Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Systems 30 (2005) 317–332
www.elsevier.com/locate/infosys

Towards practical meta-querying ☆

Jan Van den Bussche[a], Stijn Vansummeren[a,1,*], Gottfried Vossen[b]

[a] *Limburgs Universitair Centrum, Universitaire Campus, B-3590 Diepenbeek, Belgium*
[b] *University of Muenster, D-48159 Muenster, Germany*

Received 31 March 2004; accepted 13 April 2004

Abstract

We describe a meta-querying system for databases containing queries in addition to ordinary data. In the context of such databases, a meta-query is a query about queries. Representing stored queries in XML, and using the standard XML manipulation language XSLT as a sublanguage, we show that just a few features need to be added to SQL to turn it into a fully-fledged meta-query language. The good news is that these features can be directly supported by extensible database technology.

Keywords: Databases; Meta-querying; Stored queries; XML; XSLT; Extensible database technology

1. Introduction

Enterprise databases often contain not only ordinary data, but also queries. Examples are view definitions in the system catalog; usage logs or workloads; and stored procedures as in SQL/PSM or Postgres [1]. Unfortunately, these queries are typically stored as long strings, which makes it hard to use standard SQL to express *meta-queries:* queries about queries. Meta-querying is an important activity in situations such as advanced database administration, database usage monitoring, and workload analysis. Examples of meta-queries to a usage log are:

(1) Which queries in the log do the most joins?
(2) Which queries in the log return an empty answer on the current instance of the database?
(3) View expansion: replace, in each query in the log, each view name by its definition as given in the system catalog.
(4) Given a list of new view definitions (under the old names), which queries in the log give a different answer on the current instance under the new view definitions?

Query 1 is *syntactical:* it only queries the stored queries on the basis of their expressions. Query 2 is *semantical:* its answer depends on the results of dynamically executing the stored queries. Query 3

☆ Recommended by M. Lenzerini

*Corresponding author. Tel.: +32-11-268219; fax: +32-11-268299.
E-mail address: stijn.vansummeren@luc.ac.be (S. Vansummeren).
[1] Research Assistant of the Fund for Scientific Research - Flanders (Belgium).

0306-4379/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.is.2004.04.001

is again syntactical, but more so than query 1 in that it also performs syntactical transformations. Query 4 is syntactical and semantical together.

To express meta-queries, database administrators and other advanced users typically resort to a programming language like Perl, in combination with Dynamic SQL. It would be much nicer if they would not have to "leave" the database system and could express their meta-queries directly in Interactive SQL. Indeed, already in 1993, in his SIGMOD Innovations Award speech, Jim Gray urged the database community to lower the wall between data and programs. In the same vein, the Asilomar Report puts the unification between programs and data high on the database research agenda [2]. As queries are an important kind of program in the context of databases, support for meta-querying thus seems to be a step in the right direction towards understanding how we can unify programs and data in database systems.

In this paper, we present a practical meta-querying system based on the relational model. Our main design goal was to use current DBMS technology and only extend standard SQL with specific meta-querying features where necessary. Stored queries are represented as syntax trees in XML format. This representation provides a convenient basis for syntactical meta-querying. Indeed, rather than reinventing the wheel and designing a new sublanguage for syntactical manipulation of stored queries, it allows us to use the standard XML transformation language XSLT for this purpose. Many syntactical meta-queries can then directly be expressed simply by allowing XSLT function calls within SQL expressions.[2]

This combination of SQL and XSLT gives us a basic level of expressive power, but for more complex syntactical meta-queries we need a bit more. To this end, we enrich the SQL language with *XML variables* which come in addition to SQL's standard range variables. XML variables range not over the rows of a table, but rather over the subelements of an XML tree. The range can be narrowed by an XPath expression. (XPath is the sublanguage of XSLT used to locate subelements of XML documents.) XML variables thus allow us to go from an XML document to a set of XML documents. Conversely, we also add *XML aggregation* [4], which allows us to go from a set of XML documents to a single one.

SQL combined with XSLT and enriched with XML variables and aggregation offers all the expressive power one needs for ad-hoc syntactical meta-querying. To allow for semantical meta-querying as well, it now suffices to add an *evaluation function*, taking the syntax tree of some query as input, and producing the table resulting from executing the query as output. We note that a similar evaluation feature was already present in the Postgres system.

What we obtain is *Meta-SQL*: a practical meta-query language. Meta-SQL has as advantage that it is *not* "yet another query language": it is entirely compatible with modern SQL implementations offered by contemporary extensible database systems. Indeed, these systems already support calls to external functions from within SQL expressions, which allows us to implement the XSLT calls. Furthermore, XML variables and the evaluation function can be implemented using *set-valued* external functions. As we will show, the powerful feature of "lateral derived tables", part of the SQL:1999 standard, turns out to be crucial to make this work. XML aggregation, finally, can be implemented as a user-defined aggregate function.

We emphasize again that we are not proposing yet another database language. Instead, our main design goal was to stick as closely as possible to standard SQL. Of course, a drastic alternative is to abandon the relational model altogether and move to, e.g., an XML-XQuery environment, where meta-querying does not pose any problem. However, given the widespread use of relational databases, a conservative approach such as ours remains important.

This paper is further organized as follows. In Section 2, we combine SQL with XSLT. In

[2] We embrace XSLT because it is the most popular and stable standard general-purpose XML manipulation language to date. When other languages, notably XQuery [3], will take over this role, it will be an easy matter to substitute XSLT by XQuery in our overall approach.