Integrating XML Data in the TARGIT OLAP System

Dennis Pedersen  Jesper Pedersen  Torben Bach Pedersen
TARGIT  TARGIT  Aalborg University
dp@targit.com  jesper_p@targit.com  tbp@cs.aau.dk

Abstract

This paper presents work on logical integration of OLAP and XML data sources, carried out in cooperation between TARGIT [4], a Danish OLAP client vendor, and Aalborg University. A prototype has been developed that allows XML data on the WWW to be used as dimensions and measures in the OLAP system in the same way as ordinary dimensions and measures, providing a powerful and flexible way to handle unexpected or short-term data requirements as well as rapidly changing data. Compared to earlier work, this paper presents several major extensions that resulted from TARGIT’s requirements. These include the ability to use XML data as measures, as well as a novel multigranular data model and query language that formalizes and extends the TARGIT data model and query language.

1. Introduction

The Danish OLAP vendor TARGIT’s practical experience with the OLAP industry and their expectations for its future development [1] has provided valuable input to earlier work [2], resulting in several important theoretical extensions and refinements as well as the development of a prototype closely integrated with TARGIT’s current product.

The most significant extension is the ability to define logical measures from external XML data. The ability to use external XML data as dimensions [2] has been adapted to the TARGIT framework. Another major contribution is the formalization and extension of TARGIT’s internal data model and query language. Unlike traditional multidimensional data models and query languages, the ones presented here are multigranular, i.e., they allow fact data to have varying granularity over a dimension. The extended architecture and query processing issues are also described. Experiments with the prototype indicate an acceptable overhead for using external dimensions and measures for a large range of queries. We believe these contributions to be novel and interesting to both the database research and industry communities. Further details can be found in [3].
set, and the occurrence of lower/higher level aggregate values may even vary over a dimension. The latter feature distinguishes the TARGIT model and language from traditional multidimensional models such as the well-known SQL CUBE operator. The examples below are based on a case study concerning B2B portals [2].

The OLAP data model defines 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. The data models allows that a lower-level value may roll up to more than one higher-level value, i.e., dimensions may be non-strict [2]. Problems associated with this, such as incorrect aggregation is handled using so-called aggregation types [3]. Intuitively, a tuple in a fact table \( F \) captures the measure values associated with one combination of dimension values, not necessarily from the bottom levels, i.e., fact table can also contain higher-level aggregates.

Example 2.1 From the case study we construct a three-dimensional cube, Purchases, with the dimensions, levels, and ordering of dimension values depicted in Figure 2. The symbols denote all of a dimension. An example fact table, with higher-level aggregate values, i.e., multigranular data, is seen in Table 1.

The cube algebra for the OLAP data model contains two operators: generalized projection and selection. Formal definitions are given in the full paper [3].

The generalized projection (GP) operator \( \Pi \) aggregates measure values in a cube such that the resulting fact table contains a specified set of dimension value combinations.

Any measures or dimensions that are not specified, are not part of the resulting cube. The operator’s parameters are a set of dimension values for each dimension in the result and a set of measures. The bottom-most dimension values specified for each dimension defines the bottom levels of the resulting cube. If no dimension values are listed for a dimension, this dimension is aggregated to the top level and removed from the cube. Each new measure value is calculated by applying the given aggregate function to the corresponding measure value for all tuples in the old fact table that contain bottom values that roll up to the dimension values of the new fact table. Rolling up to a higher level may result in duplicated facts if the hierarchy is non-strict, i.e. if a dimension value has more than one parent. To ensure correct aggregation, we explicitly check for this in each dimension [2] and prohibit further aggregation if necessary.

The selection operator \( \sigma \) is used to slice the cube so that it contains only facts that satisfy a given predicate. The predicates we consider here are restricted to the subset of the usual SQL operators that are allowed in the TSQL language, namely \( =,\prec,\succ,\prec\succ \), and \( \text{IN} \) which can be combined with \( \text{AND} \). (OR is not used in TSQL predicates.) These operators can be used to compare constants and levels. A selection only affects the tuples in the fact table. Hence, selection returns a cube with the same fact type and the same set of dimensions. All tuples for which the predicate does not hold are removed, i.e. their measures are set to NULL. For any tuples that are computed from other tuples the measure values are recalculated to reflect the lower-level changes.

An example query in TARGIT’s TSQL language is given below (the syntax is modified slightly for readability).

```
SELECT [Suppliers], [ECs], [Units], [Cost]
FROM [Purchases]
WHERE ([ECs],[EC] in ('EC1', 'EC3', 'EC4')
DRILL DOWN ([Suppliers],[US],CHILDREN,
[Suppliers],[UK],CHILDREN)
```

A TSQL query consists of a SELECT-FROM-WHERE-DRILL DOWN construct. The SELECT clause lists the dimensions and measures that should be part of the result, while the FROM clause specifies the cube. The optional WHERE clause is used to slice the cube by specifying a predicate over the dimension values. DRILL DOWN lists the set of dimension values that should be present in the result using the DimValue, CHILDREN notation, which means the set of all dimension values that are children of DimValue.

### 3. Integration with External XML Data

External data may be used as either dimension or measure values in a cube. Using external data as dimension values allows the existing cube data to be grouped in new ways, e.g., purchases may be grouped by the city in which the sup-
plier is located, even if this information is not available in the cube but only in an external XML document.

An external dimension is defined to be a relationship between dimension values from a single level in a cube and nodes in an XML document, based on the XPath data model. This is similar to, but more flexible than, the property concept used in many commercial OLAP systems, where a descriptive value is attached to each dimension value. The reason external dimensions are more flexible, is that they allow more than one external value per dimension value, which is usually not possible with properties. As we will see later, a cube can be “decorated” with the information represented by an external dimension, creating a new cube in which the external dimension values appear as a new dimension. External dimensions are specified by giving an XPath expression in which the cube’s level names can be used as variables. For example, the expression /SupplierCities/Supplier [SupplierName = $Supplier]/City identifies the cities in which a supplier is located given the supplier’s name.

An important problem when using external values is non-strictness, i.e. what to do when there are more than one external node for each dimension value. Several different semantics are appropriate in different situations, but the following have been found to cover most applications: using all of the nodes as distinct values, concatenating all the nodes into a single value, and picking an arbitrary node.

Rather than redefining the existing operators to work also on external dimensions, we add a new operator $\delta^m$ that merges a cube and an external dimension into a new cube. The operator is called decoration since it “decorates” a dimension in a cube with additional information.

Extending cubes with new external measures (extension) is harder than with dimensions, since a measure depends on all the existing dimensions. External measures are only allowed to contain values for a single combination of levels because of the semantical problems caused if there is inconsistency between the external values. For example, if the external data specifies non-strict hierarchies, there can be several ways to compute a higher-level aggregate, each resulting in different but equally (in)valid aggregate values. It is possible to handle this problem, e.g. by checking the consistency of the external data before using it, but that is outside the scope of this work. However, we do not require the external measure to be specified for the bottom combination of levels, which means that external values at different levels can be handled by creating multiple independent external measures. Since external measures need not be specified for the bottom level, the measure values may be missing for some combinations of levels. This is similar to the behavior of the drill-across operation for joining multiple cubes, where some measures may not be defined for all combinations of levels in the result. We define an external measure to be a relation between dimension values for a combination of levels and a single external value [3]. External measures are specified similarly to external dimensions. However, it is necessary to define values for a combination of levels instead of just for one level. If there are more than one value for each combination of dimension values, they must be transformed to a single value, typically by aggregating them or by selecting one of the values.

The actual extension of a cube with the new measure is performed by the extension operator $\delta^m$. When extending a cube with an external measure, only the existing facts are extended. Extension only affects the fact table. Intuitively, each fact is extended with any external information that is available, either directly from the external measure or by calculating it from lower-level values in the external measure. If this is not possible, the measure value is NULL.

TSQL’s SELECT construct is changed so that external dimensions and measures can now be referenced in the SELECT clause, and levels in external dimensions can be used in the WHERE clause. Notice that dimension values in external dimensions cannot be used in the DRILL DOWN clause. This is because external dimensions only have a single level below the top level (apart from the bottom level which is always identical to the bottom level of the dimension being decorated) and both of these levels are implicitly part of the DRILL DOWN clause. In addition, two CREATE statements, and corresponding DROP statements, are defined to provide a practical way to create and drop new dimensions and measures in a cube based on external XML data. A CREATE EXTERNAL DIMENSION statement specifies an external dimension, while the CREATE EXTERNAL MEASURE specifies an external measure. Of course, since the data is located externally, the data is not retrieved and the dimensions or measures are not created until a query is actually evaluated. The values specified in each of the statements correspond closely to the elements of the dimension and measures specifications. We will show the syntax by the two examples below. Here, SupplierNameMappings refers to the name of a table containing supplier-alias pairs.

```
CREATE EXTERNAL DIMENSION SupplierCities
IN CUBE Purchases FROM Cities.xml
IDENTIFIED BY /SupplierCities/Supplier[SupplierName = $Supplier]/City USING Supplier AS $Supplier
WITH ALIASES SupplierNameMappings
WITH SEMANTICS CONCAT

CREATE EXTERNAL MEASURE OverheadExpenses
IN CUBE Purchases FROM OverheadExpenses.xml
IDENTIFIED BY /OverheadExpenses/Supplier[@Name=$Supplier]/EC[@Name=$EC]
USING (Supplier AS $Supplier, EC AS $EC)
WITH SEMANTICS SUM
```
Example 3.1 The CREATE statements above create the external dimension seen in Figure 3 (note the non-strictness), and the fact table shown in Table 1. □

Figure 3. The new SupplierCities dimension

Table 1. Fact table with external OE measure

<table>
<thead>
<tr>
<th>Cost</th>
<th>Units</th>
<th>OE</th>
<th>Suppliers</th>
<th>ECs</th>
</tr>
</thead>
<tbody>
<tr>
<td>9800</td>
<td>3000</td>
<td>200</td>
<td>A.A., Supplier</td>
<td>FF Class</td>
</tr>
<tr>
<td>9400</td>
<td>3000</td>
<td>800</td>
<td>B.B., Supplier</td>
<td>FF Class</td>
</tr>
<tr>
<td>14400</td>
<td>4000</td>
<td>400</td>
<td>C.C., Supplier</td>
<td>FF Class</td>
</tr>
<tr>
<td>16700</td>
<td>5000</td>
<td>1000</td>
<td>C.C., Supplier</td>
<td>G Class</td>
</tr>
<tr>
<td>19200</td>
<td>6000</td>
<td>1000</td>
<td>US Country</td>
<td>FF Class</td>
</tr>
<tr>
<td>14400</td>
<td>4000</td>
<td>400</td>
<td>UK Country</td>
<td>FF Class</td>
</tr>
<tr>
<td>16700</td>
<td>5000</td>
<td>1000</td>
<td>UK Country</td>
<td>G Class</td>
</tr>
<tr>
<td>33600</td>
<td>10000</td>
<td>1400</td>
<td>T T Supplier</td>
<td>FF Class</td>
</tr>
<tr>
<td>16700</td>
<td>5000</td>
<td>1000</td>
<td>T T Supplier</td>
<td>G Class</td>
</tr>
<tr>
<td>9800</td>
<td>3000</td>
<td>200</td>
<td>A.A., Supplier</td>
<td>T T EC</td>
</tr>
<tr>
<td>9400</td>
<td>3000</td>
<td>800</td>
<td>B.B., Supplier</td>
<td>T T EC</td>
</tr>
<tr>
<td>31100</td>
<td>9000</td>
<td>1400</td>
<td>C.C., Supplier</td>
<td>T T EC</td>
</tr>
<tr>
<td>19200</td>
<td>6000</td>
<td>1000</td>
<td>US Country</td>
<td>T T EC</td>
</tr>
<tr>
<td>31100</td>
<td>9000</td>
<td>1400</td>
<td>UK Country</td>
<td>T T EC</td>
</tr>
<tr>
<td>50300</td>
<td>15000</td>
<td>2400</td>
<td>T T Supplier</td>
<td>T T EC</td>
</tr>
</tbody>
</table>

Architecture: The overall architecture is shown in Figure 1(b) with an indication of which technologies are used in the prototype. As current OLAP systems do not allow online addition of new dimensions and measures, the Cube TSQL Evaluator processes TSQL queries by fetching data from the OLAP and XML components and combining it in a temporary component. A relational DBMS is used for the temporary component because the result can easily be computed by joining the fact table resulting from the OLAP component query, and tables containing the external data. Another auxiliary component stores metadata used in the evaluation of a TSQL query such as specifications of the external dimensions and measures. Both the temporary and the metadata component use the MS SQL Server DBMS.

Query processing: Given this overall architecture, the evaluation of a TSQL query is performed in these main steps: 1) split the TSQL query into a pure OLAP query and a set of XML queries; 2) evaluate these queries in parallel; 3) join the OLAP fact table and the tables containing external data as specified by the decoration and extension operators. 4) perform any additional grouping and/or selection on the combined result. However, to avoid excessive transfer of temporary data, we employ a number of optimizations techniques. First, we cache some of the otherwise temporary tables produced during the evaluation of a TSQL query. The caching strategy is based on assigning a score to each of the temporary tables and using this to determine which tables to expunge when the cache fills up. Second, the optimal evaluation strategy, depends on, e.g., the order in which decorations and extensions are performed. Generally, extension is performed before decoration which means that the dimension table is only joined with the combined fact table, rather than with both the OLAP result and the external measure table. To decide the order of decorations and extensions, the cost of each approach is estimated using a cost model consisting of three submodels, one for the OLAP component, one for the XML components, and one for the temporary component. Third, external predicate inlining [2] is used to evaluate selection predicates with references to external dimensions more efficiently by transforming the TSQL predicate such that it can be evaluated in the cube query without actually decorating with the external dimension. The basic idea is to evaluate the original predicate on the external dimension table, getting back a set of literal data values, and construct a new predicate that refers only to these literal values. For example, a predicate “[SupplierCities] IN (‘Los Angeles’, ‘New York’)” is translated to “[Supplier] IN (‘A.A.’, ‘B.B.’) which can be evaluated entirely in the OLAP component, resulting in better performance. We have performed a set of experiments with the prototype, using TPC-H data. The experiments showed that for many queries, our federated approach is a viable alternative to physical integration. Especially the inlining and caching techniques provided dramatic performance gains.

4. Conclusion and Future Work

We have presented a flexible extension to the TARGIT OLAP system that allows dimensions and measures to be based on XML data, e.g., on a Web-page. These dimensions and measures can then, transparently to the end user, be used as ordinary dimensions and measures in the cube.

Future work will focus on evaluating and improving the prototype in order to include it in the TARGIT Analysis product, which is expected to happen over the next few releases. Another interesting direction could be to let the system choose between logical and physical integration of external data based on the usage pattern, the source’s update frequency, the amount of processing required to perform the physical integration etc.

References