Optimization of Pathfinder Queries (Opaque)  
[public version]

Actualized research proposal NWO Open Competitie 2004 - round 2

Maurice van Keulen

January 17, 2006

**Project title:** Optimization of Pathfinder Queries  
**Project acronym:** Opaque  
**Principal investigator:** Dr.ir. M. van Keulen  
**Classification:** Informatica, discipline 2: Data- en kennisystemen, 2.1: Databasesystemen.

**Composition of the research team:**

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dr.ir. Maurice van Keulen</td>
<td>Assistant Professor (UD), CTIT, University of Twente.</td>
</tr>
<tr>
<td>vacancy</td>
<td>OiO, promotor Prof.dr. P.M.G. Apers, CTIT, Univ. of Twente.</td>
</tr>
</tbody>
</table>

**Close cooperation with:**

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ing. Jan Flokstra</td>
<td>Scientific programmer, CTIT, Univ. of Twente. Works in the context of the Bsik-project MultimediaN on the development of the XQuery engine Pathfinder on top of CWI's MonetDB.</td>
</tr>
<tr>
<td>Jens Teubner</td>
<td>PhD student, Technical Univ. of Munich, Germany. PhD topic is the evaluation of XQuery using a relational DBMS, Pathfinder.</td>
</tr>
<tr>
<td>Prof.dr. Torsten Grust</td>
<td>Database Systems, Technical Univ. of Munich, Germany. Supervisor of Jens Teubner.</td>
</tr>
<tr>
<td>Dr. Peter A. Boncz</td>
<td>Centre for Mathematics and Computer Science (CWI), Amsterdam. Project leader AmbientDB, subproject of MultimediaN.</td>
</tr>
</tbody>
</table>

**Research school:** School voor informatie- en kennisystemen (SIKS)

**Summary:**

The Opaque project focuses on query optimization for relational XQuery engines. Advances in techniques for storage and query processing for XML using existing relational engines show that this approach has great potential for being able to manage the ever growing volumes of XML data. Many storage models, index structures, and algorithms have been proposed to speed up queries. Currently, attention is shifting towards query optimization techniques to address the remaining performance problems, to which this research hopes to make a contribution.

To be able to integrate XML query processing with traditional query processing, the proposed approach is based on extending existing query optimizer technology. It involves
extending relational algebra to incorporate some XML-specifics not easily captured in
relational terms, development of a query rewriter based on equivalence rules in the
algebra, and a cost model that as accurately as possible predicts query execution cost
based on statistics.

Samenvatting:
XML is een flexibel formaat voor gegevensuitwisseling dat in de afgelopen jaren zeer pop-
ulair is geworden. XML heeft een brede toepasbaarheid, bijvoorbeeld, voor uitwisseling
van berichtjes tussen computerapplicaties, voor (meta-)beschrijvingen van (multimedia)
gegevens, en als standaard-formaat voor documenten in een digital library, etc.

Steeds meer gegevens zijn in XML-formaat opgeslagen. O.a. het bevragen (query-en)
van grote hoeveelheden XML-data wordt met gangbare gereedschappen onacceptabel
inefficient. Voor ‘normale’ gegevens zou je in zo’n situatie een relationele database
nemen. Een logische gedachte is dus om XML-gegevens in een database op te slaan
en het bevragen ervan op een efficiënte database-manier aan te pakken. De vraagtaal
die Opaque gebruikt is XQuery, een working draft van het WorldWideWeb Consortium
(W3C). Verwacht wordt dat, wat SQL is voor een relationele database, XQuery voor
XML gaat worden. Technieken voor opslag van XML in een database zijn inmiddels
bekend, alsook deels hoe XQuery daarop uit te voeren (ongeoptimaliseerd).

Een database heeft normaliter de keuze uit vele alternatieven voor het uitvoeren van
een query, de zogenaamde query-plannen. De query optimizer, een component van de
database, is doorgaans in staat om daaruit de (nagenoeg) snelste te kiezen. Voor de
XQuery query’s loopt dit echter heel vaak spaak, omdat (1) de database er niet voor
ontworpen is, (2) de query plannen zeer complex worden en (3) de technieken voor het
schatten van de snelheid van query-plannen ontoereikend zijn in XML-context.

Opaque gaat over het uitbreiden van query optimizer technologie, zodat XQuery query’s
efficient uitgevoerd kunnen worden in een normale relationele database.
1 Description of the Proposed Research

1.1 Introduction

With XML, the two separate worlds of document exchange (as a descendant of SGML) and data exchange came together. The flexibility of XML made it applicable in a wide range of applications: as format for document storage, messaging between applications, generic configuration, metadata description, and many more. This wide applicability is largely attributable to its inherent semi-structuredness and its hierarchical data model. Availability of generic tools for parsing and manipulation of XML makes it very attractive for developers to use it.

With the gaining popularity of XML, the need for more and more powerful tools grows. One of the first needs that arose was querying XML documents. Standards committees were founded which, among others, defined the widely used XPath language [4] and, more recently, the full-fledged query language XQuery [5]. XPath has found its way in many tools (e.g., XSLT processors, query engines, DBMSs). The first query engines supporting XQuery have appeared, most notably Galax [18] developed by committee members themselves as a vehicle to validate the chosen semantics for XQuery constructs.

Most of the tools for XML can cope with only a restricted volume of XML data. In several application areas, one is confronted with ever growing volumes of XML-documents. For example, digital libraries with documents stored in XML-format. Or, application messages are stored in logs for tracing, measurement, etc. Such applications need solutions that are scalable in the amount of data, in other words, these applications need database technology.

Efforts on database technology for XML are often divided into native and non-native XML DBMSs. A native DBMS is in essence a DBMS that takes XML as its central data model around which full DBMS-functionality is developed. An example of such a system is Tamino [45]. The definition of ‘native’ is, however, not very precise as many systems called native apply relational technology in their kernels, which others proclaim is distinctive for non-native systems. In the past, the database community was confronted with a similar situation, namely having to develop scalable data management technology for object-oriented data. In retrospect, we observe that OODBMSs have not made it, but that OO-technology has found its way into relational DBMSs. One can discuss about the causes, but a significant cause is that reaching a level of maturity comparable to that of a commercial RDBMS requires enormous effort. Therefore, we believe that existing relational technology injected with XML-specific techniques has great potential to manage the growing amounts of XML data.

A vital issue is how to bridge the gap between the hierarchical / semistructured nature of XML data and the tabular / structured nature of relational data. Much research is directed at relational storage schemes and efficient evaluation of XPath and XQuery inside an RDBMS.

There are effectively two classes of approaches to storage schemes for XML. First, shredding-based approaches which store XML data in many relations based on tag names of elements. For example, data on elements <employee> is stored in a relation called Employee. A representative of this approach is [46]. The disadvantage of these approaches is that they only work well for data-centric XML, i.e., highly structured XML with schema, a relatively low number of different tag names, and no mixed content nodes. The other class of approaches to storage schemes views an XML document as a tree and is centered around a few relations storing data on a per-node basis. Representatives of this class are [33, 25]. These approaches work for arbitrary XML, i.e., also document-centric XML. The only significant downside to these approaches is that they generally have difficulty managing updates. In
Pathfinder, the update problems have been largely solved at the expense of a minor reduction in query performance [10, 6].

A significant jump forward in managing document-centric XML in a relational DBMS was made with the (re)discovery of the Dietz-numbering [16] originally designed for maintaining the order in lists under a sequence of insert and delete operations. It appeared that in the XPath Accelerator approach, which uses a slight adaptation of this numbering scheme, most XPath queries could be evaluated efficiently in an ordinary RDBMS [25]. Many other approaches and applications were inspired by this numbering scheme.

It has been observed by many (e.g., [25]) that performance of XQuery queries evaluated in an RDBMS is generally good but should be improved, and that in some cases, performance is far from optimal. Causes can be found in the fact that the RDBMS ordinarily can only deploy generic index structures and join algorithms, and, secondly, sometimes makes bad choices in constructing a query plan. The DBMS is in a sense unaware of the specific properties of trees and much can be gained by injecting tree-awareness [22].

The many index structures and numbering schemes that have been proposed to speed up XML queries in relational backends, often come with associated special-purpose operators (e.g., [61, 2, 33, 23]). These require, however, changes to the database kernel to be able to extend it with these XML-specific index structures and operations. In the Pathfinder project (see Section 1.6), special attention is paid to keeping such changes minimal, because they may interfere with the integration with normal query processing, and as a consequence, reduce chances of industrial adoption of the technology [37, 38, 22].

The second cause, bad query plan construction, has not received as much attention, yet. In research on evaluating XPath queries using a relational database, it has been observed that the DBMS sometimes makes rather bad choices [51, 23]. Since the query optimizer has no knowledge of the tree-properties of XML, its methods for estimating query cost fall short. Moreover, the FLWOR-construct of XQuery is bound to generate plans with many expensive cartesian products that are hard to optimize away. Furthermore, since XQuery does not have an explicit GROUP-BY construct, the optimizer may not properly recognize it and fail to employ the appropriate operators. As a last example of a performance problem, query plans generated from XQuery queries are often too complex (see [27, 28]) for an ordinary relational query optimizer to handle.

Because of these performance problems, query (plan) optimization is gaining attention quickly, for example, [60] investigates choosing join order for structural joins of the stack-tree family. The choice is based on the cost model of [59, 58]. In the context of main-memory RDBMSs, [43] investigates a cost model for the XPath Accelerator approach used in the Pathfinder project. Both are largely histogram-based.

The Opaque project focuses on query optimization for relational XQuery engines. This project is to be carried out in cooperation with other projects who, under the Pathfinder flag, contribute to the development of a full-fledged XQuery engine on top of existing relational technology (see Section 1.6). From the two aforementioned classes of storage structure approaches, we follow the per-node approach. Without loss of generalization, we select the XPath Accelerator as a concrete and readily available representative.

Preliminary ideas for the direction in which the research in this project could go, were published in [54]. The paper shows among other things that, for the benefit of logical query optimization, only minor extensions to standard relational query optimization techniques suffice to perform join selection and projection and selection pushdown in the presence of XML-specific operators. In the meantime, more algebraic optimization techniques for Pathfinder
have been proposed [57, 26].

1.2 Research questions and envisioned results

The main research question is

**How can XQuery queries effectively be optimized in the context of a per-node based storage structure using (an extension of) relational algebra?**

This main question gives rise to the following subquestions:

- **What is a suitable algebra for XQuery processing and optimization?**
  
  An algebra for Pathfinder has been proposed in [28] as an improvement of translating XQuery directly to SQL [27]. This algebra, however, is inspired by ease of translation only, not by query optimization. Independently, a similar algebra inspired by query optimization only, was proposed in [54]. Both aim for database-like collection-based processing, data independence, and little deviation from standard relational algebra.

- **What properties does the algebra have and what equivalences hold?**
  
  Equivalences within an algebra represent alternative evaluation strategies among which a query optimizer can choose.

- **How to estimate the cost of a query plan?**
  
  Probably one of the most challenging questions is how to compare query plans at compile-time, before execution has taken place. This requires among others research on
  
  - Answer size estimation
  - Cost estimation
  - Maintenance of statistics (sizes of tables, selectivity, occurrences of certain values)

- **How can suitable candidate query plans be produced and compared efficiently?**
  
  It is a known problem in relational query optimization, that the number of alternative query plans grows very large for even simple queries, such that it is not fruitful to produce and compare them all. The query optimization process should be designed to be effective and efficient itself. Note that it is often not necessary nor feasible to find the optimal query plan; quickly producing a suboptimal one usually suffices.

Envisioned results in research terms are advances in and publications on the above mentioned topics. In addition to this, a query optimizer for the Pathfinder project should be developed to be able to validate the ideas experimentally.

1.3 Research Approach

Many database textbooks (e.g. [32]) pay attention to the topic of relational query optimization. Figure 1 shows the typical query processing steps. First, an SQL query is parsed and translated into a relational algebra expression. This expression is the query plan. Then, the plan is rewritten into several equivalent candidate query plans. This is called logical query optimization and makes use of the fact that under certain conditions, the result is unaffected by a change in the order in which joins, projections, and selections are performed. Based on statistics of the data and effects of certain operations thereon, the cost of each candidate plan can be estimated and compared. The plan with the least cost is finally executed.
In Opaque, we strive for query optimization for XQuery that integrates well with this standard architecture. First, [28] shows that XQuery expressions can be translated to an extended relational algebra. Secondly, [54] shows that selection of special-purpose joins can be integrated well with standard logical optimization techniques such as selection and projection pushdown. The expectation is that all desired optimization can be done in (extended) relational algebra without interfering with normal logical query optimization.

A promising optimization technique is based on XPath symmetries [42]. For example, \( p/descendant::n/parent::m \) is equivalent with the XPath expression \( p/descendant-or-self::m[child::n] \). For both XPath expressions, one can generate a (different) query plan according to, for example, [28]. It is a theoretical necessity that equivalence of the XPath expressions implies equivalence of the query plans. It remains to be seen if it is beneficial to have a separate XPath optimization step or that such optimizations can be easily incorporated in relational optimization.

Choosing between the candidate query plans requires being able to roughly estimate the query processing cost (usually in terms of query execution time or similar measure). A DBMS maintains statistics on, for example, table sizes, selectivity, etc. To estimate query plans that use special-purpose joins, similar statistics need to be maintained to appropriately estimate result sizes. [43] made the first proposals for this. The execution time of special-purpose joins in the main-memory DBMS MonetDB appeared to be accurately predictable by feeding the result size estimations into the cost model of [35, 34].

Choosing between the candidate query plans requires being able to roughly estimate the query processing cost (usually in terms of query execution time or similar measure). A DBMS maintains statistics on, for example, table sizes, selectivity, etc. To estimate query plans that use special-purpose joins, similar statistics need to be maintained to appropriately estimate result sizes. [43] made the first proposals for this. The execution time of special-purpose joins in the main-memory DBMS MonetDB appeared to be accurately predictable by feeding the result size estimations into the cost model of [35, 34].

Figure 2 shows an adapted query processing architecture that integrates relational and XML query processing, which will become possible with Opaque. The OiO is expected to make advances in both logical query optimization as well as cost estimation in order to realize this query processing architecture.

### 1.4 Research Methodology

Query optimization is known to be partly theoretical (investigation of algebra's properties, equivalences, proofing correctness of the optimization process, etc.) and partly empirical
(performance experiments, guidance of cost estimation work by realistic data and queries, etc.) Both go hand-in-hand, because empirical results should guide the efforts to focus on the most problematic aspects. Therefore, we strive for quick development of an experimentation environment based on a prototype. This also effectively contributes to working in the OiO.

Research within Pathfinder is directed at the development of a full-fledged XML DBMS with an XQuery query engine called MonetDB/XQuery[6, 52]. It is developed on top of the RDBMS MonetDB [9, 8], which is capable of querying huge amounts of XML data efficiently. MonetDB/XQuery will be used in Opaque as experimentation platform. The OiO will coordinate his development efforts with members of the other projects contributing to Pathfinder (see Section 1.6). The aim is to not require significant changes to the RDBMS, but to reuse and exploit relational technology as much as possible. Opaque contributes to Pathfinder and strives for the same aim concerning query optimization.

1.5 Related Work

The introduction already refers to some research related to this project. In this section, we discuss some more related work.

Several survey papers are available on XML storage schemes and query processing (e.g., [19] and [36]). Moreover, [14] is a nice survey on relational query optimization.

XML is a semi-structured data model, so benefit can be found in pre-XML research on semi-structured data. An important source is the work around the graph-based OEM-model (object exchange model) underlying the Lore system [1]. Research has been done on indexing techniques (DataGuides [21]), query optimization, and cost estimation [39].

Several early systems focused on publishing (object-)relational data as XML. A common approach is to define an XML view over the relational data. XPERANTO [13] is such a system. It uses the language XML-QL for querying and view definition. Queries are optimized on several levels including an algebraic level which uses specific XML-inspired operators such as ‘bind’ and ‘navigate’. Another important system is SilkRoute [17], which uses XQuery to define XML views and also addresses query optimization. Our own dMoa system [55] works in a similar way with the important difference that it uses an extensible NF2 algebra as an intermediary, whose inherent nesting capabilities are fruitful for optimizing XML views.

Storing XML documents in a relational DBMS starts with finding a good storage scheme. More techniques in the ‘shredding’ category that can be mentioned are, for example, the inlining techniques of Shanmugasundaram [47], the binary fragmentation of [44], and the XML mapping of TIMBER [31]. In the context of TIMBER, important query optimization work has been done. TIMBER employs multi-level optimization, namely on the level of the TAX intermediate algebra and on the level of a relational physical algebra.

Often, query optimization on the level of the query language itself is attempted. Quilt [36], a precursor to XQuery, is a good example. XPath offers opportunities for optimization as well. We already mentioned the XPath symmetries-based work of [42]. A thorough investigation of the structural properties of XPath for the purpose of simplification and optimization of XPath expressions, can be found in [3]. Other notable work is based on the manipulation of a query tree, a tree-representation of a query, often supporting some subset of XPath. Query trees are manipulated by transforming certain patterns and evaluated using ‘holistic twig joins’ [12].

Much work is devoted to dealing with the problem of the hierarchical nature of XML. Shredding-based approaches put data of similarly structured elements into one relation. The per-node based storage schemes completely flatten the tree into one relation using labeling
schemes to encode the tree structure. The XPath Accelerator is one of many labeling schemes that share similar properties [15, 33]. The other significant difference between XML and the relational model, is that XML is ordered. [41] addresses the topic of order optimization. The work of Hidders et al [29] on avoiding unnecessary duplicate removal and sorting also addresses this issue.

Unfortunately, besides the work mentioned in the introduction, there is little more on the subject of answer size and cost estimation for XML that can be mentioned.

1.6 Embedding of the Proposed Research

As mentioned before, Opaque has a strong connection with Pathfinder. Pathfinder is the name of the consortium that jointly develops and performs research for the XML DBMS MonetDB/XQuery in the context of various projects.

Pathfinder originates at the DBIS group of the University of Konstanz with Torsten Grust and his PhD student Jens Teubner [25]. During a visiting professorship in Konstanz of the principle investigator of Opaque, collaboration between the database groups of the Universities of Twente and Konstanz was established [24, 23, 53, 22]. The CIRQUID project [30] (NWO) adopted the XPath Accelerator approach soon after it started in November 2002. CIRQUID attempts to integrate relevance-oriented querying of semi-structured data (i.e., XML) with traditional querying. Cooperation on this topic with CWI in CIRQUID was strengthened with the Bsik-project MultimediaN (start April 2004). One of the first goals of the sub-project AmbientDB is to develop a Pathfinder front-end for MonetDB supporting full XQuery according to the latest ideas for evaluating FLWOR-expressions in a DBMS [27, 28].

Work on cost estimation in the context of Pathfinder started with the Master’s project of Henning Rode (University of Konstanz) supervised by the principle investigator of Opaque [43]. Subsequent investigation for Opaque concerned logic query optimization for XQuery using relational algebra [54, 57, 26]. Opaque will combine the ideas of these two efforts and further develop them.

In the SUMMER-project, the shredding-approach (see Section 1.1) was followed to establish XML-views over existing relational databases, which can be queried with XQuery [55]. Important in this project was the use of an intermediary NF² algebra, called dMoa.

2 Work Program

2.1 Planning

This section provides a division of the project into two phases and further details concerning the tasks to be done in these phases (see Figure 3). The idea behind the two phases is to first aim for carrying out experiments with a fully functional but restricted prototype system, and then iteratively fine-tune and enhance. Phase 1 will take at maximum 1.5 years.

The OiO will first perform a literature study on query optimization and cost estimation in general and investigates the state of the art concerning these topics in the context of XML. To give this task a tangible deliverable and deadline, a technical report will be written on the state of the art to be completed after six months. Subsequently, the OiO will, in cooperation with the PhD student in Konstanz (Jens Teubner), establish a base algebra definition, define equivalence rules, and define a cost model. Since these definitions play a central part in this research, they will be written down in a technical report and kept up to date during the
project. For a large part in parallel with this, prototype software modules will be constructed for a query plan rewriter based on algebra and equivalence rules, and a query plan evaluator based on the cost model. This software development happens for a large part in parallel, because it provides valuable early feedback on the more theoretical task of defining algebra, equivalence rules, and cost model. At maximum 1.5 years from the start, a fully functional but restricted prototype system should be ready. In Phase 2, performance experiments are carried out, bottlenecks are being investigated, solutions developed in terms of refinements of algebra, equivalence rules, and cost model. This is a highly iterative process in which the experimental results provide focus and direction. Phase 2 will take the remaining 2.5 years of the project.

Opaque has no overlap with other efforts in the context of Pathfinder. At its earliest, the project will start in spring 2006. By that time, the PhD research of Jens Teubner is expected to enter into its final stages. No research on query optimization will be done in the AmbientDB-project. It focuses on turning Pathfinder into a usable product and on its application in schema and data integration for an ambient environment.

### 2.2 Educational aspects

The term OiO states clearly that he/she is a researcher in training. It is policy that for all PhD students, a “AiO opleidings- en begeleidingsplan” is written before the PhD student begins his/her research. Since the plan requires discussion and agreement with the PhD student, it is sometimes written in the first or second month, for example, when the PhD student comes from abroad and is simply not available beforehand. The plan contains:

- General aim.
- Supervision, evaluation, and other formal matters.
Task and responsibility concerning research and education.

- How to organize working in and handling deficiencies in skills and knowledge.

Solutions for the latter often entail following certain courses at the university, research school courses, PhD-level summer school (although no formal right, for many years now, every PhD student of the DB-group has got the opportunity to go to a summer school once, e.g., EDBT), and doing private study. Please note that the plan is considered as a contract between PhD student and university and, hence, contains responsibilities for both sides.

3 Literature

As presented in Section 1.6, the Pathfinder work started with Grust’s paper on the XPath Accelerator for SIGMOD 2002 [25]. The collaboration during van Keulen’s visiting professorship resulted in two important publications, one on the experiences with the XPath Accelerator on MonetDB for the ACM TODS journal [24], and one on the staircase join for VLDB 2003 [23].

Boncz’ main research theme is main-memory databases with many important papers; one of the latest is for CIDR 2005 [11]. Within the AmbientDB project, van Keulen and Boncz moved into the area of data management for ambient intelligence. A selection of the first publications is worth mentioning as they provide a good indication of the kind of applications we have in mind for the Pathfinder XML DBMS to which this research contributes: one paper on peer-to-peer databases [7] and one on probabilistic data management for data integration [56].

The above paragraph describes important papers of the ‘research team’ consisting of not only the OiO and his supervisor, but also the people the OiO closely cooperates with. Taking the viewpoint of ‘research team’ consisting of only the OiO and his supervisor, there are a few more papers of van Keulen to be mentioned besides the already mentioned papers [24, 23, 56].

First, an algebraic approach to XML data management based on shredding was explored in a paper for DEXA 2003 [55] (see also Section 1.6). Second, he investigates the vision of personal schema-based querying as a new way of querying XML data on the internet [48, 50, 49].
Appendix A  Introduction to the XPath Accelerator [54]

The XPath Accelerator [25, 24] is based on a numbering scheme where each node \( v \) is assigned a pre-order rank \( \text{pre}(v) \) and a post-order rank \( \text{post}(v) \). As many have noted, these numbers efficiently characterize XPath location steps as document regions, also called query windows, e.g.,

\[
\begin{align*}
\forall v' & \text{ is a descendant of } v \\
& \iff \text{pre}(v) < \text{pre}(v') \land \text{post}(v') < \text{post}(v)
\end{align*}
\]

Other properties of the numbering scheme can be more easily seen, when the nodes are placed in a pre/post plane according to their pre-order (x-axis) and post-order (y-axis) ranks (see Figure 4). For example, for any node \( v \), when moving to the right, one first encounters nodes below \( v \) and then nodes solely above \( v \). Such tree-specific properties can be exploited in tree-aware algorithms computing the query windows.

XPath expressions can be translated to SQL-queries by simply using the query window associated with the axis step in the WHERE-clause (see Figure 5). Other conditions like node and name tests simply result in additional predicates in the WHERE-clause.

```sql
SELECT DISTINCT v2.*
FROM context c, accel v1, accel v2
WHERE c.pre < v1.pre AND v1.post < c.post
AND v1.name = n
AND v2.pre < v1.pre AND v2.post < v1.post
AND v2.par = v1.par
AND v2.kind = text
ORDER BY v2.pre ASC
```

![Figure 4: Pre/post plane and node encoding table accel. Lines indicate document regions as seen from context nodes f (•—•) and i (..), resp.](image)

![Figure 5: SQL-query for the XPath expression /descendant-or-self::n/preceding-sibling::text() (context is a table containing the root node).](image)

'b', which is a child of the root node. We refer to [24, 25] for more details.

The storage scheme described in [24] represents each node by a row in the table accel with the following attributes: (1) \( \text{pre} \) (pre-order rank), (2) \( \text{post} \) (post-order rank), (3) \( \text{par} \) (pre-order rank of parent), (4) \( \text{kind} \) (node kind), and (5) \( \text{name} \) (tag name of an element or attribute). Other information, such as the text/value of text and attribute nodes, is stored in other tables (not used here). For example, the tuple \(<\text{pre} = 1, \text{post} = 1, \text{par} = 0, \text{kind} = \text{elem}, \text{name} = \text{‘b’}>\) describes an element with tag name.
References


M. van Keulen. Relational Approach to Logical Query Optimization of XPath. In Mihajlovic and Hiemstra [40], pages 52–58.


