

XML Views

Murali Mani
Computer Science Dept, WPI
mmani@cs.wpi.edu

SYNONYMS

XML Publishing

DEFINITION

Database applications provide an XML view of their data so that the data is available to other applications, especially web applications. Database systems provide support for the client applications to use (query and/or manipulate) the data. The operations specified by the client applications are composed with the view definitions by the database system, thus performing these actions. The internal data model used by the database application, as well as how the operations are performed are transparent to the client applications; they see only an XML view of the entire system. XML views help the database systems to maintain their legacy data, as well as utilize the optimization features present in legacy systems (especially SQL engines), and at the same time make the data accessible to a wide range of web applications.

HISTORICAL BACKGROUND

Views (external schema) are a feature [12] present universally in almost all database systems. Views provide data independence as well as the ability to control access of portions of data to different classes of users. With XML [2] becoming the standard for information exchange over the web since 1998, database applications have used XML views to publish their data and to make the data accessible to web applications. Nowadays, XML views are supported by most major database engines like Microsoft SQL server, Oracle and IBM DB2.

SCIENTIFIC FUNDAMENTALS

The views that database systems support can either be virtual or materialized [12]. When the view is virtual, only the view definition is stored in the system. Whenever a client application accesses the data by issuing a query over the view, the database system composes the user query with the view definition and this combined query is executed over the underlying data. The advantage of virtual views are that the data is never out-of-date as the data is stored, accessed from and manipulated in only one place. Materialized views on the other hand store the data for the view along with the view definition. Therefore the same data is now in more than one

location, and thus the different copies of the data need to be kept consistent by maintaining the materialized views whenever the underlying data changes. Incremental and efficient maintenance of materialized views have been studied [6, 11, 9] and are supported by most commercial SQL engines. The advantage of materialized views is that the user query can potentially be answered faster as the user query can be directly answered from the materialized view, and does not require composing it with the view definition. In this article, we will consider both virtual and materialized XML views that are defined primarily over relational data, the state-of-the-art techniques, and open problems.

Mapping between the XML view and the underlying data

The mapping between the XML view and the underlying data is used for publishing the data, as well as for performing the user requested actions (such as answering queries) when the view is virtual. Different ways of specifying mappings between the XML view and the underlying data are possible. The *canonical mapping* [13] is a very simple one where there is a 1-1 mapping between the relational tuples and the XML elements in the view. An XML element is constructed for every row in every table in the relational database. Thus the entire data in the relational database is captured in this canonical XML view. Slightly more complex mappings that still capture the entire relational data in the XML view are studied in [8], where the key-foreign key constraints are used to nest XML elements within each other. The translation of queries (especially navigation queries as in XPath) in the above mapping schemes is quite straight forward.

However, often times, a database application needs to publish their data as an XML view that conforms to a standard schema. In such cases, we need a more complex mapping scheme as we might not need all of the underlying data, also the underlying data might need to be restructured to conform to the schema. In [10], the authors study how the user can specify relationships between the underlying schema and the view schema diagrammatically. Based on some assumptions, the system then analyzes the user specified relationships and translates them into meaningful mappings that can be understood by the system (such as SQL queries). Also the user is consulted when there is potential ambiguity. In [13, 5], the mapping is specified using XQuery language [15]. This is similar to the scenario that is well-understood by SQL engines (where view definitions are specified in SQL). The database administrator can specify the XML view using an XQuery expression that conforms to the required schema.

User Queries over XML Views

The systems that support XML views need to provide the capability for users to query the data. In [13, 5], the authors study how the user queries specified using XQuery can be answered efficiently in the scenario where the XML view is also specified using XQuery. The architecture for such a system as described in [13] is shown in Figure 1. One of the main assumptions made by these systems is that the SQL engine is best equipped to handle computations efficiently (those computations that an SQL engine can handle); therefore one needs to push down as much computation as possible within the SQL engine. The view composer shown in Figure 1 composes

the user query with the view query. The computation pushdown module then re-arranges the combined query plan so that the SQL portion is at the bottom of the query plan, and it includes everything that can be done by the SQL engine. In such a case, the middle ware only needs to do tagging and this is done in a single pass over the data returned by the SQL engine.

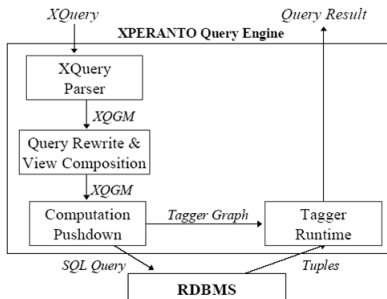


Figure 1: Typical Architecture for Processing User Queries over XML Views

User updates over XML Views

While a lot of work has focussed on how to answer user queries over XML views, very little work has focussed on handling user updates over virtual XML views. As the view is virtual, the view update needs to be performed by updating the base data in such a way that the effect expected by the user is achieved. A common semantics used for view updates is the side-effect free semantics shown in Figure 2, as described in [3, 7]. In the figure, D represents the database instance, DEF_v represents the view definition, V is the view instance, u is the user specified view update. $u(V)$ represents the effect that the user wants to achieve on the view, the view update problem therefore requires us to find an update U over the base data such that the user desired effect is achieved on the view. It is possible that there does not exist such an update U , in which case the view update cannot be performed. In some cases, there could be a unique U , and in other cases, it is possible that there exists multiple such updates over the base data, in which case the ambiguity needs to be resolved using heuristics, by the user, or by rejecting the view update.

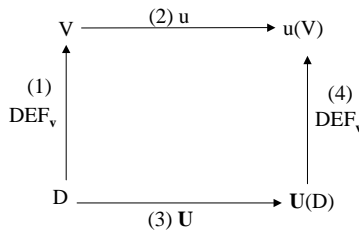


Figure 2: Illustrating side-effect free semantics

Updating SQL views itself is considered a hard problem, and the existing solutions handle only a subset of the view definitions. When a user specifies updates over view definitions that use

”non-permissible” operators (such as aggregation operators), the system rejects these updates. Most solutions, including commercial ones, use a schema level analysis to perform/reject the view update, where they utilize the base schema, the view definition and the user specified update statement. Some solutions examine the base data also, in which case fewer view updates need to be rejected.

The main approaches that study updates over XML views of relational databases include [1, 14]. In [1], the authors translate the XML view into a set of SQL views; now the solutions for SQL views can be utilized. In [14], the authors identify that the XML view update problem is harder than the SQL view update problem – SQL view update problem boils down to the case where the XML view schema has only one node. For a general XML view, given an update (such as delete) to be performed on an XML view element, the authors partition the XML view schema nodes into three categories – for one of the categories, the authors utilize the results from the SQL view update research, for the other two categories, the authors propose new approaches to check for side-effects. As follow up work to [14], the authors have studied how data level analysis can be used for the XML view update problem as well.

Maintenance of Materialized XML Views

In order to keep materialized views consistent with the underlying base data, one approach is to recompute the view every time there is a base update. However, this approach is not efficient as the base updates are typically very small compared to the entire base data. It would be efficient if we could instead perform *incremental view maintenance*, where only the update to the view is computed. Incremental view maintenance consists of typically two steps – in the *propagate* phase, the delta change to the view is computed using a *incremental maintenance plan*, and in the *apply* phase, the view is refreshed using this delta change to the view.

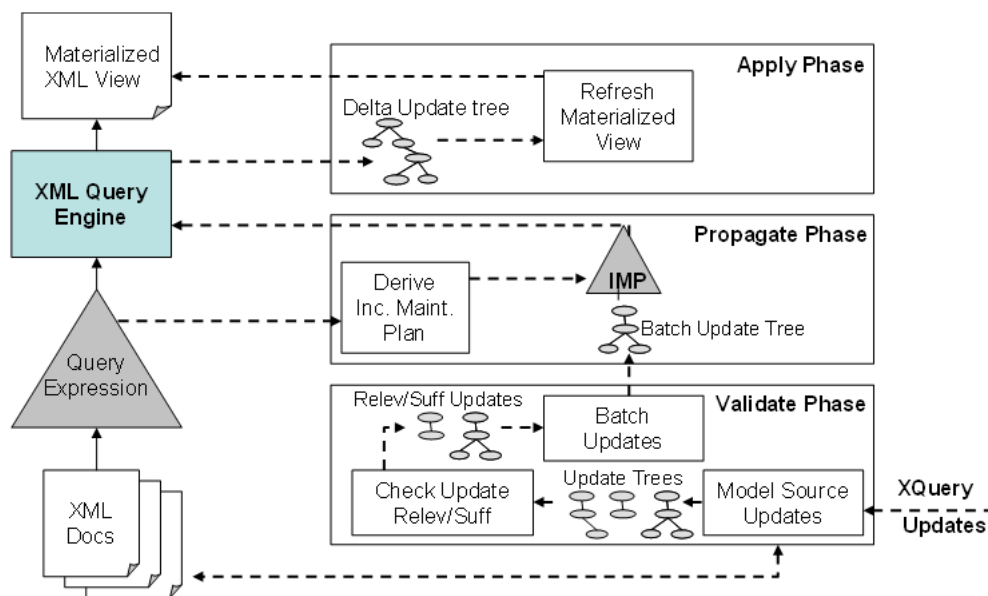


Figure 3: Architecture for Maintaining Materialized XML Views

Incremental maintenance of XML views is more complex than maintenance of SQL views, because of the more complex features of XML including nested structure and order, and of XML query languages such as XQuery. In [4], the authors study how to incrementally maintain XML views defined over underlying data sources that are also XML (note that the solutions apply to the case where the underlying data sources are relational as well). The architecture of their approach is shown in Figure 3. As the underlying base is XML, the base updates can come in many different granularities – this requires a *validate phase* which combines the update with the base data to make it a complete update. This is then fed to the incremental maintenance plan in the *propagate phase*, which computes the delta change to the view. In the *apply phase*, the view is refreshed using the delta change to the view.

KEY APPLICATIONS

XML views are useful to any database application that wishes to publish their data on the web.

FUTURE DIRECTIONS

XML views are already being used widely by database applications, and their usage is expected to increase further in future. There are still several issues that need to be studied to make such systems more efficient. For processing queries, query composition will result in several unnecessary joins. Therefore the query optimizer must be able to remove unnecessary joins, this requires the query optimizer to be able to infer key constraints in the query plan. For processing view updates, a combined schema and data analysis promises to be an efficient approach and needs to be investigated. Incremental view maintenance further requires new efficient approaches for handling aggregate operations in XQuery languages.

CROSS REFERENCE

XPath/XQuery, XML query processing, XML integration, XML publishing, XML updates

RECOMMENDED READING

- [1] V. P. Braganholo, S. B. Davidson and C. A. Heuser, From XML View Updates to Relational View Updates: Old Solutions to a New Problem, VLDB Conference, 2004
- [2] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler and F. Yergeau, Extensible Markup Language (XML) 1.0, W3C Recommendation, <http://www.w3.org/XML>
- [3] U. Dayal and P. A. Bernstein, On the Correct Translation of Update Operations on Relational Views, ACM Transactions of Database Systems, 7 (3), September 1982, 381 – 416
- [4] M. El-Sayed, E. A. Rundensteiner and M. Mani, Incremental Maintenance of Materialized XQuery Views, IEEE ICDE, 2006
- [5] M. Fernandez, Y. Kadiyska, D. Suciu, A. Morishima, and W-C. Tan, SilkRoute: A Framework for Publishing Relational Data in XML, ACM Transactions of Database Systems, 27 (4), December 2002, 438 – 493

- [6] T.Griffin and L. Libkin, Incremental Maintenance of Views with Duplicates, ACM SIGMOD, 1995
- [7] A. M. Keller, Algorithms for Translating View Updates to Database Updates for Views Involving Selections, Projections and Joins, ACM PODS, 1985
- [8] D. Lee, M. Mani, F. Chiu and W. W. Chu, NeT & CoT: Translating Relational Schemas to XML Schemas using Semantic Constraints, ACM CIKM, 2002
- [9] T. Palpanas, R. Sidle, R. Cochrane and H. Pirahesh, Incremental Maintenance for Non-Distributive Aggregate Functions, VLDB Conference, 2002
- [10] L. Popa, Y. Velegakis, R. J. Miller, M. A. Hernandez and R. Fagin, Translating Web Data, VLDB Conference, 2002
- [11] D. Quass, Maintenance Expressions for Views with Aggregates, Views Workshop, 1996
- [12] R. Ramakrishnan and J. Gehrke, Database Management Systems, McGraw Hill, 2002.
- [13] J. Shanmugasundaram, J. Kiernan, E. Shekita, C. Fan and J. Funderburk, Querying XML Views of Relational Data, VLDB Conference, 2001
- [14] L. Wang, E. A. Rundensteiner and M. Mani, Updating XML Views Published over Relational Databases: Towards the Existence of a Correct Update Mapping, Document and Knowledge Engineering Journal, 58(3), 2006, 263 – 298
- [15] W3C XQuery Working Group, <http://www.w3.org/XML/Query/>