Normalization Theory for XML

Leonid Libkin

School of Informatics, University of Edinburgh
libkin@inf.ed.ac.uk

Abstract. Specifications of XML documents typically consist of typing information (e.g., a DTD), and integrity constraints. Just like relational schema specifications, not all are good – some are prone to redundancies and update anomalies. In the relational world we have a well-developed theory of data design (also known as normalization). A few definitions of XML normal forms have been proposed, but the main question is *why* a particular design is good. In the XML world, we still lack universally accepted query languages such as relational algebra, or update languages that let us reason about storage redundancies, lossless decompositions, and update anomalies. A better approach, therefore, is to come up with notions of good design based on the intrinsic properties of the model itself. We present such an approach, based on Shannon's information theory, and show how it applies to relational normal forms as well as to XML design, for both native and relational storage.

1 Introduction

Data organization is one of the most fundamental topics in the study of databases. In fact, the concept of normalization was proposed by Codd [5] in 1971 – a mere year after he introduced the relational model. By 1974, the standard 2nd, 3rd, and Boyce-Codd [6] normal forms (2NF, 3NF, BCNF) had been developed. Bernstein's work on 3NF in the mid 1970s [3] is often viewed as the birth of database theory. It was understood very early by both database practitioners and theoreticians that having well-organized and well-designed databases is absolutely crucial for storing, querying, and updating data. Already in the 1980s, the standard normal forms such as 3NF and BCNF were covered by the majority of database texts.

After three decades of relational dominance, we have seen a new data format that is extremely widely used and can seriously challenge relational databases. Thanks to the proliferation of data on the web, much of it now appears in various markup language formats, of which XML is the most common one. Given the amount of data available in XML, it is natural to expect that some of XML designs will exhibit problems similar to those of relational designs, and indeed this is the case. As a simplest example, we can represent an arbitrary relational schema $R_1(A_1^1, \ldots, A_{n_1}^1), \ldots, R_1(A_1^k, \ldots, A_{n_k}^k)$ in XML by means of the following DTD D_1:

$$db \to R_1, \ldots, R_k$$
$$R_i \to tuple_i^*, \quad i \le k$$

D. Barbosa et al. (Eds.): XSym 2007, LNCS 4704, pp. 1–13, 2007.

that declares $A^i_1, \ldots, A^i_{n_i}$ to be the attributes of $tuple_i$. This way, a bad relational design translates into a bad XML design, inheriting its problems such as redundancies and update anomalies. But there are other ways to have designs prone to update anomalies due to the *hierarchical* nature of XML. Consider, for example, the following DTD D_2 for storing information about conference publications:

$$\begin{gathered} db \to conf^* \\ conf \to paper^* \\ paper \to author^+, title, year \end{gathered}$$

with *author*, *title*, and *year* elements each carrying an attribute with its value. Now suppose we know – and this is a reasonable assumption – that all papers in a conference have the same publication year. Then the *year* information is redundant, as it is stored repeatedly for all papers in the conference. In addition it is likely to lead to update problems: if a year needs to be changed, one cannot do it just once; instead it needs to be changed for every paper in the conference.

To build foundations of good XML design, we need to answer the following two questions:

1. How do we recognize poor XML designs, and how do we convert them to good designs? In other words, we want to develop a theory of normalization for XML.
2. What constitutes a good XML design? In other words, we want to formulate criteria for good XML designs. In the relation case, one usually appeals to the intuitive notions of redundancy and update anomaly. But this approach is problematic in the case of XML for three reasons:
   - First, due to the complicated hierarchical structure of XML documents, it is harder to see when a schema contains redundancies.
   - Second, the notion of an update is not nearly as clean as the notion of relational updates, which makes it hard to say what constitutes an update anomaly (especially in the absence of a universally accepted notion of XML updates).
   - Third, there is no query language with the same yardstick status as relational algebra has for relational databases. The process of normalization needs to be lossless (meaning that the original data can be recovered from a differently designed schema). Thus, the notion of losslessness depends on a query language.

   Thus, we need an approach based on "standards-free" XML concepts, that is, concepts that will not change even if the W3C comes up with a new query or update language for XML tomorrow.

Several XML normal forms have been proposed recently, see, e.g., [1,18,17,19,8]. They differ in terms of schema and constraint description, but are based on essentially the same set of transformations, first proposed in [1]. As for the work on justification of XML normal forms, an approach was proposed in [2] based on information theory. The idea of the approach is that we measure the amount of redundancy in a design *regardless* of any query/update language for a data model.