τBench: Extending XBench with Time

Stephen W. Thomas, Richard T. Snodgrass, and Rui Zhang

March 16, 2013

TR-92

A TIMECENTER Technical Report
Title: τBench: Extending XBench with Time

Copyright © 2013 Stephen W. Thomas, Richard T. Snodgrass, and Rui Zhang. All rights reserved.

Author(s): Stephen W. Thomas, Richard T. Snodgrass, and Rui Zhang

Publication History: December 2010: Initial release.
March 2013: Updated content, to align with Software: Practice and Experience publication and release of tBench 0.2.

TIMECENTER Participants

Michael H. Böhlen, University of Zurich, Switzerland; Curtis E. Dyreson, Utah State University, USA; Fabio Grandi, University of Bologna, Italy; Christian S. Jensen (codirector), Aarhus University, Denmark; Vijay Khatri, Indiana University, USA; Gerhard Knolmayer, University of Berne, Switzerland; Carme Martín, Technical University of Catalonia, Spain; Thomas Myrach, University of Berne, Switzerland; Mario A. Nascimento, University of Alberta, Canada; Sudha Ram, University of Arizona, USA; John F. Roddick, Flinders University, Australia; Keun H. Ryu, Chungbok National University, Korea; Simonas Šaltenis, Aalborg University, Denmark; Dennis Shasha, New York University, USA; Richard T. Snodgrass (codirector), University of Arizona, USA; Paolo Terenziani, University of Piemonte Orientale “Amedeo Avogadro,” Alessandria, Italy; Stephen W. Thomas, Queen’s University, Canada; Kristian Torp, Aalborg University, Denmark; Vassilis Tsotras, University of California, Riverside, USA; Fusheng Wang, Emory University, USA; Jef Wijsen, University of Mons-Hainaut, Belgium; and Carlo Zaniolo, University of California, Los Angeles, USA

For additional information, see The TIMECENTER Homepage:
URL: <http://www.cs.aau.dk/TimeCenter>

Any software made available via TIMECENTER is provided “as is” and without any express or implied warranties, including, without limitation, the implied warranty of merchantability and fitness for a particular purpose.

The TIMECENTER icon on the cover combines two “arrows.” These “arrows” are letters in the so-called Rune alphabet used one millennium ago by the Vikings, as well as by their predecessors and successors. The Rune alphabet (second phase) has 16 letters, all of which have angular shapes and lack horizontal lines because the primary storage medium was wood. Runes may also be found on jewelry, tools, and weapons and were perceived by many as having magic, hidden powers.

The two Rune arrows in the icon denote “T” and “C,” respectively.
## Contents

Table of Contents  
List of Figures  
List of Tables  

1 Introduction  
  1.1 Anatomy of a Benchmark  
  1.2 XBench  
    1.2.1 Data Generation  
    1.2.2 Data Size  
    1.2.3 DC/SD: The Book Store Catalog (catalog.xml)  
  1.3 adultsXSchema  
  1.4 adultsXQuery  
  1.5 Persistent Stored Modules  
  1.6 adultsPSM  
  1.7 Benchmark Frameworks  

2 The Benchmarks of adultsBench  
  2.1 The DC/SD XBench Benchmark  
    2.1.1 Schema  
    2.1.2 Data  
    2.1.3 Workload  
  2.2 The adultsXBench Benchmark  
    2.2.1 Schema  
    2.2.2 Data  
    2.2.3 Workload  
  2.3 The adultsPSM and adultsPSM-DB2 Benchmarks  
    2.3.1 Data  
    2.3.2 Schema  
    2.3.3 Workload  
  2.4 The adultsPSM and Related Benchmarks  
    2.4.1 Data  
    2.4.2 Schema  
    2.4.3 Workload  
  2.5 The adultsXSchema Benchmark  
    2.5.1 Schema  
    2.5.2 Data  
    2.5.3 Workload  

3 Supporting Tools  
  3.1 adultsGenerator  
    3.1.1 Simulation Description  
    3.1.2 Maintaining the Original Schema Constraints  
    3.1.3 Shredding XML Data into Relations  
    3.1.4 Input  
    3.1.5 Output  
    3.1.6 Compilation and Usage  
  3.2 Validation Tools  
    3.2.1 Validating the Output of adultsGenerator  
    3.2.2 Corrupting Data  
    3.2.3 Checking Primary Keys of Shredded Relations  
    3.2.4 Checking Referential Integrity of Shredded Relations  

i
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The three components of a benchmark.</td>
</tr>
<tr>
<td>2</td>
<td>A subset of the Book Store schema.</td>
</tr>
<tr>
<td>3</td>
<td>The Benchmark-Relationship diagram of the τBench framework.</td>
</tr>
<tr>
<td>4</td>
<td>The cardinality constraint (C5) of XBench’s XML Schema.</td>
</tr>
<tr>
<td>5</td>
<td>Query Q2 of the XBench benchmark.</td>
</tr>
<tr>
<td>6</td>
<td>Query Q2 of the τXQuery benchmark, with a validtime statement modifier.</td>
</tr>
<tr>
<td>7</td>
<td>Two versions of a simplified &lt;author&gt; temporal element.</td>
</tr>
<tr>
<td>8</td>
<td>Cardinality constraint C5 of the PSM benchmark schema.</td>
</tr>
<tr>
<td>9</td>
<td>Query Q2 of the PSM benchmark.</td>
</tr>
<tr>
<td>10</td>
<td>Query Q2 of the PSM-DB2 benchmark.</td>
</tr>
<tr>
<td>11</td>
<td>Cardinality constraint C5 of the τPSM benchmark schema.</td>
</tr>
<tr>
<td>12</td>
<td>Query Q2 of the τPSM benchmark.</td>
</tr>
<tr>
<td>13</td>
<td>Query Q2 of the τPSM-DB2 benchmark.</td>
</tr>
<tr>
<td>14</td>
<td>The non-sequenced cardinality constraint (C5) of τXSchema’s XML Schema.</td>
</tr>
<tr>
<td>15</td>
<td>Overview of the τGenerator process.</td>
</tr>
<tr>
<td>16</td>
<td>Commutativity validation.</td>
</tr>
<tr>
<td>17</td>
<td>A simplified slice of the temporal catalog.xml document.</td>
</tr>
<tr>
<td>18</td>
<td>The item table creation and constraint definitions in SQL.</td>
</tr>
<tr>
<td>19</td>
<td>The author table creation and constraint definitions in SQL.</td>
</tr>
<tr>
<td>20</td>
<td>The item_author table creation and constraint definitions in SQL.</td>
</tr>
<tr>
<td>21</td>
<td>The item_publisher table creation and constraint definitions in SQL.</td>
</tr>
<tr>
<td>22</td>
<td>The publisher table creation and constraint definitions in SQL.</td>
</tr>
<tr>
<td>23</td>
<td>The related_items table creation and constraint definitions in SQL.</td>
</tr>
<tr>
<td>24</td>
<td>The item table creation and constrain definitions in SQL.</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A description of the τBench benchmark framework.</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>The features highlighted by the queries in the XBench Benchmark. Reproduced from [52].</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>The characteristics of DS0.*. DS0.\texttt{HUGE} is excluded since it is too large to run in the current implementation of τGenerator. Note that the sizes of the input \texttt{catalog.xml} files and the output \texttt{output.date.xml} files differ, even though they have the same content, because τGenerator outputs the XML files with indentation, whereas XBench output is more compressed.</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>The characteristics of the \texttt{LARGE} class of the four temporal datasets we define in τBench.</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>The two versions of the \texttt{&lt;author&gt;} temporal element, shredded into two tuples.</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>The implicit and explicit schema constraints in the XBench benchmark.</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>The additional schema constraints in the τ\texttt{XSchema} benchmark.</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>The inputs into τ\texttt{Generator}.</td>
<td>25</td>
</tr>
<tr>
<td>9</td>
<td>The output files of τ\texttt{Generator}.</td>
<td>25</td>
</tr>
</tbody>
</table>
1 Introduction

Time is God’s way of keeping everything from happening at once.
-Anonymous

Since the advent of the computer in the 1960s, humans have recorded increasingly large amounts of data into various digital formats, such as relational databases, spreadsheets, and flat text files. The extensible markup language (XML) has become one of the more popular methods for storing and exchanging documents and data [45], due to its flexibility and self-describing nature. One source of evidence for XML’s popularity is the huge number of tools that have recently been developed to store, maintain, query, and validate XML documents.

X Bench is a family of benchmarks typically used to evaluate existing XML tools [33, 51, 52]. The authors of X Bench analyzed several real-world datasets to measure their characteristics (such as typical data types and length of records), and the X Bench datasets reflect these measurements. Having such a benchmark creates an environment for fair, accurate, realistic, and reproducible comparisons amongst competing XML tools and algorithms.

However, X Bench only provides a single snapshot of a dataset, even though the data typically modeled in XML is known to change over time: companies evolve with new products and employees; new scientific discoveries are made; stock markets fluctuate on a minute-by-minute basis; and social networks are expanded. As such, temporal (or time-varying) data naturally arises in many XML applications. A temporal XML document records the evolution of an XML document over time, i.e., all of the versions of the document. Capturing a document’s evolution is vital to supporting time travel queries that delve into a past version, and incremental queries that involve the changes between two versions.

Despite the prevalence of temporal data in real-world applications, no temporal benchmarks currently exist. Several strategies for managing temporal XML documents have been proposed [13, 2, 3, 4, 7, 10], but there is no easy way to compare them in a practical setting. DBMS vendors are starting to add (limited) temporal support, but again, there is no standard mechanism for comparing the performance of two DBMSes.

To create a standard comparison mechanism for temporal applications, we introduce a benchmark framework called \( \tau \) Bench. A benchmark framework is an ecosystem of benchmarks that are inter-connected in semantically-rich ways [43]. \( \tau \) Bench consists of a suite of temporal and non-temporal benchmarks, all derived from X Bench, as well as a suite of tools for generating and validating each benchmark. We describe our goals for creating such a suite and our mechanical processes for generating and validating each benchmark. We have made the \( \tau \) Bench suite of benchmarks and tools publicly available [44].

In addition to temporal XML, \( \tau \) Bench contains benchmarks for persistent stored modules (PSM) in SQL, and their temporal counterparts, \( \tau \) PSM.

A cornerstone of the \( \tau \) Bench tool suite is a temporal data generation tool, \( \tau \) Generator, which can randomly generate time-varying documents in several data formats. We describe the user-specified inputs into the tool and describe the internal simulation that randomly changes data elements at specified time intervals.

Finally, we describe an initial set of benchmarks that we have created in \( \tau \) Bench, which currently subsumes technologies such as XML Schema, X Query, \( \tau \) X Query, \( \tau \) X Schema, persistent stored modules (PSM), and \( \tau \) PSM.

1.1 Anatomy of a Benchmark

As depicted in Figure 1, a benchmark consists of three components: data, schema for the data, and workloads to perform on the data. This general definition of a benchmark allows for the data to take on various forms and formats, the schema to loosely or tightly constrain the characteristics of the data, and any sort of workload to be performed (for example, queries, constraint checking, or bulk loads).

To define a benchmark, one must define all three components of the benchmark. Providing only one of the components without the others does not define a usable benchmark and is therefore of limited value. In this report, we introduce a suite of benchmarks, each with these three components.

1.2 X Bench

X Bench is a family of benchmarks for XML database management systems (DBMSes), as well as a data generation tool for creating the benchmarks [52]. The X Bench benchmarks have been widely used as standard datasets to test the performance and functionalities of new and existing XML tools and DBMSes.
To create realistic and appropriate datasets, the authors of XBench analyzed several real-world XML datasets to quantify their characteristics and relationships. As a result of this analysis, XBench characterizes database applications along two dimensions: application characteristics and data characteristics. Application characteristics indicate whether the database is data-centric or text-centric. In data-centric (DC) XML, the set of XML vocabularies represent data that is more tightly structured than in text-centric (TC) XML. Text-centric XML is used when authoring loosely structured natural language documents such as articles or blogs. In terms of data characteristics, two classes are identified: single document (SD) and multiple document (MD). In the single document case, the database consists of a single document with complex structures, while the multiple document cases contain a set of XML documents. Thus, XBench consists of four different categories that cover DC/SD, DC/MD, TC/SD, and TC/MD respectively.

1. **TC/SD.** The text-centric, single-document category is represented by a single document that is text-dominated. This category contains repeated similar entries, deep nesting, and references between entries. This category is based on analysis of the Oxford English Dictionary [49] and the GNU version of The Collaborative International Dictionary of English (GCIDE) [22].

2. **TC/MD.** The text-centric, multiple-document category is represented by numerous small documents that are text-dominated. This category contains references between elements and recursive elements. This category is based on analysis of the Reuters news corpus and Springer digital library.

3. **DC/SD.** The document-centric, single-document category is represented by a single document that contains little text. This category includes mostly transactional data where the element tags are more descriptive and contain less text content. This category is based on the analysis of TPC-W [29] benchmark.

4. **DC/MD.** The document-centric, multiple-document category is represented by five documents that each contain little text. Like the DC/SD category, the DC/MD category includes mostly transactional data where element tags are more descriptive and contain less text content and is based on the analysis of TPC-W benchmark.

For the initial purposes of XBench, we use the DC/SD category of documents from XBench since valid-time concepts are more applicable than in the other three categories.

### 1.2.1 Data Generation

XBench populates each data element and attribute with random data obtained from ToXgene [6]. ToXgene is a template-based tool that generates synthetic XML documents according to one or more templates.

The actual text content of each data element or attribute is a randomly generated string or number, depending on the specification in the ToXgene template. For example, an instance of the `<last_name>` subelement of an `<author>` could be the string “BABABABAOGREAT”. Thus, in the general case, it is not useful to consider or analyze the actual data content of the elements and attributes. However, a `<related_ID>` element will always point to an ID of the correct form (i.e., the letter “I” followed by a positive integer); other schema constraints can be enforced within ToXgene.
1.2.2 Data Size

For scalability purposes, XBench has defined four data size classes for each of the categories: small (10MB), normal (100MB), large (1GB), and huge (10GB).

1.2.3 DC/SD: The Book Store Catalog (catalog.xml)

The DC/SD XML document created by XBench, called catalog.xml, contains information about a book store. The full XML Schema is listed in Appendix B.1; an important subset is shown in Figure 2. In short, <item> (i.e., book) elements contain a list of <author>s, a <publisher>, and a list of <related_items> of related books.

It is worth emphasizing that there is a strict one-to-many relationship between <item>s and <author>s: although an item can have several authors, an author is only the author of a single item (no two items share an author). Similarly, there is a strict one-to-one relationship between <item>s and <publisher>s: an item has exactly one publisher, and each publisher publishes exactly one item.

1.3 τXSchema

τXSchema (Temporal XML Schema) is a language and set of tools that enable the construction and validation of temporal XML documents [13, 15, 24, 38, 41]. τXSchema extends XML Schema with the ability to define temporal element types. A temporal element type denotes that an element can vary over time, describes how to associate temporal elements across slices (or snapshots, which are individual versions of a document), and provides some temporal constraints that broadly characterize how a temporal element can change over time.

In τXSchema, any element type can be turned into a temporal element type by including a simple logical annotation (stating whether an element or attribute varies over valid time or transaction time, whether its lifetime is described as a continuous state or a single event, whether the item itself may appear at certain times and not at others, and whether its content changes) in the type definition. So a τXSchema document is just a conventional XML Schema document with a few annotations.

τXSchema provides a suite of tools to construct and validate temporal documents. A temporal document is validated by combining a conventional validating parser with a temporal constraint checker. To validate a temporal document, a temporal schema is first converted to a representational schema, which is a conventional XML Schema document that describes how the temporal information is represented. A conventional validating parser is then used to validate the temporal document against the representational schema. Then, the temporal constraint checker is used to validate any additional temporal constraints specified by the user in the temporal schema.

1.4 τXQuery

τXQuery adds temporal support to XQuery [47] by extending its syntax and semantics [19]. τXQuery moves the complexity of handling time from the user/application code to the query processor.
In particular, τXQuery adds two temporal statement modifiers to the XQuery language: current and sequenced. A current query is a query on the current state of the XML data (i.e., elements and attributes that are valid now) and has the same semantics as a regular XQuery query applied to the current state of the XML data. Sequenced queries, on the other hand, are queries applied to each point in time, resulting in a sequence of temporal elements. Finally, τXQuery also has representational (also termed non-sequenced) queries, which query the time-line of the XML data irrespective of time. No statement modifiers are needed for representational queries.

1.5 Persistent Stored Modules

Persistent stored modules (PSM) is the Turing-complete portion of SQL in which stored programs are compiled and stored in a schema and can be executed on the SQL server by queries [16, 28]. PSM consists of stored procedures and stored functions, which are collectively referred to as stored routines. The SQL/PSM standard [16] provides a set of control-flow constructs (features) for SQL routines.

1.6 τPSM

τPSM extends PSM to allow temporal queries [19]. The approach requires minor new syntax beyond that already in SQL/Temporal to define PSM routines. τPSM enables current, sequenced, and non-sequenced semantics of queries and of PSM routines to be realized.

1.7 Benchmark Frameworks

Benchmark frameworks are fully described elsewhere [43]; here we only provide a brief introduction.

A benchmark framework is an ecosystem of benchmarks that are inter-connected in semantically-rich ways. For example, the data used in one benchmark might be derived from the data of another (e.g., by increasing the size of the data), while the workloads and schemas of the two benchmarks are identical.

Like software frameworks, the creation of a benchmark framework is not mechanical. Instead, they are created iteratively: beginning with a root (or foundation) benchmark B, a new benchmark B′ is created by changing one or more of the six components of B. A relationship between B and B′ is made to describe the changes. In this way, a DAG of benchmarks emerges, where nodes are benchmarks and each edge is the relationship between each pair of benchmarks. This graph exactly describes a benchmark framework. At this point, benchmark frameworks depart in design from software frameworks such as the Java Collections framework. To build the Collections framework, the designers had to first boil the ocean: almost the entire framework had to be in place before any of it was useful. In contrast, benchmark frameworks are useful immediately, as new benchmarks can easily be derived whether there are two benchmarks or two hundred present in the framework.
2 The Benchmarks of τBench

Using the methodology fully described elsewhere [43], we have created a benchmark framework called τBench. As we describe below, τBench makes use of the organizational insight of using statement modifiers, commutativity validation, and temporal upward compatibility when deriving temporal benchmarks from non-temporal benchmarks.

Depicted in Figure 3, τBench currently consists of ten individual benchmarks, spanning various technologies including XML, XML Schema [30, 31], XQuery [47], τXSchema [13, 15, 14, 24, 38, 41], τXQuery [20, 21, 19], PSM [16, 28], and τPSM [19, 39]; the Benchmark-Relationship diagram is given in Figure 3 and the main components are summarized in Table 1.

2.1 The DC/SD XBench Benchmark

XBench is a family of benchmarks for XML DBMSes, as well as a data generation tool for creating the benchmarks [51]. The XBench family has been widely used as standard datasets to investigate the functionalities of new and existing XML tools and DBMSes. (Other XML benchmarks include MBench [34], MemBeR [1], Transaction Processing over XML (TPoX) [32], XMach-1 [9], XMark [35], XPathMark [17], and XOO7 [11]. Most of these benchmarks focus on XQuery processing, as does XBench.)

To create the datasets, the authors of XBench analyzed several real-world XML datasets to quantify their characteristics and relationships. As a result of this analysis, XBench characterizes database applications along two independent dimensions: application characteristics (i.e., whether the database is data-centric or text-centric) and data characteristics (i.e., single-document vs. multi-document). Thus, XBench consists of four different categories that cover DC/SD, DC/MD, TC/SD, and TC/MD, respectively.

The DC/SD XBench benchmark is one of the four constituent benchmarks of the XBench family, based strictly on the DC/SD category. We use this benchmark as the foundation of τBench, due to its combination of...
2.1.1 Schema

The schema of DC/SD XBench is specified as an XML Schema [30, 31]. The schema defines the library catalog, where <item> (i.e., book) elements contain a list of <author>s, a <publisher>, and a list of <related_items> of related books (Figure 2). Each <item>, <author>, and <publisher> has a unique id attribute. The schema specifies several constraints, such as the uniqueness of the id attributes and that items can be related to between 0 and 5 other items.

The schema is given in Appendix B.1 and Table 6 summarizes the (implicit and explicit) constraints defined by the schema. For simplicity, the table does not list data type constraints nor implied (default) cardinality constraints (i.e., all sub-elements of an element must occur exactly once). It is these constraints that we preserve as we derive the schemas for the other benchmarks. We also observe and maintain these constraints during the temporal simulation. (Details are provided in Section 3.1.)

As a running example throughout this Section, consider Figure 4. The listing shows the XML Schema cardinality constraint that says: “Each <authors> element must have between one and four <author> subelements.” The constraint is specified via the minOccurs and maxOccurs attributes of the <element> element of XML Schema. As new benchmarks are added to the framework, we will show how this constraint gets transformed into new schema languages and models.

![Figure 4: The cardinality constraint (C5) of XBench’s XML Schema.](image)

2.1.2 Data

XBench populates each data element and attribute with random data generated from ToXgene [6]. ToXgene is a template-based tool that generates synthetic XML documents according to one or more user-defined templates.

The data in DC/SD XBench consists of a single document, catalog.xml, which contains information about a library catalog, i.e., items (books), authors, publishers, and related items. The actual text content of each element or attribute is a randomly generated string or number, depending on the specification in the ToXgene template. For example, an instance of the <last_name> of an <author> could be the string “BABABABAO-.
for $item in input()/catalog/item
where $item/authors/author/name/first_name = "Ben"
return
$item/title

Figure 5: Query Q2 of the XBench benchmark.

<table>
<thead>
<tr>
<th>Description</th>
<th>Highlighted Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Return the item that has matching item id attribute value (I1).</td>
</tr>
<tr>
<td>Q2</td>
<td>Find the title of the item which has matching author first name (Ben).</td>
</tr>
<tr>
<td>Q3</td>
<td>Group items released in a certain year (1990), by publisher name and calculate the total number of items for each group.</td>
</tr>
<tr>
<td>Q4</td>
<td>List the item id of the previous item of a matching item with id attribute value (I2).</td>
</tr>
<tr>
<td>Q5</td>
<td>Return the information about the first author of item with a matching id attribute value (I3).</td>
</tr>
<tr>
<td>Q6</td>
<td>Return item information where some authors are from certain country (Canada).</td>
</tr>
<tr>
<td>Q7</td>
<td>Return item information where all its authors are from certain country (Canada).</td>
</tr>
<tr>
<td>Q8</td>
<td>Return the publisher of an item with id attribute value (I4).</td>
</tr>
<tr>
<td>Q9</td>
<td>Return the ISBN of an item with id attribute value (I5).</td>
</tr>
<tr>
<td>Q10</td>
<td>List the item titles ordered alphabetically by publisher name, with release date within a certain time period (from 1990-01-01 to 1995-01-01).</td>
</tr>
<tr>
<td>Q11</td>
<td>List the item titles in descending order by date of release with date of release within a certain time range (from 1990-01-01 to 1991-01-01) but do not have FAX number.</td>
</tr>
<tr>
<td>Q12</td>
<td>Get the mailing address of the first author of certain item with id attribute value (I6).</td>
</tr>
<tr>
<td>Q13</td>
<td>Return the names of publishers who publish books between a period of time (from 1990-01-01 to 1991-01-01) but do not have FAX number.</td>
</tr>
<tr>
<td>Q14</td>
<td>Return the ids of items whose descriptions contain a certain word (&quot;hockey&quot;).</td>
</tr>
<tr>
<td>Q15</td>
<td>Retrieve the item titles related by certain item with id attribute value (I7).</td>
</tr>
<tr>
<td>Q16</td>
<td>Retrieve the item title whose size (length<em>width</em>height) is bigger than certain number (500000).</td>
</tr>
</tbody>
</table>

Table 2: The features highlighted by the queries in the XBench Benchmark. Reproduced from [52].

GREAT”. (However, some content is not random. For example, a `<related_ID>` element will always point to an existing item ID.)

A crucial point to note is that, because of the data generation process, there is a strict one-to-many relationship between items and authors: although an item can have several authors, an author is only the author of a single item (no two items share an author). Similarly, there is a strict one-to-one relationship between items and publishers: an item has exactly one publisher, and each publisher publishes exactly one item. These constraints are not, however, explicit in the schema, but must be satisfied by tools that manipulate the data, as we shall see in Section 3.1.

2.1.3 Workload

The workload consists of 17 XQuery [47] queries. Each query tests a different feature of the query engine, such as existential quantifiers (query q6: “Return items where some authors are from Canada”), sorting by string types (query q10: “List the item titles ordered alphabetically by publisher name”), and uni-gram searching (query q17: “Return the ids of items whose descriptions contain the word hockey”). Each such query can thus be considered a microbenchmark [34], designed to test the performance of a specific language construct. Other microbenchmarks include IMbench [27] (addressing system latency), MemBeR [1] (also XQuery), and MBench [34] (also XQuery).

A description of each query and its highlighted functionality is shown in Table 2. Figure 5 shows an example query (Q2): “Find the title of the item which has matching author first name (Ben)”. In the query, `input()` is the input document (assumed to already be opened).
2.2 The $\tau$XBench Benchmark

This benchmark adds a temporal dimension to the XBench DC/SD benchmark, so that temporal databases and query languages can be evaluated.

In particular, $\tau$XQuery is a temporal query language that adds temporal support to XQuery by extending its syntax and semantics [20, 21, 19]. $\tau$XQuery moves the complexity of handling time from the user/application code to the query processor. $\tau$XQuery adds two temporal statement modifiers [8] to the XQuery language: current and sequenced. A current query is a query on the current state of the XML data (i.e., elements and attributes that are valid now) and has the same semantics as a regular XQuery query applied to the current state of the XML data. Sequenced queries, on the other hand, are queries applied to each point in time, resulting in a sequence of temporal elements. Finally, $\tau$XQuery also has representational (also termed non-sequenced) queries, which query the time-line of the XML data irrespective of time. No statement modifiers are needed for representational queries.

The $\tau$XBench benchmark was created to investigate $\tau$XQuery and other temporal query languages.

2.2.1 Schema

The schema of $\tau$XBench is a temporal XML schema, a concept defined in the $\tau$XSchema framework [14, 13]. A temporal schema is intuitively similar to an XML Schema, except with annotations to specify which elements (i.e., books, authors, and publishers) may vary over time. In practice, a temporal schema contains three types of schemas: a conventional XML schema to describe each XML slice; logical and physical annotations to specify the logical and physical timestamps, respectively; and a representational schema to describe the physical representation of the temporal XML document.

The conventional schema, temporal annotations, and representational schemas for this benchmark are given in Appendices B.1, B.2, and B.4, respectively.

To define the temporal schema, we set the XML Schema from XBench DC/SD as the sole conventional schema; we do not specifying any logical annotations (and therefore use the sequenced semantics of each of the original constraints in the conventional schema) and place the physical timestamps at the <item>, <author>, <publisher>, and <related_author> elements, since that’s how our data generation tool works (Section ??); and we use the representational schema that is automatically created using SCHEMA_MAPPER, which is part of the $\tau$XSchema tool suite. (The representational schema is derived from the conventional schema and physical annotations.)

Validation. Since the conventional schema is duplicated from the XBench schema, it does not require validation.

It is important to note that even though the logical annotations do not contain any explicit temporal constraints, there are implicit constraints present. By design, a temporal schema preserves the sequenced semantics of its conventional schemas. Thus, the semantics of the original XML schema of XBench DC/SD), and in particular its integrity constraints, are preserved in the temporal schema. As an example, the non-temporal constraint in Figure 4 would still apply to each of the XML data slices, and thus at each point of time of the temporal document.

The representational schema describes how the temporal data is represented (e.g., <item_RepItem> and <item_Version>) and is created automatically by the $\tau$XSchema tool suite based on the temporal XML Schema.

All three of these parts of the temporal schema are validated by triangulation of the validation of the data: if each slice of a temporal document is validated against the conventional schema, and if the temporal document is validated against the temporal schema, that helps ensure that the temporal schema and representational schemas are themselves valid. Also, the temporal schema is validated against its XML Schema schema. In testing, we frequently encountered validation failures of the data that indicated subtle problems in other components, whether the generation code or one of the schema specifications. Each of these failures gave us reassurance that the sum total of the validation required a high degree of correspondence and consistency between the many parts.

2.2.2 Data

The data of $\tau$XBench is the result of applying a temporal simulation, $\tau$Generator (Section 3), to the non-temporal XBench DC/SD dataset. This derived temporal dataset consists of books whose titles, authors, and publishers change over time, authors whose names and addresses change over time, and publishers whose lists of published books changes over time.
We describe in detail the workings of $\tau$Generator in Section 3. For now, it is sufficient to know that $\tau$Generator takes as input the non-temporal XML data from XBench, along with some user-defined simulation parameters, and outputs four sets of files: a temporal XML document; a set of non-temporal XML slices that correspond to the temporal XML document; a set of shredded temporal relations; and a set of shredded non-temporal relation slices that correspond to the temporal relations.

We define four datasets, each of which is generated by $\tau$Generator, three of which are temporal, and all of which are summarized in Tables 3 and 4. Each dataset can come in any size, depending on the class of the input XBench dataset (small, medium, large, or huge). We name each dataset DS#$\cdot$.class, where ‘#’ is a unique identifier for the dataset and class is the class of the input XBench dataset. (Thus, DS1.LARGE is a temporal dataset generated from the catalog.xml document in the large DC/SD XBench dataset.) We use the notation DS#.* to refer to a dataset generated with any of the classes.

Because of the design of $\tau$Generator, which divides an input dataset into an initial slice and a “pool” of elements for use during the simulation, an input dataset must be large enough to accommodate the requested number of changes during the temporal simulation. This means that, for example, we would not be able use the SMALL XBench dataset to generate one thousand 1GB slices. Thus, depending on the characteristics of the defined temporal datasets, only certain input datasets make sense. We specify these below where applicable.

**Dataset 0 (DS0.*)** This non-temporal dataset consists of the original, unmodified DC/SD XBench catalog.xml, along with the corresponding shredded non-temporal relations.

**Dataset 1 (DS1.*)** This temporal dataset consists of DS0.* subjected to the temporal simulation with a weekly (7 day) time step and 700 changes per time step over a one year period. The initial slice of the temporal data consists of 10% of the elements from DS0.*; the remaining 90% are placed in a selection pool to be used by the simulation for the temporal changes. At each time step, the temporal changes are spread equally amongst the change types (insert, update, and delete) and temporal elements ($<$item$>$, $<$author$>$, $<$publisher$>$, and $<$related_item$>$). The total number of changes in the simulation is thus 36,400, over 52 time steps. We set the element selection type to “uniform” so that every element has an equal probability of being changed.

This dataset requires either the LARGE or HUGE XBench dataset as input.

**Dataset 2 (DS2.*)** This temporal dataset is the same as DS1.*, except with a Gaussian element selection type. We make this choice to create a hot-spot of activity at the mean of the Gaussian distribution: elements with IDs near the mean will change frequently, while elements with IDs far from the mean will likely never change.

This dataset requires either the LARGE or HUGE XBench dataset as input.

**Dataset 3 (DS3.*)** This temporal dataset is the same as DS2.*, except with a shorter time step (one day instead of seven days) and fewer changes per time step (100 versus 700). Since the number of required changes is the same, DS3.* will result in the same data characteristics as DS2.*, that is, 36,400 changes, except with seven times as many slices (364 versus 52).

This dataset requires either the LARGE or HUGE XBench dataset as input.

**Dataset 4 (DS4.*)** This temporal dataset is created in order to provide a dataset with smaller slices and a substantive selection pool size to allow a large number of slices to be created. DS4.* is the same as DS1.*, except with only 0.4% of the elements from the input XBench dataset being selected for the initial slice (and thus 99.6% placed in the selection pool) and only 10 temporal changes per time step (Note that the time step is still 7 days). The required number of slices is set to 100 (or, equivalently, 1000 changes are required), although this parameter could easily be changed for evaluation purposes (for example, varying the number of slices to see the effect on the given application).

**Validation.** The integrity of the datasets is crucial for the purposes of creating a benchmark. We therefore subject the generated datasets to a robust validation process.

The temporal XML document is validated in three ways:

- By the representational schema (created using the $\tau$XSchema tool suite [13]), to ensure it has the correct form.
## Input Size

<table>
<thead>
<tr>
<th></th>
<th>DS0.SMALL</th>
<th>DS0.MEDIUM</th>
<th>DS0.LARGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total input size (catalog.xml)</td>
<td>11MB</td>
<td>104MB</td>
<td>1.1G</td>
</tr>
</tbody>
</table>

## Output Sizes - XML documents

<table>
<thead>
<tr>
<th></th>
<th>output.date.xml size (elements)</th>
<th>item.date.csv size (rows)</th>
<th>author.date.csv size (rows)</th>
<th>publisher.date.csv size (rows)</th>
<th>related.item.date.csv size (rows)</th>
<th>item_author.date.csv size (rows)</th>
<th>item_publisher.date.csv size (rows)</th>
<th>TOTAL output size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16.1MB (2500)</td>
<td>1.1MB (2500)</td>
<td>2.9MB (6264)</td>
<td>384KB (2500)</td>
<td>70K (6401)</td>
<td>70K (6264)</td>
<td>28K (2500)</td>
<td>21MB</td>
</tr>
<tr>
<td></td>
<td>161MB (25000)</td>
<td>11MB (25000)</td>
<td>29MB (62497)</td>
<td>3.8MB (25000)</td>
<td>803KB (62636)</td>
<td>817KB (62497)</td>
<td>321KB (25000)</td>
<td>205MB</td>
</tr>
<tr>
<td></td>
<td>1.6GB (250000)</td>
<td>106MB (250000)</td>
<td>282MB (625057)</td>
<td>38MB (250000)</td>
<td>9.1MB (624685)</td>
<td>9.2MB (625057)</td>
<td>3.7MB (250000)</td>
<td>2.1GB</td>
</tr>
</tbody>
</table>

Table 3: The characteristics of DS0.*. DS0.HUGE is excluded since it is too large to run in the current implementation of τGenerator. Note that the sizes of the input catalog.xml files and the output output.date.xml files differ, even though they have the same content, because τGenerator outputs the XML files with indentation, whereas XBench output is more compressed.

## Input Parameters

<table>
<thead>
<tr>
<th></th>
<th>DS1.LARGE</th>
<th>DS2.LARGE</th>
<th>DS3.LARGE</th>
<th>DS4.LARGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIAL_PERCENTAGE</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0.4</td>
</tr>
<tr>
<td>TIME_STEP</td>
<td>7</td>
<td>7</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>Changes per time step</td>
<td>700</td>
<td>700</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>REQUIRED_CHANGES</td>
<td>36400</td>
<td>36400</td>
<td>36400</td>
<td>–</td>
</tr>
<tr>
<td>REQUIRED_SLICES</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>100</td>
</tr>
<tr>
<td>SELECTION_TYPE</td>
<td>uniform</td>
<td>gaussian</td>
<td>gaussian</td>
<td>uniform</td>
</tr>
<tr>
<td>SELECTION_STDDEV</td>
<td>–</td>
<td>100</td>
<td>100</td>
<td>–</td>
</tr>
</tbody>
</table>

## Output Sizes - XML documents

<table>
<thead>
<tr>
<th></th>
<th>output.*.xml size (elements)</th>
<th>item.final.csv size (rows)</th>
<th>author.final.csv size (rows)</th>
<th>publisher.final.csv size (rows)</th>
<th>related_item.final.csv size (rows)</th>
<th>item_author.final.csv size (rows)</th>
<th>item_publisher.final.csv size (rows)</th>
<th>TOTAL output size (XML and shredded)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>160MB (25000)</td>
<td>12.5MB (28306)</td>
<td>12.5MB (28306)</td>
<td>4MB (119875)</td>
<td>3MB (77236)</td>
<td>1.3MB (38224)</td>
<td>&lt; 0.1MB (1297)</td>
<td>12.5GB</td>
</tr>
<tr>
<td></td>
<td>160MB (25000)</td>
<td>12.5MB (28306)</td>
<td>12.5MB (28306)</td>
<td>4MB (119691)</td>
<td>3MB (77212)</td>
<td>1.4MB (38224)</td>
<td>&lt; 0.1MB (1297)</td>
<td>12.5GB</td>
</tr>
<tr>
<td></td>
<td>160MB (25000)</td>
<td>12.5MB (28306)</td>
<td>12.5MB (28200)</td>
<td>4MB (120581)</td>
<td>3MB (76723)</td>
<td>1.3MB (37800)</td>
<td>&lt; 0.1MB (1297)</td>
<td>81GB</td>
</tr>
<tr>
<td></td>
<td>6.5MB (10000)</td>
<td>0.5MB (1100)</td>
<td>0.5MB (1100)</td>
<td>0.2MB (1297)</td>
<td>0.1MB (3010)</td>
<td>0.1MB (1297)</td>
<td>–</td>
<td>860MB</td>
</tr>
</tbody>
</table>

Table 4: The characteristics of the LARGE class of the four temporal datasets we define in τBench.
Figure 6: Query Q2 of the ρXQuery benchmark, with a \texttt{validtime} statement modifier.

- By the sequenced and non-sequenced constraints in the temporal schema (also created using the ρXSchema tool suite), to ensure both types of constraints are ensured.
- By unsquashing [13] it and comparing each resulting slice to the generated slices with X-Diff [48] (see Section 3.2.5).

Should the temporal XML document pass all of these tests, we know that it is well formed and satisfies all of the constraints in the original XBench dataset.

The XML slices are validated by subjecting each to the original XBench schema using XMLLINT (see Section 3.2.1). The temporal relations are validated by checking the primary key and referential integrity constraints of each relation (see Sections 3.2.3 and 3.2.4). The relation slices are validated by checking the primary key and referential integrity constraints of each relation at each time period (see Sections 3.2.3 and 3.2.4).

2.2.3 Workload

We define a set of 85 ρXQuery queries: a current and non-sequenced query for each of the original 17 queries from XBench DC/SD, and three sequenced queries for each of the original 17 XBench queries: a short period query that only considers 10% of the time line, a medium period query that considers 50%, and a long period query that considers 90%.

The queries themselves are the same as those in the XBench DC/SD workload, except with temporal statement modifiers prepended to the query. Semantically, the queries are expanded, but we prefer in the Benchmark-Relationship diagram to emphasize that an automated mapping (such as with TXL [12]) could convert the original queries into new queries by simply prepending the temporal statement modifiers. These new queries can then be executed by any tool that understands the statement modifiers, such as the ρXQuery engine or by a source-to-source translator from ρXQuery to XQuery.

Validation. Since we only add simple statement modifiers, no validation is required.

At a different level though, if a way to evaluate ρXQuery queries existed, we could evaluate each on a temporal document, then compare the result of that query with that of the analogous non-temporal query on the slices. This process would provide a validation of the \textit{evaluator}, but also indirectly of the ρXQuery queries themselves.

2.3 The PSM and PSM-DB2 Benchmarks

Persistent stored modules (PSM) is the Turing-complete portion of SQL in which stored programs are compiled and stored in a schema and can be executed on the SQL server by queries [16, 28]. PSM consists of \textit{stored procedures} and \textit{stored functions}, which are collectively referred to as \textit{stored routines}. The SQL/PSM standard [16] provides a set of control-flow constructs (\textit{features}) for SQL routines.

We adopted the PSM benchmark from the XBench DC/SD benchmark to assess DBMSes that implement PSM functionality. The PSM-DB2 benchmark is a simple transformation of the schema and workload of the PSM benchmark so that the resulting benchmark is compatible and thus can be executed in IBM DB2.

2.3.1 Data

The data is reused as XBench DC/SD, and translated into six relations.

\textit{Shredding} is the process of transforming an XML tree into a set of two dimensional DBMS relations [36]. Doing so often requires generating additional tables that contain relational or hierarchical information. For example, an additional table is required to list the relations between items and authors in the XBench DC/SD data. This relationship is implicitly captured in XML by making author a subelement of item, but must be explicitly captured in its own relation during the shredding process.
We shred the time-varying catalog.xml into six relations: four for the original four temporal elements, one to map author IDs to item IDs, and one to map publisher IDs to item IDs. In general, each column of each shredded relation corresponds to the text content of a subelement of the corresponding temporal element. For example, the (simplified) <author> temporal element (note: not actually generated by τGenerator) with two versions shown in Figure 7 would be shredded into the two tuples shown in Table 5.

Figure 7: Two versions of a simplified <author> temporal element.

<table>
<thead>
<tr>
<th>first_name</th>
<th>date_of_birth</th>
<th>name_of_city</th>
<th>begin</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tandy</td>
<td>1972-06-29</td>
<td>Oakville</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Tandy</td>
<td>1972-06-29</td>
<td>Lincoln</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Table 5: The two versions of the <author> temporal element, shredded into two tuples.
CREATE ASSERTION number_authors CHECK
(NOT EXISTS (SELECT IA.item_id, count(*) cnt
FROM item_author IA
HAVING count(*) > 4))

Figure 8: Cardinality constraint C5 of the PSM benchmark schema.

We define foreign keys in each relation as follows.

- **related_items.** The **item_id** and **related_id** columns both refer to an existing **id** in the **item** table.
- **item_author.** The **item_id** refers to an existing **id** in the **item** table and the **author_id** refers to an existing **author_id** in the **author** table.
- **item_publisher.** The **item_id** refers to an existing **id** in the **item** table and the **publisher_id** refers to an existing **publisher_id** in the **publisher** table.

An SQL assertion was created for each of the original cardinality constraints. For example, Figure 8 shows the cardinality constraint C5 of XBench as an SQL assertion.

**Validation.** To validate the table schemas, we load the data and tables into the DBMS. If the data loads successfully (during which assertions such as in Figure 8 are checked by the DBMS), then we know the schemas are valid.

For further validation, we have written a tool called τCorruptor (Section 3.2.2) to randomly corrupt the data in the relations in such a way that one of the pre-defined constraints is violated. We run τCorruptor to corrupt the data and load it into the DBMS. If the schemas catch the violation, then we can be sure that the constraints are working correctly.

### 2.3.3 Workload

The workload is derived from the original 17 queries in the XBench DC/SD workload, translated to be in the form of relational DML (i.e., SQL/PSM). Six queries were omitted, since they had no conceptual counterpart in PSM, such as testing the access of relatively ordered XML elements. The workload was thus designed to be a microbenchmark, in that each PSM query highlights an individual construct of PSM, so that for example the performance of each construct can be studied in isolation, while retaining as much as possible the semantics of the original XBench query from which it was derived. Some queries, such as q2 were also changed to highlight a different feature, such as multiple **SET** statements in q2b and nested **FETCH** statements in q7c. See also q7b. Also we added three queries q17, q18, and q19, and one variant, q17b.

- Query q2 highlights the construct of **SET** with a **SELECT** row,
- Query 2b highlights multiple **SET** statements,
- Query q3 highlights a **RETURN** with a **SELECT** row,
- Query q5 highlights a function in the **SELECT** statement,
- Query q6 highlights the **CASE** statement,
- Query q7 highlights the **WHILE** statement,
- Query q7b highlights the **REPEAT** statement,
- Query q7c highlights the nested **FETCH** statements,
- Query q8 highlights a loop name with the **FOR** statement,
- Query q9 highlights a **CALL** within a procedure,
- Query q10 highlights an **IF** without a **CURSOR**
Figure 9: Query Q2 of the PSM benchmark.

- Query q11 highlights creation of a temporary table,
- Query q14 highlights a local cursor declaration with associated FETCH, OPEN, and CLOSE statements,
- Query q17 highlights the LEAVE statement,
- Query q17b highlights a non-nested FETCH statement,
- Query q18 highlights a function called in the FROM clause, and
- Query q19 highlights a SET statement.

The result is a set of 17 queries.

The PSM-DB2 workload utilizes a source-to-source translation of standard PSM into the syntactic variant supported by IBM DB2.

The queries for the PSM benchmark are manually derived by the second and third authors from the XQuery queries of the XBench benchmark. The authors began with the description of the XBench queries and manually implemented a semantically equivalent query in PSM.

Figure 9 shows query Q2 mapped from the original XBench query in Figure 5.

We have developed a tool (see Section 3.2.7) that compares the output of running XQuery on an XBench query and running PSM on a PSM query. If the results are textually equivalent (after some modest reformatting), then we say that the queries are equivalent. We compare each pair of queries in this way (except Q17b, since it is new to the PSM benchmark) to determine the equivalence of the query sets.

We have translated the queries to conform to the DB2 syntax requirements, as depicted in Appendix E. Figure 10 shows Q2 after the translation.

2.4 The τPSM and Related Benchmarks

Temporal SQL/PSM is a temporal extension to SQL/PSM that supports temporal relations and queries in conjunction with PSM functionality [19, 39]. The approach is similar to τXQuery in that it requires minor new syntax beyond that already in SQL/Temporal to define temporal PSM routines. Temporal SQL/PSM enables current queries (which are evaluated at the current time), sequenced (which are evaluated logically at each point in time independently), and non-sequenced queries (which are the most general of temporal semantics), as well as these three variants of PSM routines, similar to those described earlier for τXQuery.

The τPSM benchmark was created to test and validate Temporal SQL/PSM. There have been two different time slicing techniques proposed for implementing Temporal SQL/PSM: maximally-fragmented slicing and per-statement slicing [19, 39]. These slicing techniques are each a source-to-source translation of the sequenced queries and PSM routines of the workloads into conventional queries and PSM routines that explicitly manage the
Figure 10: Query Q2 of the PSM-DB2 benchmark.

timestamps in the data, to optimize the temporal queries in various ways. Thus we created the \( \tau \)PSM-MaxMapped and \( \tau \)PSM-PerStmtMapped benchmarks for these techniques. We also used the translation discussed above to create the schemas and workloads for the two DB2 variants, so that the resulting queries can be executed in DB2. (We note that the SQL:2011 standard [40] and IBM DB2 10 very recently now support temporal tables; our transformations are on conventional, that is, non-temporal, tables and queries.)

2.4.1 Data

The data of \( \tau \)PSM is the result of translating (shredding) the temporal XML data of \( \tau \)XBench into six (temporal) relations (see the previous section). We defined three datasets of different temporal volatility: DS1, DS2, and DS3. DS1 contains weekly changes, thus it contains 104 slices over two years, with each item having the same probability of being changed. Each time step experiences a total of 240 changes; thus there are 25K changes in all. DS2 contains the same number of slices but with rows in related tables associated with particular items changed more often (using a Gaussian distribution), to simulate hot-spot items. DS3 returns to the uniform model for the related tuples to be changed, but the changes are carried out on a daily basis, or 693 slices in all, each with 240 changes, or 25K changes in all (the number of slices was chosen to render the same number of total changes). These datasets come in different sizes: SMALL (e.g., DS1.SMALL is 12MB in six tables), MEDIUM (34MB), and LARGE (260MB).

The data for the two \( \tau \)PSM-*Mapped and two \( \tau \)PSM-*Mapped-DB2 variants is equivalent to that of \( \tau \)PSM.

2.4.2 Schema

Listed in Appendix C.2, the \( \tau \)PSM schemas are derived from the schemas of PSM. We extend each table with a simple temporal statement modifier (i.e., ALTER TABLE ADD VALIDTIME ...) to make the table time-varying. In addition, we extend each primary key to include the begin.time column and extend each assertion to be a sequenced assertion, thereby applying independently at each point in time.

The schema for \( \tau \)PSM-*Mapped is mapped from the temporal domain to the non-temporal domain (since DBMSes do not yet support the statement modifiers of \( \tau \)PSM), by removing ALTER TABLE ADD VALIDTIME keywords from the table creation commands.

The schema for \( \tau \)PSM-*Mapped-DB2 is mapped to meet the syntax requirements of DB2.

Validation. As we only added simple statement modifiers to the schemas, no validation is required. We have modified the \( \tau \)PSM schemas to satisfy DB2 syntax.

2.4.3 Workload

We define 85 queries for \( \tau \)PSM that correspond to the 17 queries in the PSM benchmark: 51 sequenced queries (a short period, medium period, and long period sequenced query for each of the original 17), 17 non-sequenced, and
CREATE ASSERTION number_authors VALIDTIME CHECK
(NOT EXISTS (SELECT IA.item_id, count(*) cnt
FROM item_author IA
HAVING count(*) > 4))

Figure 11: Cardinality constraint C5 of the \(\tau\)PSM benchmark schema.

---
-- Q2:  
-- Find the title of the item which has matching author  
-- first name (Ben).  
-- Feature:  
-- SET with SELECT single row
---

CREATE FUNCTION get_author_name(aid CHARACTER(10))
RETURNS CHARACTER(50)
READS SQL DATA
LANGUAGE SQL
BEGIN
DECLARE fname CHARACTER(50);
SET fname = (SELECT first_name
FROM author
WHERE author_id = aid);
RETURN fname;
END;

VALIDTIME SELECT i.title
FROM item i, item_author ia
WHERE i.id = ia.item_id
AND get_author_name(ia.author_id) = 'Ben';

Figure 12: Query Q2 of the \(\tau\)PSM benchmark.

17 current. The queries are derived directly from the PSM queries by adding simple temporal statement modifiers (i.e., VALIDTIME for sequenced queries, NONSEQUENCED VALIDTIME for non-sequenced queries, and no change for current queries) to each of the queries.

The workload for four \(\tau\)-PSM-*Mapped* variants are the result of an automated transformation, using the previously-mentioned Maximally-Fragmented and Per-Statement slicing techniques [19, 39], expressed in TXL, to create 170 queries in total.

We have translated the queries in to conform to the DB2 syntax requirements, as depicted in Appendix E. Figure 13 shows the sequenced Q2 after the translation.

Validation. Since we only add statement modifiers, no validation is required.

---
-- Q2:  
-- Find the title of the item which has matching author  
-- first name (Ben).  
-- Feature:  
-- SET with SELECT single row
---

CREATE FUNCTION db2_orig_get_author_name(aid CHARACTER(10))
RETURNS CHARACTER(50)
NO EXTERNAL ACTION
BEGIN
DECLARE fname CHARACTER(50);
SET fname = (SELECT first_name FROM author WHERE author_id = aid);
RETURN fname;
END;

VALIDTIME SELECT i.title
FROM item i, item_author ia
WHERE i.id = ia.item_id
AND get_author_name(ia.author_id) = 'Ben';

Figure 13: Query Q2 of the \(\tau\)PSM-DB2 benchmark.
2.5 The \( \tau \)XSchema Benchmark

\( \tau \)XSchema (Temporal XML Schema) is a language and set of tools that enable the construction and validation of temporal XML documents \([13, 15, 24, 38, 41]\). \( \tau \)XSchema extends XML Schema \([30, 31]\) with the ability to define temporal element types. A temporal element type denotes that an element can vary over time, describes how to associate temporal elements across slices (or snapshots, which are individual versions of a document), and provides some temporal constraints that broadly characterize how a temporal element can change over time.

In \( \tau \)XSchema, any element type can be rendered as temporal by including in the schema a simple logical annotation stating whether an element or attribute varies over valid time or transaction time, whether its lifetime is described as a continuous state or a single event, whether the item itself may appear at certain times and not at others, and whether its content changes. So a \( \tau \)XSchema document is just a conventional XML Schema document with a few annotations.

\( \tau \)XSchema provides a suite of tools to construct and validate temporal documents. A temporal document is validated by combining a conventional validating parser with a temporal constraint checker. To validate a temporal document, a temporal schema is first converted to a representational schema, which is a conventional XML Schema document that describes how the temporal information is represented. A conventional validating parser is then used to validate the temporal document against the representational schema. Then, the temporal constraint checker is used to validate any additional temporal constraints specified by the user in the temporal schema.

We have created the \( \tau \)XSchema benchmark to test the functionality and performance of \( \tau \)XSchema.

2.5.1 Schema

The schemas are the same as those in \( \tau \)XBench, augmented with seven additional manually-specified constraints. These constraints provide a simple but comprehensive set for testing the four types of XML constraints (referential integrity, data type, identity, and cardinality) with the various time modifiers (sequenced and non-sequenced). For example, one of the additional constraints specifies that “over a period of a year, an item may have up to six authors,” exercising a non-sequenced cardinality constraint.

For example, Figure 14 shows the non-sequenced constraint: “In any given year, an <item> may have up to six <authors>s”. The constraint is specified using the syntax of \( \tau \)XSchema \([13]\). The constraint uses the minOccurs and maxOccurs attributes of the <nonSeqCardinality> element to give value bounds for the cardinality constraint. Additionally, the dimension attribute specifies the time dimension (valid time or transaction time), the evaluationWindow attribute gives the time window over which the constraint should be checked, and the slideSize attribute gives the size of slide between each successive evaluation window. In this case, since evaluationWindow and slideSize are both set to “year”, the constraint is checked once for each calendar year. Should the slideSize have been set to, say, one day, then the constraint would be checked 364 times per calendar year: once for the fiscal year starting on January 1, once for the fiscal year starting January 2, etc.

Validation. As the conventional schema is duplicated, it does not require validation.

The temporal XML schema is the same as the temporal XML schema for \( \tau \)XBench, except that it has some additional sequenced and non-sequenced constraints. These constraints are defined in the \( \tau \)XSchema language \([13]\), and thus are validated automatically by the \( \tau \)XSchema tool suite.

Similar to the \( \tau \)XBench benchmark, the representational schema for \( \tau \)XSchema is automatically generated from the \( \tau \)XSchema tool suite, and thus does not require manual validation.
### Table 6: The implicit and explicit schema constraints in the XBench benchmark.

<table>
<thead>
<tr>
<th>Type</th>
<th>Defined</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1 Cardinality</td>
<td>Implicitly</td>
<td>The <code>&lt;related_items&gt;</code> subelement of <code>&lt;item&gt;</code> must occur exactly once.</td>
</tr>
<tr>
<td>C2 Cardinality</td>
<td>Implicitly</td>
<td>The <code>&lt;authors&gt;</code> subelement of <code>&lt;item&gt;</code> must occur exactly once.</td>
</tr>
<tr>
<td>C3 Cardinality</td>
<td>Implicitly</td>
<td>The <code>&lt;publisher&gt;</code> subelement of <code>&lt;item&gt;</code> must occur exactly once.</td>
</tr>
<tr>
<td>C4 Cardinality</td>
<td>Explicitly</td>
<td>There must be at least one <code>&lt;item&gt;</code> element.</td>
</tr>
<tr>
<td>C5 Cardinality</td>
<td>Explicitly</td>
<td>The <code>&lt;authors&gt;</code> element must have between 1 and 4 <code>&lt;author&gt;</code> subelements.</td>
</tr>
<tr>
<td>C6 Identity</td>
<td>Explicitly</td>
<td>The <code>id</code> attribute of <code>&lt;author&gt;</code> must be unique.</td>
</tr>
<tr>
<td>C7 Cardinality</td>
<td>Explicitly</td>
<td>The <code>&lt;FAX_number&gt;</code> subelement of <code>&lt;publisher&gt;</code> must occur 0 or 1 times.</td>
</tr>
<tr>
<td>C8 Cardinality</td>
<td>Explicitly</td>
<td>The <code>&lt;related_item&gt;</code> subelement of <code>&lt;related_items&gt;</code> must occur between 0 and 5 times.</td>
</tr>
<tr>
<td>C9 Cardinality</td>
<td>Explicitly</td>
<td>The <code>&lt;street_address&gt;</code> subelement of <code>&lt;street_information&gt;</code> must occur 1 or 2 times.</td>
</tr>
</tbody>
</table>

### Table 7: The additional schema constraints in the τXSchema benchmark.

<table>
<thead>
<tr>
<th>Time</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1 Sequenced</td>
<td>Cardinality</td>
<td>An <code>&lt;item&gt;</code> must have between 1 and 4 <code>&lt;author&gt;</code>s.</td>
</tr>
<tr>
<td>C2 Sequenced</td>
<td>Referential Integrity</td>
<td>A <code>&lt;related_item&gt;</code> should refer to a valid <code>&lt;item&gt;</code>.</td>
</tr>
<tr>
<td>C3 Sequenced</td>
<td>Identity</td>
<td>The <code>&lt;ISBN&gt;</code> of an <code>&lt;item&gt;</code> are unique.</td>
</tr>
<tr>
<td>C4 Sequenced</td>
<td>Data type</td>
<td>The <code>&lt;number_of_pages&gt;</code> of an <code>&lt;item&gt;</code> must be of type short.</td>
</tr>
<tr>
<td>C5 Non-sequenced</td>
<td>Cardinality</td>
<td>Over a period of a year, an <code>&lt;item&gt;</code> may have up to 6 <code>&lt;author&gt;</code>s.</td>
</tr>
<tr>
<td>C6 Non-sequenced</td>
<td>Referential Integrity</td>
<td>A <code>&lt;related_item&gt;</code> should refer to a valid <code>&lt;item&gt;</code> (possibly not currently in print).</td>
</tr>
<tr>
<td>C7 Non-sequenced</td>
<td>Identity</td>
<td>An <code>&lt;item&gt;</code> <code>item_id</code> is unique and may not ever be re-used.</td>
</tr>
</tbody>
</table>

2.5.2 Data

The data is the same as the data in τXBench.

2.5.3 Workload

Because τXSchema is a schema language and tool system, and because the main goal of testing such a system is to test its schema constraints, there is no workload in this benchmark. (In effect, the schema is the workload, and a microbenchmark at that.)
3 Supporting Tools

To facilitate the creation and validation of the individual benchmarks in τBench, we have developed a suite of tools that generate temporal data, validate the data in various ways, and validate the correctness of the queries. The tool suite, along with the rest of τBench, is available online [42] and is distributed according to the terms of the GNU Lesser General Public License [18].

The message of this section is that tools are some of the most important glue connecting the benchmarks together into a framework. Some of the tools map a component, e.g., data or workload, in one benchmark into another, creating that component from hole cloth. Other tools perform consistency checks within a benchmark. Tools that perform cross-validation are especially critical, in that they ensure consistency between say the data components of two related benchmarks in the framework. Such validation leverage the mapping and within-benchmark consistency checking tools, because all the tools much work correctly for the end result to validate. These tools in concert thus tied the benchmarks together into a coherent whole, raising the quality of all of the individual benchmarks within the framework.

3.1 τGenerator

τGenerator is a simulation that creates XML temporal data from the non-temporal XML data (Figure 15). To create the temporal data, τGenerator consists of a user-specified number of time steps along with a user-specified number of changes to the data within each time step. At each time step, a subset of the elements from the original document are changed, creating a new slice of the temporal document. The simulation continues until all of the required changes are made.

3.1.1 Simulation Description

We designate the <item>, <author>, <publisher>, and <related_items> elements of the DC/SD XBench XML data to be temporal elements, that is, their values can change over time. The overall goal of the simulation is to randomly change, at specified time intervals, the content of these four temporal elements. The simulation consists of the following four steps.

1. Setup. The simulation reads the user-created parameters file to initialize the simulation parameters. The simulation then uses DOM [46] to load the user-specified XBench document (in this case, catalog.xml) into memory. The simulation then creates an internal representation of the temporal items (all instances of the four temporal elements), which at this point consist of only a single version.

2. Initial snapshot selection. The simulation selects a user-specified percentage of <item> temporal elements (and all of their sub-elements) for the initial slice. The simulation selects the elements uniformly at random from catalog.xml. Elements that are selected are called active; those that are not selected for the initial slice are called inactive and put into a pool for later use.

The simulation sets the begin and end time of the first (and only, at this point) version of each selected temporal element to the user-specified begin date and the user-specified forever date, respectively.

3. Time step loop. The simulation increases the current time by the user-specified time step. The simulation then:

- Randomly selects a set of active elements to change, based on user-specified parameters.
Changes the selected elements, as well as adds new elements and deletes existing elements, based on user-specified parameters. New elements, and the new values for changed elements, come from the pool of inactive elements.

Outputs the current slice in XML format. A temporal document, i.e., all slices merged together, is also maintained.

Checks the stop criteria of the simulation. If the simulation has exceeded the user-specified number of necessary changes, the simulation exits the time step loop and continues to Step 4. Otherwise, it returns to Step 3.

4. Output. The simulation outputs the final temporal document.

3.1.2 Maintaining the Original Schema Constraints

During the above simulation process, it is important to maintain the original constraints defined and implied in the DC/SD XBench XML schema. This will ensure that the generated data validates to the sequenced semantics of the original schema. In addition, we add additional checks in the simulation to maintain the set of non-sequenced constraints in the $\tau$XSchema benchmark.

We need not consider the original data type constraints, because the simulation does not change the type of data at any point. Similarly, we need not consider identity constraints (which ensure, for example, that the item’s id attribute is globally unique) because the simulation does not generate any ids of its own—all items keep their originally-assigned ids. Thus, as long as the original data is correct, this portion of the generated data will also be correct.

However, we do need to consider the referential integrity constraint between item ids and related items. Since only a subset of items are active at any given time, there is no guarantee that related items will point to active item ids. Thus, the simulation must provide this guarantee explicitly. The simulation satisfies this by adding a check at the end of each time step. The check builds a list of all active item ids. Then, for any currently active related items that refer to an item id not in the list, the simulation replaces the value with an id in the list. Note that this ensures both sequenced and non-sequenced referential integrity constraints between item ids and related items.

We also need to consider the cardinality constraints placed on authors. Since the simulation has the ability to add new authors and delete existing authors at any point in time, the simulation must be careful not to violate the maxOccurs and minOccurs constraints in the original schema. This is true for both the sequenced (i.e., at any given time, an item must have between 1 and 4 authors) and non-sequenced (i.e., in any given year, an item can have up to 6 authors) variants of this constraint. The simulation achieves these by maintaining an author counter for each item and each time period. For the sequenced constraint, the counter is incremented and decremented as the simulation adds or removes authors. For the non-sequenced, the counter is only incremented when an author is inserted. Finally, when an item is selected by the simulation to insert or remove an author, the simulation first consults this counter; if the counter indicates that such an action will violate a constraint, the simulation chooses another item to change. The simulation will continue trying to choose a viable author for a predefined number of iterations, and then will halt to avoid infinite loops.

In this way, the resulting temporal data is made consistent for both sequenced and non-sequenced constraints.

3.1.3 Shredding XML Data into Relations

As mentioned above, we have created a tool to shred the XBench DC/SD data into six relations. If the data to be shredded is temporal, the resulting relations will have two additional columns representing the begin and end time for each row.

3.1.4 Input

Table 8 lists the names of the input parameters for $\tau$Generator along with a brief description of each. The user places the input parameters and their desired values into a parameters file, which is a conventional text file. The parameter file should have one parameter-value pair on each line. A parameter-value pair is a line containing the name of a parameter, some white space, and the desired value of the parameter. For example, to set the value of the TIME_STEP parameter to 15 days, then the following line should appear somewhere the parameters file.
Note that order is not important in the parameters file, and if a parameter is listed more than once, then the last instance will be used. If a parameter is not listed in the parameters file, then the parameter takes the default value specified in Table 8. Comment lines are also allowed, which are preceded with the `#` character:

```plaintext
# This is a comment line.

Blank lines and extra white space are allowed:

# This is line 1.

# This is line 3.
```

An example parameters file is listed in Appendix D.

### 3.1.5 Output

\( \tau \)Generator has the ability to output a temporal XML document and corresponding shredded relations into a user-specified output directory. If the the user-specified slice output parameters are set, then at each time step, \( \tau \)Generator will also output the current version of each of the six files.

The output files are summarized in Table 9.

### 3.1.6 Compilation and Usage

\( \tau \)Generator has been made publicly available on the TimeCenter website [44]. The Xerces [50] library is also required and comes bundled with the \( \tau \)Generator distribution.

\( \tau \)Generator ships with the following files:

- README.txt. A helper file that indicates how to compile and run \( \tau \)Generator.
- tGenerator.java. The single source code file.
- catalog.xml. A sample bookstore catalog generated by XBench (10MB).
- xerces-2.2.1.jar. The Xerces distribution.

Once downloaded, \( \tau \)Generator can be compiled with the following command.

```plaintext
javac tGenerator.java -cp ./xerces-2.2.1.jar
```

Or simply:

```
compile
```

This will create a Java class file named tGenerator.class in the current directory.

\( \tau \)Bench can be executed with the following command.

```plaintext
java -cp ./xerces-2.2.1.jar:. tGenerator parameters.txt
```

The format of the parameters.txt file is given in Section 3.1.4. For larger scenarios of XBench, the memory of the Java Virtual Machine (JVM) needs to be increased. The following command, for example, will provide the JVM with 28 gigabytes of memory.

```plaintext
java -Xmx28g -cp ./xerces-2.2.1.jar:. tGenerator parameters.txt
```

We have found that running \( \tau \)Generator with XBench’s small (10MB) and normal (100MB) class sizes requires less than 2GB of memory while the large (1GB) class requires 28GB of memory. We have not yet tested the huge (10GB) class.

A sample catalog.xml and parameters file is bundled with the \( \tau \)Generator distribution.
3.2 Validation Tools

We have developed several tools to help us validate our translations and simulations.

3.2.1 Validating the Output of $\tau$Generator

We have created a tool which validates the XML slices generated by $\tau$Generator against their XML schema using XMLLint [25]. In addition, the tool validates the generated temporal document against its temporal schema using $\tau$XMLLint [13]. These checks ensure that each generated slice is well-formed and consistent with the schema constraints, and that the generated temporal document is well-formed and consistent with the sequenced and non-sequenced semantics of the conventional schema.

3.2.2 Corrupting Data

Constraints should be tested on both valid and invalid data. We thus corrupt the data generated by $\tau$Generator as a testing phase to determine if the validators and associated schemas can detect the corruptions. We have developed a tool, called $\tau$Corruptor, that takes as input a temporal XML document and outputs another temporal XML document which is identical to the input document except for one victim element. The victim element is modified to directly contradict one of the constraints specified in the original schema.

Specifically, the input into $\tau$Corruptor is a temporal XML document generated by $\tau$Generator, as well as an integer parameter specifying which of the seven temporal constraints to invalidate.

1. Sequenced cardinality. Duplicates an entire author four times (but keeps each author id unique).
2. Sequenced referential integrity. Changes a related item’s item_id attribute to a random string.
4. Sequenced datatype. Changes the number of pages to a random string.
5. Non-sequenced identity. Copies the id of an item $i_1$ at time $t_1$ to the id of an item $i_2$ at time $t_2$, where $i_1 \neq i_2$ and $t_1 \neq t_2$.
6. Non-sequenced referential integrity. Changes a related item’s item_id to a random string.
7. Non-sequenced cardinality. Duplicates an entire $<\text{author RepItem}>$ six times (but keeps each author id unique).

$\tau$Corruptor will randomly select a victim node and apply the changes specified by the input. It will then output the modified temporal XML document.

$\tau$Corruptor can be compiled with the following command.

```java
javac tCorruptor.java -cp ./xerces-2.2.1.jar
```

Or simply:

```bash
compile
```

This will create a Java class file named tCorruptor.class in the current directory.

$\tau$Bench can be executed with the following command.

```java
java -cp ./xerces-2.2.1.jar: tCorruptor input.xml actionCode
```

3.2.3 Checking Primary Keys of Shredded Relations

We have created a tool that checks each shredded relation to ensure that each primary key is unique across the relation. It does so by building a hash table of primary keys in each relation. If a row is encountered whose key is already defined in the hash table, then the tool reports an error. Relations whose primary key are defined across multiple columns require multi-dimensional hash tables.
3.2.4 Checking Referential Integrity of Shredded Relations

We have created a tool that checks each shredded relation to ensure that each referential integrity constraint is valid across the relation. It does so by first reading the entire relation and creating a hash table of the referenced primary keys. A second pass ensures that each value in a foreign key column is defined in the hash table.

3.2.5 Comparing XML Slices to the Temporal XML Document

We have created a tool that ensures that the generated temporal XML document and the corresponding XML slices are consistent. The last tool makes use of UNSQUASH, which is part of the τXSchema tool suite [13]. UNSQUASH extracts each XML slice from a given temporal XML document. Our tool then compares each generated slice with each extracted slice using X-Diff [48] to ensure that they are equivalent. If all the slices are equivalent, then our tool reports that the temporal XML document and generated slices are equivalent.

3.2.6 Testing Queries

Query Executor is a tool providing a simple way to quickly execute multiple queries for a variety of DBMSes. After the DBMS and password is entered the GUI will display a list of active query directories and allow for additional directories to be added, or for old directories to be removed. Each query directory contains a parameters file (queryexecutor) that specifies a connection to the appropriate database. Queries within the selected directory appear in a separate pane and the tool allows a query to be run individually, all of the queries within the directory to be run, or a selected group of the queries to be executed. The results of each query are stored in files and Query Executor can run a diff on the last two executions. If one or more of the previous two runs was of more than a single query the diff will be performed only on the queries from both runs that have the same query number. Query Executor also allows for the results of any run or diff to be exported to a file. This tool has proven very convenient for developing the queries within each benchmark.

3.2.7 Comparing XQuery and PSM Queries

We have created a tool that performs a pair of queries and compares their output. First, it runs an XQuery query on the non-temporal XML data and saves the output. Then, it runs the corresponding PSM query on the shredded relational data and saves the output. Since the output formats are different, the tool reformats the PSM output to be of the same format as the XQuery output. If the two results are textually identical, then the queries achieved the same functionality.

This tool is part of our larger commutativity validation process. Commutativity validation compares, for example, the output of performing a temporal query on a temporal database with the output of performing a conventional query on a slice of the temporal database (Figure 16). Such a comparison simultaneously validates several tasks and tools: slicing the temporal database; executing temporal queries; slicing temporal relations; and executing conventional queries. Only if all of these steps are working perfectly will the results be the same, thus a strong level of validation is ensured. This cross-validation relies on snapshot equivalence and temporal upward compatibility [5], as well as the sequenced semantics [8, 37], which relate in very simple ways non-temporal and temporal data, queries, and constraints.
Figure 16: Commutativity validation.
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT_FILE</td>
<td>The string name of the input file (produced by XBench).</td>
<td>catalog.xml</td>
</tr>
<tr>
<td>OUTPUT_DIR</td>
<td>The string name of the directory in which to place the generated output files.</td>
<td>output</td>
</tr>
<tr>
<td>OUTPUT_XML</td>
<td>A flag (0 or 1) indicating if the simulation should output the temporal XML file when the simulation ends.</td>
<td>1</td>
</tr>
<tr>
<td>OUTPUT_RELATIONS</td>
<td>A flag (0 or 1) indicating if the simulation should output the shredded relations when the simulation ends.</td>
<td>0</td>
</tr>
<tr>
<td>OUTPUT_XML_SLICES</td>
<td>A flag (0 or 1) indicating if the simulation should output the XML slice at each time step.</td>
<td>0</td>
</tr>
<tr>
<td>OUTPUT_RELATION_SLICES</td>
<td>A flag (0 or 1) indicating if the simulation should output the shredded relations at each time step.</td>
<td>0</td>
</tr>
<tr>
<td>INITIAL_PERCENTAGE</td>
<td>The positive floating-point percentage of elements in INPUT_FILE to be placed in the initial slice.</td>
<td>10</td>
</tr>
<tr>
<td>START_TIME</td>
<td>The starting date of the simulation.</td>
<td>2010-01-01</td>
</tr>
<tr>
<td>FOREVER_TIME</td>
<td>The date of “forever”; a time not reachable.</td>
<td>2099-12-31</td>
</tr>
<tr>
<td>TIME_STEP</td>
<td>The number of days between time steps.</td>
<td>7</td>
</tr>
<tr>
<td>REQUIRED_CHANGES</td>
<td>The positive integer number of changes before the simulation terminates.</td>
<td>100</td>
</tr>
<tr>
<td>REQUIRED_SLICES</td>
<td>The positive integer number of slices before the simulation terminates. If non-zero, overrides REQUIRED_CHANGES.</td>
<td>0</td>
</tr>
<tr>
<td>ITEM_TIME_STEP_INSERTS</td>
<td>The non-negative integer number of &lt;item&gt; s that are inserted at each time step.</td>
<td>1</td>
</tr>
<tr>
<td>ITEM_TIME_STEP_DELETES</td>
<td>The non-negative integer number of &lt;item&gt; s that are deleted at each time step.</td>
<td>1</td>
</tr>
<tr>
<td>ITEM_TIME_STEP_UPDATES</td>
<td>The non-negative integer number of &lt;item&gt; s that are updated at each time step.</td>
<td>1</td>
</tr>
<tr>
<td>AUTHOR_TIME_STEP_INSERTS</td>
<td>The non-negative integer number of &lt;author&gt; s that are inserted at each time step.</td>
<td>1</td>
</tr>
<tr>
<td>AUTHORTIME_STEP_DELTES</td>
<td>The non-negative integer number of &lt;author&gt; s that are deleted at each time step.</td>
<td>1</td>
</tr>
<tr>
<td>AUTHOR_TIME_STEP_UPDATES</td>
<td>The non-negative integer number of &lt;author&gt; s that are updated at each time step.</td>
<td>1</td>
</tr>
<tr>
<td>PUBLISHER_TIME_STEP DELETES_INSERTS</td>
<td>The non-negative integer number of &lt;publisher&gt; s that are deleted/inserted at each time step.</td>
<td>1</td>
</tr>
<tr>
<td>RELATED_TIME_STEP DELETES_INSERTS</td>
<td>The non-negative integer number of &lt;related_item&gt; s that are deleted/inserted at each time step.</td>
<td>1</td>
</tr>
<tr>
<td>SELECTION_TYPE</td>
<td>The string name of the distribution for selecting elements to change; options are gaussian or uniform.</td>
<td>uniform</td>
</tr>
<tr>
<td>SELECTION_STDDEV</td>
<td>For SELECTION_TYPE of gaussian, the floating-point positive standard deviation of the distribution. (The mean is set by the simulation to half of the number of elements initially selected.)</td>
<td>10.0</td>
</tr>
<tr>
<td>NULL_VALUE</td>
<td>The string value to use for NULL columns when shredding the XML documents into CSV.</td>
<td>null</td>
</tr>
</tbody>
</table>

Table 8: The inputs into \( \tau \)Generator.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>output.final.xml</td>
<td>XML</td>
<td>Contains the final version (i.e., all slices) of the temporal XML document.</td>
</tr>
<tr>
<td>output.t.xml</td>
<td>XML</td>
<td>Contains the slice at time ( t ) of the temporal XML document.</td>
</tr>
<tr>
<td>*_.final.csv</td>
<td>CSV</td>
<td>Contains the final version of the shredded relations (i.e., each row is a tuple and each column is an attribute of that tuple).</td>
</tr>
<tr>
<td>*_.t.csv(^1)</td>
<td>CSV</td>
<td>Contains the slice at time ( t ) of the shredded relations.</td>
</tr>
</tbody>
</table>

\(^1\) = \{item, author, publisher, related_items, item_author, and item_publisher\}

Table 9: The output files of \( \tau \)Generator.
4 Conclusions and Future Research

In this report, we have introduced τBench, a benchmark framework for evaluating temporal schema and query languages on XML and relational data [43]. Although τBench spans many technologies (from XML Schema to Temporal SQL/PSM) and is used by several distinct projects, the benchmarks are easy to understand and creating new, related benchmarks is straightforward and reliable; this is one of the powers of benchmark frameworks. An early implementation of τBench is available on our project website [42]. Our experience with this framework is that it was easier to develop than a family of only loosely related benchmarks, due to extensive sharing of components and tools between constituent benchmarks.

The τBench benchmark suite currently consists of ten benchmarks, each consisting of schemas, datasets, and workloads. The τBench tool suite, which was built to aid the construction and validation of the benchmark framework, consists of a temporal data generation tool (τGenerator), a data corruption tool for testing purposes (τCorruptor), and a set of scripts that validate the generated data in various ways.

The resulting τBench suite is an ecosystem of related benchmarks. In this ecosystem, components of existing benchmarks flow into the construction of new benchmarks, all the while being validated by several measures. As each new benchmark is constructed and validated, the foundational benchmarks gain strength because their components are again being validated, this time in a new light. And while all the benchmarks are tightly interconnected (in that they are derived from each other and share data, schemas, and workloads), their use in research and industry is completely independent (in that any single benchmark can be used without any of the others).

Future work for τBench includes defining additional benchmarks for new and existing database technologies, and extending the datasets to include other categories of XBench datasets. We also want to use suitable language-to-language translators when they become available to create executable workloads for τXBench-Galax and τPSM-DB2, to enable validation of these workloads.

Future work for τGenerator includes:

- Addressing the inefficiency of DOM. The execution time of some routines in τGenerator could be vastly improved with a better use of DOM iterators. Also, the memory requirements of τGenerator should be addressed by reusing data structures where possible.
- Porting to τDOM [26]. τDOM contains temporal support for manipulating DOM objects in memory. Much of the internal complexities of τGenerator could be reduced with the use of τDOM.
- Considering other output categories of XBench. Currently, τGenerator is built on the DC/SD category of XBench output, but the other three may also be useful. This would involve generalizing the internals of τGenerator.

Finally, we encourage the XBench, SPEC and TPC organizations to consider refining their respective benchmark families into more structured and more convenient benchmark frameworks.
References


A Temporal Data Model

For the purpose of modeling temporal data in τBench, we adopt the temporal data model employed by τXSchema [13]. In τXSchema, the history of an XML document is composed of individual slices (or snapshots), which are members of a sequenced set of non-overlapping versions of a temporal (or time-varying) XML document. Each slice is associated with a time period during which it is valid. The time period of a slice does not overlap with the time period of any other slice.

Individual elements within the XML document that change over time are called temporal elements. Temporal elements are represented as items, which contain a series of versions with associated timestamps. A new version of an item is created whenever any part of the item changes, including its children (unless the changed child is itself a temporal element). Thus, to extract a slice of a temporal XML document at time t, we just select the version associated with time t of all temporal elements.

In this report, we chose to create four temporal elements out of the following four original catalog.xml elements: <item>, <author>, <publisher>, and <related_items>. These elements give us flexibility for defining the workloads of our benchmarks in later sections. An outline of the resulting temporal representation of XBench’s catalog.xml document is shown in Figure 17.

```xml
<catalog>
  <item_RepItem>
    <item_Version begin="2006-01-01" end="2009-07-05">
      <item>
        ...
      </item>
    </item_Version>
  </item_RepItem>
  ...
</catalog>
```

Figure 17: A simplified slice of the temporal catalog.xml document.

In τXSchema, the time period of all temporal subelements must be contained in the time period of their parent temporal elements. Using the example above, the time period of an <author_Version> must be contained in its associated <item_Version>.

Adopting the temporal data model used by τXSchema not only brings the advantages of the data model itself, but it also allows us to use the τXSchema tool suite to construct and validate our benchmarks.
B XML Schemas

B.1 Conventional XML Schema (DCSD.xsd)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="FAX_number" type="xs:string"/>
  <xs:element name="ISBN" type="xs:string"/>
  <xs:element name="attributes" type="xs:string">
    <xs:complexType>
      <xs:all>
        <xs:element ref="ISBN"/>
        <xs:element ref="number_of_pages"/>
        <xs:element ref="type_of_book"/>
        <xs:element ref="size_of_book"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="author" type="xs:string">
    <xs:complexType>
      <xs:all>
        <xs:element ref="first_name"/>
        <xs:element ref="middle_name"/>
        <xs:element ref="last_name"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="date_of_birth" type="xs:string"/>
  <xs:element name="biography" type="xs:string"/>
  <xs:element name="contact_information" type="xs:string">
    <xs:complexType>
      <xs:all>
        <xs:element ref="mailing_address" type="xs:string">
          <xs:complexType>
            <xs:all>
              <xs:element ref="street_information"/>
              <xs:element ref="name_of_city"/>
              <xs:element ref="name_of_state"/>
              <xs:element ref="zip_code" type="xs:string"/>
              <xs:element ref="name_of_country" type="xs:string"/>
            </xs:all>
          </xs:complexType>
        </xs:element>
        <xs:element ref="phone_number" type="xs:string"/>
        <xs:element ref="email_address" type="xs:string"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="author_id" type="xs:string"/>
  <xs:element name="authors" type="xs:string">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="author" maxOccurs="4"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="biography" type="xs:string"/>
  <xs:element name="catalog" type="xs:string">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="item" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="ISBNUnique" type="xs:string">
    <xs:complexType>
      <xs:selector xpath=".//item/attributes"/>
      <xs:field xpath="ISBN"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="ISBNUnique" type="xs:string">
    <xs:complexType>
      <xs:selector xpath=".//item/attributes"/>
      <xs:field xpath="ISBN"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
<xs:complexType>
  <xs:all>
    <xs:element ref="thumbnail"/>
    <xs:element ref="image"/>
  </xs:all>
</xs:complexType>

<xs:element name="middle_name" type="xs:string"/>
<xs:element name="name_of_city" type="xs:string"/>
<xs:element name="name_of_country" type="xs:string"/>
<xs:element name="name_of_state" type="xs:string"/>

<!-- Sequenced Datatype: -->
<!-- The number_of_pages must be of type short. -->
<xs:element name="number_of_pages" type="xs:short"/>
<xs:element name="phone_number" type="xs:string"/>

<xs:complexType>
  <xs:all>
    <xs:element ref="suggested_retail_price"/>
    <xs:element ref="cost"/>
    <xs:element ref="when_is_available"/>
    <xs:element ref="quantity_in_stock"/>
  </xs:all>
</xs:complexType>

<xs:element name="publisher">
  <xs:complexType>
    <xs:all>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="contact_information">
        <xs:complexType>
          <xs:all>
            <xs:element name="mailing_address">
              <xs:complexType>
                <xs:all>
                  <xs:element ref="street_information"/>
                  <xs:element ref="name_of_city"/>
                  <xs:element ref="name_of_state"/>
                  <xs:element ref="zip_code"/>
                  <xs:element ref="country"/>
                </xs:all>
              </xs:complexType>
            </xs:element>
            <xs:element ref="FAX_number" minOccurs="0"/>
            <xs:element ref="phone_number"/>
            <xs:element ref="web_site"/>
          </xs:all>
        </xs:complexType>
      </xs:element>
    </xs:all>
    <xs:attribute name="publisher_id" type="xs:string"/>
  </xs:complexType>
</xs:element>

<xs:element name="quantity_in_stock" type="xs:byte"/>

<xs:complexType>
  <xs:all>
    <xs:element ref="item_id"/>
  </xs:all>
</xs:complexType>

<xs:element name="related_items">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="related_item" minOccurs="0" maxOccurs="5"/>
    </xs:sequence>
    <xs:attribute name="related_items_id" type="xs:string"/>
  </xs:complexType>
</xs:element>

<xs:element name="size_of_book">
  <xs:complexType>
    <xs:all>
      <xs:element ref="length"/>
      <xs:element ref="width"/>
      <xs:element ref="height"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="street_address" type="xs:string"/>
<xs:element name="street_information"/>
<xs:sequence>
  <xs:element ref="street_address" maxOccurs="2"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="subject" type="xs:string"/>
<xs:element name="suggested_retail_price">
  <xs:simpleContent>
    <xs:extension base="xs:decimal">
      <xs:attribute name="currency" type="xs:string" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="thumbnail">
  <xs:complexType>
    <xs:all>
      <xs:element ref="data"/>
    </xs:all>
  </xs:complexType>
</xs:element>
<xs:element name="title" type="xs:string"/>
<xs:element name="type_of_book" type="xs:string"/>
<xs:element name="when_is_available" type="xs:date"/>
<xs:element name="width">
  <xs:simpleContent>
    <xs:extension base="xs:decimal">
      <xs:attribute name="unit" type="xs:string" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="zip_code" type="xs:string"/>
B.2 Temporal XML Schema

```xml
<?xml version="1.0" encoding="utf-8"?>
<temporalSchema
    xmlns="http://www.cs.arizona.edu/tau/TSSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.cs.arizona.edu/tau/TSSchema/etc/TSSchema.xsd">

<conventionalSchema>
    <include schemaLocation="DCSD.xsd" />
</conventionalSchema>

<annotationSet>
    <logical>
        <item target="catalog/item" />
        <item target="catalog/item/authors/author" />
        <item target="catalog/item/publisher" />
        <item target="catalog/item/related_items" />
    </logical>

    <physical>
        <default>
            <format plugin="XMLSchema" granularity="days" />
        </default>
        <!-- default timestamps are at logical items -->
    </physical>
</annotationSet>
</temporalSchema>
```
B.3 Conventional XML Schema with Added Constraints

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="FAX_number" type="xs:string"/>
  <xs:element name="ISBN" type="xs:string"/>
  <xs:element name="attributes">
    <xs:complexType>
      <xs:all>
        <xs:element ref="ISBN"/>
        <xs:element ref="number_of_pages"/>
        <xs:element ref="type_of_book"/>
        <xs:element ref="size_of_book"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="author">
    <xs:complexType>
      <xs:all>
        <xs:element name="name">
          <xs:complexType>
            <xs:all>
              <xs:element ref="first_name"/>
              <xs:element ref="middle_name"/>
              <xs:element ref="last_name"/>
            </xs:all>
          </xs:complexType>
        </xs:element>
        <xs:element ref="date_of_birth"/>
        <xs:element ref="biography"/>
        <xs:element name="contact_information">
          <xs:complexType>
            <xs:all>
              <xs:element name="mailing_address">
                <xs:complexType>
                  <xs:all>
                    <xs:element ref="street_information"/>
                    <xs:element ref="name_of_city"/>
                    <xs:element ref="name_of_state"/>
                    <xs:element ref="zip_code"/>
                    <xs:element name="name_of_country" type="xs:string"/>
                  </xs:all>
                </xs:complexType>
              </xs:element>
              <xs:element ref="phone_number"/>
              <xs:element ref="email_address"/>
            </xs:all>
          </xs:complexType>
        </xs:element>
        <xs:attribute name="author_id" type="xs:string"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="authors">
    <xs:complexType>
      <xs:sequence>
        <!-- Sequenced Cardinality: -->
        <!-- An item must have between 1 and 4 authors. -->
        <xs:element ref="author" maxOccurs="4"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="biography" type="xs:string"/>
  <xs:element name="catalog">
    <xs:complexType>
      <xs:sequence>
        <!-- Sequenced Identify Constraint: -->
        <!-- Item ISBNs are unique. -->
        <xs:unique name="ISBNUnique">
          <xs:selector xpath="//item/attributes"/>
          <xs:field xpath="ISBN"/>
        </xs:unique>
        <!-- Sequenced Referential Integrity: -->
        <!-- A related item should refer to a valid item -->
        <xs:key name="itemID">
          <xs:selector xpath="//item"/>
          <xs:field xpath="@id"/>
        </xs:key>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
<xs:element name="cost">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:decimal">
        <xs:attribute name="currency" type="xs:string" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="country">
  <xs:complexType>
    <xs:all>
      <xs:element name="name" type="xs:string"/>
      <xs:element ref="exchange_rate"/>
      <xs:element ref="currency"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="currency" type="xs:string"/>

<xs:element name="data" type="xs:string"/>

<xs:element name="date_of_birth" type="xs:date"/>

<xs:element name="date_of_release" type="xs:date"/>

<xs:element name="description" type="xs:string"/>

<xs:element name="email_address" type="xs:string"/>

<xs:element name="web_site" type="xs:string"/>

<xs:element name="exchange_rate" type="xs:decimal"/>

<xs:element name="first_name" type="xs:string"/>

<xs:element name="height">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:decimal">
        <xs:attribute name="unit" type="xs:string" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="image">
  <xs:complexType>
    <xs:all>
      <xs:element ref="data"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="item">
  <xs:complexType>
    <xs:all>
      <xs:element ref="title"/>
      <xs:element ref="authors"/>
      <xs:element ref="date_of_release"/>
      <xs:element ref="publisher"/>
      <xs:element ref="subject"/>
      <xs:element ref="description"/>
      <xs:element ref="related_items"/>
      <xs:element ref="media"/>
      <xs:element ref="pricing"/>
      <xs:element ref="attributes"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="item_id" type="xs:IDREF"/>

<xs:element name="last_name" type="xs:string"/>

<xs:element name="length">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:decimal">
        <xs:attribute name="unit" type="xs:string" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="media">
  <xs:complexType>
    <xs:all>
      <xs:element ref="thumbnail"/>
    </xs:all>
  </xs:complexType>
</xs:element>
<xs:element name="middle_name" type="xs:string"/>
<xs:element name="name_of_city" type="xs:string"/>
<xs:element name="name_of_country" type="xs:string"/>
<xs:element name="name_of_state" type="xs:string"/>

<!-- Sequenced Datatype: -->
<!-- The number_of_pages must be of type short. -->
<xs:element name="number_of_pages" type="xs:short"/>
<xs:element name="phone_number" type="xs:string"/>

<xs:complexType>
  <xs:all>
    <xs:element ref="suggested_retail_price"/>
    <xs:element ref="cost"/>
    <xs:element ref="when_is_available"/>
    <xs:element ref="quantity_in_stock"/>
  </xs:all>
</xs:complexType>
</xs:element>

<xs:element name="publisher">
  <xs:complexType>
    <xs:all>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="contact_information">
        <xs:complexType>
          <xs:all>
            <xs:element name="mailing_address">
              <xs:complexType>
                <xs:all>
                  <xs:element ref="street_information"/>
                  <xs:element ref="name_of_city"/>
                  <xs:element ref="name_of_state"/>
                  <xs:element ref="zip_code"/>
                  <xs:element ref="country"/>
                </xs:all>
              </xs:complexType>
            </xs:element>
            <xs:element ref="FAX_number" minOccurs="0"/>
            <xs:element ref="phone_number"/>
            <xs:element ref="web_site"/>
          </xs:all>
        </xs:complexType>
      </xs:element>
      <xs:attribute name="publisher_id" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="quantity_in_stock" type="xs:byte"/>
<xs:element name="related_item">
  <xs:complexType>
    <xs:all>
      <xs:element ref="item_id"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="related_items">
  <xs:complexType>
    <xs:sequence minOccurs="0" maxOccurs="5">
      <xs:element ref="related_item"/>
    </xs:sequence>
    <xs:attribute name="related_items_id" type="xs:string"/>
  </xs:complexType>
</xs:element>

<xs:element name="size_of_book">
  <xs:complexType>
    <xs:all>
      <xs:element ref="length"/>
      <xs:element ref="width"/>
      <xs:element ref="height"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="street_address" type="xs:string"/>
<xs:element name="street_information">
  <xs:complexType>
    <xs:sequence maxOccurs="2">
      <xs:element ref="street_address"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:complexType>
  <xs:element name="subject" type="xs:string"/>
  <xs:element name="suggested_retail_price">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:decimal">
          <xs:attribute name="currency" type="xs:string" use="required"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="thumbnail">
    <xs:complexType>
      <xs:all>
        <xs:element ref="data"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="title" type="xs:string"/>
  <xs:element name="type_of_book" type="xs:string"/>
  <xs:element name="when_is_available" type="xs:date"/>
  <xs:element name="width">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:decimal">
          <xs:attribute name="unit" type="xs:string" use="required"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="zip_code" type="xs:string"/>
</xs:complexType>
B.4 Representational XML Schema

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"

elementFormDefault="qualified"
attributeFormDefault="unqualified">

<xs:element name="catalog_RepItem">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="catalog_Version" minOccurs="1" maxOccurs="unbounded"/>
      <xs:attribute name="isItem" type="xs:string" use="required"/>
      <xs:attribute name="originalElement" type="xs:string" use="required"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="catalog_Version">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="catalog"/>
    </xs:sequence>
    <xs:attribute name="begin" type="xs:date" use="required"/>
    <xs:attribute name="end" type="xs:date" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="item_RepItem">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="item_Version" minOccurs="1" maxOccurs="unbounded"/>
      <xs:attribute name="isItem" type="xs:string" use="required"/>
      <xs:attribute name="originalElement" type="xs:string" use="required"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="item_Version">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="item"/>
    </xs:sequence>
    <xs:attribute name="begin" type="xs:date" use="required"/>
    <xs:attribute name="end" type="xs:date" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="author_RepItem">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="author_Version" minOccurs="1" maxOccurs="unbounded"/>
      <xs:attribute name="isItem" type="xs:string" use="required"/>
      <xs:attribute name="originalElement" type="xs:string" use="required"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="author_Version">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="author"/>
    </xs:sequence>
    <xs:attribute name="begin" type="xs:date" use="required"/>
    <xs:attribute name="end" type="xs:date" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="publisher_RepItem">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="publisher_Version" minOccurs="1" maxOccurs="unbounded"/>
      <xs:attribute name="isItem" type="xs:string" use="required"/>
      <xs:attribute name="originalElement" type="xs:string" use="required"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="publisher_Version">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="publisher"/>
    </xs:sequence>
    <xs:attribute name="begin" type="xs:date" use="required"/>
    <xs:attribute name="end" type="xs:date" use="required"/>
  </xs:complexType>
</xs:element>
```

45
</xs:complexType>
</xs:element>

<xs:element name="related_items_RepItem">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="related_items_Version" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="isItem" type="xs:string" use="required"/>
    <xs:attribute name="originalElement" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="related_items_Version">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="related_items"/>
    </xs:sequence>
    <xs:attribute name="begin" type="xs:date" use="required"/>
    <xs:attribute name="end" type="xs:date" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="FAX_number" type="xs:string"/>
<xs:element name="ISBN" type="xs:string"/>
<xs:element name="attributes">
  <xs:complexType>
    <xs:all>
      <xs:element ref="ISBN"/>
      <xs:element ref="number_of_pages"/>
      <xs:element ref="type_of_book"/>
      <xs:element ref="size_of_book"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="author">
  <xs:complexType>
    <xs:all>
      <xs:element name="name">
        <xs:complexType>
          <xs:all>
            <xs:element ref="first_name"/>
            <xs:element ref="middle_name"/>
            <xs:element ref="last_name"/>
          </xs:all>
        </xs:complexType>
      </xs:element>
      <xs:element ref="date_of_birth"/> 
      <xs:element ref="biography"/>
      <xs:complexType name="contact_information">
        <xs:all>
          <xs:element name="mailing_address">
            <xs:complexType>
              <xs:all>
                <xs:element ref="street_information"/>
                <xs:element ref="name_of_city"/>
                <xs:element ref="name_of_state"/>
                <xs:element ref="zip_code"/>
                <xs:element name="name_of_country" type="xs:string"/>
              </xs:all>
            </xs:complexType>
          </xs:element>
          <xs:element ref="phone_number"/> 
          <xs:element ref="email_address"/>
        </xs:all>
      </xs:complexType>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="authors">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="author_RepItem" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="biography" type="xs:string"/>
<xs:element name="temporalRoot"/>
<xs:complexType>
  <xs:sequence>
    <xs:element ref="temporalSchemaSet"/>
    <xs:element ref="catalog_RepItem"/>
  </xs:sequence>
  <xs:attribute name="begin" type="xs:string"/>
  <xs:attribute name="end" type="xs:string"/>
</xs:complexType>

<xs:element name="temporalSchemaSet">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="temporalSchema" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="temporalSchema">
  <xs:complexType>
    <xs:sequence>
      <xs:attribute name="schemaLocation" type="xs:string" use="required"/>
      <xs:attribute name="begin" type="xs:string"/>
      <xs:attribute name="end" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="catalog">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="item_RepItem" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="item">
  <xs:complexType>
    <xs:choice minOccurs="10" maxOccurs="unbounded">
      <xs:element ref="cost"/>
      <xs:element ref="catalog"/>
      <xs:element ref="country"/>
      <xs:element ref="currency"/>
      <xs:element ref="image"/>
      <xs:element ref="item"/>
      <xs:element ref="item"/>
      <xs:element ref="item"/>
      <xs:element ref="item"/>
      <xs:element ref="item"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name="cost">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:decimal">
        <xs:attribute name="currency" type="xs:string" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="catalog">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="item_RepItem" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="currency">
  <xs:complexType>
    <xs:all>
      <xs:element ref="currency"/>
      <xs:element ref="currency"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="currency">
  <xs:complexType>
    <xs:all>
      <xs:element ref="currency"/>
      <xs:element ref="currency"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="country">
  <xs:complexType>
    <xs:all>
      <xs:element ref="exange_rate"/>
      <xs:element ref="currency"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="exchange_rate">
  <xs:complexType>
    <xs:all>
      <xs:element ref="exchange_rate"/>
      <xs:element ref="exchange_rate"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="currency">
  <xs:complexType>
    <xs:all>
      <xs:element ref="currency"/>
      <xs:element ref="currency"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="exchange_rate">
  <xs:complexType>
    <xs:all>
      <xs:element ref="exchange_rate"/>
      <xs:element ref="exchange_rate"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="image">
  <xs:complexType>
    <xs:all>
      <xs:element ref="data"/>
      <xs:element ref="data"/>
      <xs:element ref="data"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="item">
  <xs:complexType>
    <xs:all>
      <xs:element ref="item"/>
      <xs:element ref="item"/>
      <xs:element ref="item"/>
    </xs:all>
  </xs:complexType>
</xs:element>
<xs:element ref="item_id"/>
</xs:all>
</xs:complexType>
</xs:element>
<xs:element name="related_items">
<xs:complexType>
<xs:sequence>
<xs:element ref="related_item" minOccurs="0" maxOccurs="5"/>
</xs:sequence>
<xs:attribute name="related_items_id" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="size_of_book">
<xs:complexType>
<xs:all>
<xs:element ref="length"/>
<xs:element ref="width"/>
<xs:element ref="height"/>
</xs:all>
</xs:complexType>
</xs:element>
<xs:element name="street_address" type="xs:string"/>
</xs:element>
<xs:element name="street_information">
<xs:complexType>
<xs:sequence>
<xs:element ref="street_address" maxOccurs="2"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="subject" type="xs:string"/>
</xs:element>
<xs:element name="suggested_retail_price">
<xs:complexType>
<xs:simpleContent>
<xs:extension base="xs:decimal">
<xs:attribute name="currency" type="xs:string" use="required"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="thumbnail">
<xs:complexType>
<xs:all>
<xs:element ref="data"/>
</xs:all>
</xs:complexType>
</xs:element>
<xs:element name="title" type="xs:string"/>
</xs:element>
<xs:element name="type_of_book" type="xs:string"/>
</xs:element>
<xs:element name="when_is_available" type="xs:date"/>
</xs:element>
<xs:element name="width">
<xs:complexType>
<xs:simpleContent>
<xs:extension base="xs:decimal">
<xs:attribute name="unit" type="xs:string" use="required"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="zip_code" type="xs:string"/>
</xs:element>
</xs:schema>
B.5 Temporal XML Schema with Added Constraints

```xml
<?xml version="1.0" encoding="utf-8"?>
<temporalSchema
    xmlns="http://www.cs.arizona.edu/tau/TSSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.cs.arizona.edu/tau/TSSchema/etc/TSSchema.xsd"
>
    <conventionalSchema>
        <include schemaLocation="DCSD.xsd" />
    </conventionalSchema>

    <annotationSet>
        <logical>
            <!-- Non-sequenced Identify Constraint: -->
            <!-- Item IDs are unique for books and may not ever be re-used. -->
            <item target="catalog">
                <nonSeqKey
                    name="bookIDKey" dimension="validTime" evaluationWindow="lifetime">
                    <selector xpath="item" />
                    <field xpath="@id" />
                </nonSeqKey>
            </item>

            <!-- Non-sequenced Referential Integrity: -->
            <!-- A related item should refer to a valid item
                (may not currently be an item in print). -->
            <item target="catalog/item">
                <nonSeqKeyref
                    name="relatedItemRI" refer="bookIDKey" dimension="validTime">
                    <selector xpath="." />
                    <field xpath="related_items/related_item/item_id" />
                </nonSeqKeyref>
            </item>

            <!-- Non-sequenced Cardinality: -->
            <!-- In any given year, an item may have up to 6 authors. -->
            <item target="catalog/item">
                <nonSeqCardinality
                    name="bookAuthorsNSeq" minOccurs="1" maxOccurs="6" dimension="validTime"
                    evaluationWindow="year" slideSize="year">
                    <selector xpath="." />
                    <field xpath="authors/author" />
                </nonSeqCardinality>
            </item>

            <item target="catalog/item/authors/author" />
            <item target="catalog/item/publisher" />
            <item target="catalog/item/related_items" />
        </logical>

        <physical>
            <default>
                <format plugin="XMLSchema" granularity="days" />
            </default>

            <!-- default timestamps are at logical items -->
        </physical>
    </annotationSet>
</temporalSchema>
```
C Relational Schemas

C.1 Non-temporal Relational Schemas

```sql
CREATE TABLE item (
    item_id CHARACTER(10) NOT NULL,
    ISBN VARCHAR(20),
    title CHARACTER(100),
    subject VARCHAR(200),
    number_of_pages INTEGER,
    type_of_book CHARACTER(50),
    length FLOAT,
    width FLOAT,
    height FLOAT,
    suggested_retail_price DECIMAL(10,2),
    cost DECIMAL(10,2),
    when_is_available DATE,
    quantity_in_stock INTEGER,
    date_of_release DATE,
    description VARCHAR(500))

ALTER TABLE item ADD PRIMARY KEY (item_id);
```

Figure 18: The `item` table creation and constraint definitions in SQL.

```sql
CREATE TABLE author(
    author_id CHARACTER(10) NOT NULL,
    first_name CHARACTER(50),
    middle_name CHARACTER(50),
    last_name CHARACTER(50),
    date_of_birth DATE,
    biography VARCHAR(500),
    street_address VARCHAR(100),
    name_of_city VARCHAR(50),
    name_of_state VARCHAR(50),
    zip_code CHARACTER(8),
    name_of_country VARCHAR(50),
    phone_number VARCHAR(50),
    email_address VARCHAR(100))

ALTER TABLE item_author ADD PRIMARY KEY (author_id);
```

Figure 19: The `author` table creation and constraint definitions in SQL.

```sql
CREATE TABLE item_author (
    item_id CHARACTER(10) NOT NULL,
    author_id CHARACTER(10) NOT NULL)

ALTER TABLE item_author ADD PRIMARY KEY (item_id, author_id);
ALTER TABLE item_author ADD FOREIGN KEY (item_id) REFERENCES item(item_id);
ALTER TABLE item_author ADD FOREIGN KEY (author_id) REFERENCES author(author_id);
```

Figure 20: The `item_author` table creation and constraint definitions in SQL.
CREATE TABLE item_publisher (  
item_id CHARACTER(10) NOT NULL,  
publisher_id CHARACTER(10) NOT NULL)

ALTER TABLE item_publisher ADD PRIMARY KEY (item_id, publisher_id);  
ALTER TABLE item_publisher ADD FOREIGN KEY (item_id) REFERENCES item(item_id));  
ALTER TABLE item_publisher ADD FOREIGN KEY (publisher_id) REFERENCES publisher(publisher_id));

Figure 21: The item_publisher table creation and constraint definitions in SQL.

CREATE TABLE publisher (  
publisher_id CHARACTER(10) NOT NULL,  
name VARCHAR(100),  
street_address VARCHAR(100),  
name_of_city VARCHAR(50),  
name_of_state VARCHAR(50),  
zip_code CHARACTER(8),  
exchange_rate FLOAT,  
currency CHARACTER(50),  
phone_number VARCHAR(50),  
web_site VARCHAR(40),  
FAX_number VARCHAR(20) NOT NULL)

ALTER TABLE publisher ADD PRIMARY KEY (publisher_id);

Figure 22: The publisher table creation and constraint definitions in SQL.

CREATE TABLE related_item (  
item_id CHARACTER(10) NOT NULL,  
related_id CHARACTER(10) NOT NULL)

ALTER TABLE related_item ADD PRIMARY KEY (item_id, related_id);  
ALTER TABLE related_item ADD FOREIGN KEY (item_id) REFERENCES item(item_id));  
ALTER TABLE related_item ADD FOREIGN KEY (related_id) REFERENCES item(item_id));

Figure 23: The related_items table creation and constraint definitions in SQL.
C.2 Temporal Relational Schemas

To accommodate temporal relational data, we must add valid time columns to each relation and alter the primary keys so that they contain the `begin_time` column. Namely, the following changes must be made to the relational schemas presented in Appendix C.1.

```sql
ALTER TABLE item ADD VALIDTIME (DATE);
ALTER TABLE author ADD VALIDTIME (DATE);
ALTER TABLE publisher ADD VALIDTIME (DATE);
ALTER TABLE related_item ADD VALIDTIME (DATE);
ALTER TABLE item_author ADD VALIDTIME (DATE);
ALTER TABLE item_publisher ADD VALIDTIME (DATE);
ALTER TABLE item ADD PRIMARY KEY (item_id, begin_time)
ALTER TABLE author ADD PRIMARY KEY (author_id, begin_time)
ALTER TABLE item_author ADD PRIMARY KEY (item_id, author_id, begin_time)
ALTER TABLE publisher ADD PRIMARY KEY (publisher_id, begin_time)
ALTER TABLE item_publisher ADD PRIMARY KEY (item_id, publisher_id, begin_time)
ALTER TABLE related_item ADD PRIMARY KEY (item_id, related_id, begin_time)
```

Figure 24: The `item` table creation and constrain definitions in SQL.
D Example Parameters File for \( \tau \)Generator

INITIAL_PERCENTAGE 10
START_TIME 2010-01-01
FOREVER_TIME 2099-12-31
TIME_STEP 7
REQUIRED_CHANGES 36400
ITEM_PER_TIME_STEP_INSERTS 58
ITEM_PER_TIME_STEP_UPDATES 58
AUTHOR_PER_TIME_STEP_INSERTS 58
AUTHOR_PER_TIME_STEP_UPDATES 58
PUBLISHER_PER_TIME_STEP_INSERTS 58
PUBLISHER_PER_TIME_STEP_UPDATES 58
RELATED_PER_TIME_STEP_INSERTS 59
RELATED_PER_TIME_STEP_UPDATES 59
CHANGE_TYPE gaussian
CHANGE_STDDEV 10
OUTPUT_DIR output
INPUT_FILE catalog.xml
NULL_VALUE ""
E  DB2 Differences

DB2 [23] differs from from the PSM standard in the following ways.

- **Temporary tables.** Instead of creating a temporary table and populating it from an existing table, DB2 requires two steps:

```
DECLARE GLOBAL TEMPORARY TABLE table_name
   (SELECT * FROM original_table) DEFINITION ONLY
INSERT INTO SESSION.table_name
   SELECT * FROM original_table
```

- **DECLARE statement for conditions or temporary tables.** In a DB2 user defined function (UDF), DECLARE is limited to declarations of variables and cursors only. If a condition or temporary table is declared, the function must be converted to a procedure.

- **Function returning table type.** In a UDF definition, in order to return a table, the keyword RETURNS TABLE must be used.

- **Using a function that returns table in FROM clause.** In order to perform the following,

  ```
  SELECT ... FROM table1 t1, foo() ...
  ```

  the following syntax must be used.

  ```
  SELECT ... FROM table1 t1, TABLE(foo()) t2 ...
  ```

  Also, the call to `foo()` has to reference a column from every correlation name appearing before it in the FROM clause. So in this case, that means that the query expression for `t2` must mention a column from `t1`.

- **Input type casting.** To invoke a function or a procedure, input arguments needs to be explicitly cast. For instance, to invoke `foo('I2', '1990-07-12')` in DB2, one must specify the following.

  ```
  foo(CAST('I2' AS CHAR(10)), CAST('1990-07-12' AS DATE))
  ```

- **ATOMIC keyword.** When defining a function body, the BEGIN statement should be associated with the ATOMIC keyword. This is to make sure the actions performed in the function can be completely rolled-back if the calling transaction aborts.

- **Booleans.** Booleans are not supported at all within a UDF. In particular, returning boolean types from a function and returning boolean arguments as OUT parameters in procedure are not supported. Integers can be used to simulate boolean values.

- **Printing messages.** There is no PRINT statement in DB2. To print ad-hoc messages, one must use the following.

  ```
  SIGNAL SQLSTATE '80000' SET MESSAGE_TEXT = 'message';
  ```

  Moreover, if the message contains multiple pieces that are concatenated, e.g., several variables, the following needs to be specified. Note that non-string typed values need to be cast.

  ```
  DECLARE output_text CHARACTER(250);
  SET output_text = var1 || ' ' ||
  CAST(time1 AS CHARACTER(20)) || ' ' ||
  CAST(time2 AS CHARACTER(20));
  SIGNAL SQLSTATE '80000' SET MESSAGE_TEXT = output_text;
  ```

- **ROW ARRAY.** DB2 does not support row arrays. Instead, a dedicated temporary table must be used, outside the function or procedure.
• $a \% b$. This syntax is not supported by DB2. This operator can be replaced with $\text{MOD}(a, b)$.

• $a \text{ BETWEEN} b \text{ AND } c$. $\text{BETWEEN}$ is not supported by DB2. This operator can be replaced with two predicates, $b \leq a \text{ AND } a \leq c$. 