluation techniques are introduced. Fourth, a robust federation query engine is implemented with all the above techniques and experiments are performed with it.

The rest of the report is organized as follows. Section 2 includes a general introduction of the case study, the OLAP-XML federation, the overall architecture, and the data model. Section 3 presents the OLAP-XML query semantics and formal definitions of the physical operators. Section 4 describes how logical plans are converted to physical plans and therefore how they are executed. Section 5 consists of the descriptions of the query optimizer and its components, as well as their implementations. Next, Section 6 describes in detail the performance study with different respects. Section 7 introduces previous work and the novelties of this report. Finally, Section 8 concludes the report and points to future work.

2 Background

2.1 Case Study

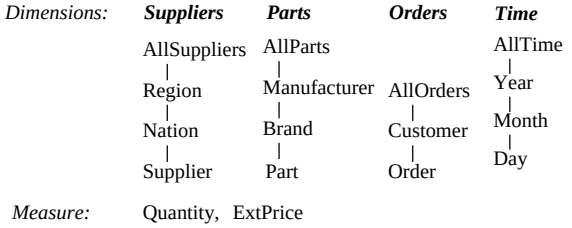


Figure 1: Cube Schema

```
<Nations>
        <Nation>
                <NationName>Denmark</NationName>
                <Population>5.3</Population>
        </Nation>
        <Nation>
                <NationName>China</NationName>
                <Population>1264.5</Population>
        </Nation>
        <Nation>
                <NationName>United Kingdom</NationName>
                <Population>19.1</Population>
        </Nation>
          ...
</Nations>
```

Figure 2: Part of the XML data

Quantity	ExtPrice	Supplier	Part	Order	Day
17	17954	S1	P3	11	2/12/1996
36	73638	S2	P5	18	5/2/1992
28	29983	S2	P4	42	30/3/1994
2	2388	S3	P3	4	8/12/1996
26	26374	S4	P2	20	10/11/1993

Figure 3: The fact table

The TPC-H-based [42] database used in the experiments and descriptions is shown in Figure 1. The OLAP database, called TC, is characterized by a Supplier dimension, a Parts dimension, an Order dimension, and a Time dimension. For each line item, Quantity and ExtendedPrice are measured. An example fact table is shown in Figure 3. The XML document is composed of the nation names and public population data about nations in millions. An example of the document is illustrated in Figure 2, where each Nation element contains two sub-elements, NationName and Population. We use the listed three lines as the example data in this report.

2.2 OLAP-XML Federations

A federation contains an OLAP *cube* and the XML documents. OLAP data is often organized in *multi-dimensional cubes*. A cube contains measured values (*measures*) that are characterized by *dimensions*. A dimension is structured using *levels* of different details. The fundamental part of an XML document is the element node, which can contain other element nodes. Another component of a federation is *links*. Links are created by users or DBAs between existing dimensions and XML data, allowing external data to characterize the OLAP measures as extra dimensions. The fundamental linking mechanism is a relation between one dimension value in a cube and one node in an XML document. Figure 4 shows an example link, Nation link (Nlink), that connects the dimension values of Nations to the Nation nodes that have the same text values in the sub-nodes, NationName, in the XML document. The plus/minus symbol in a box indicates whether the element is folded/unfolded. With Nlink, the population information about nations can be referenced in OLAP queries.

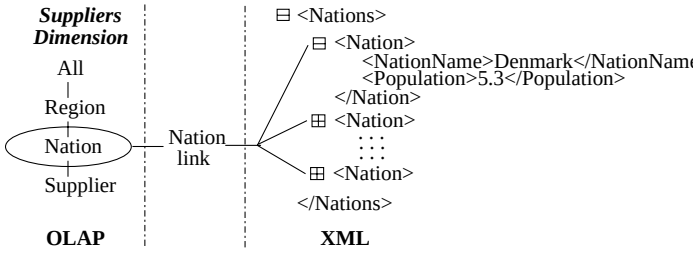


Figure 4: Linking OLAP and XML

The federation query language is called "XML-extended Multidimensional SQL" (SQL_{XM}), which has basic clauses similar to SQL, i.e., SELECT, FROM, WHERE, GROUP BY, and HAVING, and uses *level expressions* (defined below) for referencing external XML data. Based on the cube schema in Figure 1, the example SQL_{XM} query shown in Figure 5 shows the total quantity of the parts of each brand sold by each nation, where a nation is decorated with its population. Brand(Part) is a *roll-up expression*, which rolls up the cube to the Brand level from the Part level in the Parts dimension. Nation[ANY]/Nlink/Population is a level expression, where Population is a relative XPath expression applied to the XML nodes in Nlink to identify new nodes. A level expression allows decoration of dimension values (e.g., nation names) with XML values (e.g., populations) in the context defined through links.

SELECT	SUM(Quantity),Brand(Part),
	Nation[ANY]/Nlink/Population
FROM	TC
WHERE	Nation[ANY]/Nlink/Population<30
GROUP BY	Brand(Part),Nation[ANY]/Nlink/Population

Figure 5: An example SQL_{XM} query

2.3 The OLAP-XML System

In this section, we give an overview of the OLAP-XML federation system, the federation query language and the basic query evaluation process. The overall architecture of the prototype is shown in Figure 6. Besides the OLAP and the XML components, three auxiliary components have been introduced to hold meta data, link data, and temporary data. Queries are posed to the query engine, which coordinates the execution of queries in the data components. In the prototype, Microsoft SQL Server 2000 Enterprise

Edition with SP3 is used. More specifically, the temporary component is the temporary database on SQL Server, and the OLAP component uses MS Analysis Services, and is queried with SQL [24]. The XML component is the local file system based on the XML data retrieved from the Web with MS SQLXML [26] on top.

As shown in Figure 6, the query engine has three components: *query analyzer*, *query optimizer* and *query evaluator*. Given a query, the query engine parses and analyzes the query, and generates the initial logical plan. The plan is expressed in the logical algebra (see Section 3.1). The query optimizer generates a plan space for the initial plan, where all the logical plans produce the same output as the original one. Furthermore, the optimizer converts all the logical plans into physical plans by converting the composing logical operators into physical operators. Then, costs of the plans can be estimated. Finally, the optimizer searches for the best execution plan which has the least execution time and passes the plan on to the query evaluator. The evaluator executes the operators in the given plan and generates the final result. Generally, the component queries are evaluated in the OLAP and XML components in parallel and the data is transferred to the temporary component. Sometimes, the selection predicates on level expressions can be rewritten to new predicates with only references to dimension values and constants, therefore can be evaluated in the OLAP component. We term this technique *inlining* (see Section 3.3). Therefore, in such a situation, some XML queries have to be evaluated before the construction of OLAP queries so as to rewrite the predicates. Moreover, the underlying OLAP cube may be sliced and aggregated, which leads to less inter-component data transfer. There are also physical operators in the execution plan that model the processing of the temporary data in the temporary component. There, SQL operations are used to calculate the final result on the gathered data. Finally, the final result is produced in the temporary component.

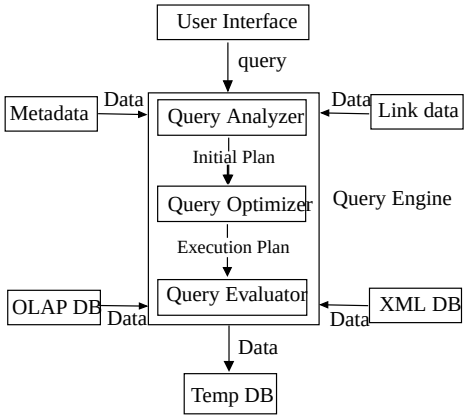


Figure 6: Architecture of the query engine

2.4 Data Models

The *cube model* is defined in terms of a multidimensional cube consisting of a *cube name*, dimensions and a *fact table*. Each dimension comprises two partially ordered sets (posets) representing hierarchies of levels and the ordering of *dimension values*. Each level is associated with a set of dimension values. That is, a *dimension* D_i is a two-tuple (L_{D_i}, E_{D_i}), where L_{D_i} is a poset of levels and E_{D_i} is a poset of dimension values. More specifically, L_{D_i} is the four-tuple $(LS_i, \sqsubseteq_i, \top_i, \bot_i)$, where $LS_i = \{l_{i1}, \ldots, l_{ik}\}$ is a set of levels, $\sqsubseteq_i$ is a partial order on these levels. $\bot_i$ is the bottom level, while $\top_i$ is the unique "ALL" level. We shall use $l_{ij} \in D_i$ as a shorthand meaning that the level l_{ij} belongs to the poset of levels in dimension D_i. Each pair of levels has a containment relationship, that is, the *partial order* of two levels, $l_1 \sqsubseteq_i l_{i2}$, which holds if elements in L_{i2} can be said to contain the elements in L_{i1}. Furthermore, $l_{i1} \sqsubset_i l_{i2}$ holds

if $l_{i1} \sqsubseteq_i l_{i2} \wedge l_{i1} \neq l_{i2}$. Here, L_{ik} is the dimension values of level l_{ik}, that is, $L_{ik} = \{e_{ik_1}, \ldots, e_{ik_{l_{ik}}}\}$. Similarly, we say that $e_1 \sqsubseteq_{D_i} e_2$ if e_1 is logically contained in e_2 and $l_{ij} \sqsubseteq_i l_{ik}$ for $e_1 \in L_{ij}$ and $e_2 \in L_{ik}$. E_{D_i} is a poset $(\bigcup_j L_{ij}, \sqsubseteq_{D_i})$, consisting of the set of all dimension values in the dimension and a partial ordering defined on these. For each level l we assume a function Roll-up$_l : L \times LS_i \mapsto \mathcal{P}(D_i)$, which given a dimension value in L and a level in LS_i returns the value's ancestor in the level. That is, Roll-up$_l(e_{ik_h}, l_{ij}) = \{e' | e_{ik_h} \sqsubseteq_{D_i} e' \wedge e' \in L_{ij}\}$. In the federation query, a roll-up expression $l_{ij}(l_{ik})$ uses the Roll-up function to aggregate the cube from a lower level l_{ik} to a higher level l_{ij}, i.e. $l_{ik} \sqsubset_i l_{ij}$.

A *fact table* F is a relation containing one attribute for each dimension and one attribute for each measure; thus, $F = \{(e_{\perp_1}, \ldots, e_{\perp_n}, v_1, \ldots, v_m) | (e_{\perp_1}, \ldots, e_{\perp_n}) \in \perp_1 \times \perp_n \wedge (v_1, \ldots, v_m) \in M \subseteq T_1 \times \ldots \times T_m\}$, where $n \geq 1$, $m \geq 1$ and T_j is the domain value for the j'th measure. We will also refer to the j'th measure as $M_j = \{(e_{\perp_1}, \ldots, e_{\perp_n}, v_j)\}$. Each measure M_j is associated with a *default aggregate function* $f_j : \mathcal{P}(T_j) \mapsto T_j$, where the input is a multi-set. Aggregate functions ignore NULL values as in SQL. There may be NULL values for measures in the logical definition, but in a physical implementation only the non-empty tuples would be stored in the fact table. An *n-dimensional cube*, C, is given as: $C = (N, D, F)$, where N is the cube name, $D = \{D_1, \ldots, D_n\}$ is a set of dimensions, and F is the fact table. A *federation* is the data structure on which we perform logical federation operations, e.g. selections, aggregations and decorations. A federation $\mathcal{F}$ is a three-tuple: $\mathcal{F} = (C, Links, X)$, where C is an OLAP cube, X are the referenced XML documents, and $Links$ is a set of *links* (defined below) between levels in C and documents in X.

A link is a relation that connects dimension values with nodes in XML documents. For example, a link $Nlink = \{(Denmark, n1), (China, n2), (United\,Kingdom, n3)\}$ maps each dimension value to a node in the example XML document, here, $n1$ is the Nation node with the sub-node NationName having the string value "Denmark," $n2$ is the Nation node with the sub-node NationName having the string value "China," and similarly for $n3$.

An XPath expression [5] is a path that selects a set of nodes in an XML document. To allow references to XML data in SQL$_{XM}$ queries, links are used with XPath expressions to define level expressions. A level expression $l[SEM]/link/xp$ consists of a *starting level* l, i.e., the dimension level to be decorated, a decoration semantic modifier SEM, a link $link$ from l to nodes in one or more XML documents, and a relative XPath expression xp which is applied to these nodes to identify new nodes. For example, Nation[ANY]/Nlink/Population connects the dimension value "Denmark" with its population data "5.3" (million) which is the string value of the node Population in the context of $n1$. SEM represents the *decoration semantics*, ALL, ANY, and CONCAT which specify how many decoration values should be used when several of them are found for a dimension value through $link$ and xp. The ALL semantics connect each dimension value with all the linked decoration values, and the ANY semantics just use an arbitrary decoration value for each dimension value, whereas the CONCAT semantics concatenate all the possible decoration values into one.

A hierarchy is *strict* if no dimension value has more than one parent value from the same level [20]. Non-strict hierarchy can lead to incorrect aggregation over a dimension, e.g., some lower-level values will be double-counted. Three types of data are distinguished: c, data that may not be aggregated because fact data is duplicated and may cause incorrect aggregation, α, data that may be averaged but not added, and Σ, data that may also be added. A function AggType:$\{M_1, \ldots, M_m\} \times D \mapsto \{\Sigma, \alpha, c\}$ returns the aggregation type of a measure M_j when aggregated in a dimension $D_i \in D$. Considering only the standard SQL functions, we have that $\Sigma = \{$SUM, AVG, MAX, MIN, COUNT$\}$, $\alpha = \{AVG, MAX, MIN, COUNT\}$, and $c = \emptyset$.

3 Query Semantics

3.1 Logical Algebra and Query Semantics

In previous work [32], a logical algebra over federations was proposed which is the basis of our work. In this section, a brief background introduction to the logical algebra, and the original SQL$_{XM}$ query semantics are given. We then propose a simplified version of the original query semantics.

Decoration A decoration operator, $\delta_{l_z[SEM]/link/xp}$, builds a decoration dimension using the XML data referenced by a level expression. The decoration dimension consists of the unique top level, the mid-level which is also called the *decoration level* and composed of external XML data, and the bottom level to which the starting level of the level expression (e.g., Nation for Nation[ANY]/Nlink/Population) belongs is defined as the bottom level of the decoration dimension. For example, Population is the decoration level of the virtual dimension yielded by Nation[ANY]/Nlink/Population and Supplier is the bottom level. The mid-level decorates the measures of the OLAP database through the bottom level, whose values and their ancestors in the mid-level are related using the relationship defined by the level expression. The decoration operator enables the external XML data to become virtually a part of the cube, thereby allowing the following operations involving XML data to be performed on the federation.

Federation Selection A federation selection operator, $\sigma_{Fed[\theta]}$, allows the facts from the cube to be filtered using the external XML data as well as the regular cube data. The predicate can reference any dimension levels in the federation, including decoration levels. After selection, the cube schema is not changed. Only facts in the fact table are affected. Note that when referenced by a federation selection or generalized projection (see below) operator, a decoration level appears in the form of the level expression that yields the virtual dimension containing the decoration level.

Federation Generalized Projection The generalized federation projection operator, $\Pi_{Fed[\mathcal{L}]<F(M)>}$, also lets the federated cube be aggregated over the external XML data. Here, $\mathcal{L}$ is a set of levels to which the federation will be rolled up to, intuitively, the levels in the GROUP BY clause where level expressions can be present. *F(M)* is a set of aggregate functions over the specified measures, i.e., the aggregates in the SELECT clause. Given a set of argument levels, the generalized projection first removes the dimensions in which no argument levels are present (like the SQL projection), and then each dimension is rolled up to the specified level, replacing the original dimension values in the facts by their ancestor values in the levels in $\mathcal{L}$. Finally, facts in the fact table are grouped, the specified measures are aggregated, and other measures not specified in the arguments are removed.

Semantics of the SQL$_{XM}$ Query Language The semantics of an SQL$_{XM}$ query can be expressed in terms of the algebra defined above. In the following, suppose: $\mathcal{F} = (C, Links, X)$ is a federation. $\{\perp_p, \ldots, \perp_q\} \subseteq \{\perp_1, \ldots, \perp_n\}$ and $\{l_s, \ldots, l_t\}$ are levels in C such that $\perp_s \sqsubset_s l_s, \ldots, \perp_t \sqsubset_t l_t$. le is used to represent a level expression, $l[SEM]/link/xp$, where the decoration semantic modifier is $SEM \in \{ANY, ALL, CONCAT\}$, l is a level in C, $link \in Links$ is a link from l to documents in X, and xp is an XPath expression. $pred_{where}$ represents the predicates in the WHERE clause. $pred_{having}$ represents the predicates in the HAVING clause. $LE_{\Pi} = \{le_{u_{\Pi}}, \ldots, le_{v_{\Pi}}\}$ are the level expressions in the SELECT and GROUP BY clauses. $LE_{\sigma_{where}} = \{le_{u_{\sigma_{where}}}, \ldots, le_{v_{\sigma_{where}}}\}$ are the level expressions in the WHERE clause. $LE_{\sigma_{having}} = \{le_{u_{\sigma_{having}}}, \ldots, le_{v_{\sigma_{having}}}\}$ are the level expressions in the HAVING clause. $f_x, \ldots, f_y$ are the aggregation functions. A sequence of decoration operations is denoted by Δ, that is: $\Delta_{\{le_i,\ldots,le_j\}}(\mathcal{F}) = \delta_{le_i}(\ldots(\delta_{le_j}(\mathcal{F})))$. Here is a prototypical SQL$_{XM}$ query: SELECT $f_x(M_x), \ldots, f_y(M_y), \perp_p, \ldots, \perp_q, l_s(\perp_s), \ldots, l_t(\perp_t), le_{u_\Pi}, \ldots, le_{v_\Pi}$ FROM $\mathcal{F}$ WHERE $pred_{where}$ GROUP BY $\perp_p, \ldots, \perp_q, l_s(\perp_s), \ldots,$

$l_t(\perp_t), le_{u_\Pi}, \ldots, le_{v_\Pi}$ HAVING $pred_{having}$, which can be represented in the logical algebra proposed by [32] as shown below.

$$\Pi_{Fed[\perp_p,\ldots,\perp_q,l_s(\perp_s),\ldots,l_t(\perp_t),le_{u_\Pi},\ldots,le_{v_\Pi}]<f_x(M_x),\ldots,f_y(M_y)>}($$
$$\sigma_{Fed[pred_{having}]}($$
$$\Delta_{LE_{\sigma_{having}}}($$
$$\Pi_{Fed[\perp_p,\ldots,\perp_q,l_s(\perp_s),\ldots,l_t(\perp_t),le_{u_\Pi},\ldots,le_{v_\Pi}]<f_x(M_x),\ldots,f_y(M_y)>}($$
$$\Delta_{LE_\Pi}($$
$$\sigma_{Fed[pred_{where}]}($$
$$\Delta_{LE_{\sigma_{where}}}(\mathcal{F})))))))$$

The semantics above implies an SQL$_{XM}$ query can be evaluated in four major steps. First, the cube is sliced as specified in the WHERE clause, possibly requiring decorations with XML data. Second, the cube is decorated for the level expressions in the SELECT and GROUP BY clauses which creates a number of new dimensions in the cube. Then all dimensions, including the new ones, are rolled up to the levels specified in the GROUP BY clause. Third, the resulting cube is sliced according to the predicate in the HAVING clause, which may require additional decorations. Fourth, the top generalized projection projects the decorations not required by the SELECT and GROUP BY clause and gives out the final result cube.

3.2 Simplified Query Semantics

The query semantics have a great impact on generating the initial plan, as the semantics take the form of a logical query tree in the query engine when an SQL$_{XM}$ query is parsed and analyzed. As the semantics indicate, duplicate decoration operators are generated when a level expression exists in several sub-clauses, e.g., the SELECT clause and WHERE clause. As the algebra shows, an operator can take an argument federation and generate a new one; thus, repeated operators then can be detected by examining the input and output federations.

The simplified query semantics can be constructed by removing the *redundant operators* that do not change the cube semantics. An operator that generates the same federation as the argument federation is redundant; thus, the plan without redundant operators is more compact, and sometimes considerably smaller than the unsimplified version. This simplification benefits the performance of the query processing. First, during the query optimization, the equivalent plans in the plan space can be enumerated much faster. Intuitively, this process can be looked as the combinations of operators. The less operators a plan has, the less combinations it results in. Second, smaller plans lead to less logical-to-physical conversion and cost-estimation time. Third, in the execution phase, no duplicate data is retrieved, thereby leading to high reusability, and more importantly, less resource consumptions, e.g., CPU, I/O, storage, etc. The simplified algebraic query representation is shown below.

$$\sigma_{Fed[pred_{having}]}($$
$$\Pi_{Fed[\perp_p,\ldots,\perp_q,l_s(\perp_s),\ldots,l_t(\perp_t),le_{u_\Pi},\ldots,le_{v_\Pi}]<f_x(M_x),\ldots,f_y(M_y)>}($$
$$\Delta_{LE_{\Pi,\delta}}($$
$$\sigma_{Fed[pred_{where}]}($$
$$\Delta_{LE_{\sigma_{where}}}(\mathcal{F})))))$$

Here, $LE_{\Pi,\delta}$ is a set of the decoration operators that are referenced by the SELECT and GROUP BY clauses only, that is: $LE_{\Pi,\delta} \subseteq LE_\Pi \wedge LE_{\Pi,\delta} \cap LE_{\sigma_{where}} = \emptyset$. Moreover, an instance of a decoration operator for a specific level expression is unique. In other words, when a virtual dimension for a level expression already exists in the federation, no decoration operator building the same dimension is needed

again; therefore, some of the decoration operators for the WHERE clause may build the virtual dimensions required by the SELECT and GROUP BY clauses as well, that is: $LE_{\Pi} \backslash LE_{\Pi,\delta} \subseteq LE_{\sigma_{where}}$. $\Delta_{pred_{having}}$ is removed because predicates on level expressions in the HAVING clause can be put in the WHERE clause. The original top generalized projection is also removed, because the HAVING clause does not change the cube schema. An example query and the corresponding simplified logical plan tree is shown in Figure 7, where only one decoration, $\delta_{N[ANY]/Nl/P}$, exists below the federation selection, although referenced by two federation operators.

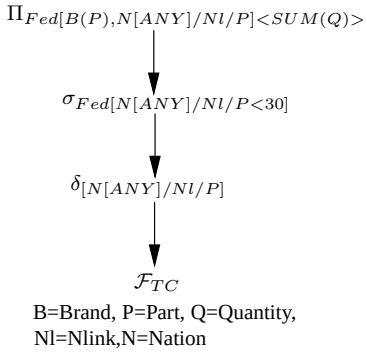


Figure 7: The initial logical plan

3.3 Introduction to the Physical Algebra

As shown in Figure 6, an execution plan is produced by the query optimizer which is used to guide the evaluator about when, where, and how the data retrieval and manipulation operations should be performed. An execution plan is an SQL$_{XM}$ query tree expressed in the physical algebra. The logical semantics of a query implies the main phases of the query evaluation, whereas a physical query tree is integrated and extended with more detailed evaluation operations. In the following sections, we introduce the new physical algebra operators and the new semantics of the existing federation operators, and show an example of a logical plan and its corresponding physical plan.

In the process of OLAP and XML federation, OLAP data is always put into a temporary component to be decorated by the decoration data loaded from XML documents. Selections and aggregations then can be performed in the temporary component over the decorated cube; therefore, the temporary component plays an important role in the practical query evaluation. Before we describe the physical operators, we extend the original federation to an extended form, on which our physical algebra is based. An *extended federation* is $\mathcal{F}_{ext} = (C, Links, X, T)$, where C is a cube, $Links$ is a set of links between levels in C and documents in X, and T is a set of temporary tables.

3.4 Querying the OLAP component

Cube operators include cube selection and cube generalized projection. They are used to model the OLAP component query which is used to retrieve the cube data from the OLAP database.

Cube Selection The cube selection operator σ_{Cube} is much like a logical federation selection operator, but has no references to level expressions in the predicates. A cube selection only affects the tuples in the fact table, thereby returning a cube with the same fact type and the same set of dimensions.

Example 3.1 Suppose the extended federation $\mathcal{F}_{TC,ext}$ has the cube schema and the fact table in Section 2.1. The cube selection operator $\sigma_{Cube[Supplier='S1' \, OR \, Supplier='S2']}(\mathcal{F}_{TC,ext}) = \mathcal{F}'_{TC,ext}$ slices the

TC cube so that only the data for the suppliers S1 and S2 are retained. Nothing but the fact table is affected in the resulting extended federation. The resulting fact table is shown in Table 1.

Quantity	ExtPrice	Supplier	Part	Order	Day
17	17954	S1	P3	11	2/12/1996
36	73638	S2	P5	18	5/2/1992
28	29983	S2	P4	42	30/3/1994

Table 1: The fact table after selection

Definition 3.1 (Cube Selection) Let $\mathcal{F}_{ext} = (C, Links, X, T)$ be an extended federation, and θ be a predicate over the set of levels $\{l_1, \ldots, l_k\}$ and measures $M_1, \ldots, M_m$. A cube selection is defined as: $\sigma_{Cube[\theta]}(\mathcal{F}_{ext}) = (C', Links, X, T)$, where $C' = (N, D, F')$, and $F' = \{t'_1, \ldots, t'_l\}$. If $t_i = (e_{\perp_1}, \ldots, e_{\perp_n}, v_1, \ldots, v_m) \in F$ then $t'_i = \begin{cases} t_i & \text{if } \theta(t_i) = tt \\ (e_{\perp_1}, \ldots, e_{\perp_n}, \text{NULL}, \ldots, \text{NULL}) & \text{otherwise.} \end{cases}$

Cube Generalized Projection The cube generalized projection operator Π_{Cube} rolls up the cube, aggregates measures over the specified levels and at the same time removes unspecified dimensions and measures from a cube. Intuitively, it can be looked as a SELECT statement with a GROUP BY clause in SQL. The difference between a cube and a federation generalized projection operator is that the first one does not involve external XML data or level expressions and is executed in the OLAP component. Intuitively, the levels specified as parameters to the operator becomes the new bottom levels of their dimensions and all other dimensions are rolled up to the top level and removed. Each new measure value is calculated by applying the given aggregate function to the corresponding value for all tuples in the fact table containing old bottom values that roll up to the new bottom values. To ensure safe aggregation in case of non-strict hierarchies, we explicitly check for this in each dimension. If a roll-up along some dimension duplicates facts we disallow further aggregation along that dimension by setting the aggregation type to c.

Example 3.2 Suppose the extended federation $\mathcal{F}_{TC,ext}$ has the cube schema and the fact table in Section 2.1. The operator $\Pi_{Cube[Supplier]<SUM(Quantity)>}(\mathcal{F}_{TC,ext}) = \mathcal{F}'_{TC,ext}$ rolls up the cube to the level Supplier and calculates the Quantity per Supplier. After the projection, only the measure Quantity and the dimension Suppliers are retained, of which the bottom level is Supplier. The resulting fact table is shown in Table 2.

Quantity	Supplier
17	S1
64	S2
2	S3
26	S4

Table 2: The resulting fact table after the cube generalized projection

Definition 3.2 (Cube Generalized Projection) Let $\mathcal{F}_{ext} = (C, Links, X, T)$ be an extended federation. $l_{i_1}, \ldots, l_{i_k}$ be levels in C such that at most one level from each dimension occurs. The measure $\{M_{j_1}, \ldots, M_{j_l}\} \subseteq \{M_1, \ldots, M_m\}$ are kept in the cube and $f_{j_1}, \ldots, f_{j_l}$ are the given aggregate functions for the specified measures, such that $\forall D'_g \in \{D_g | D_g \in D \wedge \perp_g \notin \{l_{i_1}, \ldots, l_{i_k}\}\} \forall f_{j_h} \in \{f_{j_1}, \ldots, f_{j_l}\}(f_{j_h} \in \text{AggType}(M_{j_h}, D'_g))$, meaning that the specified aggregate functions are allowed to be applied. The generalized cube projection operator Π_{Cube} over a cube C is then defined as: $\Pi_{Cube[l_{i_1},\ldots,l_{i_k}]<f_{j_1}(M_{j_1}),\ldots,f_{j_l}(M_{j_l})>}(\mathcal{F}_{ext}) = (C', Links, X, T)$, where $C' = (N, D', F')$, and $D'_{i_h} = (L'_{D_{i_h}}, E'_{D_{i_h}})$ for $h \in \{1, \ldots, k\}$. The new poset of levels in the remaining dimensions is $L'_{D_{i_h}} = (LS'_{i_h}, \sqsubseteq'_{i_h}, \top_{i_h}, l_{i_h})$,

where $LS'_{i_h} = \{l_{i_hP} | l_{i_hP} \in LS_{i_h} \wedge l_{i_h} \sqsubseteq_{i_h} l_{i_hP}\}$, and $\sqsubseteq'_{i_h} = \sqsubseteq_{i_h|LS'_{i_h}}$. Moreover, $E'_{D_{i_h}} = (\bigcup_{l_{i_h} \in LS'_{i_h}} L_{i_h}, \sqsubseteq_{D_{i_h}|\bigcup_{l_{i_h} \in LS'_{i_h}} L_{i_h}})$, where L_{i_h} is the set of dimension values of the level l_{i_h}. The new fact table is given by : $F' = \{(e'_{\perp_{i_1}}, \ldots, e'_{\perp_{i_k}}, v'_{j_1}, \ldots, v'_{j_l}) | e'_{\perp_{i_g}} \in L_{i_g} \wedge v'_{j_h} = f_{M_{j_h}}(\{v | (e_{\perp_1}, \ldots, e_{\perp_n}, v) \in M_{j_h} \wedge (e'_{\perp_{i_1}}, \ldots, e'_{\perp_{i_k}}) \in \text{Roll-up}_{\perp_{i_1}}(e_{\perp_{i_1}}, l_{i_1}) \times \ldots \times \text{Roll-up}_{\perp_{i_k}}(e_{\perp_{i_k}}, l_{i_k})\})\}$. Furthermore, if $\exists (e_{\perp_1}, \ldots, e_{\perp_n}, v_j) \in M_{j_h} \exists e \in \{e_{\perp_1}, \ldots, e_{\perp_n}\}(||\text{Roll-up}_{\perp_{i_g}}(e, l_{i_g})|| > 1 \wedge v_j \neq \text{NULL})$ then AggType(M_{j_h}, D'_{i_g})=*c*.

3.5 Data Transfer Between Components

This section presents the definitions of fact-, dimension-, and XML-transfer operators. These operators are used to transfer data between components. The fact-transfer operator transfers fact data from the OLAP to the temporary component, whereas a dimension-transfer operator only transfers dimension data. An XML-transfer operator connects the temporary and XML components, transferring the referenced XML data into a temporary table.

Fact-Transfer In a physical execution plan, the fact-transfer operator is above the cube operators. The resulting fact data from the cube operators is transferred to the temporary component through the fact-transfer operator. Then, SQL operations, e.g., selections and joins, can be performed over the temporary fact table; therefore, the fact-transfer operator separates the cube operators from the other operators, e.g. federation selection and generalized projection.

Definition 3.3 (Fact-Transfer) Let $\mathcal{F}_{ext} = (C, Links, X, T)$ be an extended federation. The fact-transfer operator is $\phi(\mathcal{F}_{ext}) = (C, Links, X, T')$, where $T' = T \cup \{R_F\}$, R_F is the copy of the fact table in the temporary component.

Dimension-Transfer When a non-bottom level is referred by the federation operations in the temporary component, dimension values of the non-bottom level are required. The dimension transfer operator ω is used at this time to load the dimension values for the given dimension levels into a table in the temporary component, which then can be used by federation selection and generalized projection operators.

Example 3.3 A roll-up expression, Nation(Supplier), yields a dimension transfer. The two input parameters are Nation and Supplier. The dimension values for the two levels are loaded into a temporary table R_1 shown in Table 3.

Nation	Supplier
Denmark	S1
Denmark	S2
China	S3
United Kingdom	S4

Table 3: The temporary table R_1 for Nation and Supplier

Definition 3.4 (Dimension-Transfer) Let $\mathcal{F}_{ext} = (C, Links, X, T)$ be an extended federation, where the cube is $C = (N, D, F)$. Let l_{ix}, l_{iy} be two levels in dimension D_i, and $l_{ix} \sqsubseteq_i l_{iy}$. The dimension-transfer operator is defined as: $\omega_{[l_{ix}, l_{iy}]}(\mathcal{F}_{ext}) = (C, Links, X, T')$, where $T' = T \cup \{R\}$, $R = \{(e_{ix}, e_{iy}) | e_{ix} \in L_{ix} \wedge e_{iy} \in L_{iy} \wedge e_{ix} \sqsubseteq_{D_i} e_{iy}\}$,

In the following, a temporary table for l_{ix} and l_{iy} by a dimension-transfer operator is denoted as: $R_{\omega_{[l_{ix}, l_{iy}]}}$. In Example 3.3, the temporary table R_1 can be denoted as $R_{\omega_{[Supplier, Nation]}}$. According to the definition, the temporary component T' has a new element, $R_{\omega_{[Supplier, Nation]}}$.

XML-Transfer At query execution time, XML data is required in the temporary component to allow grouping, decoration, or selection on the cube according to the referenced level expressions. Intuitively, the XML-transfer operator connects the temporary component and the XML component, transferring the XML data into the temporary component. The input parameter is a level expression, which specifies the dimension values to be decorated and the corresponding decoration XML values selected by the relative XPath expression and the link in the level expression. The operator yields a new table in the temporary component.

Definition 3.5 (XML-Transfer) Let $\mathcal{F}_{ext} = (C, Links, X, T)$ be an extended federation, where $C = (N, D, F)$. Let $l_z[SEM]/link/xp$ be a level expression, where $l_z \in D_z$, $link \in Links$ is a link from l_z to X and xp is an XPath expression over X. The XML-transfer operator is defined as: $\tau_{l_z[SEM]/link/xp}(\mathcal{F}_{ext}) = (C, Links, X, T')$, where $T' = T \cup \{R\}$, here R is the temporary table containing the dimension values and the decoration XML values found through the XML documents with the decoration semantics, ALL, ANY or CONCAT, specified by the semantic modifier SEM. At query execution time, the ALL semantics yield the temporary table having multiple rows with the same dimension value but different decoration values, whereas the table for the ANY semantics has only one row for a dimension value and an arbitrary decoration value linked through the level expression. Similarly, a dimension value decorated with the CONCAT semantics also takes up one row, but the decoration column is the concatenation of all the decoration values. In the following descriptions, R is denoted as $R_{\tau_{l_z[SEM]/link/xp}}$ and formally $R_{\tau_{l_z[SEM]/link/xp}} =$

- $\{(e_z, e_{xp}) | \forall (e_z, s) \in link(\forall s' \in xp(s)(e_{xp} = StrVal(s')))\}$, if $SEM = ALL$.
- $\{(e_z, e_{xp}) | \exists (e_z, s) \in link(e_{xp} = StrVal(s') for\ some\ s' \in xp(s))\}$, if $SEM = ANY$.
- $\{(e_z, e_{xp}) | (e_z, s) \in link \wedge e_{xp} = Concat(StrVal(s_1), \ldots, StrVal(s_k)) \wedge s_i \in S_{e_z}\}$, where $S_{e_z} = \{s | \forall (e, s') \in link(s \in xp(s'))\}$, for each $e_z \in L_z$, if $SEM = CONCAT$.

Example 3.4 The operator $\tau_{Nation[ANY]/Nlink/Population}(\mathcal{F}_{TC,ext})$ generates a new extended federation $\mathcal{F}'_{TC,ext} = (C, Links, X, T')$, where T' contains a new temporary table $R_{\tau_{Nation[ANY]/Nlink/Population}}$. The table has two columns, one for the dimension values of Nation and the other for the decoration values Population. A decoration value is the string value of a Population node in the context of the nodes in $Nlink$. Each nation has one population as specified by the decoration semantics, ANY. The resulting temporary table $R_{\tau_{Nation[ANY]/Nlink/Population}}$ using the XML data from Figure 2 is shown in Table 4.

Nation	Population
Denmark	5.3
China	1264.5
United Kingdom	19.1

Table 4: The temporary table for Nation and Population

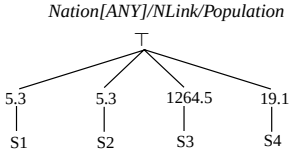
3.6 Querying the Temporary Component

This section presents the definitions of the operators that are performed in the temporary component. They are the decoration, federation selection and generalized projection operators, which allow the OLAP data to be decorated, selected and grouped by the external XML data.

Decoration The cube is decorated in the temporary component using the decoration operator δ, which has a level expression as the parameter. The operator generates a decoration dimension. The new dimension has the unique top level, the middle decoration level, and the bottom level of the dimension containing the starting level of the parameter level expression; therefore, the new dimension has the same aggregation type as the referred dimension with each measure. Values of the levels are derived from a temporary table, which

is composed of the decoration values and the bottom values of the referred dimension. The decoration dimension is derived according to the cube semantics, that the fact table contains the bottom levels of all dimensions. Moreover, since the cube definition does not allow duplicate dimensions, no changes are made if an identical dimension already exists in the cube. In the evaluation of the decoration operator, the temporary table created by the XML-transfer operator having the same parameter level expression is used. The new dimension follows the same decoration semantics specified by the level expression. Correct aggregations on such a decoration dimension is ensured by the federation generalized projection operator in Definition 3.8. A physical decoration operator may have more than one child operator. For example, one child operator could be an XML-transfer operator with the same level expression as the input parameter, thereby providing the XML data in a temporary table. The formal definition is below.

Example 3.5 The decoration operator for Nation[ANY]/Nlink/Population generates a decoration dimension containing the top level $\top$, the middle level Population, and the bottom level Supplier which is the bottom level of the dimension having the starting level Nation. The dimension values are derived from the result of a SQL inner join on the temporary tables of Examples 3.3 and 3.4. The dimension hierarchy is strict since a supplier in a nation only has one corresponding population number. Figure 8 shows the dimension and the temporary table.



Supplier	Population
S1	5.3
S2	5.3
S3	1264.5
S4	19.1

Figure 8: The decoration dimension and the temporary table Nation/Population

Definition 3.6 (Decoration) Let $Op_1, \ldots, Op_n$ be the child operators of a decoration operator $\delta_{l_z[SEM]/link/xp}$. Let $(C, Links, X, T_1), \ldots, (C, Links, X, T_n)$ be their output federations, where $C = (N, D, F)$. Let $l_z[SEM]/link/xp$ be a level expression, where $l_z \in D_z$, $link \in Links$ is a link from l_z to X and xp is an XPath expression over X. The physical decoration operator is defined as: $\delta_{l_z[SEM]/link/xp}($ $\mathcal{F}_{ext}) = (C', Links, X, T')$ where $\mathcal{F}_{ext} = (C, Links, X, T)$ is the input, $T = T_1 \cup \ldots \cup T_n$ is the union of the temporary tables from the child operators. In the output federation, $T' = T \cup \{R_{D_{n+1}}\}$, $R_{D_{n+1}}$ is a temporary table holding the dimension values of the bottom level $\perp_z$, and the XML level l_{xp}, n is the number of the existing dimensions before the decoration. More precisely, suppose $R_{\tau_{l_z[SEM]/link/xp}} \in T$ is a temporary table loaded by an XML-transfer operator, $R_{\omega_{[\perp_z, l_z]}}$ is a temporary table loaded by a dimension-transfer operator, then $R_{D_{n+1}} = \pi_{\perp_z, l_{xp}}(R_{\tau_{l_z[SEM]/link/xp}})$ *if* $l_z = \perp_z$, otherwise, $R_{D_{n+1}} = \pi_{\perp_z, l_{xp}}(R_{\tau_{l_z[SEM]/link/xp}} \bowtie R_{\omega_{[\perp_z, l_z]}})$*if* $\perp_z \sqsubseteq_z l_z$, where π is the regular SQL projection, and $\bowtie$ is the natural join. The resulting cube is given by: $C' = (N, D', F)$, where $D' = \{D_1, \ldots, D_n\} \cup \{D_{n+1}\}$ and $D_{n+1} = \{L_{D_{n+1}}, E_{D_{n+1}}\}$. Here, $L_{D_{n+1}} = (LS_{n+1}, \sqsubseteq_{n+1}, \top_{n+1}, \perp_{n+1})$, where $LS_{n+1} = \{\top_{n+1}, l_{xp}, \perp_{n+1}\}$, $\sqsubseteq_{n+1} = \{(\perp_{n+1}, l_{xp}), (l_{xp},$ $\top_{n+1}), (\perp_{n+1}, \top_{n+1})\}$, and $\perp_{n+1} = \perp_z$. The poset of dimension values is $E_{D_{n+1}} = (\bigcup_{(e_i, e_j) \in R_{D_{n+1}}} \{e_i, e_j\} \cup$ $\{\top_{n+1}\}, \sqsubseteq_{D_{n+1}})$, where $\sqsubseteq_{D_{n+1}} = \{(e_{\perp_{n+1}}, e_{xp}) | (e_1, e_2) \in R_{D_{n+1}} \wedge e_{\perp_{n+1}} = e_1 \wedge e_{xp} = e_2\} \cup \{(e_{\perp_{n+1}},$ $\top_{n+1}) | (e_1, e_2) \in R_{D_{n+1}} \wedge e_{\perp_{n+1}} = e_1\} \cup \{(e_{xp}, \top_{n+1}) | (e_1, e_2) \in R_{D_{n+1}} \wedge e_{xp} = e_2\}$. For each measure M_h in M the aggregation type of D_{n+1} is: AggType(M_h,D_z).

Federation Selection Intuitively, the physical federation selection operator σ_{Fed} is a SQL selection over the join of several tables, including the fact table, decoration dimension tables and temporary dimension tables for non-bottom levels referenced by the predicates. Similarly to the cube selection, the federation selection returns a cube with the same fact types and the same set of dimensions, and only affects the tuples

of the fact table, however, in the temporary component. A federation selection operator may have several child operators, e.g., dimension-transfer and decoration operators, to provide the values required by the predicates. The temporary tables produced by the child operators are collected and will be used in the join.

Example 3.6 Suppose the temporary fact table in $\mathcal{F}_{TC,ext}$ is the copy of the fact table in Figure 3. For the federation selection $\sigma_{Fed[Nation[ANY]/Nlink/Population<30]}(\mathcal{F}_{TC,ext})$, the decoration values Population are needed to filter the fact data; therefore, a SQL SELECT statement is issued against the join of the temporary table in Figure 8 and the temporary fact table, with the predicate on the decoration level and all the columns from the fact table in the SELECT clause. See Figure 9 for the query and the fact table.

```
SELECT   Fact.*
FROM     Fact F,
         Nation[ANY]/Nlink/Population P
WHERE    F. Supplier=P. Supplier
         AND  P.Population<30
```

Quantity	ExtPrice	Supplier	Part	Order	Day
17	17954	S1	P3	11	2/12/1996
36	73638	S2	P5	18	5/2/1992
28	29983	S2	P4	42	30/3/1994
26	26374	S4	P2	20	10/11/1993

Figure 9: The SQL query and the resulting fact table

Definition 3.7 (Federation Selection) Let $Op_1, \ldots, Op_n$ be the child operators of a federation selection operator, $(C, Links, X, T_1), \ldots, (C, Links, X, T_n)$ be their output federations, where $C = (N, D, F)$. Let θ be a predicate over the levels in C. The federation selection operator is defined as: $\sigma_{Fed[\theta]}(\mathcal{F}_{ext}) = (C', Links, X, T')$, where the input is $\mathcal{F}_{ext} = (C, Links, X, T)$, and $T = T_1 \cup \ldots \cup T_n$ is the union of the temporary tables from the child operators. In the output federation, $T' = T \backslash \{R_F\} \cup \{R'_F\}$ means the temporary fact table R_F is replaced by R'_F. The resulting cube is $C' = (N, D, F')$, where the new fact table is $F' = \{t_i | t_i \in R'_F\}$. Suppose S_θ is the set of levels referenced by θ. The new temporary fact table is $R'_F = \sigma_\theta(R_F)$, if $S_\theta = \{\perp_1, \ldots, \perp_l\}$ meaning the predicates only contain the bottom levels. Otherwise, if S_θ has roll-up or level expressions, that is, $\{l_x(\perp_x), \ldots, l_y(\perp_y)\} \subseteq S_\theta$, and $\{l_u[SEM_j]/link_j/xp_j, \ldots, l_v[SEM_k]/link_k/xp_k) \subseteq S_\theta$, then $R'_F = \pi_{R_F.*}(\sigma_\theta(R_F \bowtie R_{\omega_{[\perp_x, l_x]}} \bowtie \ldots \bowtie R_{\omega_{[\perp_y, l_y]}} \bowtie R_{\tau_{l_u[SEM_j]/link_j/xp_j}} \bowtie \ldots \bowtie R_{\tau_{l_v[SEM]/link_k/xp_k}}))$.

Federation Generalized Projection Similar to the federation selection, the federation generalized projection operator Π_{Fed} is also implemented as a SQL SELECT statement over a set of temporary tables. More specifically, a roll-up is a join between the fact table and the temporary table containing the bottom level and the target level, where the common bottom level is taken as the key of the join. Likewise, showing the decoration values together with OLAP values in the result also can be implemented as a roll-up from the bottom level to the decoration level of a decoration dimension. Finally, a SQL aggregation calculates the given aggregate functions of the measures over the grouped facts according to the SELECT and GROUP BY arguments. Note that when performing roll-ups, correct aggregation must be ensured by detecting hierarchy strictness explicitly, e.g., the dimension values of the two levels. If a roll-up along some dimension duplicates facts we disallow further aggregation along that dimension by setting the aggregation type to not available.

Example 3.7 Suppose the temporary fact table in $\mathcal{F}_{TC,ext}$ is the copy of the fact table in Figure 3. For the operator $\Pi_{Fed[Nation[ANY]/Nlink/Population]<SUM(Quantity)>}(\mathcal{F}_{TC,ext})$, the temporary decoration table containing values of Population and Supplier is needed to perform the roll-up, while the other dimensions and measures (not specified in the SELECT clause) will be removed from the cube; therefore, a SQL query is issued against the temporary table from Figure 8 and the temporary fact table with only Population and SUM(Quantity) in the SELECT and GROUP BY clauses. Table Nation/Population is strict, therefore further aggregation is allowed along this decoration dimension. See Figure 10 for the query and the fact table.

SELECT	SUM(Quantity), Population
FROM	Fact F,
	Nation[ANY]/Nlink/Population P
WHERE	F.Supplier=P. Supplier
GROUP BY	Population

Quantity	Population
81	5.3
2	1264.5
26	19.1

Figure 10: The SQL query and the resulting fact table

Definition 3.8 (Federation Generalized Projection) Let $Op_1, \ldots, Op_n$ be the child operators of a federation generalized projection operator, $(C, Links, X, T_1), \ldots, (C, Links, X, T_n)$ be their output federations, where the cube is $C = (N, D, F)$. Let $l_u[SEM_j]/link_j/xp_j, \ldots, l_v[SEM_k]/link_k/xp_k$ be level expressions, $\perp_p, \ldots, \perp_q$ be bottom levels, $l_s(\perp_s), \ldots, l_t(\perp_t)$ be roll-up expressions, and $D_j, \ldots, D_k$ be the dimensions built for the preceding level expressions containing the decoration levels $l_{xp_j}, \ldots, l_{xp_k}$. Furthermore, let $f_x, \ldots, f_y$ be aggregate functions over the levels $\{M_x, \ldots, M_y\} \subseteq \{M_1, \ldots, M_m\}$ such that $\forall f_z \in \{f_x, \ldots, f_y\} \forall D_g \in \{D_s, \ldots, D_t, D_j, \ldots, D_k\}(f_z \in AggType(M_z, D_g)\}$. The Π_{Fed} operator is defined as: $\Pi_{Fed[\perp_p,\ldots,\perp_q,l_s(\perp_s),\ldots,l_t(\perp_t),\ l_u[SEM_j]/link_j/xp_j,\ldots,l_v[SEM_k]/link_k/xp_k]<f_x(M_x),\ldots,f_y(M_y)>}(\mathcal{F}_{ext}) = (C', Links, X, T')$, where $\mathcal{F}_{ext} = (C, Links, X, T)$ is the input, $T = T_1 \cup \ldots \cup T_n$ is the union of the temporary tables from the child operators. In the output federation, $C' = (N, D', F')$ is the updated cube. After the projection, only the temporary table containing the values required by the federation projection are retained, that is, $T' = \{R'_F, R_{\omega_{[\perp_s,l_s]}}, \ldots, R_{\omega_{[\perp_t,l_t]}}, R_{D_j}, \ldots, R_{D_k}\}$, where $R_{D_j}, \ldots, R_{D_k}$ are built by the decoration operators for $l_u[SEM_j]link_j/xp_j, \ldots, l_v[SEM_k]link_k/xp_k$. Unspecified dimensions are also rolled up to the top level and projected away; therefore, the set of dimensions is given as: $D' = \{D_p, \ldots, D_q, D'_s, \ldots, D'_t, D'_j, \ldots, D'_k\}$, where the ordering of dimension values, the hierarchies of levels and the aggregation types are updated in the same way as for the cube generalized projection operator. Moreover, the fact table is given as: $F' = \{t_i | t_i \in R'_F\}$, where the temporary fact table is $R'_F = \{t_i | t_i \in {}_{\perp_p,\ldots,\perp_q,l_s,\ldots,l_t,l_{xp_j},\ldots,l_{xp_k}}\mathcal{G}_{f_x(M_x),\ldots,f_y(M_y)}(R_{F,intermediate})\}$, where $R_{F,intermediate} = R_F \bowtie R_{\omega_{[\perp_s,l_s]}} \bowtie \ldots \bowtie R_{\omega_{[\perp_t,l_t]}} \bowtie R_{D_j} \bowtie \ldots \bowtie R_{D_k}$, $\mathcal{G}$ is the SQL aggregation.

3.7 Inlining XML Data

The inlining operator ι is used to rewrite the selection predicates such that a referenced level expression can be integrated into a predicate by creating a more complex predicate that contains only references to regular dimension levels and constants. Without inlining, the OLAP and XML components can be accessed in parallel, followed by computation of the final result in the temporary component, e.g., selection of the OLAP data according to XML data; therefore, when selection predicates refer to decoration values, a large

amount of OLAP data has to be transferred into the temporary component before it could be filtered. In this situation, it is often advantageous to make the OLAP query dependent on the XML queries. That is, for the predicates referring to level expressions, the XML data and the decorated dimension values are first retrieved. After this, the level expressions are inlined into the predicates which then only refer to dimension levels and constants but have the identical effects as the original ones; thus, the selection can be performed over the cube, and thereby reducing the cube size effectively before the data is transferred to the temporary component.

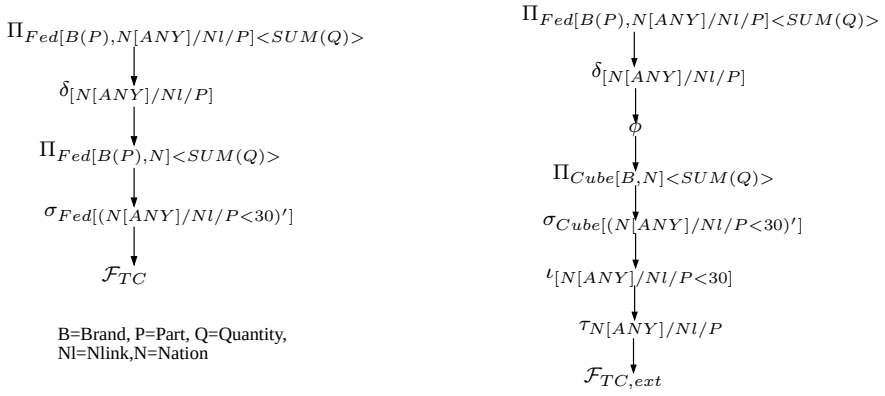
Example 3.8 Consider the predicate: $Nation[ANY]/Nlink/Population < 30$. The decoration data and the decorated dimension values are retrieved from the XML-transfer operator in Example 3.4. Using the result, the predicate is transformed to: $Nation =$'*Denmark*' OR *Nation* ='*United Kingdom*' .

Definition 3.9 (Inlining) Let $\theta_1, \ldots, \theta_n$ be predicates referencing level expressions. θ_i has the following possible formations:

$$\theta_i = \begin{cases} 1. & l_z[SEM]/link/xp\ po\ K, \text{ where } K \text{ is a constant} \\ 2. & l_z[SEM]/link/xp\ po\ l_w, \text{ where } l_w \text{ is a level} \\ 3. & l_z[SEM]/link/xp\ po\ M, \text{ where } M \text{ is a measure} \\ 4. & l_z[SEM_1]/link_1/xp_1\ \ po\ \ l_w[SEM_2]/link_2/xp_2 \\ 5. & l_z[SEM]/link/xp\ \ IN\ \ (K_1, \ldots, K_n), \text{ where } K_i \\ & \text{is a constant value.} \\ 6. & NOT\ (\theta_{i1}) \\ 7. & \theta_{i1}\ bo\ \theta_{i2} \end{cases}$$

where, the binary operator bo is AND or OR, the predicate operator po is one of : $=, <, >, <>, >=, <=$, and LIKE. Let $\mathcal{F} = (C, Links, X)$ be a federation, where $C = (N, D, F)$ is the original cube. $l_j/link_1/xp_1, \ldots, l_k/link_m/xp_m$ are the level expressions referenced by $\theta_1, \ldots, \theta_n$. The ι operator is defined as: $\iota_{[\theta_1,\ldots,\theta_n]}(\tau_{l_j/link_1/xp_1}(\mathcal{F}_{ext,1}), \ldots, \tau_{l_k/link_m/xp_m}(\mathcal{F}_{ext,m})) = (C, Links, X, T')$, where $\mathcal{F}_{ext,i} = \{C, Links, X, T_i\}$ is an extended federation with a temporary component T_i which is an empty set, and $\tau_{l_j/link_1/xp_1}(\mathcal{F}_{ext,1}), \ldots, \tau_{l_k/link_m/xp_m}(\mathcal{F}_{ext,m})$ are the XML-transfer operators used to load decoration values referenced by the level expressions into T_i, $1 \leq i \leq m$. The resulting temporary component T' has new temporary tables, that is: $T' = T \cup \{R_{\tau_{l_j/link_1/xp_1}}, \ldots, R_{\tau_{l_k/link_m/xp_m}}\}$. The inlining operator is positioned at the bottom of a physical plan above the XML-transfer operators and rewrites the predicates in its parameter list to $\theta'_1, \ldots, \theta'_n$. As a consequence, the other occurrences of these predicates in the plan change accordingly. To provide the decoration values required by the inlining operator, the child XML-transfer operators are always executed first, the rest part of the plan is executed after the inlining processes are finished. The *transforming function* $\mathcal{T}(\theta_i)$ rewrites θ_i to θ'_i, which returns the rewritten predicate for each listed formation, respectively. That is, $\theta'_i =$

1. $l_z\ IN\ (t_1, \ldots, t_n)$, where $t_i \in \{e_z | (e_z, e_{xp}) \in R_{\tau_{l_z[SEM]/link/xp}} \wedge e_{xp}\ po\ K = true\}$.
2. $l_z = e_{z1}\ AND\ e_{xp1}\ po\ l_w\ OR\ \ldots\ OR\ l_z = e_{zn}\ AND e_{xpn}\ po\ l_w$, where $(e_{zi}, e_{xpi}) \in R_{\tau_{l_z[SEM]/link/xp}}$.
3. $l_z = e_{z1}\ AND\ e_{xp1}\ po\ M\ OR\ \ldots\ OR\ l_z = e_{zn}\ AND e_{xpn}\ po\ M$, where $(e_{zi}, e_{xpi}) \in R_{\tau_{l_z[SEM]/link/xp}}$.
4. $(l_z = e_{z1}\ AND\ l_w = e_{w1}\ AND\ e_{xp_1 1}\ po\ e_{xp_2 1}\ OR\ \ldots\ OR\ l_z = e_{z1}\ AND\ l_w = e_{wn}\ AND\ e_{xp_1 1}\ po\ e_{xp_2 n})\ OR\ \ldots\ OR\ (l_z = e_{zm}\ AND\ l_w = e_{w1}\ AND\ e_{xp_1 m}\ po\ e_{xp_2 1}\ OR\ \ldots\ OR\ l_z = e_{zm}\ AND\ l_w = e_{wn}\ AND\ e_{xp_1 m}\ po\ e_{xp_2 n})$, where $(e_{zi}, e_{xp_1 i}) \in R_{\tau_{l_z[SEM]/link_1/xp_1}}$, $(e_{wi}, e_{xp_2 i}) \in R_{\tau_{l_w[SEM]/link_2/xp_2}}$.
5. $\mathcal{T}(l_z[SEM]/link/xp = K_1)\ OR \ldots\ OR\ \mathcal{T}(l_z[SEM]/link/xp = K_n)$.



(a) Logical plan (b) Physical plan

Figure 11: A logical plan and its corresponding physical plan

6. $NOT\ (\mathcal{T}(\theta_{i1}))$.
7. $\mathcal{T}(\theta_{i1})\ \ bo\ \ \mathcal{T}(\theta_{i2})$.

Example 3.9 In this example, we show a logical plan and its corresponding physical plan. The plan is enumerated by the query optimizer for the query in Figure 5. The plan is selected so that it is possible to show more physical operators. The logical plan always yields a unique physical plan. A logical operator in a certain context can only be converted to one corresponding physical operator accompanied by other operators that provide data or construct new predicates; thus, a logical plan can be deterministically converted to a physical plan.

In Figure 11(a), the predicate "$N/Nl/P < 30$" is marked to be rewritten and no longer refers to the level expression at execution time. It means the federation selection can then be executed directly in the OLAP component. The bottom two federation operators perform the selection and partial aggregation on the federation before the cube is decorated. Besides the measure Quantity, only the dimensions Part and Suppliers are retained after the projection, but it still allows the decoration afterwards. The top federation generalized projection rolls up the dimensions to the specified levels and calculates the aggregate functions over the specified measure.

The corresponding physical plan is shown in Figure 11(b). The plan is executed in a bottom-up fashion. The XML-transfer operator retrieves the dimension values and their decoration data, followed by the inlining operator which is required by the predicate in the logical plan which is marked to be rewritten. After the predicate is rewritten by the inlining operator, the cube selection slices the cube using the new predicates, followed by the cube generalized projection operator which rolls up the cube to levels Brand and Nation. As an optimized plan, it aggregates the cube as much as possible, therefore unspecified dimensions are rolled up to the top level and removed. The dimension Part is rolled up to Brand as it is required in the SELECT clause. The dimension Suppliers is rolled up to Nation and it is still possible to perform the decoration afterwards. The two cube operators are converted from the two corresponding federation operators at the bottom of the logical plan, which do not refer to external data and can be executed in the OLAP component to reduce the data transferred between components. The fact-transfer operator is responsible for transferring the returned OLAP data to the temporary component after the cube operators process the cube. The decoration and the federation generalized projection operators are performed in the temporary component. Since the starting level Nation is the current bottom level of Suppliers and Nation[ANY]/Nlink/Population is already evaluated by the bottom XML-transfer operator, the decoration operator directly uses the dimension values for Nation from Nlink and the corresponding population data to generate the decoration dimension. If the bottom level is Supplier, a dimension-transfer is required as a child operator of δ to create a table for Nation

and Supplier which is then joined with the table for the level expression to create the decoration dimension linking the facts and the decoration data. The top federation generalized projection operator utilizes SQL operations to roll up the cube furthermore to the decoration level Population. Also, the dimension Orders is rolled up to the top level and removed.

4 Query Evaluation

This section describes the topics related to the physical algebra in the following sequence. First, the logical-to-physical plan conversion is introduced, where the conversion rules transforming logical operators to physical operators and the conversion algorithm manipulating the rules and building the full corresponding physical plans are described. Second, how a physical plan is executed is introduced. There, the evaluation algorithm describes the execution of a whole plan, whereas the following part describes how each operator is evaluated.

4.1 Conversion to Physical Plans

This section describes the conversion of logical query plans into query plans expressed in the physical algebra. A logical query plan is composed of logical operators, where tasks are described on an abstract level. The conversion uses physical operators to replace the logical operators in order to integrate more concrete operational details, e.g., predicate rewriting and intermediate table generations.

Algorithms for Converting Logical Query Plans The algorithm for converting logical plans is shown in Figure 12 in pseudo-code with C-like "// " for comments. Operators in the algorithm are manipulated through memory references. The *convert* function takes the root operator of a logical query plan and returns the root operator of the resulting physical query plan. Given the root, *convert* builds a physical query plan in a bottom-up fashion, after first recursively descending to bottom. The bottom federation is first turned into an extended federation with a temporary component to store the temporary data. The algorithm then converts the operator above to a physical operator by first examining the context and then applying the suitable rules. The context refers to the current operator type, the operator's position, the parameters and the existence of certain operators in the plan. New operators can also be generated to accomplish the desired effect. For example, an operator which applies the inlining technique is needed if a predicate is marked to be rewritten. This process goes on until the root logical operator is converted. The algorithm also partitions the physical plan into different parts so as to schedule the execution tasks in different components. For example, the lower cube operators select and aggregate the cube, whereas the top federation selections and generalized projections generate the final result based on the temporary data. The fact-transfer operator in between connects the two partitions, allowing the data from the OLAP component to be used in the temporary component.

In the following descriptions, child operators will be expressed as arguments of the parent operator and an extended federation produced by an operator can be used to represent a sub-plan. For example, the plan rooted at the decoration operator in Figure 11(b) is represented as: $\delta_{Nation[ANY]/Nlink/Population}(\mathcal{F}_{TC,ext,\phi})$, where $\mathcal{F}_{TC,ext,\phi}$ is the output extended federation by the fact-transfer ϕ below, representing the sub-plan with the fact-transfer operator as the root.

Expanding Logical Decoration According to Definition 3.6, a temporary table or a join of two temporary tables is needed to generate a decoration dimension having the decoration level and the bottom level which will be used as the key to join the temporary fact table; therefore, transfer operators, i.e., XML-transfer and

```
operator convert(operator op)
1)   {
2)       if op is a federation
3)         turn op into an extended federation and return op;
4)       else
5)         get the child operator op_child of op;
6)         operator op'_child = convert(op_child);
7)         put op'_child below op;
8)         if op is a federation selection operator
9)           if the predicates of op should be rewritten to reference only dimension values
10)            replace op with a cube selection using Rule 4;  // to select from the cube in the OLAP component.
11)            generate an inlining operator using Rule 5;      // to inline XML data at execution time.
12)          else
13)            if op does not reference any level expressions and is below all decorations
14)              replace op with a cube selection using Rule 4;  // to select the cube in the OLAP component.
15)            else
16)              expand op using Rule 3;  // add dimension-transfer operators as children
17)                                       // of op to load required temporary values.
18)        else
19)          if op is a federation generalized projection operator
20)            if op does not reference any level expressions and is below all decorations
21)              replace op with a cube generalized projection using Rule 4;  // to aggregate the cube
22)                                                                           // in the OLAP component.
23)            else
24)              expand op using Rule 2;  // add dimension-transfer operators as children
25)                                       // of op to load required temporary values.
26)          else
27)            if op is a decoration operator
28)              expand op using Rule 1;  // add XML-transfer and dimension-transfer operators as children
29)                                       // of op to load required temporary values.
30)            if op is the lowest decoration operator
31)              generate a fact-transfer operator below op using Rule 6;  // put a fact-transfer operator in
32)                    //the plan to transfer the result of the cube operators to the temporary component.
33)        if op is the root operator and no fact-transfer exists in the tree  // when no fact-transfer operator is
34)                                       //created because no decoration operators are present.
35)          generate a fact-transfer operator using Rule 6;  // execute all the operators in the OLAP component.
36)      return the root operator of this sub-plan;
37)  }
```

Figure 12: Transforming the logical query tree

dimension-transfer operators, are generated as child operators below a decoration operator, when temporary tables containing the desired values are not present.

Rule 1 (Expanding Logical Decoration) *The logical decoration operator over $\mathcal{F} = (C, Links, X)$ is $\delta_{l_z[SEM]/link/xp}(\mathcal{F}) = \mathcal{F}'$, where $\mathcal{F}' = (C', Links, X)$ is the logical federation generated by the logical decoration operator. During conversion, the extended federation generated by the lower physical operator is $\mathcal{F}_{ext} = (C, Links, X, T)$. The corresponding physical operator has the following formations according to different contexts.*

- *If the values used to construct the bottom level and the decoration level of the decoration dimension are not pre-sent in the temporary component, that is if $\perp_z \neq l_z \wedge R_{\omega_{[\perp_z,l_z]}} \notin T \wedge R_{\tau_{l_z[SEM]/link/xp}} \notin T$, the physical decoration operator is $\delta_{l_z[SEM]/link/xp}(\tau_{l_z[SEM]/link/xp}(\mathcal{F}_{ext,1}), \omega_{[\perp_z,l_z]}(\mathcal{F}_{ext,2}), \mathcal{F}_{ext}) = (C', Links, X, T')$, where $\mathcal{F}_{ext,1} = (C, Links, X, T_1)$ and $\mathcal{F}_{ext,2} = (C, Links, X, T_2)$, here T_1 and T_2 are the empty sets to store temporary tables, and $T' = T \cup \{R_{\tau_{l_z[SEM]/link_i/xp}}, R_{\omega_{[\perp_{\perp_z,l_z}]}}\}$.*
- *If the values of the bottom level are already present or the values can be retrieved by evaluating the level expression, i.e., if $(\perp_z = l_z \vee R_{\omega_{[\perp_z,l_z]}} \in T) \wedge R_{\tau_{l_z[SEM]/link/xp}} \notin T$, the physical decoration operator is $\delta_{l_z[SEM]/link/xp}(\tau_{l_z[SEM]/link/xp}(\mathcal{F}_{ext,1}), \mathcal{F}_{ext}) = (C', Links, X, T')$, where $\mathcal{F}_{ext,1} = (C, Links, X, T_1)$, T_1 is an empty set for temporary tables, and $T' = T \cup \{R_{\tau_{l_z[SEM]/link_i/xp}}\}$.*
- *If only the values of the bottom level are not present, i.e., if $\perp_z \neq l_z \wedge R_{\omega_{[\perp_z,l_z]}} \notin T \wedge R_{\tau_{l_z[SEM]/link/xp}} \in T$, the physical decoration operator is $\delta_{l_z[SEM]/link/xp}(\omega_{[\perp_z,l_z]}(\mathcal{F}_{ext,1}), \mathcal{F}_{ext}) = (C', Links, X, T')$, where $\mathcal{F}_{ext,1} = (C, Links, X, T_1)$, T_1 is an empty set used to store the temporary tables, and $T' = T \cup \{R_{\omega_{[\perp_z,l_z]}}\}$.*
- *Otherwise, the physical decoration operator has no new child operators, because the values for building the decoration dimension are already present in T. The physical decoration operator is: $\delta_{l_z[SEM]/link/xp}(\mathcal{F}_{ext}) = (C', Links, X, T')$, where no temporary tables is built, that is, $T' = T$.*

Example 4.1 According to the rule, the logical decoration operator in Figure 11(a) yields the decoration operator in the corresponding physical plan in Figure 11(b). The dimension-transfer operator is not needed because the starting level Nation of the level expression is now the bottom level of the dimension Suppliers and the values for Nation can be retrieved from the link data. Since the required XML-transfer operator $\tau_{Nation[ANY]/Nlink/Population}$ is already present at the bottom of the physical plan tree (see Rule 5), the temporary table it generates can be directly used by the decoration operator.

Converting Logical Federation Generalized Projection and Selection to Physical Operators The physical federation generalized projection and selection operators need to join the fact table with temporary tables to roll up the cube to, or evaluate the predicate on higher levels, i.e., the levels whose values do not exist in the temporary fact table; therefore, in the process of converting a logical federation generalized projection operator, new dimension-transfer operators are created as child operators to load the dimension values for the non-bottom levels in the logical federation projection's parameters, if the required temporary values do not exist in the temporary component.

Rule 2 (Converting Logical Federation Generalized Projection) *The logical federation generalized projection operator (FGP) over $\mathcal{F} = (C, Links, X)$ is $\Pi_{Fed[\perp_p,...,\perp_q,l_s(\perp_s),\ ...,l_t(\perp_t),le_u,...,le_v]<f_x(M_x),...,f_y(M_y)>}(\mathcal{F}) = \mathcal{F}'$, where $\mathcal{F}' = (C', Links, X)$ is the output federation, $\{l_s(\perp_s), \ldots, l_t(\perp_t)\}$ is a set of roll-up expressions, and $le_u, \ldots, le_v$ are level expressions which yields the decoration operators below in the logical plan. During conversion, $\mathcal{F}_{ext} = (C, Links, X, T)$ is the output of the lower sub-tree, where logical operators are transformed to physical operators. The corresponding physical FGP operator is*

$\Pi_{Fed[\perp_p,\ldots,\perp_q,l_s(\perp_s),\ldots,l_t(\perp_t),le_u,\ldots,le_v]\ <f_x(M_x),\ldots,f_y(M_y)>}(\omega_{[\perp_i,l_i]}(\mathcal{F}_{ext,1}),\ldots,\omega_{[\perp_j,l_j]}(\mathcal{F}_{ext,n}),\mathcal{F}_{ext}) = (C', Links, X, T')$*, where, the roll-up expressions* $\{l_i(\perp_i),\ldots,l_j(\perp_j)\} \subseteq \{l_s(\perp_s),\ldots,l_t(\perp_t)\}$ *yield the parameters for the dimension-transfer operators, which are then used to load the values not existing in* T*, that is,* $\forall l_k(\perp_k) \in \{l_i(\perp_i),\ldots,l_j(\perp_j)\}(R_{\omega_{[\perp_k,l_k]}} \notin T)$*. Moreover,* $\mathcal{F}_{ext,1} = (C, Links, X, T_1),\ldots,\mathcal{F}_{ext,n} = (C, Links, X, T_n)$*, where* $T_1,\ldots,T_n$ *are empty sets and* $n = |\{l_i(\perp_i),\ldots,l_j(\perp_j)\}|$*. In the output extended federation,* C' *is the same as in* $\mathcal{F}'$*. And* $T' = T \cup \{R_{\omega_{[\perp_i,l_i]}},\ldots,R_{\omega_{[\perp_j,l_j]}}\}$*.*

Similarly, when converting a logical federation selection operator, new dimension-transfer operators are created as child operators to load the dimension values for the non-bottom levels referred by the predicates, if the required temporary values are not loaded in the temporary database.

Rule 3 (Converting Logical Federation Selection) *The logical federation selection operator over* $\mathcal{F} = (C, Links, X)$ *is* $\sigma_{Fed[\theta]}(\mathcal{F}) = \mathcal{F}'$*, where* $\mathcal{F}' = (C', Links, X)$ *is the output federation. Let* S_θ *be the set of levels in* θ *and* $\{l_x(\perp_x),\ldots,l_y(\perp_y)\} \subseteq S_\theta$ *be a set of roll-up expressions. During conversion,* $\mathcal{F}_{ext} = (C, Links, X, T)$ *is the output of the lower sub-tree, where logical operators are transformed to physical operators. The corresponding physical federation selection is:* $\sigma_{Fed[\theta]}(\omega_{[\perp_i,l_i]}(\mathcal{F}_{ext,1}),\ldots,\omega_{[\perp_j,l_j]}(\mathcal{F}_{ext,n}),\mathcal{F}_{ext})) = (C', Links, X, T')$*, where, the roll-up expressions* $\{l_i(\perp_i),\ldots,l_j(\perp_j)\} \subseteq \{l_x(\perp_x),\ldots,l_y(\perp_y)\}$ *yield the parameter levels for the new dimension-transfer operators, which are then used to load values not existing in T, that is,* $\forall l_k(\perp_k) \in \{l_i(\perp_i),\ldots,l_j(\perp_j)\}(R_{\omega_{[\perp_k,l_k]}} \notin T)$*. Moreover,* $\mathcal{F}_{ext,1} = (C, Links, X, T_1),\ldots,\mathcal{F}_{ext,n} = (C, Links, X, T_n)$*, where* $T_1,\ldots,T_n$ *are empty sets and* $n = |\{l_i(\perp_i),\ldots,l_j(\perp_j)\}|$*. In the output extended federation,* C' *is the same as in* $\mathcal{F}'$*. And* $T' = T \cup \{R_{\omega_{[\perp_i,l_i]}},\ldots,R_{\omega_{[\perp_j,l_j]}}\}$*.*

Example 4.2 The top federation generalized projection operator in Figure 11(a) is converted to the operator at the top of the corresponding physical plan in Figure 11(b). Because the lower federation generalized projection operator in the logical plan has rolled up the dimension Parts to the level Brand, there is no dimension-transfer operators below the physical federation generalized projection. Otherwise, $\Pi_{Fed[Brand(Part),Nation[ANY]/Nlink/Population]}(\mathcal{F}_{TC})$ is converted to $\Pi_{Fed[Brand(Part),Nation[ANY]/Nlink/Population]}(\omega_{[Part,Brand]}(\mathcal{F}_{TC,ext,\delta,1}),\mathcal{F}_{TC,ext,\delta})$, if Parts has Part as the bottom level. Here $\mathcal{F}_{TC,ext,\delta}$ is the output extended federation of the lower decoration operator, $\mathcal{F}_{TC,ext,\delta,1}$ is the same as $\mathcal{F}_{TC,ext,\delta}$ except that the set of temporary tables is empty.

Replacing Federation Generalized Projection and Federation with Cube Operators A logical federation selection can be replaced by a cube selection if the federation selection operator does not reference level expressions, i.e., selection can be performed directly in the OLAP component when no external XML data is required. A logical federation selection can also be replaced if the predicate is rewritten at execution time to reference only dimension values. Using cube operators can reduce the amount of data being transferred between the OLAP and the temporary components. Similarly, a logical federation generalized projection can be replaced by a cube generalized projection when no level expressions are referred by the projection.

Rule 4 (Replacing Logical Operators by Cube Operators) *Suppose the logical federation selection over* $\mathcal{F} = (C, Links, X)$ *is* $\sigma_{Fed[\theta]}(\mathcal{F}) = (C', Links, X)$*. Let* S_θ *be the set of levels in* θ *and has no references to level expressions. During conversion,* $\mathcal{F}_{ext} = (C, Links, X, T)$ *is the output of the lower sub-tree, where operators are transformed to physical. The cube selection operator,* $\sigma_{Cube[\theta]}(\mathcal{F}_{ext}) = (C', Links, X, T)$*, is used to replace the logical federation selection.*

Suppose the logical federation generalized projection operator over $\mathcal{F} = (C, Links, X)$ *is* $\Pi_{Fed[\perp_p,\ldots,\perp_q,l_s(\perp_s),\ldots,l_t(\perp_t)]<f_x(M_x),\ldots,f_y(M_y)>}(\mathcal{F}) = (C', Links, X)$ *which does not refer to level expressions. During conversion,* $\mathcal{F}_{ext} = (C, Links, X, T)$ *is the output of the lower sub-tree, where operators are physical. The cube generalized projection operator,* $\Pi_{Cube[\perp_p,\ldots,\perp_q,l_s,\ldots,l_t]<f_x(M_x),\ldots,f_y(M_y)>}(\mathcal{F}_{ext}) = (C', Links, X, T)$*, is used to replace the logical federation generalized projection.*

Example 4.3 According to the rule, the federation selection and generalized projection operators in Figure 11(a) are replaced by the cube selection and generalized projection operators in Figure 11(b), respectively. The cube selection inherits the same parameter, whereas the parameters of the other operator are the highest levels that the cube can be rolled up to. At execution time, when the marked predicate is rewritten, the two cube operators are executed in the OLAP component.

Generating an Inlining Operator An inlining operator is needed to rewrite a predicate of a federation selection to reference only dimension levels and constants at execution time.

Rule 5 (Generating an Inlining Operator) *Suppose in the process of conversion that θ_p is the predicate of a logical federation selection σ_{Fed} to be rewritten by inlining the level expressions, $l_j/link_{1_j}/xp_{1_j}, \ldots, l_k/link_{m_k}/xp_{m_k}$. The logical federation selection is first replaced by a cube selection σ_{Cube} and outputs $\mathcal{F}_{ext,\sigma_{Cube}} = \{C_{\sigma_{Cube}}, Links, X, T_{\sigma_{Cube}}\}$. An inlining operator and required XML-transfer operators are generated at the bottom of the query plan as $\iota_{[\theta_p]}(\tau_{l_j/link_{1_j}/xp_{1_j}}(\mathcal{F}_{ext,1_j}), \ldots, \tau_{l_k/link_{m_k}/xp_{m_k}}(\mathcal{F}_{ext,m_k})) = (C, Links, X, T')$, where the components are described in Definition 3.9. Moreover, the temporary tables in T' propagate upwards along the plan tree and add themselves into the sets of temporary tables in the output extended federations of the operators passed by, until σ_{Cube} is reached and the output extended federation is changed to $\mathcal{F}_{ext,\sigma_{Cube}} = (C_{\sigma_{Cube}}, Links, X, T'_{\sigma_{Cube}})$, where $T'_{\sigma_{Cube}} = T_{\sigma_{Cube}} \cup T'$, meaning the new temporary tables can be used by other operators afterwards. If an lining operator is already generated at the bottom, then θ_p is added into the parameters of the existing inlining operator and the required XML-transfer operators are added as children operators, that is, $\iota_{[\theta_1,\ldots,\theta_n,\theta_p]}(\tau_{l_u/link_{1_u}/xp_{1_u}}(\mathcal{F}_{ext,1_u}), \ldots, \tau_{l_v/link_{m_v}/xp_{m_v}}(\mathcal{F}_{ext,m_v}), \tau_{l_j/link_{1_j}/xp_{1_j}}(\mathcal{F}_{ext,1_j}), \ldots, \tau_{l_k/link_{m_k}/xp_{m_k}}(\mathcal{F}_{ext,m_k}))$, where $\theta_1, \ldots, \theta_n$ are the existing predicates of the inlining operator and $\tau_{l_u/link_{1_u}/xp_{1_u}}(\mathcal{F}_{ext,1_u}), \ldots, \tau_{l_v/link_{m_v}/xp_{m_v}}(\mathcal{F}_{ext,m_v})$ are the existing XML-transfer operators at the bottom of the plan. The new temporary tables will propagate upwards in the same way as explained above.*

Example 4.4 The federation selection operator in Figure 11(a) is first replaced by the cube selection operator in Figure 11(b). Then, the inlining operator is generated below to instantiate the inlining process. The bottom XML-transfer operator loads the dimension values for the starting level and the decoration XML data into the table $R_{\tau_{Nation[ANY]/Nlink/Population}}$, which becomes a member of the sets of temporary tables in the extended federations produced by the above cube selection and the inlining operator, and later used by the decoration operator $\delta_{Nation[ANY]/Nlink/Population}$ to build the decoration dimension. At execution time, the inlining operator will first load the linked values and rewrite the marked predicate. The cube selection then can be executed in the OLAP component.

Partitioning Physical Plans We partition a query plan into three parts. The lowest part includes the inlining operator and the lower XML-transfer operators, which are the first to be executed to enable the following cube selections to be performed in the OLAP component. The middle part is constructed in such a way that the OLAP cube will be aggregated as much as possible, while still allowing the decorations to be performed. This part is evaluated in the OLAP component using the OLAP component queries. The top part is mainly composed of operations performed in the temporary component and implemented in SQL operations, e.g., decoration, federation selection and generalized projection. The federation selection and generalized projection operators depend on the result set of the OLAP component queries, therefore can only be executed after the middle part. A special case of partitioning is that when no level expressions needs inlining, the plan only has the top two parts.

In a physical query tree, the fact-transfer operator is used to separate the operations in the OLAP component and the temporary component. Facts are copied to the temporary component after the OLAP cube is sliced or aggregated, which enables further operations, e.g., decoration, to be performed in the temporary

component. When no decoration operators exists in the plan, it means the cube data does not need to federate with external data and the plan can be executed entirely in the OLAP component. The fact-transfer operator in this case is used to generate the final query result in the temporary database.

Rule 6 (Generation of Fact-transfer Operator) *After the above rules are applied, a fact-transfer is then generated below the lowest decoration operator. If no decoration operators exist in the plan, a fact-transfer operator is generated below the lowest federation selection operator referring to measures if such selection operators exist in the plan, otherwise a fact-transfer operator is generated above all the operators.*

Note that the federation selection operators with predicates referring to measures cannot be executed in the OLAP component by the current query engine (due to a limitation in MS SQL Server, see Section 4.2), therefore they cannot be put below the fact-transfer operator.

Example 4.5 According to the rule, a fact-transfer operator is generated below the decoration operator in the plan in Figure 11(b) to separate the operations in the temporary component from the lower two cube operations. At execution time, the fact-transfer operator transfers the result of the lower operators to the temporary component, thereby enabling the above operations.

4.2 Physical Plan Evaluation

In this section, we first describe the evaluation algorithm for a physical execution plan and then the evaluation methods for the physical operators.

Evaluation Algorithm The evaluation algorithm is shown in pseudo-code below.

void OpEvaluation(Operator Op)
1) {
2) if Op is a $\mathcal{F}_{ext}$
3) **return**;
4) get all the child operators $Op_1, \ldots, Op_n$ below Op;
5) perform each OpEvaluation(Op_i) in separate threads;
6) wait until all threads return;
7) find the required tables in $TempTable$;
8) execute Op;
9) add an entry for the output in $TempTable$;
10) return;
11) }

Figure 11(b) shows an execution plan, where a leaf is an extended federation. In lines 2 and 3, the algorithm just returns when it reaches the bottom of a plan tree (which always consists of a federation) as no operations need to be performed on these. When the algorithm returns from the bottom, the real execution starts; therefore, the algorithm follows the idea of the conventional pull-based iterator model [12], where the lower part of the plan tree provides data for higher operators. However, data is not directly transferred between operators through pipes or shared memory. Instead, temporary tables are used. A hashtable, $TempTable$, is used to record the temporary tables, where the creator information composes the key to identify each entry. Sibling operators in the plan tree can be executed in parallel, therefore a multi-threaded technique is adopted in implementation, as line 5 shows. After all the sub-threads are finished, the real execution of Op begins. Here, queries may be constructed for different components, and furthermore could be evaluated. Data can also be transformed into temporary tables. After the execution, the output is registered in $TempTable$. For

some operators, e.g., a dimension-transfer, the result might be a real table. But for a federation selection or generalized projection, it is a SQL query string, e.g., as in Example 3.7, which then can be nested into the query string of a higher operator and evaluated later at some point in batch-mode.

Evaluation of Operators Queries on the OLAP component is dependent to a large extent on the underlying OLAP server, in our case, Microsoft Analysis Services (AS). We use the SQL SELECT statement supported by AS [24] instead of Microsoft Multidimensional Expressions (MDX), because it is easier and faster to build up the temporary fact table in relational form. Based on the supported SQL syntax definition, the part below the fact-transfer operator in the general form $\Pi_{Cube[\mathcal{L}]<F(M)>}(\sigma_{Cube[\theta]}(\mathcal{F}_{ext}))$ constructs a query: SELECT F(M), $\mathcal{L}$ FROM N WHERE θ GROUP BY $\mathcal{L}$, where, θ is a predicate, $\mathcal{L}$ represents levels that the cube is rolled up to, $F(M)$ stands for the aggregate functions over the measures, and N is the cube name. A dimension-transfer also constructs an OLAP query. For example, for a dimension-transfer operator $\omega_{[\perp_z,l_z]}(\mathcal{F}_{ext})$, the following query is used, SELECT DISTINCT $\perp_z$, l_z FROM N. Note that the HAVING clause is not supported in the syntax definition, therefore selections on measures can only be performed in the temporary database. This leads to the restriction in Rule 6 which then can be ignored when an OLAP component fully supporting the HAVING clause is used.

Fact-transfer is implemented with Microsoft Transact-SQL function OPENQUERY. As pointed out by [25], the OPENQUERY function provides the best results for pass-through queries from SQL Server to Analysis Services. In our early stage of looking for data loading techniques, experiments showed that the function is the stablest with the highest transfer speed, compared with BULK INSERT [27] and prepared statements. Other efficient inter-process communication techniques for programmers, e.g., pipes, have not been found available for data transferring between Microsoft Analysis Services and SQL Server. Similar to OPENXML, the OPENQUERY function can also be used in a SQL SELECT INTO statement, where the result set returned by the Analysis Services are taken as a table in the FROM clause.

Federation selection and generalized projection operators construct regular SQL queries. For a federation selection operator, $\sigma_{Fed[\theta]}(\mathcal{F}_{ext})$, this SQL query is constructed: SELECT * FROM F WHERE θ, where F is the temporary fact table. If θ references levels not in F, a join table of the fact table and several tables built by dimension-transfer or decoration operators are used instead. Similarly, the SQL query constructed for a federation generalized projection $\Pi_{Fed[\mathcal{L}]<F(M)>}(\mathcal{F}_{ext})$ is: SELECT $F(M)$,$\mathcal{L}$ FROM F GROUP BY $\mathcal{L}$, where F can be the fact table or a join table. Following Definition 3.6, the decoration $\delta_{l_z[SEM]/link/xp}(\mathcal{F}_{ext})$ which decorates the non-bottom level l_z with the XML values referenced by xp leads to the SQL query: SELECT DISTINCT l_z, l_{xp} INTO $R_{\delta_{l_z[SEM]/link/xp}}$ FROM R_1 INNER JOIN R_2 ON $R_1.l_z = R_2.l_z$, where $l_z \neq \perp_z$, and R_1 and R_2 are the tables containing the dimension values of l_z and decoration values. If l_z is a bottom level, then the decoration dimension can be built directly from $link$ and the XML values pointed at by xp in the XML documents; therefore, only one table needs to be present in the FROM clause, which is $R_{\tau_{l_z[SEM]/link/xp}}$.

The XML-transfer operator is implemented using Microsoft's OPENXML technique [23]. The referenced XML documents referenced by level expressions are loaded directly into the temporary component. The XML component query is a SQL INSERT INTO statement with a nested SELECT statement which maps an XML document into a table through a schema definition by the function OPENXML.

The inlining operator uses the *Recordset* objects from Microsoft Active Data Objects (ADO) to load the required values into memory. To select these values, SQL queries like: SELECT * FROM $R_{l_z[SEM]/link/xp}$ are evaluated on the tables built by the XML-transfer operators. The new predicate string is composed as in Definition 3.9. In our implementation, the SQL queries are executed in SQL Server through ADO *Connection* or *Command* objects synchronously.

5 Query Optimization

This section presents the topics related to query optimization. First, the whole optimization process is explained and illustrated with a figure showing the structure of the optimizer. Then, the transformation rules used in plan rewriting are introduced with examples. The third part part describes the cost model for a physical plan and the cost components. The last part presents how the main functions of the query optimizer are implemented with a concrete example and algorithms in pseudo-code.

5.1 Architecture of the Optimizer

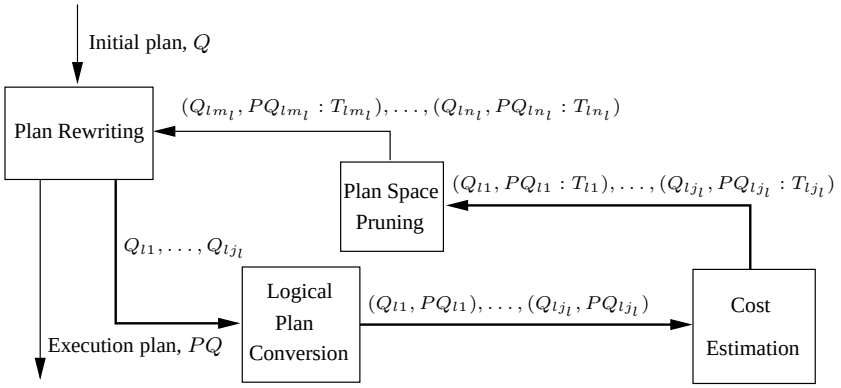


Figure 13: Inner structure of the query optimizer

As the architecture of the query engine in Figure 6 shows, the query optimizer takes the initial plan and generates the final execution plan. The query optimization is a Volcano-like [13], *rule-based* process. However, the optimizer is a totally different implementation specialized for OLAP-XML federations, including: 1) faster logical plan enumeration (by a factor of five or more), because the logical plans are considerably smaller than physical plans without integrating the detailed data retrieval and manipulation algorithms, 2) novel *transformation rules* (see below) specialized for OLAP-XML federations, 3) a novel cost-model for the federation components (see below), which have a high degree of autonomy , and 4) the inlining technique that rewrites selection predicates as introduced above.

Figure 13 shows the structure of the optimizer. There are four main phases. First, *plan rewriting*, where the logical plans generating the equivalent federations are enumerated for the input logical plan, we call these *equivalent plans*. *Transformation rules* are applied to enumerate equivalent plans (see below). A typical transformation rule inspects adjacent operators in a query tree and checks if a plan with the same output federation can be constructed by switching operators and adjusting corresponding parameters. In Figure 13, $Q_{l1}, \ldots, Q_{lj_l}$ are the equivalent plans for query plan Q_l. At the second phase, a physical query plan is generated for each logical plan. Physical query plans have more concrete query evaluation information, therefore they are generated in order to estimate the evaluation costs. In Figure 13, after conversion, each logical plan Q_{li} is accompanied by a physical plan PQ_{li}, $i \in \{1, \ldots, j_l\}$. At the *cost estimation* phase, the optimizer traverses through each physical plan, estimates the cost of the operators, and generates the overall cost using *cost functions*. Now, in the same figure, each physical plan PQ_{i_l} is labeled with the estimated execution time. Usually, the equivalent plans of an original plan are generated on the basis of the equivalent plans of its sub-plans; therefore, it is necessary to prune the plan space when more and more plans are being generated as the process going on. The fourth phase, *plan space pruning*, removes the expensive and meaningless plans using cost-based pruning techniques (see below). In Figure 13, after

pruning, some plans are removed from the plan space, and only a subset of the original plans are kept, that is, $\{Q_{lm_l}, \ldots, Q_{ln_l}\} \subseteq \{Q_{l1}, \ldots, Q_{lj_l}\}$.

The optimization for the initial plan is performed operator by operator through a bottom-up fashion; therefore the process iterates through the four phases several times. The plan rewriting process first starts from the bottom operator of the initial plan, generates the equivalent plans which are then processed through the remaining three phases. The higher operator, which is just above the bottom one, constructs new trees on top of the result plans from the first iteration. The second iteration then starts from generating equivalent plans for these new plans, and proceeds until the pruning phase is finished; therefore, the process of iterations goes on until it reaches the top. When the pruned plan space for the root operator is generated, the physical plan with the least cost is selected as the execution plan. In Figure 13, l stands for the index of the operators in the initial plan tree, Q, starting from the bottom. It is 1 for the first iteration.

5.2 Transformation Rules

In this section, we present a collection of transformation rules for the logical federation algebra. Although some rules are variants of existing rules in relational systems [36], all the rules are specialized for OLAP-XML federation with the hierarchical structure and the virtual dimensions taken into consideration. Following this, Rules 7, 8, 13, and 14 can be thought of as being developed from well-known relational rules on select and group-by. Rules 9, 11, and 12 aim to optimize cases when the special OLAP-XML decoration operator is involved, and thereby are novel. Rule 10 which concerns inlining, is also novel.

The transformation rules are given as equivalences that express that two plans in logical algebraic expressions generate exactly the same federation semantics, i.e. given the same federation, the output federations are equivalent. That is, they have the identical XML documents, links, measures and the dimensions rolled up to the same levels. Also, the fact data are filtered by the predicates with identical effects (but not necessarily identical specifications). Heuristics are also used in the rules to construct new plans. A typical heuristics is to reduce the size of the intermediate cube to the largest possible extent, which will at execution time lead to less usage of temporary space and less data being transferred, and also less operator execution time. For example, selection and generalized projection operations are performed to the maximum possible extent in the OLAP component rather than the temporary component. Another example is, redundant operators are always to be removed. In the formal presentation, a left-to-right rule (denoted by $\rightarrow$) can only reconstruct the plan expression on the left into the form of the right side, while each side in a bidirectional rule (denoted by $\leftrightarrow$) can be reconstructed to the other side.

In the rules, we use the following denotations. Let:

- MaxStrict: $\mathcal{P}(Levels) \rightarrow Levels$ be a function that given a set of levels from the same dimension returns the uppermost such level (max-strict) that do not introduce *non-strictness* when rolled up to from the bottom level. A hierarchy of levels is strict if no dimension value has more than one parent value from the same level [20, 33]. When a dimension is rolled up to the level returned by MaxStrict from the bottom level, safe aggregation is secured, that is, no values from the lower levels are duplicated; therefore, further aggregation is still allowed along that hierarchy.

- S_θ be the levels referenced by the predicate θ.

- $\mathcal{F}$ be a federation or a placeholder representing the sub-query tree below the current operator. For example, during transformation, the logical query tree in Figure 11(a) can be written as: $\Pi_{Fed[Brand(part),\ Nation[ANY]/Nlink/Population]<SUM(Quantity)>}(\mathcal{F}_{TC})$, where $\mathcal{F}_{TC}$ is the output of the lower decoration operator and used to represent the lower sub-query tree when the detail information of the lower operators is not as important as the top federation generalized projection operator.

5.2.1 Rules Involving Federation Selection and Generalized Projection

In some relational optimization systems, heuristics like "perform selection operations as early as possible" and "perform projections early" [40] are used to reduce intermediate tuples. Such ideas can still be applied on our federation system to reduce the intermediate cube. However, the hierarchical structure of the data must be taken into consideration during repositioning of the operators, e.g., a federation generalized projection cannot roll up the cube higher than the levels referenced by the federation selection executed afterwards because these levels would no longer exist after the roll-up.

The commutativity rule of the federation selection and generalized projection operators holds if such requirement for the hierarchical data is fulfilled. However, the rule brings in the uncertainty of which operator should actually be executed first since both of them can reduce the cube's size. The decisions can be made through *cost-based* strategies, where estimated costs of the two plans are compared on the basis of the factors, e.g., the selectivities of the predicates and the size of multidimensional aggregates (see Section 5.3).

Rule 7 (Commutativity of Federation Generalized Projection and Selection) *A federation generalized projection operator and a selection operator are commutative if the levels referred by the selection are not projected away by the projection, that is, if θ does not reference measures, and for each level l_i in θ there exists a level $l'_i \in \mathcal{L}$ such that $l'_i \sqsubseteq_i l_i$, the following rule holds:*

$$\Pi_{Fed[\mathcal{L}]<F(M)>}(\sigma_{Fed[\theta]}(\mathcal{F})) \leftrightarrow \sigma_{Fed[\theta]}(\Pi_{Fed[\mathcal{L}]<F(M)>}(\mathcal{F}))$$

Proof sketch: Since a single fact in the fact table will satisfy the selection predicate both before and after grouping, exactly the same facts will be selected by the selection operation in the two cases. However, measure values are very possibly changed by the generalized projection, thereby providing a different level of numerical summary.

Example 5.1 $\Pi_{Fed[Order]<SUM(Quantity)>}(\sigma_{Fed[Customer= \text{ "}Customer\sharp 01\text{"}]}(\mathcal{F}_{TC}))$ is equivalent to $\sigma_{Fed[Customer= \text{ "}Customer\sharp 01\text{"}]}(\Pi_{Fed[Order]<SUM(Quantity)>}(\mathcal{F}_{TC}))$, as the orders belonging to a customer do not change before and after grouping.

Rule 7 does not apply for the special cases where a federation selection cannot be pulled above a projection as the referred levels no longer exist after the cube is aggregated. However, part of the projection can still be pushed down to allow aggregation of the cube to some extent such that the predicate must still be able to be evaluated. In addition, it must still be possible to roll up the cube to the levels specified in the original generalized projection, therefore the new generalized projection can only roll up along a strict hierarchy, otherwise further aggregation to the original projection is prohibited.

Rule 8 (Pushing Federation Generalized Projection Below Selection) *A part of a federation generalized projection can be performed prior to the federation selection below if the original projection rolls up the cube to the levels higher than those referred by the selection's predicate, meaning that the cube can be rolled up to certain levels without interfering the selection afterwards. That is, if θ does not reference measures, and for a level l_i in θ there exists a level $l'_i \in \mathcal{L}$ such that $l_i \sqsubseteq_i l'_i$, the following rule holds:*

$$\Pi_{Fed[\mathcal{L}]<F(M)>}(\sigma_{Fed[\theta]}(\mathcal{F})) \rightarrow \Pi_{Fed[\mathcal{L}]<F(M)>}(\sigma_{Fed[\theta]}(\Pi_{Fed[\mathcal{L}']<F(M)>}(\mathcal{F}))$$

where, $\mathcal{L}' = \{l_j | l_j \in \mathcal{L} \wedge \exists l'_j \in S_\theta \backslash \mathcal{L}_\theta (l_j \sqsubseteq_j l'_j)\} \cup \{MaxStrict(\{l\}) | l \in \mathcal{L}_\theta\}$, and $\mathcal{L}_\theta = \{l_i | l_i \in S_\theta \wedge \exists l'_i \in \mathcal{L}(l_i \sqsubseteq_i l'_i)\}$, representing the levels in the predicate that are lower than the levels in the GROUP BY clause. Here, the levels that the cube is going to be rolled up to prior to the federation selection are divided into two subsets: the levels that roll up the cube to the maximum possible extent and those that allow federation selection to be performed afterwards.

Proof sketch: If a dimension value in a fact aggregated by the new projection satisfies the selection predicate then all facts in the original cube that correspond to children of that dimension value will also satisfy the predicate. However, measure values associated with different set of levels will possibly be aggregated to different results.

Example 5.2 $\Pi_{Fed[Customer,Brand]<SUM(Quantity)>}(\sigma_{Fed[\quad Order=\text{“}Order\sharp 01\text{”}]}(\mathcal{F}_{TC}))$ is equivalent to $\Pi_{Fed[Customer,\ Brand]<SUM(Quantity)>}(\sigma_{Fed[Order=\text{“}Order\sharp 01\text{”}]}(\Pi_{Fed[Order,\ Brand]<SUM(Quantity)>}(\mathcal{F}_{TC})))$. In the *TC* cube, each order is supposed to belong to one customer. The new generalized projection rolls up the cube to Order and Brand, making it possible to perform the selection on Order and roll up to Customer afterwards.

5.2.2 Rules Involving Federation Selection and Decoration

A federation selection operator slices the cube so that only the facts satisfying the given predicate are retained; therefore, moving federation selections down leads to, at execution time, less temporary space and less data transfer between the OLAP component and the temporary component.

Rule 9 (Pushing Federation Selection Below Decoration) *A federation selection operator can be pushed below a decoration operator if the predicate does not refer to the decoration level, i.e., if* $l_z[SEM]/link/xp \notin S_\theta$*, the following rule holds:*

$$\sigma_{Fed[\theta]}(\delta_{l_z[SEM]/link/xp}(\mathcal{F})) \rightarrow \delta_{l_z[SEM]/link/xp}(\sigma_{Fed[\theta]}(\mathcal{F}))$$

Proof sketch: If no level expression is involved in the selection, the predicate will only be evaluated on the existing data. Moreover, selections do not change the federation schema nor the dimension data, thus the decoration can be moved inside or outside of the selection's argument. The right to left transformation is not introduced because we want to push the federation selection as low as possible so it can be evaluated in the OLAP component to reduce inter-component data transfer.

Example 5.3 $\sigma_{Fed[Customer=\text{“}Customer\sharp 01\text{”}]}(\delta_{Nation[ANY]/\ \ Nlink/Population}(\mathcal{F}_T))$ is equivalent to $\delta_{Nation[ANY]/Nlink/Population}(\sigma_{Fed[Customer=\text{“}Customer\sharp 01\text{”}]}(\mathcal{F}_T))$. The second plan first filters the facts before it is decorated and the same decoration dimension can still created using the unchanged dimension values and linked data.

As Rule 9 implies, a federation selection operator cannot be pushed below a decoration operator if the predicate references the decoration level. However, the rule can still be applied if there is a way to rewrite the predicate to fulfill the requirement. The inlining technique is therefore used. The new predicate no longer references the level expression but still has the same effect on the facts. The following rule applies inlining and then uses Rule 9 to push the selection down.

Rule 10 (Inlining of Decoration in Federation Selection) *A federation selection on a decoration level can be pushed below the decoration operator which builds the decoration dimension, while the predicate is assigned with a special mark and will be rewritten at execution time to a new one with the same effect and only references to regular dimension levels and values. That is, if the predicate* θ *contains references to the level expression* $l_z[SEM]/link/xp$*, the following rule holds:*

$$\sigma_{Fed[\theta]}(\delta_{l_z[SEM]/link/xp}(\mathcal{F})) \rightarrow \delta_{l_z[SEM]/link/xp}(\sigma_{Fed[\theta']}(\mathcal{F}))$$

where, θ' no longer refers to XML data, and is a placeholder at optimization time for the real predicate having the same filtering effects as θ, with references only to regular dimension levels and constants.

Proof sketch: This is possible because the modified predicate is expressed in terms of constant values resulting from evaluating the level expressions instead of referring to the level expressions directly.

Example 5.4 $\sigma_{Fed[Nation[ANY]/Nlink/Population=1017645163]}(\delta_{Nation[ANY]/Nlink/Population}(\mathcal{F}_T))$ is equivalent to $\delta_{Nation[ANY]/Nlink/Population}(\sigma_{Fed[(Nation[ANY]/Nlink/\ Population=1017645163)']}(\mathcal{F}_T))$. The predicate $Nation[ANY]\ /Nlink/Population = 1017645163$ becomes a placeholder in the second plan and will be rewritten to $Nation =$ "$India$" by inlining at execution time. The two predicates have the same effect on the facts because India, and only India has exactly a population of 1017645163. The marked predicate is at optimization time considered to no longer reference decoration values; therefore, the federation selection can be pushed down and performed on the OLAP cube, which at execution time leads to less fact data transfer between components.

5.2.3 Rules Involving Federation Generalized Projection and Decoration

A logical federation generalized projection removes all dimensions that are not present in the parameters, and rolls up the remaining dimensions to the specified levels. A decoration dimension is considered as a regular dimension and would therefore be projected away by a federation generalized projection that does not reference the decoration level of that dimension using a level expression. This could happen if the decoration level is only used in a predicate to select the cube. Moreover, if the decoration operator which generates the decoration dimension is right below such a federation generalized projection operator, the decoration operator itself can be removed. For example, after Rule 10 is applied, the decoration operator is moved above the federation selection operator. If the query plan only uses the XML data which is inlined in the new predicate for selecting in the cube, then the decoration operator and its resulting dimension are no longer valuable to the above operators and thus considered redundant.

Rule 11 (Redundant Decoration Below Federation Generalized Projection) *A decoration operator can be removed if the federation generalized projection above does not include the level expression referring to the decoration level. That is, If* $l[SEM]/link/xp \notin \mathcal{L}$ *the following holds:*

$$\Pi_{Fed[\mathcal{L}]<F(M)>}(\delta_{l[SEM]/link/xp}(\mathcal{F})) \rightarrow \Pi_{Fed[\mathcal{L}]<F(M)>}(\mathcal{F})$$

Proof sketch: If the decoration dimension does not occur in the federation generalized projection it is projected away before it is used and therefore it is not necessary to decorate the cube.

Example 5.5 $\Pi_{Fed[Customer,Brand]<SUM(Quantity)>}(\delta_{Nation[ANY]/Nlink/Population}(\mathcal{F}_{TC}))$ is equivalent to $\Pi_{Fed[Customer,Brand]<SUM(Quantity)>}(\mathcal{F}_{TC})$, because the projection operator removes all the dimensions except for those containing Customer and Brand.

There are situations where the federation projection operator above a decoration operator contains the level expression which refers to the decoration level of the dimension produced by the decoration operator below. In that case, the equivalent plan has a new federation generalized projection below the decoration. Intuitively, this means that instead of decorating a cube and then aggregating it, the cube is first aggregated to yield an intermediate cube. After this, the cube is decorated and then further aggregated to produce the final result. The new projection aggregates the cube as much as possible, while still allowing the decoration operator to be applied.

Rule 12 (Pushing Federation Generalized Projection Below Decoration) *If the federation generalized projection operator above a decoration operator rolls up the decoration dimension to the decoration level, that is, if the level expression referring to the decoration level is a projection parameter, i.e.,* $l_z[SEM]/link/xp \in \mathcal{L}$*, the following holds:*

$$\Pi_{Fed[\mathcal{L}]<F(M)>}(\delta_{l_z[SEM]/link/xp}(\mathcal{F})) \leftrightarrow \Pi_{Fed[\mathcal{L}]<F(M)>}(\delta_{l_z[SEM]/link/xp}(\Pi_{Fed[\mathcal{L}']<F(M)>}(\mathcal{F})))$$

where $\mathcal{L}' = \{l | l \in \mathcal{L} \wedge l \notin D_z \wedge l \neq l_{xp}\} \cup \{MaxStrict(\{l_z\} \cup \{l | l \in \mathcal{L} \wedge l \in D_z\})\}$. $\mathcal{L}'$ *is identical to* $\mathcal{L}$ *except for one level from the dimension* D_z *containing the starting level* l_z. *The level is selected in such a way that further aggregation by the original projection must be allowed and the starting level used to build the decoration dimension must be present in the hierarchy; therefore, the destination level is a level below or equal to the starting level and does not introduce non-strictness over the bottom level in the new hierarchy.*

Proof sketch: The extra projection introduced on the right side, aggregates the cube to the same levels as the original projection, except for the dimension D_z which is rolled up to such a level that it is always possible to apply the decoration operator afterwards. Furthermore, $\mathcal{L}'$ is always possible to construct, because there is only a one-level difference from $\mathcal{L}$ and it could be the bottom level of D_z since a bottom level does not introduce non-strictness over itself. Note that although the rule is explained in the left to right direction it also holds in the opposite direction because of the way $\mathcal{L}'$ is constructed.

Example 5.6 $\Pi_{Fed[Customer,Brand,Nation[ANY]/Nlink/Population]<SUM(Quantity)>}(\delta_{Nation[ANY]/Nlink/Population}(\mathcal{F}_{TC}))$ is equivalent to $\Pi_{Fed[Customer,Brand,Nation[ANY]/Nlink/Population]<SUM(Quantity)>}(\delta_{Nation[ANY]/Nlink/\ Population}(\Pi_{Fed[Customer,Brand,Nation]<SUM(Quantity)>}(\mathcal{F}_{TC}))$. The new projection operator also has the levels Customer and Brand. But the level expression is replaced by Nation. To allow the decoration and the final roll-up to the top level, Nation is used as it is both the starting level of the level expression and the level that can be rolled up from without introducing non-strictness. Note, we assume that each supplier has only one country, thus, the relationship between Nation and Supplier is strict.

5.2.4 Rules Involving a Single Operator

In our federation system, similar to relational selections, a conjunctive federation selection can be split up into two selections and vice versa. For example, when a predicate is a conjunction of two predicates, where only one predicate references external XML data. In this case, the left to right transformation can be applied to split the conjunctive predicate, thereby enabling the selection with references only to dimension levels to be possibly evaluated in the OLAP component. The other direction of the rule can be used to combine federation selection operators, e.g., two federation selection operators with predicates applied with inlining.

Rule 13 (Cascade of Federation Selections) *Let* θ_1 *and* θ_2 *be predicates. Then the following holds:*

$$\sigma_{Fed[\theta_1 \wedge \theta_2]}(\mathcal{F}) \leftrightarrow \sigma_{Fed[\theta_1]}(\sigma_{Fed[\theta_2]}(\mathcal{F}))$$

Proof sketch: Selection only affects tuples in the fact table. Such a tuple satisfies the conjunctive predicate exactly when it satisfies the first predicate and then the second predicate.

Example 5.7 $\sigma_{Fed[Nation[ANY]/Nlink/Population=1017645163 \wedge Nation=\text{“}Denmark\text{”}]}(\mathcal{F}_T)$ is equivalent to $\sigma_{Fed[Nation[ANY]/\ Nlink/Population=1017645163]}(\sigma_{Fed[Nation=\text{“}Denmark\text{”}]}((\mathcal{F}_T))$.

As a deduction of Rule 13 and the relational algebra equivalence $\theta_1 \wedge \theta_2 = \theta_2 \wedge \theta_1$, federation selection operators also commute. The following rule can be used to swap two federation selection operators to enable other rules, e.g., Rule 10.

Rule 14 (Commutativity of Federation Selections) *Let* θ_1 *and* θ_2 *be predicates. Then the following holds:*

$$\sigma_{Fed[\theta_1]}(\sigma_{[\theta_2]}(\mathcal{F})) \leftrightarrow \sigma_{Fed[\theta_2]}(\sigma_{[\theta_1]}(\mathcal{F}))$$

Proof sketch: Follows from Rule 13 and commutativity of conjunction.

Example 5.8 $\sigma_{Fed[Nation[ANY]/Nlink/Population=1017645163]}(\ \sigma_{Fed[Nation=\text{“}India\text{”}]}(\mathcal{F}_T))$ is equivalent to $\sigma_{Fed[Nation=\ \text{“}India\text{”}]}(\sigma_{Fed[Nation[ANY]/Nlink/Population=1017645163]}(\mathcal{F}_T))$.

5.3 Query Cost Estimation

In this section, the cost model and the cost functions of physical operators will be introduced. The cost model which expresses generally the cost of evaluating an execution plan for a federation query. The cost functions determine the costs of the physical operators, which are dependent on the limited and varying cost information of the federation components. Moreover, the current cost functions only provide approximate estimations, but have been proved capable of supporting the choice of good plans effectively by the experiments. For example, the constant overhead of evaluating an OLAP component query is not taken into account, because we are not aiming to calculate the correct absolute cost of a plan, but only what is enough to differentiate the relative performance of candidate plans.

A Cost Model for Federation Query Plans Basically, a physical plan for a federation query and the evaluation algorithm suggest how the cost can be estimated. That is, the cost of a query plan is the cost of the root operator plus the maximal execution time of the sub-plans. However, to give an intuitive overview, we divide the total cost according to the three partitions of a typical physical plan, i.e., the time for inlining, OLAP query evaluation and data transfer, and producing the final result in the temporary component. For a query plan on which the inlining technique is applied, references to level expressions can be inlined into the selection predicates and therefore can be evaluated in the OLAP component; therefore, the first period of the execution time is spent on the inlining process, i.e., XML query evaluation, XML data transfer and predicate rewriting. The second period of the total time starts from query evaluation in the OLAP component until the data is transferred into the temporary component. However, for queries not inlining all the level expressions in the selection predicates, it is the time for the slowest retrieval of data from the OLAP and XML components. Finally, the sum of the previous two periods plus the time for producing the final result in the temporary component gives the total time. However, when expressed in a practical cost formula, the cost model consists of two parts, that is:

$$t_{Plan} = t_{inlining} + t_{OLAP,Temp}$$

where, $t_{OLAP,Temp}$ expresses the time for the last two periods as a whole. Because the evaluation algorithm executes the OLAP operators as a sub-plan in parallel with dimension- or XML-transfer operators, it is convenient to estimate the OLAP operators as part of the cost of a larger plan which may be rooted at an operator performed in the temporary component, i.e., a decoration, a federation selection, or a generalized projection operator. More specifically,

- $t_{Inlining}$ includes the maximal time for loading the required XML data into the temporary component and the time for inlining this XML data. The process is started by the XML-transfer operators that load the dimension values and referred XML data for the involved level expressions into the temporary component. Then, this data is loaded into main memory and integrated into the new predicate string. The two tasks are executed sequentially, thereby yielding $t_{Inlining}$ as the sum of the time spent on each phase.

- $t_{OLAP,Temp}$ is the time of evaluating the rest of the plan. The root operator for a (sub)plan can have several child operators which execute in parallel; therefore, the cost of this (sub)plan is the time for evaluating the root operator plus the maximum time for evaluating the sub-branches. Let Op_i be the root operator of a query tree, $Op_{i1}, \ldots, Op_{in}$ be the child operators and t_{Op_i} be the time for evaluating a single operator, then $t_{OLAP,Temp}$ for the query tree rooted at Op_i is: $t_{Overall,Op_i} = t_{Op_i} + MAX(t_{Overall,Op_{i1}}, \ldots, t_{Overall,Op_{in}})$. If $Op_i = \phi$, then $t_{Overall,\phi} = t_{OLAP}$, where t_{OLAP} (see below) is the time for retrieving the temporary fact data, i.e., evaluating the cube operators and the fact-transfer operator.

How the cost of each component is evaluated in the cost model is introduced in more detail below.

Estimating $t_{\sigma_{Fed}}$ As defined by Definition 3.7, the federation selection operator σ_{Fed} is implemented by relational operators, that is, it is a regular SQL selection if the fact table contains all the levels and measures referenced in the predicates, otherwise the tables containing the values of the referenced attributes are joined, selected and finally projected again to retain only the attributes from the fact table; therefore the cost of a federation selection operator is also estimated through the SQL operations. That is, $t_{\sigma_{Fed[\theta]}(\mathcal{F})} =$

1. $t_{\sigma_\theta(F)}$, if $S_\theta \subseteq attrs(F)$.
2. $t_{\pi_{attrs(F)}(\sigma_\theta(F \bowtie T_1 \bowtie \ldots \bowtie T_n)}$, if $\exists l \in S_{\theta_i}(l \in attrs(T_i) \wedge l \notin attrs(F))$.

Here, θ, pi, $\bowtie$ are SQL selection, projection and inner natural join respectively, F is the fact table, S_θ is a the set of the attributes referred by θ, and $attrs(T)$ is a function that returns the attributes in T. Traces of the execution plans have shown that the tables are accessed through table scans and the join methods are usually hash joins. Here, the temporary fact table is the probe table of the multi-table hash join, since it has all the common columns and it is usually larger than the other tables which are hashed and later probed by the fact table.

Based on the methods in [8, 12, 40] and traces of executions in the temporary component, we approximate the costs in the following manner. The cost of the relational operator σ_θ is estimated as: $t_{\sigma_\theta} = \frac{b_F}{ScanningRate} + \frac{selectivity(\theta,F)\times|F|}{f_F \times WritingRate}$, where b_F is the number of blocks containing tuples of F, f_F is the blocking factor of F, i.e., the number of tuples of F that fit into one block, $|F|$ is the cardinality of F and $selectivity(\theta, F)$ is the predicate selectivity of θ against F. Moreover, $ScanningRate$ and $WritingRate$ are the constant factors for the speed of reading and writing blocks of tuples from/to secondary memory. The first part of the cost function is the time for reading the tuples of F into memory and the second part is the time for writing the result to disk. For the second case, where the temporary fact table is hash joined with other temporary tables as in-memory operations, the cost is estimated as: $t_{\pi_{attrs(F)}(\sigma_\theta(F \bowtie T_1 \bowtie \ldots \bowtie T_n))} = \frac{b_F + b_{T_1} + \ldots + b_{T_n}}{ScanningRate} + t_{hash_{T_1}} + \ldots + t_{hash_{T_n}} + n \times t_{hash_F} + \frac{selectivity(\theta,F)\times|F|}{f_F \times WritingRate}$, where b_{T_i} is the number of blocks of table T_i and $t_{hash_{T_i}}$ is the time for creating the hash table for the tuples in T_i, i.e., suppose $HashCostPerTuple$ is a constant factor representing the time for hashing a tuple, then hashing T_i costs $t_{hash_{T_i}} = HashCostPerTuple \times |T_i|$. Moreover, $n \times t_{hash_F}$ is the time for probing the hash tables for $T_1, \ldots, T_n$ using the tuples from F. The last part is still the same as in the cost function for the first case, because the cardinality of the resulting table is decided by the temporary fact table. More specifically, the temporary fact table F contains all the join attributes (which are also the bottom levels containing distinct dimension values) for the inner joins, and the temporary tables $T_1, \ldots, T_n$ are assumed to have strict hierarchies, thus, the join result has the same number of tuples as the temporary fact table.

Estimating $t_{\Pi_{Fed}}$ A federation generalized projection operator is a regular SQL projection and aggregation if the fact table contains all the levels and measures referenced in the SELECT clause, otherwise the tables containing the specified levels in the SELECT and GROUP BY clause is first needed to be joined with the fact table. $t_{\Pi_{Fed[\mathcal{L}]<F(M)>}(\mathcal{F})} =$

1. $t_{\mathcal{L}\mathcal{G}_{F<M>}(F)}$, if $\mathcal{L} \subseteq attrs(F)$.
2. $t_{\mathcal{L}\mathcal{G}_{F<M>}(F \bowtie T_1 \bowtie \ldots \bowtie T_n)}$, if $\exists l \in \mathcal{L}(l \in attrs(T_i) \wedge l \notin (attrs(F))$.

Here $\mathcal{G}$ is the relational aggregation operator. Similar to the federation selection operator, the tables are also assumed to be hash-joined and accessed through table scan.

The cost of the first case is: $t_{\mathcal{L}\mathcal{G}_{F<M>}(F)} = \frac{b_F}{ScanningRate} + t_{hash_F} + \frac{numDistinct(F)}{f_F \times WritingRate}$, where t_{hash_F} is the time for building the in-memory hash table on the grouping attributes, and $numDistinct(F)$ is used to return the number of distinct tuples of F because after aggregation there is only one tuple for each group.

For the second case, the cost is estimated as: $t_{\mathcal{L}\mathcal{G}_{F<M>}(F\bowtie T_1\bowtie\ldots\bowtie T_n)} = \frac{b_F+b_{T_1}+\ldots+b_{T_n}}{ScanningRate} + t_{hash_{T_1}} + \ldots + t_{hash_{T_n}} + n \times t_{hash_F} + k \times b_F \times log_2 b_F + \frac{numDistinct(F)}{f_F \times WritingRate}$, where, in addition to the cost of multi-table hash join and writing the aggregation result to disk, $k \times b_F \times log_2 b_f$ is the cost of sorting the join result on the grouping attributes according to [8] and k is a constant factor.

Estimating t_ω The dimension-transfer operator ω copies dimension values of specified levels from the OLAP to the temporary component; therefore, the time it takes is the size of the dimension values divided by a constant fact transfer rate. That is,

$$t_{\omega_{[l_{ix},l_{iy}]}} = \frac{size(R_{\omega_{[l_{ix},l_{iy}]}})}{FactTransRate}$$

where, the function $size(R)$ returns the table size and the constant $FactTransRate$ is the rate with which data can be transferred from the OLAP component to the temporary component.

Estimating $t_{Inlining}$ As discussed above, $t_{Inlining}$ is the total time spent on XML data retrieving and predicate rewriting, i.e., $t_{Inlining} = MAX(t_{\tau_{l_j/link_1/xp_1}}, \ldots, t_{\tau_{l_k/link_m/xp_m}}) + t_{\iota_{[\theta_1,\ldots,\theta_n]}}$, where $t_{\tau_{l_j/link_1/xp_1}}, \ldots, t_{\tau_{l_k/link_m/xp_m}}$ represent the time spent on each child operator of $\iota_{[\theta_1,\ldots,\theta_n]}$ (see below), and $l_j/link_1/xp_1, \ldots, l_k/link_m/xp_m$ are the level expressions referenced by $\theta_1, \ldots, \theta_n$. To rewrite a predicate, the inlining operator ι first loads the tables $R_{\tau_{l_j/link_1/xp_1}}, \ldots, R_{\tau_{l_k/link_m/xp_m}}$ which are the temporary tables built by the child XML-transfer operators into the main memory, then generates the new predicate string using the values. Considering that the in-memory string processing is trivial compared with the expensive I/O readings and writings, the cost of an inlining operator is mainly composed of the time for loading the XML data. The cost function for the inlining operator is: $t_{\iota_{[\theta_1,\ldots,\theta_n]}} = \frac{b_{R_{\tau_{l_j/link_1/xp_1}}}}{ScanningRate} + \ldots + \frac{b_{R_{\tau_{l_k/link_m/xp_m}}}}{ScanningRate}$ where b_{R_i} is the number of blocks containing tuples of the table R_i and $ScanningRate$ is the constant speed for loading records from a temporary table into the main memory. The cost function implicates that the more level expressions inlined, the more time the rewriting process takes.

Estimating t_τ The cost function for XML-transfer operators is dependent on the XML access method. Currently, the XML component query engine maps XML attributes to table columns through a schema definition; therefore, an XML document is represented as a table, and can be selected by the SQL SELECT statement, which then can be incorporated into the INSERT INTO statement to transfer the XML data into the temporary component. The cost function for the XML-transfer operators comprises the time to perform table scans over the XML documents and the time for the table insertion. That is, $t_{\tau_{l_z/link/xp}} = \frac{size(X_1)+\ldots+size(X_n)}{XMLScanRate} + \frac{size(X_1)+\ldots+size(X_n)}{RecordInsertionRate}$ where $X_1, \ldots, X_n$ are the referenced XML documents in $link$, $size(X_i)$ returns the size of the XML document X_i, $XMLScanRate$ is the rate of scanning the XML documents and $RecordInsertionRate$ is the rate of inserting the scanned data into a temporary table.

Estimating t_{OLAP} The fact data is retrieved in two steps: first, the OLAP component query is evaluated, then the data is transferred from the OLAP to the temporary component. Here, only the last factor is considered since evaluations of SQL$_{XM}$ queries have shown that it is mainly the transfer time that differs the performance in retrieving fact data using different OLAP queries; therefore, the cost is the size of the cube after it is aggregated and selected divided by a constant fact transfer rate. That is, $t_{OLAP} = \frac{size(F)\times selectivity(\theta,C)\times rollupFraction(\mathcal{L},C)}{FactTransferRate}$, where, C represents the cube, F is the fact table, θ is the predicate used to select the cube and $\mathcal{L}$ represents the specified levels to which the cube is rolled up. Moreover, $size(F)$ returns the size of the cube, which is approximately the size of the fact table since the fact data

takes most of the space of a cube. $Selectivity(\theta, C)$ returns the fraction of the total size of C that is selected by θ. This is estimated using the standard methods from [8]. The function $rollupFraction(\mathcal{L}, C)$ returns the fraction to which C is reduced in size, when it is rolled up to the levels $\mathcal{L}$. According to the method proposed by [39], where the facts are assumed to be uniformly distributed in the cube, the fraction is the number of the distinct facts after the cube is rolled up to $\mathcal{L}$ divided by the number of the original facts.

Estimating Relational Operator Cost and Statistical Information The temporary component is used as a scratch-pad by the OLAP-XML query engine which is assumed to have full access to its meta data, such as cardinalities, attribute domains and histograms; therefore, existing query optimization techniques can be adopted to provide statistical information and cost estimates. More specifically, we use the simplified variants of the methods from [6, 8, 40] to estimate, e.g., selectivities and cardinalities of physical operators. In the current implementation, the statistical information, e.g., $FactTransRate$, $ScanningRate$ and $WritingRate$, are retrieved by *probing queries* [48] and stored as constants in meta data. For example, to approximate $FactTransRate$, a simple query to retrieve the fact table can be used. The transfer time is measured from the moment query is sent until the temporary fact table is built; therefore, $FactTransRate$ is the size of the fact table divided by the transfer time. In our future implementations, the statistical information will be collected by a *Statistics Manager* dynamically.

5.4 Implementing Query Optimization

oset $rewrite$(operator op)
1) {
2) oset $plans_{op}$ = $\emptyset$;
3) operator op_{child} = the child of op;
// *rewrite the sub-plan of* op *first*
4) oset $plans_{child} = rewrite(op_{child})$;
5) **for each** plan with root op'_{child} in $plans_{child}$
6) construct a new plan with root op' by putting
a copy of op on top of op'_{child};
// *generate the plan space for the new plan*
7) oset $plans_{tmp} = matchAndApplyRules(op')$;
// *add the new plans into the plan space for* op
8) merge $plans_{tmp}$ into $plans_{op}$;
9) $prune(plans_{op})$;
10) **return** $plans_{op}$;
11) }

Figure 14: The *rewrite* function

Similar to the Volcano optimizer, the plan space for a given plan is represented by a number of equivalence classes. The elements of an equivalence class are the logical operators that generate the same federations in different plans. For example, the three operators in the initial logical plan in Figure 7 are categorized into three classes 1, 2 and 3 in Figure 16. More specifically, class 1 initially contains the decoration operator $\delta_{Nation[ANY]/Nlink/Population}$ (in short, δ), class 2 contains the federation selection operator $\sigma_{Fed[Nation[ANY]/Nlink/Population]}$ (in short, σ_{Fed}) and class 3 contains the federation generalized projection operator $\Pi_{Fed[Brand(Part),Nation[ANY]/Nlink/Population]<SUM(\ Quantity)>}$ (in short, Π_{Fed}). The operators in classes 1 and 2 generate partial results of the query, whereas class 3 contains the root operators

of the equivalent plans that generate the same final result federations. Note that to compress the plan space, no duplicate plans are allowed, and to prevent plans having redundant operators, no operator is allowed to point to another operator from the same equivalence class (see below for more discussion).

```
oset matchAndApplyRules(operator op)
1)   {
2)     oset plans_tmp = ∅;
3)     oset plans_op = ∅;
4)     for each rule_i
5)       if rule_i is applicable
           // this may generate new operators and classes
6)         apply rule_i;
           // op' is the root operator of the new plan
7)         plans_tmp = rewrite(op');
           // add the new plans into the plan space for op
8)         merge plans_tmp into plans_op;
9)     return plans_op;
10)  }
```

Figure 15: The *matchAndApplyRules* function

Figures 14 and 15 outline the pseudo-code with C-like "// " for comments for the two main functions for the plan space generation, $rewrite$ and $matchAndApplyRules$. Ordered sets (osets) are used to hold plans and preserve orders, for example, the first element in an ordered set is always the original plan and the others are the enumerated equivalent plans. Initially, the operators from the original initial plan are created in the plan space for different classes, and then $rewrite$ is invoked on the root operator. The $rewrite$ function recursively invokes itself on the child operator of the current input operator, meaning the plan space for the current plan is always based on the returned equivalent plans for the lower operator. When the bottom of the plan is reached, transformation rules are applied using the function $matchAndApplyRules$, which may yield new alternative operators generating the same federation. After this, new plans can be constructed by putting the operator from a higher class of the original plan on top of the operators, including the new ones, in the current class. Then the transformation rules are applied again on the new plans. The $matchAndApplyRules$ function goes through all the rules, uses the rule condition to identify an applicable rule and then performs the transformation. This may trigger the creation of new operators and new classes. For each of the new plans that contains these newly generated operators, $rewrite$ is invoked to generate other equivalent plans. Note that the new operators must not violate the two restrictions on the plan space, otherwise the corresponding rule is not applicable.

We first show how all the equivalent plans are generated for an initial logical plan in Figure 7 without plan pruning. Figure 16 shows the plan space, which is generated as follows. Initially, three operators representing the three query tree operators are created, with arrows going from the parent to the child operators. Then, the $rewrite$ function is invoked for the first elment of class 3, which, in turn, invokes $rewrite$ for the first element in class 2, and then again for the first element in class 1. The last call does not do anything because no rules apply to δ on its own. For the first element in class 2, however, Rule 10 is applied, which switches the decoration and the federation selection operators and marks the selection predicate for inlining at execution time; therefore, two new operators are generated, one of which is added as the second element of class 2 and the other as the second element of class 1. Then as Line 7 indicates, $rewrite$ is invoked on the new operator δ in class 2, but for this sup-plan, no applicable rules are found, thereby yielding no new operators. After this, the $rewrite$ function for the first element σ_{Fed} in class 2

returns $plans_{child} = (\sigma_{Fed}(\delta), \delta(\sigma'_{Fed}))$, $matchAndApplyRules$ is then invoked for Π_{Fed} in class 3 to generate equivalent plans for the initial plan, $\Pi_{Fed}(\sigma_{Fed}(\delta))$, and $\Pi_{Fed}(\delta(\sigma'_{Fed}))$ which is constructed by putting Π_{Fed} on top of the new decoration operator in class 2. Note that the dotted line between two operators means the connected operators in different plans have the identical parent operator. For the first plan, Rule 7 is applied on Π_{Fed} and σ_{Fed} to switch their positions, which adds σ_{Fed} to class 3 and Π_{Fed} to class 2. The bottom decoration operator δ remains the same. According to Line 7 , $rewrite$ is invoked again for the new operator σ_{Fed} in class 3. But the next rule applied is Rule 12 on Π_{Fed} in class 2 and a new federation generalized projection operator $\Pi_{Fed[Brand(Part),\ Nation]}$ (in short, Π'_{Fed}) is generated in a new class, class 4, below class 1. Moreover, the new operators also include the new Π_{Fed} in class 2 and δ in

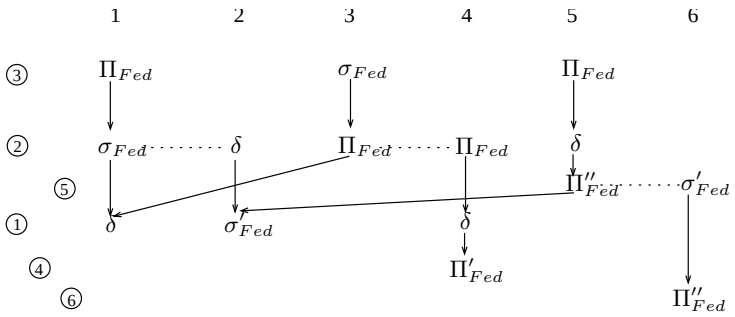


Figure 16: A logical query plan and its plan space

class 1. After this, Line 7 invokes $rewrite$ for Π_{Fed} but this time no applicable rules are found and Rule 12 cannot be applied again because no operators in the same class are allowed to be connected. Moreover, Rule 7 cannot be applied on $\sigma_{Fed}(\Pi_{Fed})$ again, because an identical plan is already present. After the $matchAndApplyRules$ function for the first plan returns $plans_{child}$ containing plans 1, 3 and 4, two more plans, plans 3 and 4, are added into the plan space $plans_{op}$. The $rewrite$ function continues to apply rules on $\Pi_{Fed}(\delta(\sigma'_{Fed}))$. This time, Rule 12 is applied to generate another federation generalized projection $\Pi_{Fed[Brand(Part),Nation]}$ (in short, Π''_{Fed}) below δ, which is added to a new class, class 5. Then, Rule 7 is applied on Π''_{Fed} and the lower σ_{Fed}, which switches the positions and yields a new operator σ'_{Fed} in class 5 and a new class, class 6, with only one operator Π''_{Fed}. Like before, Rules 12 and 7 cannot be applied on the newly generated operators again because of the two restrictions. After matching and applying rules for plan 2 returns, $plans_{op}$ contains two more plans, i.e., plans 5 and 6. At this point, the full plan space is generated.

```
oset prune(oset plans)
1)  {
2)    for each plan plan_i in plans
3)        estimate the cost c_i of the corresponding physical plan
          for plan_i;
4)    for each plan plan_i in plans
          // the first plan in the list is the original plan
5)        if  c_i > c_0
6)          remove plan_i from plans;
7)    return plans;
8)  }
```

Figure 17: Pseudo-code for pruning the plan space

Figure 17 briefly shows the cost-based pruning process for an initial plan and its corresponding plan space. The idea is to reduce the plan space by excluding the generated equivalent plans that cannot give less execution time than the original plan. The $prune$ function first estimates the costs of all the plans. This is achieved by first generating the physical plans based on the logical plans and then estimating the cost with the cost model and functions. Lines 4-6 use the cost of the first plan in the plan space (i.e. the original plan) as an upper bound. All the plans that have larger costs are removed from the plan space. For example, during the plan space generation for the first plan in Figure 16, if the pruning method is not used, all the listed plans are included in the final plan space. However, if in addition to the general reasoning that plans filtering and aggregating the cube before the cube is decorated lead to better performance, we assume $\sigma_{Fed}(\Pi_{Fed})$ is more expensive than $\Pi_{Fed}(\sigma_{Fed})$ when they have the same lower operators, and $\sigma'_{Fed}(\Pi''_{Fed})$ is more expensive than $\Pi''_{Fed}(\sigma'_{Fed})$, then plans 3 and 6 would be excluded from the plan space when the $prune$ function is used.

When all the plans are generated and costs are estimated, the optimizer selects the logical plan and its corresponding physical plan with the least cost as the final execution plan and passes the execution plan to query evaluator. For example, plan 5 in Figure 16 filters and aggregates the cube to the largest possible extent before the cube is decorated, therefore the final execution plan is the physical plan for plan 5.

6 Performance Study

The experiments were performed on a machine with an Intel Pentium III 800Mhz CPU, 512MB of RAM, 30 GB of disk and 4096MB of page file. The OS is Microsoft Windows 2000 server with SP4. In the prototype, Microsoft SQL Server 2000 Enterprise Edition with SP3 is used. More specifically, the temporary component is the temporary database on SQL Server, and the OLAP component uses MS Analysis Services, and is queried with SQL [24]. The XML component is the local file system based on the XML data retrieved from the Web with MS SQLXML [26] on top. The example cube, TC, used in the experiments is shown in Figure 1 . The cube is based on about 100MB of data generated using the TPC-H benchmark [42] .

In the following experiments, the query engine and the federation are observed w.r.t. three aspects. First, the query evaluation performance of the query engine when queries of different complexities are posed. Second, the effectiveness of the optimization techniques. For this aspect, comparisons are made between the evaluations of a) straightforward and b) optimized execution plans. Third, the feasibility of the federation system. There, the external data is put in different components, i.e., XML, OLAP, and the local relational database, which, at query execution time, are *federated*, *integrated* and *cached* external data, respectively. Comparisons are made for the queries referencing this data.

To reveal the behavior patterns of the federation and the SQL$_{XM}$ query engine, several choices have been made on the experimental environment. The selected storage mode is ROLAP for the cubes, which are queried with the federated or integrated XML data. One of the reasons to do so is that this mode is most commonly used for large data warehouses. Secondly, for queries on ROLAP cubes the execution plans can be analyzed (unlike for MOLAP cubes). Thirdly, our initial experiments with MOLAP cubes have shown that the relative performance of queries over federated versus integrated cubes is the same whether ROLAP or MOLAP is used. Moreover, our experiences with pre-aggregated cubes have shown that when pre-computed tables are only used in the evaluation of some queries, the behavioral pattern of the query engine itself is hard to see; therefore, no aggregates are pre-computed so as to see the general behavior of the query engine and performance with the least interference. Note that this does not alter the balance between querying the OLAP component with federated and integrated data. For the same purpose, in all the experiments, caches existing in the OS and SQL Server are all cleared after each run of the query engine so as not to affect the next execution.

type	no_of_dim	dim1	dim2	dim1level	dim2level
1	1	Suppliers		Nation	
2	1	Suppliers		Supplier	
3	1	Parts		Brand	
4	1	Parts		Part	
5	2	Suppliers	Parts	Nation	Brand
6	2	Suppliers	Parts	Supplier	Brand
7	2	Suppliers	Parts	Nation	Part
8	2	Suppliers	Parts	Supplier	Part
9	2	Suppliers	Orders	Nation	Customer
10	2	Suppliers	Orders	Supplier	Customer
11	2	Suppliers	Orders	Nation	Order
12	2	Suppliers	Orders	Supplier	Order
13	2	Parts	Orders	Brand	Customer
14	2	Parts	Orders	Part	Customer
15	2	Parts	Orders	Brand	Order
16	2	Parts	Orders	Part	Order

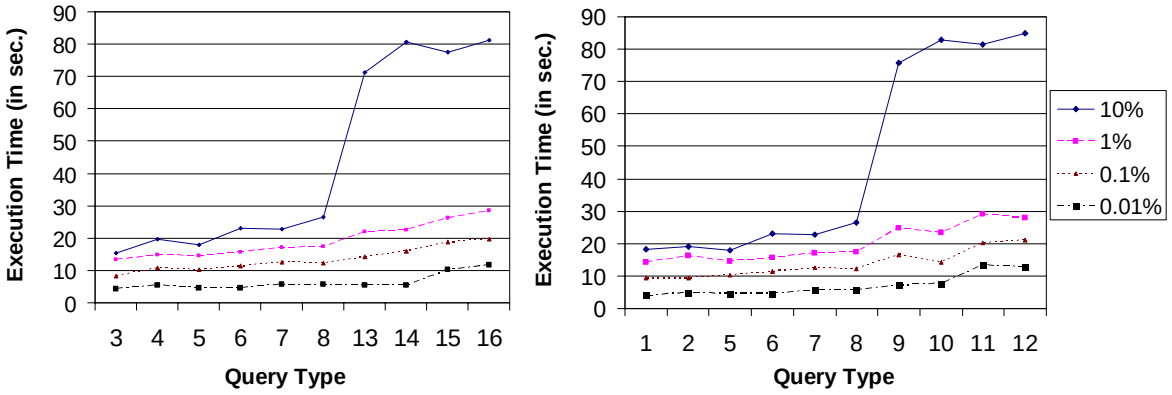
Table 5: Query types and their attribute values

6.1 Query Evaluation Performance

To study the behavior of the query engine, four groups of queries of different types and selectivities were evaluated. Groups 1 to 4 have selectivities of 0.01%, 0.1%, 1% and 10%, respectively. Each group has sixteen types of queries. Each query type is formed with the attributes: *no_of_dim* for the number of dimensions mentioned in the SELECT clause with a maximum value of two, *dim1* and *dim2* for the dimension names, and *dim1level* and *dim2level* for the argument levels in the SELECT clause. The query types are enumerated and numbered according to the sizes of the participating dimensions. In the TC cube, the dimensions participating in the queries are Suppliers, Parts and Orders, in ascending order by size. For example, query type 1 selects the middle level of the smallest dimension with the least number of values, and query type 2 selects the bottom level of the same dimension. Query type 3 selects the middle level of the second-smallest (smallest not used) dimension, Parts. Likewise, the types having multiple dimensions are also sorted with combinations of smaller dimensions before larger ones; thus, the query type represents the expected complexity. Table 5 below shows the different attributes of each query type.

The line charts in Figure 18 illustrate execution performance of queries with different selectivities. We fix one participating dimension in each chart to show how the performance is affected by the query types. Moreover, the query types all start from one of the small dimensions, Parts or Suppliers, so that the performance is more sensitive to size changes. The X axis represents the query type, and the Y axis represents the execution time in seconds. Each line represents the queries of one particular selectivity. The top-most line represents the execution of the queries with selectivity of 10%, followed by the lines for selectivities of 1%, 0.1%, and 0.01%, in the top-down direction. This indicates the more selective a query is, the less execution time it needs. More specifically, a less selective federation query takes more time in evaluating the OLAP component query and transfer of the intermediate data. Consequently, the following step for performing the relational operations in the temporary database also take more time.

In each chart of Figure 18, there is a distinct leap for the queries of selectivity 10% when the largest dimension, Orders, is involved. That is, along the top line, the query types larger than 10 takes around 40 seconds more than the smaller ones. For these queries, a couple of predicates on a higher dimension level are joined by "OR" to provide the selectivity, which causes different component execution plans from those of the same type of queries with different selectivities. Moreover, as [25] and the execution plans point out, the result set is just a union of the results of the OLAP queries using the predicates individually in the



(a) Types involving Parts (b) Types involving Suppliers

Figure 18: Line charts for query execution time of different selectivities

WHERE clause, and thereby causing the same execution plan to be executed repeatedly as many times as the number of the ORed predicates and yet another aggregation in the temporary component to coalesce the multiple rows for the duplicate grouping values; therefore, several factors, i.e., repeated executions with joins on large participating tables (even after filtering), more inter-component data transfer time and the temporary coalescing for the larger return set, cause the leap.

Nevertheless, in most of the times, the execution time grows slightly when the type number is increased. The query types are organized in such a way that queries with larger type numbers tend to return more data than the smaller ones. For example, query type 1 rolls up the smallest dimension to the middle level and removes all the other dimensions, whereas query type 2 rolls up to a relatively lower level, which leads to a larger cube for the result values. The dimension Orders is much larger than the other two dimensions, therefore, queries leaving this dimension in the cube yield a lot more data than the others and take a lot more execution time. In summary, the more the cube is reduced by selection and aggregation, the better the query performs.

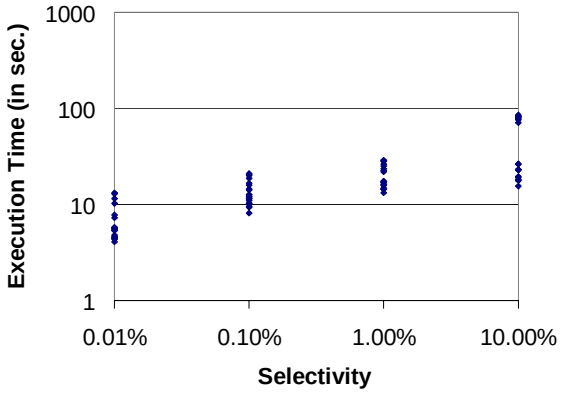


Figure 19: Scatter chart for all queries

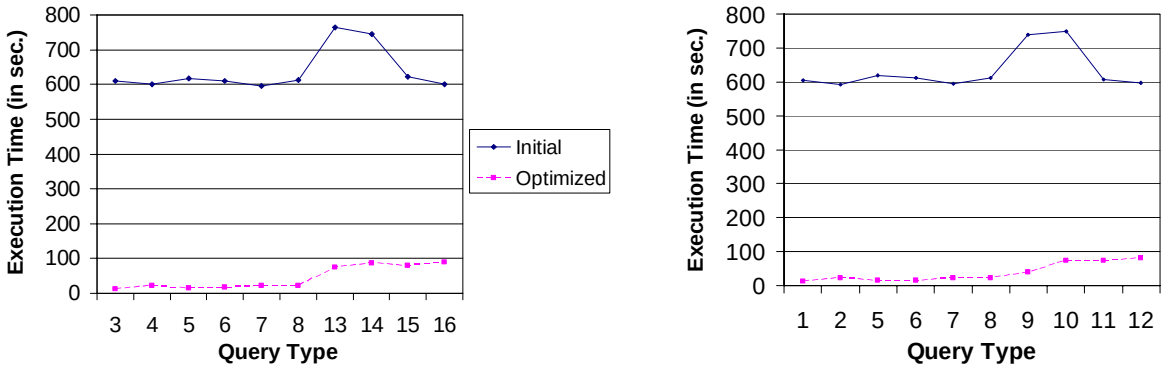
Figure 19 shows a scatter chart for the execution of all queries. The X axis represents the values of the selectivities, and the Y axis stands for the execution time in seconds. Both axes are in logarithmic scale. For each selectivity, there are sixteen data points on the column representing all the query types in Table 5. From the chart, we can see more selective queries tend to take less execution time. And as the selectivity increases, the execution time tends to grow more or less linearly. There are points, however, higher than those with

larger selectivities, because the cube after selection is not sufficiently aggregated and still contains a large amount of dimension values of bottom levels. This leads to more data transfer and component operation time. The chart depicts a conclusion consistent with the previous figures, that is, the selective queries are generally faster.

6.2 Query Optimization Effectiveness

To study the effectiveness of the optimization techniques, comparisons were made for queries evaluated on the query engines with and without optimizations. The same queries as in Table 5 were used except that the WHERE clause now refers to external XML data and has a selectivity of 10%. The initial plans of these queries require selections to be performed over the OLAP data in the temporary component, therefore it is interesting to see how the optimization can affect the final execution plan. An example query is:

```
SELECT    SUM(Quantity),Brand(Part),Supplier
FROM      TC
WHERE     Nation/Nlink/Population=45860000 OR
          Nation/Nlink/Population=59128187 OR
          Nation/Nlink/Population=31787647
GROUP BY  Brand(Part),Supplier
```

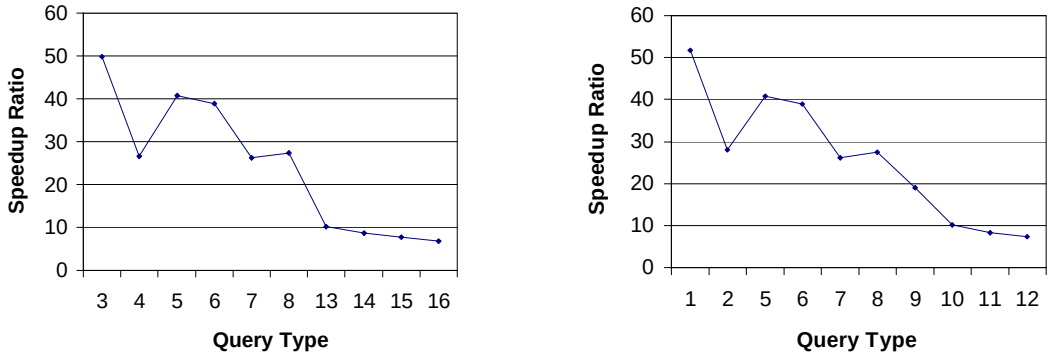


(a) Types involving Parts

(b) Types involving Suppliers

Figure 20: Charts for comparisons between straightforward and optimized executions

Figure 20 presents the executions of the initial plans (straightforward executions) and the optimized plans (optimized executions). The initial plan first performs the data transfers from the OLAP, and XML component to the temporary database, copying the fact table with the columns for all the measures and bottom levels, the dimension values for Brand and Part, and the XML values, which is then followed by a regular SQL join of the temporary tables with the selection predicates on XML values. The top lines in Figures 20(a) and 20(b) represent the cost of the straightforward executions, which consists of the fixed overhead for copying the data and the time for performing SQL operations in the temporary database. The spikes for queries 9, 10, 13, and 14 indicate that, as 10,000 customers have 150,000 orders, it is expensive to perform the roll-up from the bottom level Order to the middle level Customers by joining the fact table and the temporary roll-up table for the two levels. The bottom lines in the chart represent the executions of the optimized plans. The inlining technique rewrites the predicates so that only OLAP data is referenced. The transformation rules push the projection and the modified selections below the decoration so that selections and aggregations can be applied on the cube before OLAP data is transferred. Decoration of the cube is



(a) Types involving Parts (b) Types involving Suppliers

Figure 21: Speedup ratios of execution speed between straightforward and optimized executions

then no longer needed as the SQL_{XM} query does not require the XML data to be present together with the cube values. The line follows the trend described in Section 6.1, that is, selections and aggregations in the OLAP component boost the query performance.

The charts in Figure 21 show the *speedup ratio* of query execution when the optimized plan is used. However, not all the executions are sped up to the same extent. The lines indicate that optimized plans are executed seven to fifty times faster than the straightforward initial plans. In each chart, the line drops as the complexity increases. The last four queries are not optimized as effectively as the other queries, as these queries involve the largest dimension and return larger result sets. The most effective optimization takes place for the first query, as the query aggregates the smallest dimension to the middle level and the other dimensions to the top, therefore yielding a small result set. Note that the different aggregation algorithms for the optimized and straightforward executions lead to opposite evaluation performance. For example, in Figure 20(a), when the naive plans are executed, query 3 takes more time than query 4 which, on the contrary, needs more execution time when both queries are optimized. The optimized plans aggregate the cube in the OLAP component, whereas the naive plans lead to joins in the temporary component on the fetched OLAP data for roll-ups to higher levels. In need of more data to be transferred and joined, the second takes more time, therefore the algorithms have opposite effects. In summary, the experiments suggest that the more the cube is reduced and aggregated in the SQL_{XM} query, the more effective the optimization is.

The time for the optimizer to generate the final execution plan normally varies from less than 1 second to 2 seconds. From the above discussion, we know the executions take approximately 600-700 seconds for non-optimized plans and 5-30 seconds for the optimized ones; therefore, the query optimization time is trivial compared to the speed up and even the execution time for the optimized plans. For this reason, the optimizer performance is not a focus in this report. However, the optimization takes several seconds more when a large amount of new XML data (no statistical information ever recorded) is involved in a query and the statistical information of that data is not present in the meta data. For example, when 11.4MB XML data was used (see Section 6.3), it took four seconds more. In that case, the optimizer has to scan the XML documents for statistics, e.g., cardinality and node size. But, again, the optimization time is still fairly small compared to the plan execution time, because the larger volume of external data increases the execution time as well (about 140 seconds). In future work, the optimizer will be tuned to be faster by, e.g., using faster XML access methods, code optimization, etc..

Optimization Case Since the optimized plans are faster, it is interesting to see what happens during the process of plan rewriting. Motivated by this, we now demonstrate a case study, where the initial query plan and the equivalent plans optimized to different levels are executed to see the varying performance. The

SQL$_{XM}$ query is:

SELECT	SUM(Quantity),Brand(Part),Supplier,
	Nation/NLink/Population
FROM	TC
WHERE	Nation/NLink/Population<25
GROUP BY	Brand(Part),Supplier,Nation/NLink/Population

The first plan is the initial logical plan shown in Figure 22. The last plan is an optimized plan shown in Figure 24, which yields the physical plan with the least execution cost. The second logical plan is the intermediate plan generated during the optimization process. In the plans, $B(P)$, S, $N/Nl/P$ and Q represent, the roll-up expression Brand(Part), the bottom level Supplier, the level expression Nation/Nlink/Population (the default decoration semantic modifier is ANY) and the measure Quantity, respectively.

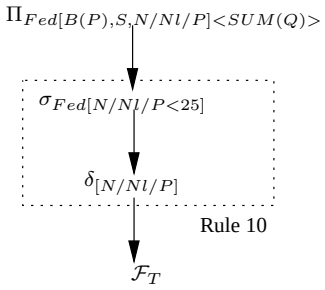


Figure 22: Initial plan

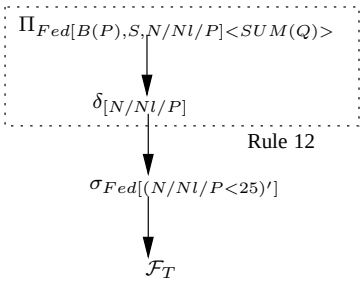


Figure 23: 2nd plan

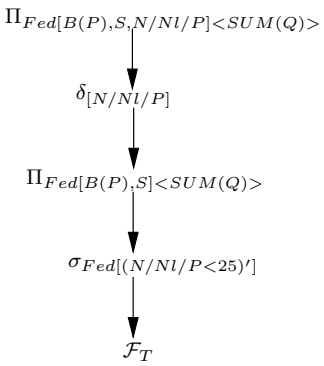


Figure 24: 3nd plan

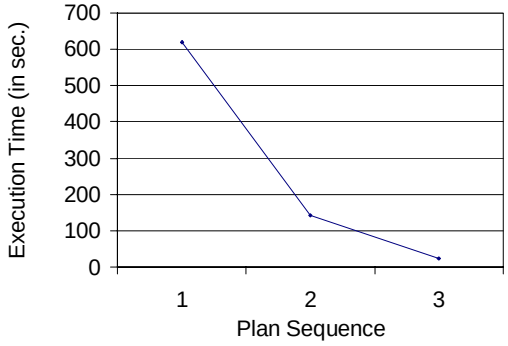


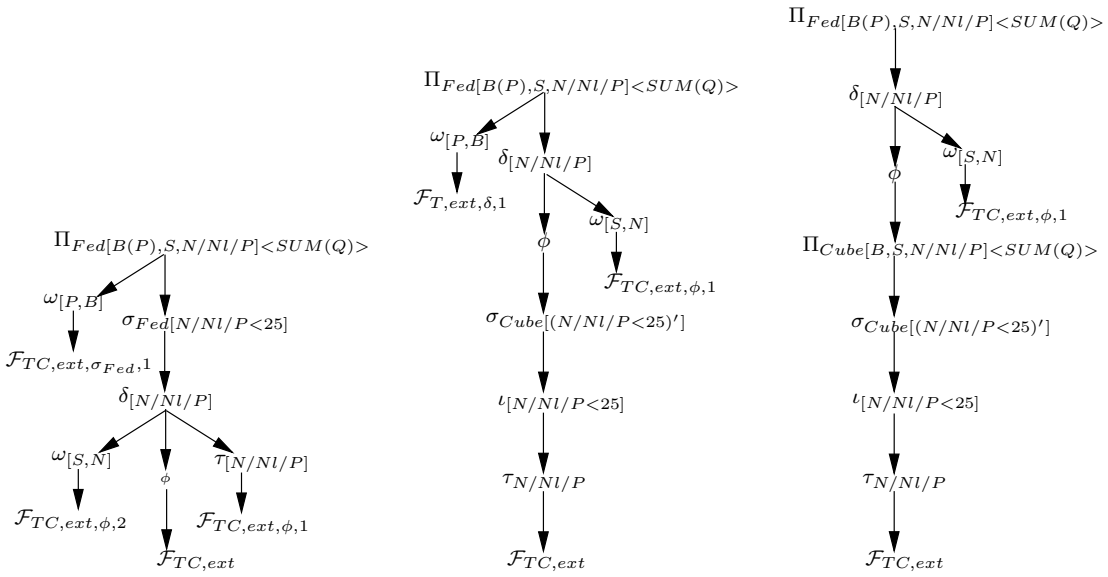
Figure 25: Execution time for plans during optimization

Figure 25 shows the execution time for each plan. The Y axis represents the time in seconds. The X axis shows the plan sequence. The initial plan models the execution process in a straightforward way, yielding the physical execution plan in Figure 26(a). The fact table containing values for all the measures and bottom levels is first copied into the temporary component. The cube is decorated by the decoration operator building a new dimension consisting of the XML values. Then selection is performed over the join of the fact table and the decoration dimension table. The following federation generalized projection removes the unspecified levels in the fact table and rolls up the cube by joining the temporary dimension tables for Brand and Part with the fact table. Finally, the regular SQL aggregation operator is performed on

the fact table. The execution time was 620 seconds for the initial plan, shown by the first point from the left in Figure 25.

The first rule applied is Rule 10 (Inlining of Decoration in Federation Selection), which yields the second plan in Figure 23. The predicate $N/Nl/P < 25$ is marked to be rewritten and no longer refers to the level expression at execution time. The federation selection can then be pushed down and executed directly in the OLAP component. The corresponding physical plan is shown in Figure 26(b). The execution starts from the bottom XML-transfer operator loading the dimension values and the decoration values for the above inlining operator, which uses the linked data to rewrite the predicate. The decoration and generalized projection are performed in the temporary component after the filtered fact table is loaded. As above, the second point from the left in Figure 25 stands for 143 seconds for executing the second plan.

Rule 12 (Pushing Federation Generalized Projection Below Decoration) is applied on the second plan, which pushes a part of the federation generalized projection below the decoration. The result plan in Figure 24 performs the selection and partial aggregation on the federation before the cube is decorated. The top federation generalized projection still performs the same task, rolling up the dimensions to the specified levels and aggregating the desired measures. The corresponding physical plan is shown in Figure 26(c). The plan is the same as the previous one except that it now contains a cube generalized projection which rolls up the cube to the maximum possible levels but still allows the decoration later on. As above, the third point from the left in Figure 25 stands for 22 seconds for executing the third physical plan.



(a) Initial plan (b) 2nd plan (c) 3rd plan

Figure 26: The physical plans

The above plans are used to show step by step how execution time for a query can be reduced by query rewriting. The path from the initial plan to the final execution plan represents the process for applying transformation rules. Each execution speed is more than five times faster than the previous one, while the structure above the decoration does not change; therefore, performing selection and aggregate as quick as possible boosts the performance. Moreover, the inlining technique also improves the performance by enabling predicates referencing external data to be evaluated in the OLAP component.

6.3 Federation Versus Integration Performance

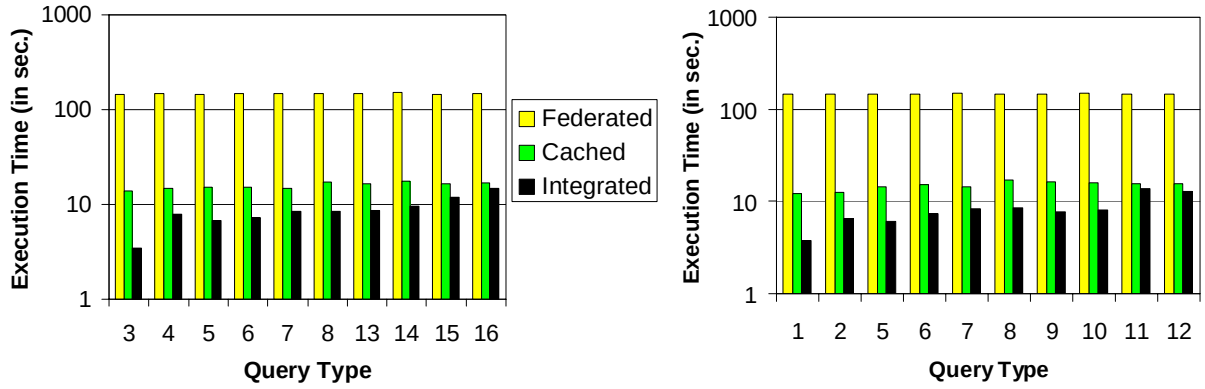
In the following two sets of experiments, comparisons are made between the queries on logically federated and physically integrated XML data in the cube to see the feasibility of the federation. The query engine is implemented with all the optimization techniques shown above.

For the first set of experiments, the XML document, *priorities.xml* (see Figure 27), is generated from TPC-H benchmark [42] about orders and their priorities, with the size of about 11.4MB. As dimensions in OLAP cubes are not typically very large, we believe the amounts of XML data used for virtual dimensions are realistic. The structure is shown below. A link $Plink$ is defined, which links the dimension values for Order to the nodes Orderkey. Meanwhile, a dimension (All-Priority-Order) is built in the cube based on

```
<Orders>
    <Order>
        <Orderkey>1</Orderkey>
        <Orderpriority>2</Orderpriority>
    </Order>
    <Order>
        <Orderkey>2</Orderkey>
        <Orderpriority>6</Orderpriority>
    </Order>
    <Order>
        <Orderkey>3</Orderkey>
        <Orderpriority>5</Orderpriority>
    </Order>
      . . .
</Orders>
```

Figure 27: The XML document, priorities.xml

the physically integrated Orderpriority in the base relational table. Moreover, the XML data is also cached in the local relational database, where a table containing two columns for both the linked dimension values and the XML values. Queries referring to this table are executed to study the performance when the XML data is retrieved and stored locally in the relational schema. The test queries follow the same criteria as in Table 5 for constructing the SELECT and the GROUP BY clauses, while the WHERE clause is composed of one of the predicates, "Order/PLink/Orderpriority=6" or "Priority=6", each of which has a selectivity of 0.1%.



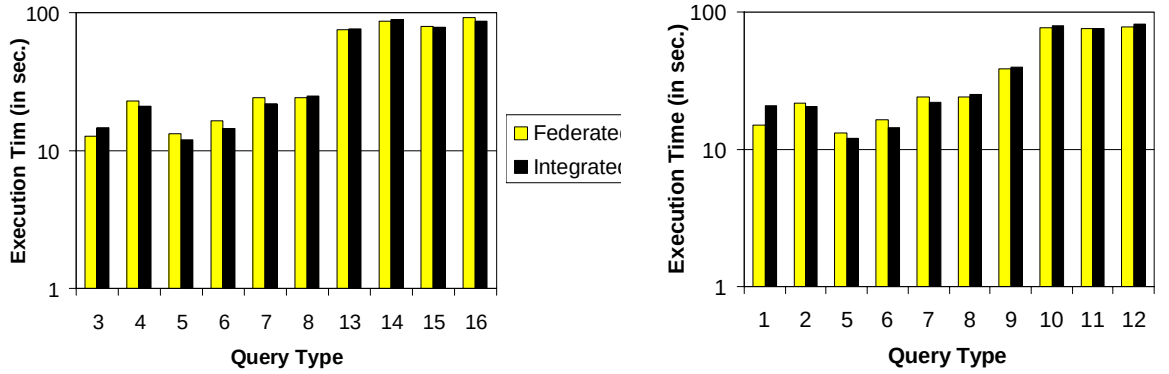
(a) Types involving Parts (b) Types involving Suppliers

Figure 28: The queries referring to federated, cached or integrated 11.4MB XML data

The bar charts in Figure 28 show the comparisons. The bars denoted as "Federated" represent the

queries using the first predicate, which federate the OLAP and the XML components and use the external data to select the cube. The "Cached" bars represent the queries using the same predicate but refer to the XML data cached in a local relational database. The "Integrated" bars show the execution time for the queries using the integrated data directly to perform the selection. The X axis and the Y axis represent the query types and the execution time in seconds in logarithmic scale, respectively.

As the charts indicate, the cost of querying the federation exceeds the cost of querying the physical integration by a factor of ten to twenty. The "Cached" bars stay in between but much closer to the "Integrated". The execution process of these federation queries can be divided into three sequential tasks. First, load the XML data into the temporary component and inline the XML values into the OLAP query. Second, perform the selection and aggregation in the OLAP component and then load the values into the temporary component. Third, generate the final result in the temporary component. Among the tasks, the first one takes much more time (about 135 seconds) so that the other two are relatively trivial; therefore, in the chart, the queries on federations seem to take up approximately the same execution time. Since the XML data is already processed and stored in the relational database on the same server as where the temporary component resides, execution of the queries on the cached XML data first rewrite the predicates and then starts the second step, and therefore boosts the execution speed. The rest of the queries referencing the dimension values which originate from the integrated XML elements skip the first step entirely. These queries are processed mostly in the OLAP component. The execution is finished after the query results of the corresponding OLAP component queries are returned and transferred to the temporary component. The performance follows the trend discussed in Section 6.1, that is, the more the size of the cube is reduced, the faster the execution is.



(a) Types involving Parts

(b) Types involving Suppliers

Figure 29: The queries referring to federated or integrated 2KB XML data

The charts in Figure 29 demonstrate comparisons of queries on two other federated/integrated levels. For this set of experiments, about 2KB of XML data was used. The XML document is composed of the nations and public population data of nations. The structure of the document is illustrated in Figure 2. Queries referencing the XML data now has the WHERE clause as "WHERE Nation[ANY]/Nlink/Population in (45860000,30205387,29250541)". Another dimension (All-Population-NationName) is built in the cube for the queries having the WHERE clause as "WHERE Population in (45860000, 30205387, 29250541)". Both predicates have the same selectivity of 10%. The two series of queries are evaluated in the same manner as the queries in the previous experiments. These two charts suggest that querying the logical federation with a virtual dimension has almost the same performance as on the physically integrated cube, when the amount of the XML data is small, i.e., a few kilobytes, and can be retrieved quickly; therefore, a federation involving such XML data can be queried just as if it was a local cube.

However, when the XML documents grow larger and larger, retrieving XML values is becoming the bottleneck for processing the federation queries. Experiments have shown that the performance can be

improved by caching the external data. That is, the XML data can be stored in relational tables, therefore reducing the time for decorating the cube for the queries using these data. As the charts in Figure 28 suggest, querying the federation of OLAP and a large amount of cached XML data can be much more efficient than querying the federation involving directly the XML documents, and thus more practical; therefore, based on the strategies proposed by [29] in handling external XML data sources under different circumstances, the cached XML data can be used by queries and provide efficient access to external data for analysis, when, e.g., the data is not out of date.

6.4 Summary of Performance Study

In summary, the federation performance is dependent on the query complexity and selectivity. The optimization strategies are effective. The more the cube is reduced and aggregated, the more effective the optimization is. Moreover, the federation approach is always good for a small amounts of XML data. For large amounts of XML data, efficient query performance can be gained by caching the external data locally, which will be the most common case in the applications of OLAP-XML federations. All in all, the logical approach to OLAP-XML federation is comparable to physical integration in terms of performance and can, unlike traditional physical integration, be the only practical solution to flexible on-line analysis involving external fast-changing data.

7 Related Work

There has been a great deal of previous work on data integration, e.g., on integrating relational data [4, 15, 17, 28, 18], object-oriented data [37], semi-structured data [9, 21, 44], a combination of relational and unstructured data [22], a combination of relational and semi-structured data [11], and a combination of object-oriented and semi-structured data [1]. However, none of these handle the advanced issues related to OLAP systems, e.g., dimensions with hierarchies and the problems related to correct aggregation. This is also true for the combined relational and XML with the query language xQuery [43], and for nD-SQL [10], which considers the federation of relational sources providing basic OLAP functionality.

An orthogonal line of work considers the correct and efficient handling of updates of dimensions [7, 16] or complete cubes [2, 19]. In contrast, we do not need to perform any updates on the underlying cube structure and data, and thus save the (still) large cost of updating the physical structures. This line of work can be considered as orthogonal to ours, and one can even imagine a combined approach where our federation approach is used when external data is first queried, followed by a physical integration of the most commonly used external data using the techniques mentioned above.

One previous paper [34] has considered the federation of OLAP and object data. In comparison, our approach is not restricted to object DBs, and their rigid schemas, but can be used on any imaginable data source as long as it allows XML wrapping. Also, we allow irregularities in the external data and offer a more general use of external data when performing decoration, selection, and grouping. Pérez et al. [35] integrated a corporate warehouse with text-rich XML documents, where XML data is always first extracted into a context warehouse and then maintained by complex Information Retrieval techniques, whereas our solution provides fast and flexible access to structured XML data, thereby enabling more efficient analysis on today's fast changing data. Cabibbo and Torlone [3] proposed the methods of integrating heterogeneous multidimensional databases, where data is of good quality, structured in a rather uniformed way, and most importantly, static. Zaman and Schneider [47] also integrated static relational and multidimensional data and presented SQL to MDX translation algorithms; however, the multidimensional data is viewed in a rather relational way and the solution does not support OLAP queries. In comparison, using XML as data source, as we do, enables the federation to be applied on any data as long as the data allows XML wrap-

ping, greatly enlarging the applicability. Query processing and optimization has been considered for data warehousing/OLAP systems [41], federated, distributed, and multi-databases [38], heterogeneous databases [6, 14], and XML and semistructured data [9]. However, previous work does not address the special case of optimizing OLAP queries in a federated environment.

The present report is a further development of the previous conference papers [30, 31, 32] which propose a logical federation of OLAP and XML systems, optimization techniques, a cost model and a partial, straight-forward implementation. In comparison, this report simplifies the query semantics, proposes a novel physical algebra modeling the actual execution tasks involved in processing an OLAP-XML query, adapts the cost model and the optimization techniques to the simplified query semantics and the physical algebra, and presents a full-function, robust query engine. Moreover, this report extends the conference paper [46] and the journal paper [45] which mainly present the physical algebra and query execution techniques. Specifically, the present paper is the first to present the transformation rules and the optimizer implementation, the logical-to-physical conversion rules and algorithm, the cost models and functions for physical plans, the query evaluation algorithm and techniques, and the full details on the experiments, along with many new examples throughout the report.

8 Conclusion and Future Work

Current OLAP systems have a common problem in handling the situations where changes in data requirements are common and data changes frequently. Physical integration of new data into OLAP systems is a long and time-consuming process, making logical integration, or federation, the better choice in many cases. The increasing use of XML suggests that the required data will often be available in XML format; therefore, a logical integration of external XML data and local OLAP databases becomes a desirable solution.

We have presented a novel practical approach to the logical federation of OLAP databases and XML documents. Our solution enables evaluation of OLAP-XML federation queries on a robust query engine integrated with optimized query processing techniques, allowing OLAP data to be decorated, selected, and aggregated by external XML data. We have covered the major issues of the OLAP-XML query engine, including the simplified query semantics, a physical algebra, query optimization, query evaluation and a series of experiments evaluating the prototypical query engine. First, the query semantics from previous work were simplified, thereby leading to a concise and compact logical query plan. Second, a physical query algebra was introduced to model the execution tasks of a federation query, where the physical operators were described with algebraic definitions and examples. Third, the process of converting a logical query plan into a physical plan was introduced. The specific execution tasks, e.g., retrieving data and performing SQL operations, are integrated into a plan by the conversion rules and algorithm. Fourth, the evaluation algorithm of a physical plan and the implementation algorithms of the physical operators were introduced. Fifth, the novel query optimizer for federation queries was described. There, the optimizer components and the optimization process were also described. Sixth, novel algebra- and heuristic-based query transformation rules were given. These rules are used to generate equivalent logical plans for the initial logical plan of a federation query. Seventh, the cost model for a physical query plan and the cost functions of the physical operators which are used to approximate the execution time of the physical candidate plans were given. Furthermore, the novel optimizer implementation algorithms were described in detail with examples. Finally, experimental results were given with respect to federation performance, optimization effectiveness and federation feasibility, suggesting that the logical OLAP-XML federation is comparable to physical integration of OLAP and XML data, and therefore can be the only practical solution to providing flexible access to fast changing data in XML format in OLAP systems.

We believe that we are the first to extend logical OLAP-XML federations by robust query optimization and evaluation techniques. Specifically, we proposed the simplified query semantics and a series of opti-

mization methods, including the optimizer architecture, the optimizer implementation, the transformation rules for rewriting logical plans based on the simplified query semantics, logical-to-physical plan conversion rules, and cost models and functions for physical plans. The query evaluation algorithm and the implementation methods for the physical operators are also new. Moreover, we believe we are the first to implement a robust query engine that parses, analyzes, optimizes and executes OLAP-XML federation queries.

Our future work will focus on three major points. First, the implementation of the query engine, including the optimization techniques, e.g., more accurate cost estimation of an execution plan, and the evaluation techniques, e.g., a plan can be evaluated in a pipelined manner as the classic pull-based SQL query evaluation. Second, as computers are more and more widely used in our daily life, e.g., in mobile phones. Integrating traditional OLAP systems with the data streams emitted by these small devices is another interesting issue as this data is increasingly available on the web in XML format. Efforts will be put into extending our system for business analysis in this area. Third, it could also be interesting to explore how the logical federation system could be used in a real software product. For example, the ability to quickly integrate XML data could be incorporated into an existing OLAP querying tool. Other components, e.g., user-friendly interface and a tool for linking XML and OLAP data, are required to be developed.

9 Acknowledgements

This work was supported by the Danish Research Council for Technology and Production Sciences under grant no. 26-02-0277.

References

[1] Kyoung-Il Bae, Jung-Hyun Kim, and Soon-Young Huh. Federated Process Framework in a Virtual Enterprise Using an Object-oriented Database and Extensible Markup Language. *Journal of Database Management*, 14(1):27–47, 2003.

[2] Mathurin Body, Maryvonne Miquel, Yvan Bédard, and Anne Tchounikine. Handling Evolutions in Multidimensional Structures. In *Proceedings of the 19th International Conference on Data Engineering, Bangalore, India, March 5-8, 2003*, pages 581–591.

[3] Luca Cabibbo and Riccardo Torlone. Integrating Heterogeneous Multidimensional Databases. In *Proceedings of the 17th international conference on Scientific and statistical database management, Berkeley, CA, USA, June 27-29, 2005*, pages 205–214.

[4] Huajun Chen, Zhaohui Wu, Heng Wang, and Yuxin Mao. RDF/RDFS-based Relational Database Integration. In *Proceedings of the 22nd International Conference on Data Engineering, Atlanta, GA, USA, April 3-8, 2006*, page 94.

[5] James Clark and Steve DeRose. XML Path Language (XPath). http://www.w3.org/TR/xpath. Current as of Oct. 27, 2006.

[6] Weimin Du, Ravi Krishnamurthy, and Ming-Chien Shan. Query Optimization in a Heterogeneous DBMS. In *Proceedings 18th International Conference on Very Large Data Bases, Vancouver, Canada, August 23-27, 1992*, pages 277–291.

[7] Johann Eder and Christian Koncilia. Changes of Dimension Data in Temporal Data Warehouses. In *Proceedings of the 3rd International Conference on Data Warehousing and Knowledge Discovery, Munich, Germany, September 5-7, 2001*, pages 284–293.

[8] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, fourth edition, 2004.

[9] Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey D. Ullman, Vasilis Vassalos, and Jennifer Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *J. Intell. Inf. Syst.*, 8(2):117–132, 1997.

[10] Frédéric Gingras and Laks V. S. Lakshmanan. nD-SQL: A Multi-Dimensional Language for Interoperability and OLAP. In *Proceedings of 24rd International Conference on Very Large Data Bases, New York City, New York, USA, August 24-27, 1998*, pages 134–145.

[11] Roy Goldman and Jennifer Widom. WSQ/DSQ: A Practical Approach for Combined Querying of Databases and the Web. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, Texas, USA, 16-18 May 2000*, pages 285–296.

[12] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, 1993.

[13] Goetz Graefe and William J. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *Proceedings of the 9th International Conference on Data Engineering, Vienna, Austria, April 19-23, 1993*, pages 209–218.

[14] Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. Optimizing Queries Across Diverse Data Sources,. In *Proceedings of the 23rd International Conference on Very Large Databases, Athens, Greece, August 25-29, 1997*, pages 276–285.

[15] Joseph M. Hellerstein, Michael Stonebraker, and Rick Caccia. Independent, Open Enterprise Data Integration. *IEEE Data Eng. Bull.*, 22(1):43–49, 1999.

[16] Carlos A. Hurtado, Alberto O. Mendelzon, and Alejandro A. Vaisman. Maintaining Data Cubes under Dimension Updates. In *Proceedings of the 15th International Conference on Data Engineering, Sydney, Austrialia, March 23-26, 1999*, pages 346–355.

[17] IBM corporation. Datajoiner. http://www-306.ibm.com/software/data/integration. Current as of Oct. 27, 2006.

[18] Holger Kache, Wook-Shin Han, Volker Markl, Vijayshankar Raman, and Stephan Ewen. POP/FED: Progressive Query Optimization for Federated Queries in DB2. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 1175–1178.

[19] Heum-Geun Kang and Chin-Wan Chung. Exploiting Versions for On-line Data Warehouse Maintenance in MOLAP Servers. In *Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong, China, August 20-23, 2002*, pages 742–753.

[20] Hans-Joachim Lenz and Arie Shoshani. Summarizability in OLAP and Statistical Data Bases. In *Proceedings of the 9th International Conference on Scientific and Statistical Database Management, Olympia, Washington, USA, August 11-13, 1997*, pages 132–143.

[21] Yu Li and Aijun An. Representing UML Snowflake Diagram from Integrating XML Data Using XML Schema. In *Proceedings of the International Workshop on Data Engineering Issues in E-Commerce, Tokyo, Japan, April 9, 2005*, pages 103–111.

[22] Imran R. Mansuri and Sunita Sarawagi. Integrating Unstructured Data into Relational Databases. In *Proceedings of the 22nd International Conference on Data Engineering, Washington, DC, USA, April 3-7, 2006*, page 29.

[23] Microsoft corporation. Performing bulk load of XML data. http://msdn.microsoft.com/library/en-us/sqlxml3/htm/ bulkload_7pv0.asp. Current as Oct. 27, 2006.

[24] Microsoft corporation. Supported sql select syntax.

[25] Microsoft corporation. Passing queries from SQL Server to a linked Analysis Server. *Books Online*, version 5.2.3790.

[26] Microsoft corporation. Sqlxml. *Books Online*, version 5.2.3790.

[27] Microsoft corporation. Using bcp and bulk insert. *Books Online*, version 5.2.3790.

[28] Oracle corporation. Gateways. http://www.oracle.com/gateways. Current as Oct. 27, 2006.

[29] Dennis Pedersen and Torben B. Pedersen. Integrating XML data in the TARGIT OLAP system. In *Proceedings of the 20th International Conference on Data Engineering, Boston, Massachusetts, USA, March 30-April 02, 2004*, pages 778–781.

[30] Dennis Pedersen, Karsten Riis, and Torben Bach Pedersen. Cost Modeling and Estimation for OLAP-XML Federations. In *Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery, Aix-en-Provence, France, September 4-6, 2002*, pages 245–254.

[31] Dennis Pedersen, Karsten Riis, and Torben Bach Pedersen. Query Pptimization for OLAP-XML Federations. In *Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP, McLean, Virginia, USA, November 8, 2002*, pages 57–64.

[32] Dennis Pedersen, Karsten Riis, and Torben Bach Pedersen. XML-Extended OLAP Querying. In *Proceedings of the 14th International Conference on Scientific and Statistical Database Management, Edinburgh, Scotland, UK, July 24 - 26, 2002*, pages 195–206.

[33] Torben Bach Pedersen and Christian S. Jensen. Multidimensional data modeling for complex data. In *Proceedings of the 15th International Conference on Data Engineering, Sydney, Austrialia, 23-26 March 1999, pages 336–345*.

[34] Torben Bach Pedersen, Arie Shoshani, Junmin Gu, and Christian S. Jensen. Extending OLAP Querying to External Object Databases. In *Proceedings of the ninth international conference on Information and knowledge management, McLean, VA, USA, November 6-11, 2000*, pages 405–413.

[35] Juan Manuel Pérez, Rafael Berlanga Llavori, María José Aramburu, and Torben Bach Pedersen. A Relevance-extended Multi-dimensional Model for a Data Warehouse Contextualized with Documents. In *Proceedings of the 8th International Workshop on Data Warehousing and OLAP, Bremen, Germany, November 4-5, 2005*, pages 19–28.

[36] Raghu Ramakrishnan. *Database Management Systems*. WCB/McGraw-Hill, third edition, 2003.

[37] Mary Tork Roth, Manish Arya, Laura M. Haas, Michael J. Carey, William F. Cody, Ronald Fagin, Peter M. Schwarz, Joachim Thomas II, and Edward L. Wimmers. The Garlic Project. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data, Montreal, Quebec, Canada, June 4-6, 1996*, page 557.

[38] Amit P. Sheth and James A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Comput. Surv.*, 22(3):183–236, 1990.

[39] Amit Shukla, Prasad Deshpande, Jeffrey F. Naughton, and Karthikeyan Ramasamy. Storage Estimation for Multidimensional Aggregates in the Presence of Hierarchies. In *Proceedings of the 22th International Conference on Very Large Data Bases, Bombay, India, September 3-6, 1996*, pages 522–531.

[40] Avi Silberschatz, Henry F. Korth, and S. Sudarshan. *Database system concepts*. McGraw-Hill, fifth edition, 2005.

[41] Erik Thomsen. *OLAP Solutions: Building Multidimensional Information Systems*. John Wiley & Sons, second edition, 2002.

[42] Transaction Processing Performance Council. TPC-H. http://www.tpc.org/tpch. Current as of Oct. 27, 2006.

[43] W3C. XQuery 1.0: an XML query language. http://www.w3.org/TR/xquery. Current as of Oct. 27, 2006.

[44] Xia Yang, Mong-Li Lee, Tok Wang Ling, and Gillian Dobbie. A Semantic Approach to Query Rewriting for Integrated XML Data. In *Proceedings of the 24th International Conference on Conceptual Modeling, Klagenfurt, Austria, October 24-28, 2005*, pages 417–432.

[45] Xuepeng Yin and Torben B. Pedersen. Evaluating XML-Extended OLAP Queries Based on a Physical Algebra. *Journal of Database Management*, 17(2):85–116, 2006.

[46] Xuepeng Yin and Torben Bach Pedersen. Evaluating XML-Extended OLAP Queries Based on a Physical Algebra. In *Proceedings of the 7th ACM international workshop on Data Warehousing and OLAP, Washington, DC, USA, November 12-13, 2004*, pages 73–82.

[47] Kazi A. Zaman and Donovan A. Schneider. Modeling and Querying Multidimensional Data Sources in Siebel Analytics: a Federated Relational System. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 822–827.

[48] Qian Zhu and Per A. Larson. Global Query Processing and Optimization in the CORDS Multidatabase System. In *Proceedings of the 12th International Conference on Data Engineering, New Orleans, Louisiana, USA, February 26-March 1, 1996*, pages 640–646.