Evaluating XML-Extended OLAP Queries Based on a Physical Algebra

Xuepeng Yin
Aalborg University
Fredrik Bajers Vej 7E
Dk-9220,Aalborg Ø
Denmark
xuepeng@cs.aau.dk

Torben Bach Pedersen
Aalborg University
Fredrik Bajers Vej 7E
Dk-9220,Aalborg Ø
Denmark
tbp@cs.aau.dk

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. In this paper, we extend previous work on the logical federation of OLAP and XML data sources by presenting a simplified query semantics, a physical query algebra and a robust OLAP-XML query engine. Performance experiments with a prototypical implementation suggest that the performance for OLAP-XML federations is comparable to queries on physically integrated data.

Categories and Subject Descriptors

C.2.4 [**COMPUTER-COMMUNICATION NETWORKS**]: Distributed Systems—*Distributed databases*; H.2.4 [**Database Management**]: Systems—*Query Processing*; H.2.5 [**Database Management**]: Heterogeneous Databases—*Data Translation*; H.2.7 [**Database Management**]: Database Administration—*Data Warehouses and Repository*

General Terms

Algorithms, Performance

Keywords

OLAP, XML, data integration, query semantics, physical algebra

1. INTRODUCTION

On-line Analytical Processing (OLAP) technology enables data warehouses to be used effectively for online analysis, providing rapid responses to iterative complex analytical queries. Usually an OLAP system contains a large amount of data, but *dynamic data* today, 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, 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, allowing *selections* and *aggregations* to be performed over the decorated cube. In this paper, we describe the foundation of a robust federation query engine with query plan generation, optimization and evaluation techniques. A novel query semantics that simplifies earlier definitions 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. The main contribution of the paper is the definition of a *physical query algebra* that, unlike the previous logical algebra, is able to model the real execution tasks of a federation query. Here, all concrete data retrieval and manipulation operations in the federation are integrated. This means that we obtain a much more precise foundation for performing query optimization and cost estimation. *Algebra-based* query optimization and evaluation techniques are also presented. Experiments with the query engine suggest that the query performance of the federation approach is comparable to physical integration.

There has been a great deal of previous work on data integration, for instance, on relational data [3, 4, 9], semi-structured data [1], and a combination of relational and semi-structured data [5, 10]. However, none of these handle the advanced issues related to OLAP systems, e.g. automatic and correct aggregation. Some work concerns integrating OLAP and object databases [17, 8], which demands rigid schemas, i.e., data is represented by classes and connected by complex associations. 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 wrapping, greatly enlarging the applicability. The most related previous work is [15], which presents a logical federation of OLAP and XML systems, where a *logical* algebra defines the query *semantics*, and a partial, straightforward implementation. In comparison, this paper presents a full-function, robust query engine and a *physical* query algebra that models the actual execution tasks involved in processing an OLAP-XML query, along with more robust query optimization techniques.

This paper makes the following novel contributions to OLAP-XML federations. First, a simplified query semantics (compared to [15]) is proposed. Second, a physical query algebra is defined. Third, algebra-based query optimization and evaluation techniques

DOLAP'04, November 12–13, 2004, Washington, DC, USA.

are introduced. Fourth, a robust federation query engine is implemented with all the above techniques and experiments are performed with it. The novel contributions correspond to Sections 6–9.

The rest of the paper is organized as follows. Section 2 introduces the cube and the XML document used in the illustrations. Section 3 introduces the overall architecture of the query engine. Section 4 defines the data models used in the federation. In Section 5, a brief introduction to the logical algebra and query semantics is given, which is followed by the simplified query semantics in Section 6. Section 7 presents the formal definitions of the physical operators. The query optimizer, query cost estimation and query evaluation techniques are briefly described in Section 8. Section 9 describe the performance study, whereas Section 10 concludes the paper and points to future work.

2. CASE STUDY

The TPC-H-based [18] database used in the experiments and illustrations is shown in Figure 1(a). 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 1. The XML document is composed of the nation codes and public population data about nations in millions. An example of the document is illustrated in Figure 1(b), where each `Nation` element contains two sub-elements, `NationName` and `Population`. We use the listed three lines as the example data in this paper. To connect the dimension values of the level `Nation` and the populations, a link, $Nlink$, is defined, which maps dimension values for `Nation` in the cube to the nodes `Nation` in the XML document (See Section 4).

Dimensions:	***Suppliers***	***Parts***	***Orders***	***Time***
	AllSuppliers	AllParts		AllTime
	Region	Manufacturer	AllOrders	Year
	Nation	Brand	Customer	Month
	Supplier	Part	Order	Day
Measure:	Quantity, ExtPrice			

(a) 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>
```

(b) Part of the XML data

Figure 1: Cube and 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

Table 1: The fact table

3. OVERALL ARCHITECTURE

In this section, we give an overview of the prototypical OLAP-XML federation system, the federation query language and the basic query evaluation process. The overall architecture of the prototype is shown in Figure 2. 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 components. In the prototype, MS 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 [13]. The XML component is the local file system based on the XML data retrieved from the Web with M S SQLXML [12] on top.

```
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
```

Table 2: An example SQL$_{XM}$ query

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. Figure 2 is an example SQL_{XM} query based on the cube in Figure 1(a), where the *roll-up function* "Brand(Part)" rolls up to the `Brand` level from the `Part` level, and the level expression "Nation[ANY]/Nlink/Population" connects the dimension level `Nation` and the decoration XML data `Population` with a link $Nlink$.

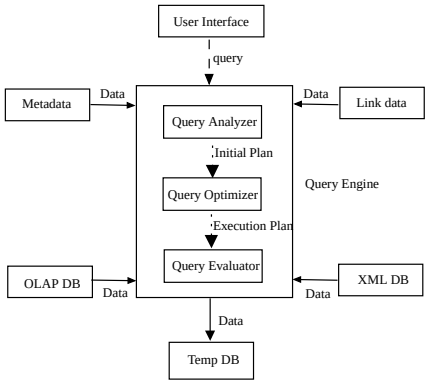


Figure 2: Architecture of the optimized query engine

As shown in Figure 2, the query engine has three components: *query analyzer*, *query optimizer* and *query evaluator*. The query engine parses and analyzes the given query, and generates the initial logical plan. The plan is expressed in the logical algebra (See Section 5). 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 evaluation 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 7). 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.

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. 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, $l_{i1} \sqsubseteq_i l_{i2}$, which holds if elements in L_{i2} can be said to contain the elements in L_{i1}. 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 $e_2 \in L_{ik}$ and $e_1 \neq e_2$. 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$_l(e_{ik_h}, l_{ij}) = \{e' | e_{ik_h} \sqsubseteq_{D_i} e' \wedge e' \in L_{ij}\}$. 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. $l_{ik} \sqsubseteq_i l_{ij} \wedge l_{ik} \neq l_{ij}$.

A *fact table* F is a relation containing one attribute for each dimension and one attribute for each measure. Thus, $F = \{(e_{\bot_1}, \ldots, e_{\bot_n}, v_1, \ldots, v_m) | (e_{\bot_1}, \ldots, e_{\bot_n}) \in \bot_1 \times \bot_n \wedge (v_1, \ldots, v_m) \in M \subseteq T_1 \times \ldots \times T_m\}$, 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 $M_j = \{(e_{\bot_1}, \ldots, e_{\bot_n}, v_j)\}$. 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 referred XML documents, and $Links$ is a set of *links* (See 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 $Nlink = \{(DK, n1), (CN, n2), (UK, n3)\}$ 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 "DK", $n2$ is the `Nation` node with the sub-node `NationName` having the string value "CN", and similarly for $n3$.

An XPath expression [2] 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, 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, $Nation[ANY]/Nlink/Population$ links the dimension value "DK" 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 [11]. 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 $c = \emptyset$.

5. LOGICAL ALGEBRA AND QUERY SEMANTICS

In previous work [15], 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, δ, builds a virtual dimension using the XML data referenced by a level expression. The virtual dimension has the hierarchy in accordance with the cube semantics, which consists of the unique top level, the mid-level of external XML data, and the bottom level linking the measures and the decoration data.

Federation Selection A federation selection, σ_{Fed}, selects data in the federated cube according to user defined conditions. The cube can have virtual dimensions built from external data, which means that XML data can also be used in the filter. The cube schema is not changed. Only facts in the fact table are affected.

Federation Generalized Projection The generalized federation projection, Π_{Fed}, also let the federated cube be aggregated over the external XML data. Given a set of argument levels, the generalized projection first removes the dimensions in which no argument levels are present, and then each dimension value is rolled up to the specified level. Finally, facts in the fact table are grouped, aggregated measures are calculated, 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. $\{\bot_p, \ldots, \bot_q\} \subseteq \{\bot_1, \ldots, \bot_n\}$ and $\{l_s, \ldots, l_t\}$ are levels in C such that $\bot_s \sqsubseteq_s l_s, \ldots, \bot_t \sqsubseteq_t l_t$ and $\bot_s \neq l_s \wedge \ldots \wedge \bot_t \neq l_t$. le is used to represent a level expression, $l[SEM]/link/xp$, where SEM is the semantic modifier, 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. $LE_\Pi = \{le_{v_\Pi}, \ldots, le_{v_\Pi}\}$ are the level expressions in the SELECT and GROUP BY clause. $LE_{\sigma_{where}}$ are the level expressions in the WHERE clause. $LE_{\sigma_{having}}$ 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 $f_x(M_x), \ldots, f_y(M_y), \bot_p, \ldots, \bot_q$,

$l_s(\perp_s), \ldots, l_t(\perp_t), le_{u_\Pi}, \ldots, le_{v_\Pi}$ 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 [15] 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, and 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.

6. SIMPLIFIED QUERY SEMANTICS

The query semantics have a great impact on the initial plan, as the semantics take the form of a logical query tree after 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 and the WHERE clauses. As the algebra shows, an operator takes an input federation and generates 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 input 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 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 shema. An example query and the corresponding simplified logical plan tree is shown in Figure 3, where only one decoration, $\delta_{N[ANY]/Nl/P}$, exists below the federation selection, although referenced by two federation operators.

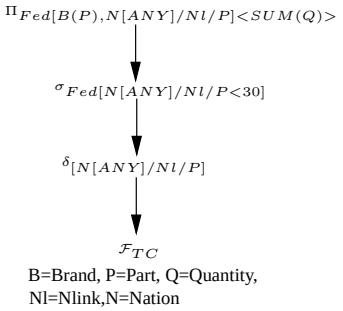


Figure 3: The initial logical query plan

7. PHYSICAL ALGEBRA

As shown in Figure 2, 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 with more detailed evaluation operations. In this section, 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.

The OLAP-XML federation decorates OLAP data in the temporary component using the decoration XML data, which then enables selections and aggregations over the decorated temporary fact data. Therefore, the temporary component plays an important role at evaluation time. 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.

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 7.1. *Suppose the extended federation $\mathcal{F}_{TC,ext}$ has the cube schema and the fact table in Section 2. The cube selection*

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 3: The fact table after 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 in the fact table are retained. The resulting fact table is shown in Table 3.

DEFINITION 7.1. **(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: $\sigma_{Cube[\theta]}(\mathcal{F}_{ext}) = (C', Links, X, T)$*, 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* $t'_i = \begin{cases} t_i & \text{if } \theta(t_i) = tt \\ (e_{\perp_1}, \ldots, e_{\perp_n}, NULL, \ldots, NULL) & \text{otherwise.} \end{cases}$

Cube Generalized Projection (CGP) 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 7.2. *Suppose the extended federation* $\mathcal{F}_{TC,ext}$ *has the cube schema and the fact table in Section 2. The CGP 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 4.*

Quantity	Supplier
17	S1
64	S2
2	S3
26	S4

Table 4: The fact table after the CGP operator

DEFINITION 7.2. **(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* $\forall D'_g \in \{D_g | D_g \in D \wedge \perp_g \notin \{l_{i_1}, \ldots, l_{i_k}\}\} \forall f_{j_h} \in \{f_{j_1}, \ldots, f_{j_l}\} (f_{j_h} \in AggType(M_{j_h}, D'_g))$*, meaning that the specified aggregate functions are allowed to be applied. The CGP 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* $LS'_{i_h} = \{l_{i_h P} | l_{i_h P} \in LS_{i_h} \wedge l_{i_h} \sqsubseteq_{i_h} l_{i_h P}\}$*, 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:* $F' = \{(e'_{\perp_{i_1}}, \ldots, e'_{\perp_{i_k}}, v'_{j_1}, \ldots, v'_{j_l}) | e'_{\perp_{i_g}} \in L_{i_g} \wedge v'_{j_h} = f_{M_{j_h}}(\{v | (e_{\perp_1}, \ldots, e_{\perp_n}, v) \in M_{j_h} \wedge (e'_{\perp_{i_1}}, \ldots, e'_{\perp_{i_k}}) \in Roll\text{-}up_{\perp_{i_1}}(e_{\perp_{i_1}}, l_{i_1}) \times \ldots \times Roll\text{-}up_{\perp_{i_k}}(e_{\perp_{i_k}}, l_{i_k})\})\}$*. Furthermore, if* $\exists (e_{\perp_1}, \ldots, e_{\perp_n}, v_j) \in M_{j_h} \exists e \in \{e_{\perp_1}, \ldots, e_{\perp_n}\} (||Roll\text{-}up_{\perp_{i_g}}(e, l_{i_g})|| > 1 \wedge v_j \neq NULL)$ *then* $AggType(M_{j_h}, D'_{i_g}) = c$*.*

Fact-Transfer In a physical execution plan, the fact-transfer operator is above the cube and below the federation operators. The resulting fact data from the cube operators is transfered to the temporary component through the fact-transfer operator. Thereafter, 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 7.3. **(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 7.3. *A roll-up function, "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 5.*

Nation	Supplier
DK	S1
DK	S2
CN	S3
UK	S4

Table 5: The temporary table for `Nation` and `Supplier`

DEFINITION 7.4. **(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 7.3, 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 evaluation time, the XML data is needed in the temporary component to allow decoration, grouping 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 7.5. **(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 specified by the semantic modifier* SEM*. At evaluation 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* $SEM = CONCAT$.

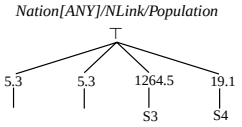
EXAMPLE 7.4. *The operator $\tau_{Nation[ANY]/Nlink/Population}(\mathcal{F}_{TC,ext})$ generates $\mathcal{F}'_{TC,ext} = (C, Links, X, T')$, where T' has a new table $R_{\tau_{Nation[ANY]/Nlink/Population}}$. The table has two columns, one for the dimension values of* Nation *and the other for the decoration 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 table $R_{\tau_{Nation[ANY]/Nlink/Population}}$ using the XML data from Figure 1(b) is shown in Table 6.*

Nation	Population
DK	5.3
CN	1264.5
UK	19.1

Table 6: The temporary table for Nation **and** Population

Decoration The cube is decorated in the temporary component using the decoration operator δ. 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 decorated level. 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. At evaluation time of the decoration operator, the temporary table created by the XML-transfer operator having the same input 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 7.8. 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.

EXAMPLE 7.5. *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 7.3 and 7.4. The dimension hierarchy is strict since a supplier in a nation only has one corresponding population number. Table 7 shows the dimension and the temporary table.*



Supplier	Population
S1	5.3
S2	5.3
S3	1264.5
S4	19.1

Table 7: The decoration dimension and the temporary table Nation/Population

DEFINITION 7.6. **(Decoration)** *Let $Op_1, \ldots, Op_n$ be the child operators of a decoration operator $\delta_{l_z[SEM]/link/xp}$, $(C, Links, X, T_1), \ldots, (C, Links, X, T_n)$ 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 prior to 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 7.6. *Suppose the temporary fact table in $\mathcal{F}_{TC,ext}$ is the copy of the fact table in Figure 1. 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 7 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 Table 8 for the query and the fact table.*

DEFINITION 7.7. **(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*

```
SELECT   Fact.*
FROM     Fact F,
         Nation/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

Table 8: The SQL query and the resulting fact table

federation selection operator is defined as: $\sigma_{Fed[\theta]}(\mathcal{F}_{ext}) = (C', Links, X, T')$ *where* $\mathcal{F}_{ext} = (C, Links, X, T)$ *is the input, 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*.* $R'_F = \sigma_\theta(R_F)$*, 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,* $\{l_x(l_{\perp_x}), \ldots, l_y(l_{\perp_y})\} \subseteq S_\theta$*, 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 (FGP) 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 function 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-up functions, 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 7.7. *Suppose the temporary fact table in* $\mathcal{F}_{TC,ext}$ *is the copy of the fact table in Figure 1. 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 7 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 Table 9 for the query and the fact table.*

```
SELECT    SUM(Quantity), Population
FROM      Fact F,
          Nation/Population P
WHERE     F.Supplier=P.Supplier
GROUP BY  Population
```

Quantity	Population
81	5.3
2	1264.5
26	19.1

Table 9: The SQL query and the resulting fact table

DEFINITION 7.8. **(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* $\perp_p, \ldots, \perp_q$ *be bottom levels,* $l_s(\perp_s), \ldots, l_t(\perp_t)$ *be roll-up expressions,* $l_u[SEM_j]/link_j/xp_j, \ldots, l_v[SEM_k]/link_k/xp_k$ *be level expressions and* $D_j, \ldots, D_k$ *be the dimensions built for the preceding level expressions. 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 FGP operator* Π_{Fed} *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,* $T' = \{R'_F, R_{\omega_{[\perp_s, ls]}}, \ldots, R_{\omega_{[\perp_t, l_t]}}, R_{D_j}, \ldots, R_{D_k}\}$*, 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 hierarchies of levels, the ordering of dimension values 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* $R_{F,intermediate} = R_F \bowtie R_{\omega_{[\perp_s, ls]}} \bowtie \ldots \bowtie R_{\omega_{[\perp_t, l_t]}} \bowtie R_{D_j} \bowtie \ldots \bowtie R_{D_k}$*,* $\mathcal{G}$ *is the SQL aggregation.*

Inlining 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 transfered 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 and dimension values linked by the level expressions 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 7.8. *For "Nation[ANY]/Nlink/Population<30", the decoration data and the decorated dimension values are retrieved from the XML-transfer operator in Example 7.4. Using the result, the predicate is transformed to "Nation='DK' OR Nation='UK' ".*

DEFINITION 7.9. *Let* $\theta_1, \ldots, \theta_n$ *be predicates referencing level expressions.* θ_i *has the following possible formations:*

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

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 child XML-transfers 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 evaluated first, the rest of the plan is evaluated 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. $\mathcal{T}(l_z[SEM]/link/xp = K_1)\ OR \ldots OR\ \mathcal{T}(l_z[SEM]/link/xp = K_n)$.
6. $NOT\ (\mathcal{T}(\theta_{i1}))$.
7. $\mathcal{T}(\theta_{i1})\ \ bo\ \ \mathcal{T}(\theta_{i2})$.

EXAMPLE 7.9. *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 2. 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. Therefore, a logical plan can be deterministically converted to a physical plan.*

In the logical plan in Figure 4, the predicate "$(N/Nl/P < 30)'$" is marked to be rewritten and no longer refers to the level expression at evaluation 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 FGP operator rolls up the dimensions to the specified levels and calculates the aggregate functions over the specified measure.*

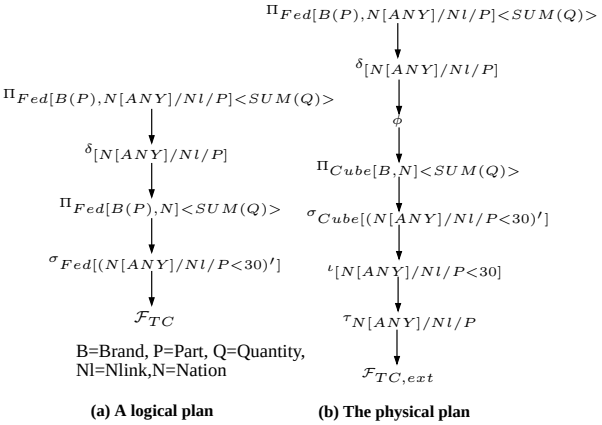


Figure 4: A logical plan and its corresponding physical plan

The corresponding physical plan is shown to the right. The plan is evaluated 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 CGP 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. 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 evaluated 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 FGP 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 uses directly the dimension values for* `Nation` *from Nlink and the corresponding population data in the XML document 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 FGP operator utilizes SQL operations to roll up the cube furthermore to the decoration level* `Population`*.*

8. QUERY OPTIMIZATION AND EVALUATION

In this section, we take a brief look into the query optimization, cost estimation and evaluation.

Architecture of The Optimizer The optimizer is based on the Volcano optimizer [6], where queries are optimized in two stages. The first phase, plan rewriting, is similar to the first stage of Volcano, where, for the input plan, the entire plan space consisting of logical expressions is generated. During the second stage of Volcano, the search for the best plan is performed. The implementation rules are used to replace operators by algorithms, and the costs of diverse sub-plans are estimated. In our optimizer, we integrate the second stage into the first stage such that logical plan enumeration

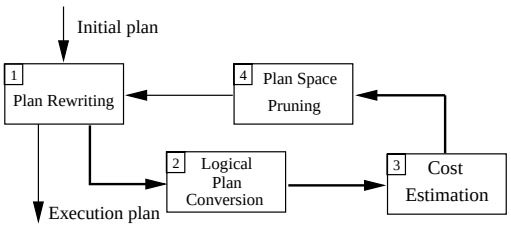


Figure 5: Inner structure of the query optimizer

is combined with plan search. Conversion into physical plans is straightforward as every logical operator can be deterministically implemented by one algorithm. After Phase 2, each logical plan is integrated with real execution tasks. Therefore the evaluation cost can be explicitly estimated. This is achieved in Phase 3 in order to perform *cost-based* plan space pruning in Phase 4. As the logical plans are considerably smaller than physical plans, plan enumeration is made much faster by enumerating logical rather than physical plans, boosting the overall optimization speed.

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. An operator constructs new trees on top of the result plans from the previous iteration, generates the logical plan space and proceeds until the pruning phase is finished. The resulting plans are then passed on to the operator above in the original plan which starts another iteration. 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.

The evaluation cost of a physical plan is estimated according to the evaluation algorithm. Therefore, we first introduce the query evaluation process to give a better understanding of cost estimation.

Query Evaluation The general evaluation algorithm is shown in pseudo-code of Figure 10. Plan b in Figure 4 shows an execution plan tree, 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 either a federation or a cube name) as no operations need to be performed on these. When the algorithm returns from the bottom, the real execution starts. The algorithm follows the idea of the conventional pull-based iterator model [7], where the lower part of the plan tree provides data for higher operators. However, data is not directly transferred between operators through pipes. 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 evaluated 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, component queries are constructed and evaluated in the appropriate component. Data can also be transferred 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, as Examples 7.6 and 7.7 have shown, it might be a query string, which then can be nested into the query string of a higher operator and evaluated later at some point in batch-mode.

Cost Estimation 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 evaluation time of the sub-plans. However, to give an intuitive overview, we divide the total cost to 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

```
void OpEvaluation(Operator Op)
1)    {
2)        if Op is a F_ext
3)          return;
4)        get all the child operators Op_1, ..., 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)   }
```

Table 10: The evaluation algorithm

be inlined into the selection predicates and therefore can be evaluated in the OLAP component. Therefore, the first period of the evaluation 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 the 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. (See [16] for a more detailed discussion).

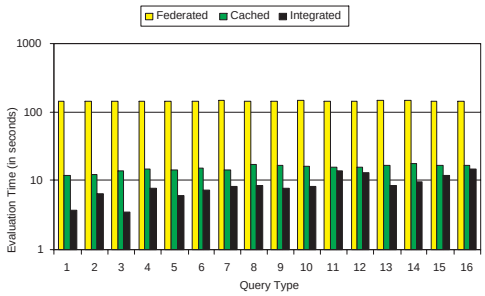
9. PERFORMANCE STUDY

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. The example cube used in the experiments is shown in Figure 1(a) . The cube is based on about 100MB of data generated using the TPC-H benchmark [18] . The following experiments observe the federation w.r.t. the practicality of the federation system. Thus, we compare the performance when the external XML data is in 1) the XML component (federated), 2) in the local, relational temporary component (cached), and 3) physically integrated in the OLAP cube itself (integrated).

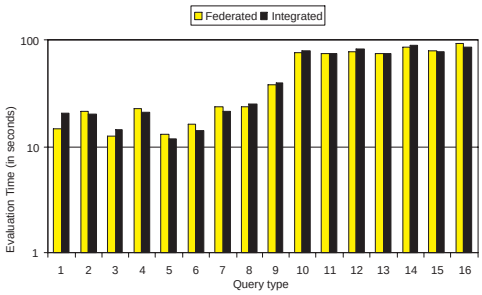
The performance of sixteen different *query types* was measured. The query types all aggregate fact data, but vary in a) whether one or two dimensions are used in the query, b) which dimensions are used, and c) which levels in these dimensions are used. Two different XML documents were used, a large (11.4 MB) document about orders and their priorities, and a small (2KB) document about nations and their populations, both generated from the TPC-H benchmark [18]. For the small document, the WHERE clause has a 10% selectivity. For the large one, the WHERE clause has a 0.1% selectivity. The selectivity does not affect the relative performance of the queries on the federated, cached and integrated data originating from the same XML document, as long as the same selectivity is used. The bar charts in Figure 6 shows the performance for the federated, cached, and integrated cases described above.

As Figure 6(a) indicates, 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 “federated” queries are evaluated in three sequential tasks. First, load the XML data into the temporary component and rewrite the predicate. Second, perform the selection and aggregation in the OLAP component, then load the values into the temporary component. Third, generate the final result in the temporary component. The first task takes much more time (about 135 sec.) so that the other two are relatively trivial. Therefore, the queries on federations seem to take approximately the same evaluation time. The “cached” queries skip the first part of the first

task and rewrite the predicates using the cached XML data, thereby boosting the execution speed. The "integrated" queries referencing the dimension values which are the integrated XML elements skip the first step, thereby evaluated mostly in the OLAP component.



(a) Queries involving the federated, cached or integrated 11.4MB XML data



(b) Queries involving the federated or integrated 2KB XML data

Figure 6: Comparisons of the queries involving different component data

The chart in Figure 6(b) demonstrates comparisons of queries on two other federated/integrated levels. The chart suggests 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. 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, thereby reducing the time for decorating the cube for the queries using these data. Based on the strategies proposed by [14] 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. In summary, the federation is good for a small amount of XML data. However, more efficient query performance can be gained by caching the external data locally, which will become the most common case in the applications of OLAP-XML federations. All in all, the logical approach can actually be a practical alternative for flexible on-line analysis involving external fast-changing data.

10. CONCLUSION AND FUTURE WORK

Current OLAP systems have a common problem in physically integrating fast changing data. As external data will most often be available in XML format, a logical integration of OLAP and XML data is desirable.

Motivated by this, we have extended previous work on OLAP-XML federations as follows. First, a simplified logical query semantics is proposed which yields more compact and concise logical query plans for SQL$_{XM}$ queries. Second, a set of physical algebra operators is presented in order to model the actual query execution tasks precisely. Third, query plan optimization is performed in four phases: query rewriting, logical query conversion, cost estimation and plan space pruning. Fourth, we describe the implementation of a robust query engine, which generates component queries for underlying data sources, and evaluate the final execution plan in a bottom-up manner. Fifth, a performance study has been performed that shows the effectiveness of our approach.

Future work will be focused on improving the query engine by developing more advanced query optimization techniques, cost estimation techniques, and query evaluation techniques, e.g., a more efficient search algorithm for query enumeration, more accurate cost formulas, and more efficient OLAP and XML data loading techniques. Also, more performance studies should be performed to reveal the behavior of the OLAP-XML query engine.

11. REFERENCES

[1] S. Chawathe et al. The TSIMMIS project: Integration of heterogeneous information sources. In *Proc. of IDS of Japan*, pp. 7-18, 1994.

[2] J. Clark and S. DeRose, XML Path Language (XPath), www.w3.org/TR/xpath. Current as of Aug. 27, 2004.

[3] Datajoiner. www-306.ibm.com/software/data/datajoiner. Current as of Aug. 27, 2004.

[4] Gateways. www.oracle.com/gateways. Current as of Aug. 27, 2004.

[5] R. Goldman and J. Widom. Wsq/dsq: A practical approach for combined querying of databases and the web. In *Proc of SIGMOD*, pp. 285-296, 2000.

[6] G. Graefe and W.J.McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *Proc. of ICDE*, pp. 209-218, 1993.

[7] G. Graefe. Query Evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73-170, 1993.

[8] J. Gu, T. B. Pedersen, and A. Shoshani. OLAP++: Powerful and Easy-to-Use Federations of OLAP and Object Databases. In *Proc of VLDB*, pp. 599-602, 2000

[9] J. M. Hellerstein, M. Stonebraker, and R. Caccia. Independent, open enterprise data integration. *IEEE Data Engineering Bulletin*, 22(1):43-49, 1999.

[10] T. Lahiri et al. Ozone - integrating semistructured and structured data. In *Proc of DBPL*, 1999.

[11] H. Lenz and A. Shoshani. Summarizability in olap and statistical data bases. In *Proc of SSDBM*, pp. 39-48, 1997.

[12] Microsoft corporation, SQLXML, www.sqlxml.org. Current as of Aug. 27, 2004.

[13] Microsoft corporation, Supported SQL SELECT Syntax, msdn.microsoft.com/library/default.asp?url=/library/en-us/olapdmpr/prsql_70e0.asp. Current as of Aug. 27, 2004.

[14] D. Pedersen and T. B. Pedersen. Integrating XML Data in the TARGIT OLAP System. In *Proc of ICDE*, pp. 778-781, 2004.

[15] D. Pedersen, K. Riis, and T. B. Pedersen. XML-Extended OLAP Querying. In *Proc of SSDBM*, pp. 195-206, 2002.

[16] D. Pedersen, K. Riis, and T. B. Pedersen. Cost modeling and estimation for OLAP-XML federations. In *Proc of DaWaK*, pp. 245-254, 2002

[17] T. B. Pedersen, A. Shoshani, J. Gu, and C. S. Jensen. Extending OLAP Querying to External Object Databases. In *Proc of CIKM*, pp. 405-413, 2000.

[18] Transaction Processing Performance Council. TPC-H. www.tpc.org/tpch. Current as of Aug. 27, 2004.