The OLAP-XML Federation System

Xuepeng Yin Torben Bach Pedersen

Department of Computer Science, Aalborg University, 9220 Aalborg Ø, Denmark
{xuepeng, tbp}@cs.aau.dk

Abstract

We present the logical "OLAP-XML Federation System" that enables the external data available in XML format to be used as *virtual dimensions*. Unlike the complex and time-consuming physical integration of OLAP and external data in current OLAP systems, our system makes OLAP queries referencing fast-changing external data possible. This demo shows the uses of XML data for selection and grouping and explains the query optimization and evaluation processes with a concrete example.

1 Introduction

The changing data requirements of todays dynamic business environments are not handled well by current On-Line Analytical Processing (OLAP) systems. Physically integrating unexpected data into such systems is a long and time-consuming process requiring the cube to be rebuilt[3, 4], thereby making logical integration the better choice in many situations. The increasing use of Extended Markup Language (XML), e.g. in business-to-business (B2B) applications, suggests that the required data will often be available as XML data. Thus, making XML data accessible to OLAP systems is greatly needed.

Our overall solution is to logically federate the OLAP and XML data sources, enabling data analysis on historical data as well as the newest information. The OLAP-XML federation system presented here decorates the OLAP cube with virtual dimensions based on external XML data, allowing selections and aggregations to be performed over the decorated cube. We present a case study based on the Transaction Processing Council (TPC) TPC-H benchmark [9]. The system is implemented in MS Visual J++ 6.0 on top of the MS SQL Server 2000 SP3 and COM[7] interfaces including MS XML and database object models.

The demonstrated system implements the following novel contributions to logical federation of OLAP and XML data sources.

- a logical algebra and query semantics for OLAP and XML federation.
- a physical query algebra that models the federation queries with concrete component data retrieval and manipulation operations.
- evaluation techniques of the federation queries, including component query construction, execution, and scheduling.
- rule-based and cost-based query optimization techniques including algebraic query rewriting and inlining.
- an OLAP-XML query engine implemented with the above techniques.

We believe that we are the first to implement a fully functioning query engine for logical OLAP-XML federations, including techniques from logical design to query evaluation and optimization.

2 OLAP-XML Federations

A federation contains an OLAP *cube*, the XML documents, and the *links* between OLAP data and XML nodes for decoration. Cubes are *multidimensional* views of data that are typically categorized into data being analyzed (*measures*) or *dimensions*, which are mostly textual and characterize the facts. Dimensions are structured using *levels* that correspond to the required levels of detail. The fundamental part of an XML document is the element node, which can contain other element nodes. Links between OLAP data and XML data make it easy to reference XML data in OLAP queries. The fundamental linking mechanism is a relation between one dimension value in a cube and one node in an XML document. Figure 1 shows an example link, Nation

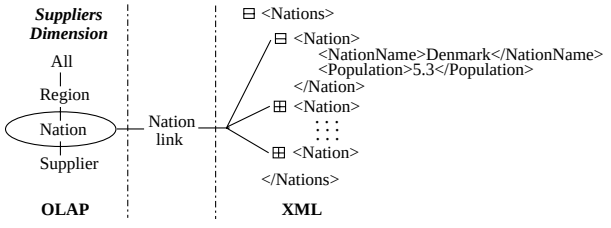

Figure 1: The link between OLAP and XML

link (Nlink), that connects the dimension values of Nations

International Conference on Management of Data
COMAD 2006, Delhi, India, December 14–16, 2006

to the Nation nodes that have the same text values in the sub-nodes, NationName, in the XML document. The dimension schema is based on the TPC-H benchmark [9] and the XML document is composed of nation names and population data (in millions). The plus/minus symbol in a box indicates whether the element is folded/unfolded. Below is an example federation query, where Brand is a level from the Parts dimension with the schema (All-Manufacture-Brand-Part).

```
SELECT    SUM(Quantity),Brand,
          Nation/Nlink/Population
FROM      TC
WHERE     Nation/Nlink/Population<30
GROUP BY  Brand, Nation/Nlink/Population
```

The above query shows the total quantity of the parts of each brand sold by each nation, where a nation is decorated with its population. Nation/Nlink/Population, is a *level expression* [10], where Population is a relative XPath [1] 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.

3 The OLAP-XML System

The overall architecture of the federation system 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. The temporary component is the temporary database on SQL Server, and the OLAP component uses MS Analysis Services, and is queried with SQL [6]. The XML component is the local file system based on the XML data retrieved from the Web with MS SQLXML [5] on top.

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 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 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. Specifically, the component queries modeled by the execution plan are evaluated in the OLAP and XML components and the data is transferred to the temporary component, which is then processed by SQL operations to produce the final result.

Generally, the component queries are evaluated in the OLAP and XML components in parallel, which sometimes causes a large amount of OLAP data to be transfered to the temporary database and then selected by predicates referencing XML data. To reduce the inter-component data transfer, an optimization, *inlining* (see [10] for a formal

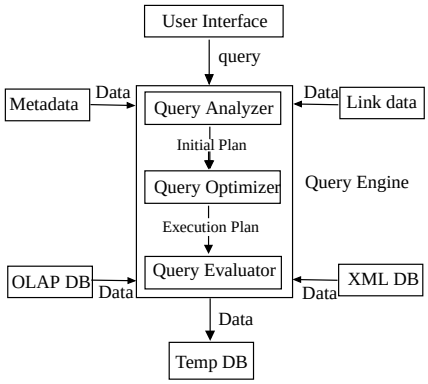

Figure 2: Architecture of the query engine

definition), retrieves the relevant XML data first and uses it to rewrite the predicates to reference dimension values and constants only but perform the same selection. Then, selections can be performed in the OLAP component, thereby reducing the amount of OLAP data to be transferred to the temporary component effectively.

Besides inlining, the query optimizer also rewrites the query plans to reduce intermediate data. The optimizer is based on the Volcano optimizer [2] and composed of four phases: *plan rewriting*, *plan conversion*, *cost estimation*, and *plan pruning*. The input is the initial logical plan as shown in Figure 2. In the first phase, plan re-writing, the logical plans generating the equivalent output federations are enumerated for the input logical plan. *Transformation rules* are applied to enumerate equivalent plans. In the second phase, physical plans are generated for the logical plans from Phase 1, which are then cost-estimated in Phase 3. Phase 4 removes expensive logical plans from the plan space, based on the cost of their physical plans. The cheap plans then again participate in plan rewriting in phase 1. This process goes on until all the operators in the initial plan are visited and selects the physical plan with the least cost as the execution plan. Experiments on query optimization effectiveness indicate that the optimized plans are executed seven to fifty times faster than the straightforward initial plans. The more the OLAP cube is reduced by selection and aggregation, the more effective the optimization is.

We have also observed the federation system with respect to the feasibility of the federation system. Experiments suggest that the federated approach has almost the same performance as the physical integration, where external data has been physically integrated in the OLAP cube itself, when the amount of the XML data is small (i.e., a few kilobytes). More efficient query performance can be gained by caching the external data locally in the local, relational component, which will become the most common case in the applications of OLAP-XML federations.

4 The Demonstration

The demonstration will first show the interface of the federation system. Next, query processing will be demonstrated

by means of concrete federation queries. We show the optimizations and component query construction, which are specialized for OLAP-XML federations. Supporting material in the form of slides and posters will be used in the demonstration.

4.1 User Interface

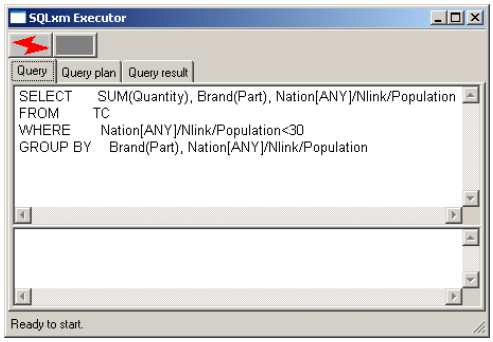

Figure 3: The GUI of the federation query engine

Figure 3 shows a screen shot of the prototypical query engine. Using the query engine, users can pose queries in the *Query* tab and execute the query by pressing the button with a red flash. The *Query plan* tab in the middle shows the execution plan of the posed query. To its right is the *Query result* tab, where the result data is shown in table format. Messages indicating the processing progress of the posed query are shown in the bottom text box. The current prototype does not have a sophisticated interface, and is only used for experimental purposes. The core techniques will later be integrated with the business analysis products of a Danish Business Intelligence (BI) tool vendor, TARGIT [8].

4.2 Query Processing

The query analyzer parses the query posed in the query tab (the same query as the example in Section 2) and generates the initial logical query plan below. The DECORATION operator instantiates the level expression and builds a virtual dimension, which consists of the population data to decorate the associated suppliers' nations through Nlink. The SELECTION operator slices the cube using the population data, and finally the PROJECTION operator rolls up the levels of dimensions to Brand, Population, and All (the top level of the dimensions not referenced in the SELECT clause), which leaves only the Parts and Suppliers dimensions in the federation, and produces the final aggregate results. Since the virtual dimension containing the decoration data (e.g., population) can only be constructed in the temporary component, in the actual evaluation process, the selection and projection operations referencing the decoration data are also performed in the temporary component.

```
PROJECTION [SUM(Quantity),
|                  Brand, Nation/Nlink/Population]
|—-SELECTION [Nation/Nlink/Population<30]
|————DECORATION [Nation/Nlink/Population]
```

The initial plan is then passed on to the optimizer and yields the optimized execution plan below (also shown in the query plan tab in Figure 4), which is integrated with the inlining optimization and aims to use the OLAP and XML components as much as possible so as to reduce the intermediate data transfered to the temporary component. The predicate in the SELECTION operator with a special mark first yields the inlining process, which evaluates the predicate in the XML component and generates a new predicate using the satisfying nation values (e.g., Nation='IRAQ' OR Nation='PERU' OR Nation='KENYA'). Then, the SELECTION and PROJECTION operators slice and aggregate the OLAP cube using the new predicate, which reduces the OLAP cube significantly and leaves only the Parts and Suppliers dimensions in the cube. The DECORATION operator builds the virtual dimension. The final PROJECTION operator further rolls up the dimensions, Parts, Suppliers, and the virtual dimension, to Brand, All, and Population, respectively, and aggregates the measures, which produces the same results as the PROJECTION in the initial plan, but is much faster based on the significantly reduced cube by the lower PROJECTION operator in the same plan.

```
PROJECTION [SUM(Quantity),
|                  Brand, Nation/Nlink/Population]
|—DECORATION [Nation/Nlink/Population]
|——PROJECTION [SUM(Quantity), Brand, Nation]
|————SELECTION [(Nation/Nlink/Population<30)′]
```

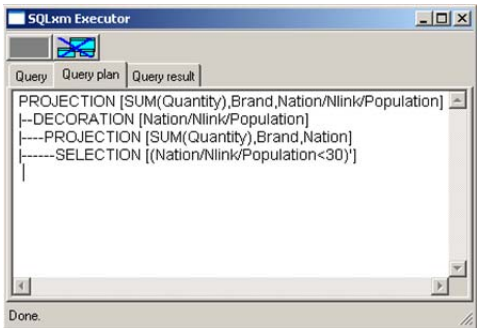

Figure 4: The query plan tab

When the above plan is passed on to the evaluator, it is processed as follows. The inlining optimization uses INSERT INTO statements, which are extended with an OPENXML function mapping an XML document into a table through a schema definition. The following query is issued against the example XML document in Figure 1.

```
INSERT INTO   tmp_pop
SELECT        DISTINCT *
FROM          OPENXML(@hdoc, '/Nations/Nation',2)
              WITH( NationName   varchar(25),
                    Population   float)
WHERE         NationName IN ('China',..., 'Kenya')
              AND Population <30
```

NationName	Population
IRAQ	23.1
KENYA	29.3
PERU	27.1

Table 1: tmp_pop

In the above query, the temporary table, tmp_pop (shown in Table 1), contains the satisfying nation names and their population data. The IN predicate in the WHERE clause ensures that only the XML nodes referenced in Nlink are selected and $@hdoc$ is the handle of the document.

Quantity	Brand	Nation
1865	Brand#11	IRAQ
1757	Brand#11	PERU
629	Brand#11	KENYA

Table 2: Example tmp_facts

The rewritten predicate by inlining is "Nation='IRAQ' OR Nation='PERU' OR Nation='KENYA'." For the SELECTION and PROJECTION operators, the evaluator issues the following query against the OLAP component, which transfers the results to the temporary component in the table, tmp_facts (partially shown in Table 2).

```
SELECT   *
INTO     tmp_facts
FROM     OPENQUERY( OLAP_SVR,
           SELECT    SUM(Quantity), Brand, Nation
           FROM      TC
           WHERE     Nation='IRAQ' OR
                     Nation='PERU' OR
                     Nation='KENYA'
           GROUP BY  Brand, Nation)
```

The DECORATION operator uses the table, tmp_pop, as the dimension table for the virtual dimension, since the population and the decorated nations are already available in one table. The final query that produces the query results in the temporary component is below. Figure 5 shows that the query results are displayed in the Query result tab of the query engine.

```
SELECT     SUM(Quantity), Brand, Population
INTO       tmp_results
FROM       tmp_facts t1, tmp_pop t2
WHERE      t1.Nation=t2.NationName
GROUP BY   Brand, Population
```

5 Conclusion

In this demo, we have explained the logical "OLAP-XML federation system" and shown the federation query optimization and processing techniques with a concrete example on an OLAP-XML query engine. We believe that we are the first to develop a full set of techniques for logical federation of OLAP and XML data sources.

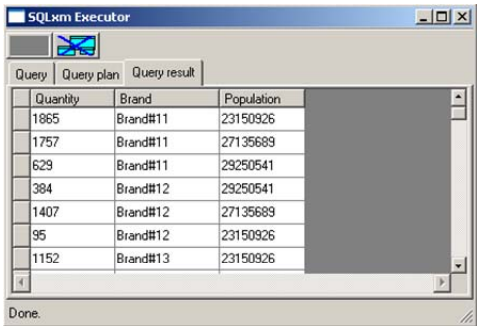

Figure 5: The query result tab

Acknowledgements

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

References

[1] J. Clark and S. DeRose. XML Path Language (XPath). www.w3.org/TR/xpath. Current as of Oct. 13, 2006.

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

[3] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, 2002.

[4] Lahiri T. et al. Ozone: Integrating Structured and Semistructured Data. In *Proc of DBLP*, pages 297–323, 1999.

[5] Microsoft corp. SQLXML. *Books Online*, version 5.2.3790.

[6] Microsoft corp. Supported SQL SELECT Syntax. *Books Online*, version 5.2.3790.

[7] Microsoft corporation. Com: Component object model technologies. http://www.microsoft.com/com. Current as of Oct. 13, 2006.

[8] D. Pedersen and T. B. Pedersen. Integrating XML data in the TARGIT OLAP system. In *Proc of ICDE*, pages 778–781, 2004.

[9] Transaction Processing Performance Council. TPC-H. www.tpc.org/tpch. Current as of Oct. 13, 2006.

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