On Database Theory and XML

Dan Suciu
University of Washington
www.cs.washington.edu/homes/suciu

Abstract

Over the years, the connection between database theory and database practice has weakened. We argue here that the new challenges posed by XML and its applications are strengthening this connection today. We illustrate three examples of theoretical problems arising from XML applications, based on our own research.

1 On Database Theory

The field of relational databases is the product of a theoretician, E.F. Codd, from the early 70s. Relational databases had to struggle for a while against the industry proposal CODASYL [58], but then became universally adopted and today we have both a strong industry and a flourishing research field. The end of 70's and early 80's were golden years for database theoreticians. Theory had two principal threads [61], relational database theory (dependency theory, universal relation theory, acyclic hypergraph theory), and transaction processing. Many contributions of this period influenced database systems and the industry. Papers from that time analyzing the state of theoretical database research and its relationship to database systems concluded that the field was in a healthy state [61, 52]. Over time, however, theoretical research became less connected to database systems. New fields, like deductive databases (recursive queries) [60, 7, 8], nested relations [1, 32, 51] theory of object-oriented query languages [35, 36, 2, 3, 4], ended up having little or no influence on the industry, despite their practical motivation and excellent results. Other areas, especially in the 90s, where directly motivated by theoretical questions and were totally ignored by practitioners: query languages and complexity classes [63, 30, 31, 59], topics in finite model theory [57, 10], topological and constraint databases [34, 47, 45].

In 1995 Christos Papadimitriou wrote a thought provoking essay entitled *Database Metatheory: Asking the Big Queries* [46], analyzing the state of database theory at that time. Two of its ideas are especially relevant to our discussion. The first is that theory in Computer Science is inevitable. Unlike traditional *natural* sciences which study an objective world, Computer Science is a science of the artificial, studying artifacts that only exists as a result of scientific activity. The second is to adapt Kuhn's *paradigm principle* for natural sciences to Computer Science. Under this principle, a science evolves in a cycle consisting of three states: from *normal science* to *crisis*, to *revolution*, then back to normal science. In Kuhn's theory the crisis occurs when new observations and measurements about the objective reality fail to agree with the the science's accepted paradigm; during a revolution a new paradigm is developed, consistent with the new observations. Clearly, this definition of a crisis does not apply to a field of Computer Science, since there is no real world to observe.

Instead, Papadimitriou suggest another definition. Consider the graph of all research units[1] in a given field. An edge from x to y represents the fact that x influenced y. *Normal* science is defined by a graph which is multiply connected (i.e. remains connected even if several nodes are removed); a science in *crisis* is defined by one which is almost disconnected, with two major connected components corresponding to theoretical and applied projects respectively. This definition is interesting for two reasons. First, it should be possible to automatically monitor the healthiness of our research field, e.g. by using the DBLP bibliography database [41]. Second, it reinforces our general belief that our science is healthy if theory and practice are strongly connected: the general feeling in the mid 90's was that this connection was weakening.

What caused this rift in a field which started with theoreticians and practitioners working so closely ? While there are probably a large number of reasons that we don't claim to know, there is one in particular that is of importance to our discussion: the tension between the theoreticians' quest to explore new avenues, branching out of Codd's pure relational framework, and the practitioners' need to improve systems in the existing framework (relational/SQL) and for existing applications (client/server). Theoreticians explore new query languages (e.g. with recursion [60]), new data models [51], new usages of data (e.g. incomplete information [29]), or entirely new frameworks (e.g. constraint databases [34]). Practitioners improve the execution of relational operators [24] or improve relational optimizers [53, 25, 26,

[1] I.e. projects, papers, group – Papadimitriou leaves purposedly the term undefined.

42, 48].

One particular theoretical result that has been obtained long time ago but never used in practice is on query containment. Consider the following two queries:

```
Q1 = SELECT DISTINCT x.name
     FROM Person x
     WHERE x.department = ''sales''

Q2 = SELECT DISTINCT x.name
     FROM Person x, Person y, Person z
     WHERE x.department = ''sales'' AND
           x.manager = y.manager AND
           y.fax = z.fax
```

Q1 and Q2 compute the same answers, but Q2 does this in a more cumbersome way: we say that Q1 and Q2 are equivalent, or, at a finer level, that Q1 is contained in Q2 and Q2 is contained in Q1. Obviously, Q1 is the simplest expression computing this answer, so Q1 is called a *minimal* query, while Q2 is not minimal. The problem of checking whether a query is contained in another and of minimizing a query has been solved long time ago by Chandra and Merlin [14] for conjunctive queries[2]. Still, none of the major commercial databases minimizes Q2 to replace it with Q1, and for a good reason: minimization is expensive, and in the most common applications SQL queries are written by users who don't write inefficient queries like Q2. However, theoreticians have continued to study this problem for a variety of other types of queries: in the presence of functional and inclusion dependencies [33], for queries with one level of negation [50, 40], with order [37, 62], with recursion [56, 15], with nested relations [39], with aggregates [44], and with regular expressions [23].

2 On XML

Internet applications and the new ways in which they handle data are changing the role of database theory and its relationship with practice. They use a new data model, the semistructured data model, with XML syntax; this in itself offers a rich source of problems for database theory, some will be illustrated below. Moreover, these applications perform data management operations that were not common in traditional database applications, such as data transformations, query translation, data transport, and stream-based processing.

XML itself and concepts related to its use are created within standards bodies, especially W3C working groups. This is a new phenomenon, since traditionally artifacts in Computer Science were created in academic and industrial research labs before being adopted by the industry or standards committees. While more academics are now getting involved in the W3C working groups, standards and concepts are sometimes created faster than research communities can validate them.

[2]Same as SQL's SELEC-DISTINCT-FROM-WHERE queries where only equality predicates are allowed in WHERE.

3 DB Theory and XML

We see a double role of theory in the Web age. One is a long term of conceptualization and rationalization which can lead to improvements in existing standards. This is a traditional role, and it will probably take years; perhaps an early example in this category is the recent work on keys for XML by Buneman et al. [11]. The second is more short term: to answer critical technical questions that may either help the working groups or the implementors. We discuss here three particular problems: XML publishing, XML typechecking, and XML storage. Their particular choice is influenced by our own previous work in [21, 20, 43, 19], and is not intended to be an exhaustive, or even representative list of XML research problems.

XML Publishing XML publishing is the problem of transforming existing, relational data into XML. Conceptually, this is the same as defining an XML view over the relational data, but this view is considerably more complex than relational views. The relational data is normalized: flat and fragmented into many relations. Often the relational schema is also proprietary, since it represents the company's internal organization (relation names may correspond to business units) or to the company's policy (whether `price` is an attribute of `Product` or of the `Product - Customer` relationship). By contrast, XML data is unnormalized: nested and monolithic. Moreover, its raison d'être is to have a public schema, shared across a community, or, at least between a few partners; for example `www.biztalk.org` lists about 458 public XML schemas.

Consider the following relational schema:

```
Product(pid, pname, price)
Customer(cid, cname, address)
Orders(oid, pid, cid, date)
Complaints(oid, text)
```

Suppose we want to export it into an XML document with the following DTD:

```
<!ELEMENT products (product)*>
<!ELEMENT product (productID, name, price,
          order*, complaint*)>
<!ELEMENT order (customerID, date,
          customerName, customerAddress)>
<!ELEMENT complaint (customerID, text)>
```

```
CREATE VIEW product AS    /* <product> nodes */
SELECT DISTINCT product1.pid, product1.name, product1.price
FROM Product product1
```

```
CREATE VIEW order AS    /* <order> nodes */
SELECT DISTINCT product2.pid, orders2.date,
       customer2.cid, customer2.name, customer2.Address
FROM   Product product2, Orders orders2,
       Customer customer2
WHERE  product2.pid = orders2.pid AND
       orders2.cid = customer2.cid
```

```
CREATE VIEW complaint AS    /* <complaint> nodes */
SELECT DISTINCT product3.pid,
       customer3.cid, product3.pid
FROM   Product product3, Orders orders3,
       Customer customer3, Complaint complaint3
WHERE  product3.pid = orders3.pid AND
       orders3.cid = customer3.cid AND
       order3.oid = complaint3.oid
```

Figure 1: Views for `<product>`, `<order>`, and `<complaint>` arranged in a viewtree

Various XML publishing systems [21, 12, 54, 49] use different formalisms for defining the XML view. What they share in common the fact that, at least conceptually, a separate relational view is defined for every element type in the DTD. For our example, Fig. 1 illustrates the views for the three of the twelve tags: `<product>`, `<order>`, and `<complaint>`. These views are arranged in a tree, called *viewtree* [21], corresponding to the hierarchical structure of the DTD.

The meaning of the viewtree is the following. Each tuple in each view corresponds to a node in the XML document (tuples in `product` correspond to `<product>` nodes, etc). For any two tuples x, y, if x's view is a parent of y's view in the viewtree, and the two tuples have the same values in the fields that form the primary key of x, then the node corresponding to x is a parent of that corresponding to y.

The interesting case for us here is when the XML view is virtual[3]. Here the system accepts XML queries over the view, and translates them into SQL. For example, consider the following XQuery [13] expression, returning all complaints filed by customers for products ordered in 1999.

```
FOR $p IN document("xmlview")/productcs/product,
    $cid IN $p/order[date="1999"]/customerID,
    $t IN $p/complaint[customerID=$cid]/text
RETURN <result> <pid> $p/productID </pid>
                <complaint> $t </complaint>
       </result>
```

The SilkRoute system [21] describes in detail how XML queries can be composed with the XML view and translated into SQL[4]. Intuitively, this corresponds to binding the XQuery variables to the nodes in the viewtree, then combining all SQL expressions of the the matched nodes. In our example all three nodes in the viewtree are touched by the query, hence the combined SQL query is:

```
SELECT DISTINCT product1.pid, complaint3.text
FROM Product product1, Product product2,
     Orders orders2, Customer customer2
     Product product3, Orders orders3,
     Customer customer3, Complaint complaint3
WHERE product1.pid = product2.pid AND
      product1.pid = product3.pid AND
      product2.pid = orders2.pid AND
      orders2.cid = customer2.cid AND
      product3.pid = orders3.pid AND
      orders3.cid = customer3.cid AND
      order3.oid = complaint3.oid AND
      order2.date = "1999"
```

The problem becomes clear now: this query performs more joins than necessary. In other words, this query is not minimal. The reason is that we generated this query automatically, by combining join conditions from three different relational views, and they happened to share some common subexpressions. Some redundant joins can be easily eliminated, for example it follows from the viewtree structure that the tuple variables `product2` and `product3` are redundant, but others may require a general-purpose minimization algorithm. We may also need to take into account constraints in the databases. In our example, if we assume that `pid, cid` is a key in `Orders`, the query can simplified to:

```
SELECT DISTINCT product1.pid, complaint3.text
FROM Product product1, Orders orders2,
     Customer customer2, Complaint complaint3
WHERE product1.pid = orders2.pid AND
      orders2.cid = customer2.cid AND
      order2.oid = complaint3.oid AND
      order2.date = "1999"
```

This phenomenon is not new: when views are unfolded in SQL queries they can also introduce repeated subexpressions (hence, non-minimal queries). However, joins across multiple views are rare in the relational world, so query minimization is not critical.

[3] Materialized XML views are discussed in [54, 20].

[4] SilkRoute uses XML-QL rather than XQuery.

By contrast, in XML publishing every node type is defined by a different relational view, and we should expect queries to frequently join many of them: here minimization, either in the engine or in some middleware, is a necessity.

While query containment (and, hence, minimization) is NP complete for conjunctive queries, recent advances in theoretical database research have created powerful tools that could lead to practical minimization algorithms. Chekuri and Rajaraman [16] have shown that containment can be checked efficiently if the queries have a small *query width*. Kolaitis and Vardi [38] established additional relationships to *tree-widths*, first order logic with bounded variables, and the constraint satisfaction problem (see also [64]).

XML Typechecking Although XML can be schema-less, most XML instances will probably be associated to some form of schema, either a DTD or an XML-Schema [9]. Given a program that generates an XML document, *typechecking* is the problem of deciding whether the generated XML document always conforms to a given output DTD (or XML-Schema); this should not be confused with validation, which checks conformance of a given XML document to a DTD (or XML-Schema). It is possible to do typechecking dynamically, by validating the generated XML document at runtime, but this creates the possibility of runtime errors, and slows down an application. *Static typechecking* is more desirable, but also more difficult, since it requires a thorough analysis of the program generating the XML document.

Static XML typechecking is undecidable if an arbitrary C++ or Java program is generating the XML document, due to Rice's theorem. The more interesting question is whether typechecking is possible for more restricted languages, like XSLT [17] or XQuery [13], since in all likelihood, most dynamically generated XML documents will be created by scripts/queries written in these languages. In the following discussion we assume the XML types to be given by DTDs: everything carries over to XML-Schemas too. Given two DTDs τ_1, τ_2, and given an XML transformation $f : XML \rightarrow XML$ expressed in some query language, the typechecking problem asks whether for every document D conforming to τ_1 (in notation: $D \in \tau_1$), $f(D)$ conforms to τ_1:

$$\forall D \in \tau_1, f(D) \in \tau_2$$

All systems that offer static typechecking today, including the XQuery Algebra [13], base their typechecking algorithm on *type inference*, which we formalize here as follows: given an input DTD τ_1 and transformation f, compute the "output DTD"

$$f(\tau_1) \stackrel{\text{def}}{=} \{f(D) \mid D \in \tau_1\}$$

If we knew how to do type inference, then type checking is easy: given τ_1, τ_2, and f, first infer the output type (i.e. compute $f(\tau_1)$), then check whether $f(\tau_1) \subseteq \tau_2$, which is possible based on the fact that containment of two regular expressions is decidable[5].

This is the point where we need some input from theory. It turns out that type inference is not possible, not even for the simplest subsets of query languages. Systems that do type inference compute an approximate output DTD that will lead to false negatives in the typechecking procedure. To see such an example, consider the input DTD τ_1:

$$\tau_1 = \; < \texttt{!ELEMENT root (elm*)} >$$

The query below iterates over the input document three times, producing first an `<a>` element for each `<elm>`, then a `<b>` element, then a `<c>` element. We give f below, expressed in XQuery:

```
f =
<result>
 FOR $x IN "doc.xml"/root/elm RETURN <a> $x/text() </a>
 FOR $x IN "doc.xml"/root/elm RETURN <b> $x/text() </b>
 FOR $x IN "doc.xml"/root/elm RETURN <c> $x/text() </c>
</result>
```

Thus, for the input XML document `<root> <elm/> <elm/> ...<elm/> </root>`, in short $\mathtt{elm}^{\mathtt{n}}$, the query will return the output document $\mathtt{a}^{\mathtt{n}}.\mathtt{b}^{\mathtt{n}}.\mathtt{c}^{\mathtt{n}}$.

Today's systems will infer the following output DTD for this query:

$$\tau = \; < \texttt{!ELEMENT result (a*, b*, c*)} >$$

While this seems reasonable, it is obviously not the correct one, which, according to the definition should be:

$$\mathtt{f}(\tau_1) = < \texttt{!ELEMENT result } (\{\mathtt{a}^{\mathtt{n}}, \mathtt{b}^{\mathtt{n}}, \mathtt{c}^{\mathtt{n}} \mid \mathtt{n} \geq 0\}) >$$

Obviously, this is not a real DTD, and perhaps not very interesting in practice, and one may wonder if one should accept, for all practical purposes, the inferred output DTD `a*,b*,c*`. The problem, however, is that with this inferred DTD one may fail to correctly typecheck. To see an example, consider the following output DTD:

$$\tau_2 = \texttt{<!ELEMENT result } (((\mathtt{a},\mathtt{a})*, (\mathtt{b},\mathtt{b})*, (\mathtt{c},\mathtt{c})*) \mid ((\mathtt{a},\mathtt{a})*, \mathtt{a}, (\mathtt{b},\mathtt{b})*, \mathtt{b}, (\mathtt{c},\mathtt{c})*, \mathtt{c})) >$$

It says that there are either an even number of `a`'s, `b`'s, and `c`'s, or an odd number of `a`'s, `b`'s, and `c`'s.

[5]In addition, both DTDs and XML-Schema restrict the regular expressions such that their associated automaton is deterministic. If τ_2 has this property, then one can check $f(\tau_1) \subseteq \tau_2$ in PTIME.

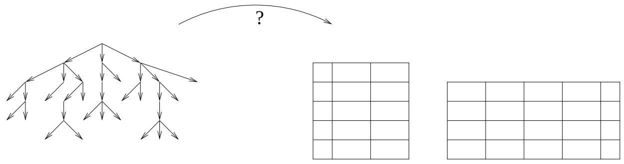

Figure 2: The XML Storage Problem

Clearly f typechecks with respect to τ_1 and τ_2, but, using the "inferred" type τ, we have $\tau \not\subseteq \tau_2$, hence the typechecking procedure based on type inference will fail.

This is a serious limitation of the type inference approach. If users want to generate XML documents that conform to the type τ_2 and correctly write the program f for that purpose, the system will reject it claiming (incorrectly) that it does not typecheck. This limitation needs to be investigated by theoretical research.

One may argue that τ_2 is not a "practical" output type. But this begs for a definition of "practical" DTDs. Even if we had one, the question remains whether type inference is complete for that restricted set of DTDs.

Another question is whether typechecking is possible without using type inference. One answer, given in [43], is a surprising "yes", for a large class of XML transformations that include recursive traversal of the XML tree, e.g. like in XSLT. However, these transformations do not allow us to do joins: this may be acceptable for XSLT, but certainly such a result is useless for query languages like XQuery, or extensions of SQL with XML publishing features. When one adds joins, typechecking becomes undecidable, even if one attempts to restrict DTD's in various ways to "practical" DTDs [6, 5].

Today the most promising approach to typechecking remains that based on type inference. The XDuce language [27, 28] defines a type inference system for a functional language with recursion; the XQuery algebra defines a type inference system using XML Schema as its type system. Since we know that this approach cannot be as robust as typechecking in general-purpose programming languages, a study of its applicability and limitations is needed.

XML Storage XML data is a labeled tree; a relation is a table. The problem of storing XML data in one or several tables, suggested in Fig. 2, is a challenging one, both for theoreticians and practicians. Since the tree is meant to describe some irregular structure while tables are by definition regular, we are attempting to store some irregular data into a regular data type. In addition to the pure combinatorial aspect, there is a logical aspect to the storage problem: given a storage mapping, one needs to be able to translate queries formulated over the XML data into relational queries formulated over the relational storage. The combination of combinatorics and logic make the problem particularly appealing.

Several approaches have been tried so far. The simplest is to store XML as a graph, in a ternary relation (two columns for the edges, the third for the labels and/or data values). This approach is explored by Florescu and Kossman in [22]. The price one pays for its simplicity is that many self-joins of the edge table are required in order to reconstruct a given XML element: one join for each subelement. Shanmugasundaram et al. [55] use the DTD (or XML-Schema) to derive a relational schema. One table is created for each element type that can occur in a collection position. This technique works well in practice whenever one has a schema for the XML document. A subtle problem is that the resulting storage is very sensitive to that schema. For example if the content of `<person>` changes from `(name, phone)` to `(name, phone*)` then we need to move all phone numbers to a separate table, although perhaps the XML document has changed very little.

The case when the XML document has no schema, or when the schema changes frequently is harder, and has a more dramatic impact on performance. An approach is proposed in [19], which uses data mining on the XML instance to infer a relational schema. The idea is to find regularities than may exists in a given XML data instance, and to organize the storage based on those regularities. Another approach is proposed in [18]. Here all data values of the XML document are stored together and a smart index allows efficient access based on the path expression and, possibly, based on the data value.

The challenge in any storage schema is that it has to be flexible enough to accommodate any XML data, yet it has to be as efficient as regular data storage when the XML data happens to be regular. Finding the largest regular subset in an irregular data instance is a problem which can be formulated and addressed theoretically. A related problem is that of quantifying the degree of irregularity in a semistructured data instance. For instance, in the simple case of flat XML data, elements can be viewed as records with variable fields; a boolean matrix S describes the structure completely, with S_{ij} equal 1 when element i contains field j, and 0 otherwise. It is easy to see that if the data instance can be stored perfectly in k separate tables then the rank of the matrix is k. Here k is a measure of semistructuredness, since the larger it is, the more irregular the data. Finding a general definition of semistructuredness remains a topic for future research.

4 Conclusions

We have described three XML research problems, inspired from our own work. XML's semistructured data model represents paradigm shift for theoretical database research. It is not the first one: for example the object-oriented data model can also be considered a paradigm shift, which generated a vast amount of theoretical and applied research. This time, however, the shift comes from outside the community (XML was imposed on us) and this, at least, settles easily the question of applicability. It offers us both a chance both to apply research on old topics (query containment) and to conduct research on new topics (typechecking).

Acknowledgment I would like to thank Gerome Miklau for his comments.

References

[1] S. Abiteboul and N. Bidoit. Non first normal form relations to represent hierarchical organized data. In *Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, April 2-4, 1984, Waterloo, Ontario, Canada*, pages 191–200. ACM, 1984.

[2] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 159–173, Portland, Oregon, 1989.

[3] S. Abiteboul and P. C. Kanellakis. Object identity as a query language primitive. *JACM*, 45(5):798–842, 1998.

[4] S. Abiteboul, P. C. Kanellakis, and E. Waller. Method schemas. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee*, pages 16–27. ACM Press, 1990.

[5] N. Alon, T. Milo, F. Neven, D. Suciu, and V. Vianu. Typechecking xml views of relational databases. In *LICS*, 2001.

[6] N. Alon, T. Milo, F. Neven, D. Suciu, and V. Vianu. Xml with data values: Typechecking revisited. In *PODS*, 2001.

[7] F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman. Magic sets and other strange ways to implement logic programs. In *Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 24-26, 1986, Cambridge, Massachusetts*, pages 1–16. ACM, 1986.

[8] F. Bancilhon and R. Ramakrishnan. An amateur's introduction to recursive query processing strategies. In *Proceedings of ACM SIGMOD Conference on Management of Data*, May 1986.

[9] D. Beech, S. Lawrence, M. Maloney, N. Mendelsohn, and H. Thompson. Xml schema part 1: Structures, May 1999. `http://www.w3.org/TR/xmlschema-1/`.

[10] M. Benedikt, G. Dong, L. Libkin, and L. Wong. Relational expressive power of constraint query languages. In *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, 1996, Montreal, Canada*, pages 5–16. ACM Press, 1996.

[11] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. In *Proceedings of the 10th WWW Conference*, pages 201–210, 2001.

[12] M. Carey, D. Florescu, Z. Ives, Y. Lu, J. Shanmugasundaram, E. Shekita, and S. subramanian. XPERANTO: publishing object-relational data as XML. In *Proceedings of WebDB*, Dallas, TX, May 2000.

[13] D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu. XQuery: a query language for XML, 2001. available from the W3C, `http://www.w3.org/TR/query`.

[14] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of 9th ACM Symposium on Thoery of Computing*, pages 77–90, Boulder, Colorado, May 1977.

[15] S. Chaudhuri and M. Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. In *Proceedings of 11th ACM Symposium on Principles of Database Systems*, 1992.

[16] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. In F. N. Afrati and P. Kolaitis, editors, *Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings*, volume 1186 of *Lecture Notes in Computer Science*, pages 56–70. Springer, 1997.

[17] J. Clark. XSL transformations (XSLT) specification, 1999. available from the W3C, `http://www.w3.org/TR/WD-xslt`.

[18] B. Cooper, N. Sample, M. Franklin, G. Hjaltason, and M. Shadmon. A fast index for semistructured data. In *VLDB*, 2001.

[19] A. Deutsch, M. Fernandez, and D. Suciu. Storing semistructured data with STORED. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 431–442, 1999.

[20] M. Fernandez, A. Morishima, and D. Suciu. Efficient evaluation of XML middle-ware queries. In *Proceedings of ACM SIGMOD Conference on Management of Data*, Santa Barbara, 2001.

[21] M. Fernandez, D. Suciu, and W. Tan. SilkRoute: trading between relations and XML. In *Proceedings of the WWW9*, pages 723–746, Amsterdam, 2000.

[22] D. Florescu and D. Kossmann. Storing and querying xml data using an rdbms. *IEEE Data Engineering Bulletin*, 22(3), 1999.

[23] D. Florescu, A. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 139–148, 1998.

[24] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, June 1993.

[25] G. Graefe and D. J. DeWitt. The exodus optimizer generator. In U. Dayal and I. L. Traiger, editors, *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference, San Francisco, California, May 27-29, 1987*, pages 160–172. ACM Press, 1987.

[26] G. Graefe and W. J. McKenna. The volcano optimizer generator: Extensibility and efficient search. In *Proceedings of the Ninth International Conference on Data Engineering, April 19-23, 1993, Vienna, Austria*, pages 209–218. IEEE Computer Society, 1993.

[27] B. C. P. Haruo Hosoya. Xduce: An xml processing language (preliminary report). In *WebDB'2000*, 2000. http://www.research.att.com/conf/webdb2000/.

[28] B. C. P. Haruo Hosoya. Regular expression pattern matching for xml. In *ACM SIGPLAN, SIGACT Symposium on Principles of Programming Languages (POPL)*, January 2001.

[29] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31:761–791, October 1984.

[30] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

[31] N. Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16:760–778, 1987.

[32] G. Jaeschke and H. J. Schek. Remarks on the algebra of non-first-normal-form relations. In *Proceedings ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, pages 124–138, Los Angeles, California, March 1982.

[33] D. S. Johnson and A. C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. In *Proceedings of the ACM Symposium on Principles of Database Systems, March 29-31, 1982, Los Angeles, California*, pages 164–169. ACM, 1982.

[34] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee*, pages 299–313. ACM Press, 1990.

[35] M. Kifer and G. Lausen. F-logic: A higher order language for reasoning about objects, inheritance, and scheme. In *Proceedings of ACM-SIGMOD 1989*, pages 46–57, June 1989.

[36] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.

[37] A. C. Klug. On conjunctive queries containing inequalities. *JACM*, 35(1):146–160, 1988.

[38] P. Kolaitis and M. Vardi. Conjunctive-query containment and constraint satisfaction. In *Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1998.

[39] A. Levy and D. Suciu. Deciding containment for queries with complex objects. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1997.

[40] A. Y. Levy and Y. Sagiv. Queries independent of updates. In R. Agrawal, S. Baker, and D. A. Bell, editors, *19th International Conference on Very Large Data Bases, August 24-27, 1993, Dublin, Ireland, Proceedings*, pages 171–181. Morgan Kaufmann, 1993.

[41] M. Ley. Computer science bibliography (dblp). `http://dblp.uni-trier.de`.

[42] G. M. Lohman. Grammar-like functional rules for representing query optimization alternatives. In H. Boral and P.-Å. Larson, editors, *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, June 1-3, 1988*, pages 18–27. ACM Press, 1988.

[43] T. Milo, D. Suciu, and V. Vianu. Typechecking for xml transformers. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 11–22, Dallas, TX, 2000.

[44] W. Nutt, Y. Sagiv, and S. Shurin. Deciding equivalences among aggregate queries. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington*, pages 214–223. ACM Press, 1998.

[45] C. Papadimitriou, D. Suciu, and V. Vianu. Topological queries in spatial databases. In *Proceedings of 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Montreal, Canada, June 1995.

[46] C. H. Papadimitriou. Database metatheory: Asking the big queries. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 22-25, 1995, San Jose, California*, pages 1–10. ACM Press, 1995.

[47] J. Paredaens, J. Van den Bussche, and D. Van Gucht. Towards a theory of spatial database queries. In *Proceedings of 13th ACM Symposium on Principles of Database Systems*, pages 279–288, Minneapolis, Minnesota, May 1988.

[48] H. Pirahesh, J. M. Hellerstein, and W. Hasan. Extensible rule-based query rewrite optimization in Starburst. *SIGMOD Record*, 21(2):39–48, June 1992.

[49] M. Rys. Bringing the internet to your database: using SQLServer 2000 and XML to build loosely-coupled systems. In *Proceedings of the International Conference on Data Engineering*, pages 465–472, 2001.

[50] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM*, 27:633–655, 1980.

[51] H.-J. Schek and M. H. Scholl. The relational model with relation-valued attributes. *Information Systems*, 11(2):137–147, 1986.

[52] P. G. Selinger. Chickens and eggs — the interrelations of systems and theory. In *Proceedings of 6th ACM Symposium on Principles of Database Systems*, pages 250–253, 1987.

[53] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 23–34, 1979. Reprinted in *Readings in Database Systems*, Morgan-Kaufmann, 1988.

[54] J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B. Lindsay, H. Pirahesh, and B. Reinwald. Efficiently publishing relational data as xml documents. In *Proceedings of VLDB*, pages 65–76, Cairo, Egipt, September 2000.

[55] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational databases for querying XML documents: limitations and opportunities. In *Proceedings of VLDB*, pages 302–314, Edinburgh, UK, September 1999.

[56] O. Shmueli. Decidability and expressiveness aspects of logic queries. In *Proceedings of ACM Symp. on Principles of Database Systems*, pages 237–249, 1987.

[57] A. P. Stolboushkin and M. A. Taitslin. Finite queries do not have effective syntax. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 22-25, 1995, San Jose, California*, pages 277–285. ACM Press, 1995.

[58] M. Stonebraker and J. Hellerstein. *Readings in Database Systems*. Morgan Kaufmann, 1998.

[59] D. Suciu and V. Breazu-Tannen. A query language for NC. In *Proceedings of 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 167–178, Minneapolis, Minnesota, May 1994.

[60] J. D. Ullman. Implementation of logical query languages for databases. *TODS*, 10(3):289–321, 1985.

[61] J. D. Ullman. Database theory: Past and future. In *Proceedings of the Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 23-25, 1987, San Diego, California*, pages 1–10. ACM, 1987.

[62] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. In *Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 2-4, 1992, San Diego, California*, pages 331–345. ACM Press, 1992.

[63] M. Y. Vardi. The complexity of relational query languages. In *Proceedings of 14th ACM SIGACT Symposium on the Theory of Computing*, pages 137–146, San Francisco, California, 1982.

[64] M. Y. Vardi. Constraint satisfaction in database theory. In *Proceedings of PODS*, pages 76–85, Dallas, TX, 2000.