Information Systems 56 (2016) 198–213

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

S2CX: From relational data via SQL/XML to (Un-)Compressed XML

Stefan Böttcher, Rita Hartel*, Dennis Wolters

University of Paderborn, Department of Computer Science, Fürstenallee 11, D-33102 Paderborn, Germany

ARTICLE INFO

Article history:
Received 26 August 2015
Accepted 3 September 2015
Available online 27 October 2015

Keywords:
SQL/XML
XML compression
XML generation from relational databases.

ABSTRACT

The gap between storing data in relational databases and transferring data in form of XML has been closed e.g. by SQL/XML queries that generate XML data out of relational data sources. However, only few relational database systems support the evaluation of SQL/XML queries. And even in those systems supporting SQL/XML, the evaluation of such queries is quite slow compared to the evaluation of SQL queries. In this paper, we present S2CX, an approach that allows to efficiently evaluate SQL/XML queries on any relational database system, no matter whether it supports SQL/XML or not. As a result to an SQL/XML query, S2CX supports different output formats ranging from plain XML to different compressed XML representations including a succinct encoding of XML data, schema-aware compressed XML to grammar compressed XML. In many cases, S2CX produces compressed XML as a result to an SQL/XML query even faster than the evaluation of SQL/XML queries into non-compressed XML as provided by Oracle 11 g and by DB2. Furthermore, our approach to query evaluation scales better, i.e., the larger the dataset, the faster is our approach compared to SQL/XML query evaluation in Oracle 11 g and in DB2.

1. Introduction

XML has emerged to a standard for data transmission in internet-based systems, even when the data source is data stored within a relational database. This leads to an increased data volume for data transmission and requires a possibility to create XML data based on relational data as e.g. described in the SQL/XML standard, which is supported by only a few database systems.

We have developed S2CX, an approach that supports SQL/XML queries on every SQL database system. Furthermore, S2CX can provide the SQL/XML query result not only as traditional uncompressed XML documents, but also directly in compressed XML format, which reduces data transmission volume to remote destinations. Even more, S2CX supports different compressed XML representations as output formats, i.e., beyond [1] S2CX can now also generate grammar compressed XML, i.e. the most strongly compressed updateable XML format.

S2CX uses the SQL/XML syntax to describe the transformation of relational data to XML data, but does not require a relational database system to support SQL/XML. Based on a given SQL/XML query, our approach extracts the generated XML structure and derives a plain SQL query for collecting the relevant data. Using this generic XML representation consisting of the plain SQL query result and the extracted XML structure, we can create different XML representations, as e.g. plain XML, a SAX event stream, but also different compressed XML formats. The compressed XML formats that can be generated by S2CX range from encoding-based compression formats generated by

* Corresponding author.
E-mail addresses: stb@uni-paderborn.de (S. Böttcher), rst@uni-paderborn.de (R. Hartel), dennis.wolters@uni-paderborn.de (D. Wolters).

http://dx.doi.org/10.1016/j.is.2015.09.011
0306-4379/

Succinct XML [2], over schema-based compression formats generated by XSDS [3] to grammar-based compression formats generated by CluX [4], BPLEX [5], or TreeRePair [6]. Thereby, in scenarios like e.g. SOAP data transmission, product catalog transmission, or sensor data transmission, where transferring compressed XML is preferred in comparison to transferring plain XML, S2CX can save the additional XML compression step that would be required, if SQL/XML queries are executed on databases like Oracle 11 g or DB2 that can output uncompressed XML only.

Contributions

We present S2CX, an approach that fast and flexibly answers SQL/XML queries with the following properties:

- Given an SQL/XML query and a relational database, S2CX can generate the query result in different XML formats: as SAX event stream or in different compressed XML formats, like e.g. the encoding-based Succinct format, the schema-based XSDS format or the grammar-based format of CluX, BPLEX, and TreeRePair.
- From a given SQL/XML query, we derive a generic XML representation consisting of the tree structure of the XML result that can be derived from the SQL/XML query and additional information and text data that is derived from the result of a plain SQL query that is evaluated on the underlying relational database. We use this generic representation to generate either XML or SAX events or different compressed XML formats, like e.g. Succinct, XSDS or CluX/BPLEX/TreeRePair.
- Our performance evaluations show that our approach generates compressed XML faster than evaluating an SQL/XML query and compressing its result.
- In most cases, our approach generates compressed XML even faster than Oracle or DB2 can generate uncompressed XML. Even more, in a majority of cases, our approach together with an additional decompression step generates uncompressed XML faster than evaluating SQL/XML queries directly with Oracle 11 g or DB2. Additionally, our approach scales better, such that S2CX's performance speed-up compared to the Oracle 11 g and the DB2 implementations becomes larger with increasing database size.
- Finally, as our approach does not need an SQL/XML query engine, it provides a useful technique to answer SQL/XML queries also on every SQL database system that does not support the SQL/XML standard.

2. The concept

2.1. Considered subset of SQL/XML

In this paper, we consider the "raw" XML data and ignore additional XML constructs like comments or processing instructions. Therefore, we restrict the XML publishing functions of SQL/XML to the functions described in Table 1. These publishing functions are used in combination with the SQL elements SELECT, FROM, WHERE, ORDER BY and GROUP BY to form SQL/XML queries. We allow the queries to be nested at any extent.

2.2. Our example

To describe our approach, we use the following minimized example consisting of the relational database shown in Fig. 1, which provides the input data for the SQL/XML query given in Fig. 2 in order to generate the output XML document given in Fig. 3.

A subquery can be embedded into an SQL/XML query, as long as this subquery returns a single value as a result. An example for a subquery can be seen in the lines 6–13 of Fig. 2. This subquery determines the employee assigned to a given project.

Executing the query of Fig. 2 on the example database of Fig. 1 results in the XML document displayed in Fig. 3. For each project in the example database, a corresponding element is created (see lines 2–7 in Fig. 3). First, the project "Invisibility", line 2 in Fig. 3, is retrieved by the query because of the ORDER BY clause in line 16 of Fig. 2. The subquery in lines 6–13 of Fig. 2 creates the employee of line 3 in Fig. 3. There is no "comment" element for this project because the value of the column "Comment" is *null*, and therefore, the XMLFOREST function does not

Table 1
Considered XML publishing functions of SQL/XML.

SQL/XML function	Description
`XMLELEMENT`	Creates an XML element. The XMLELEMENT function has the element name as first parameter, the attribute list, i.e. an XMLATTRIBUTES function call, as an optional second parameter, and the content as a list of optional further parameters.
`XMLATTRIBUTES`	Creates XML attributes. The XMLATTRIBUTES function has a list of attributes as parameters, where the first part of each attribute is the column name from where to read the attribute value and the second part is the attribute name.
`XMLFOREST`	Creates a forest of XML trees. The XMLFOREST function has a list of its contents as parameters, where the first part of each parameter is the content definition and the second part is the label of the tag by which the content is surrounded.
`XMLCONCAT`	Combines a list of individual XML values to create a single value containing an XML forest.
`XMLAGG`	Aggregates multiple rows, each containing a single XML value, to create a single value containing an XML forest. The XMLAGG function has a call to an XML publishing function as first parameter and an ORDER BY clause that defines the order of its content as optional second parameter.