An XML Database Benchmark:

"Transaction Processing over XML (TPoX)"

Version 1.2
http://tpox.sourceforge.net/

June 2008

Matthias Nicola, IBM Silicon Valley Lab, mnicola@us.ibm.com
Irina Kogan, IBM Toronto Lab, ikogan@ca.ibm.com
Rekha Raghu, rekha.raghu@intel.com
Agustin Gonzalez, agustin.gonzalez@intel.com
Berni Schiefer, IBM Toronto Lab, schiefer@ca.ibm.com
Kevin Xie, IBM Toronto Lab, kxie@ca.ibm.com

Contents

About This Document

This document was developed by IBM's DB2 Performance and Development group with input from Intel. It proposes an XML database benchmark called "**T**ransaction **P**rocessing **o**ver **X**ML" (TPoX). This document intends to stimulate discussion and solicit feedback from other interested groups. In collaboration with them, the goal is to extend, refine and promote the benchmark.

1 Introduction

1.1 Motivation

XML database functionality has been emerging in "XML-only" databases as well as in the major relational database products. Currently there is no industry standard XML database benchmark. To the best of our knowledge at the time of writing of this document, neither the Transaction Processing Council (TPC, *tpc.org*) nor the Standard Performance Evaluation Corporation (SPEC, *spec.org*) have specific plans to develop and standardize an XML database benchmark. Several benchmarks for the evaluation of XQuery performance have been proposed by the research community. These benchmark proposals include XMach-1 [4], XMark [14], XPathMark [7], XOO7 [6], XBench [17], MBench [13], and MemBeR [3],[10]. Some of these are predominantly *application oriented*, such as XMach-1 and XBench, while others are designed as abstract *micro-benchmarks*, e.g. MBench and MemBeR. XMark, XPathMark and X007 can be viewed as a *blend* because their data and queries represent a fictive application scenario but they also try to exercise all relevant aspects of the XQuery and XPath languages. More detailed analysis and comparison of the benchmarks can be found in [12],[1],[5],[11] and hence is not repeated here.

With the exception of X-Mach-1, these benchmarks focus mainly on XQuery processing but not on evaluating a database system in its entirety. Most of the benchmarks define queries only, no inserts, update or delete operations [12]. Most of them are also designed as single-user tests on a single XML document. These tests are very useful as micro-benchmarks to evaluate design alternatives and optimizations in an XQuery processing engine. However, these benchmarks are not enough to evaluate the overall performance of a full-fledged XML database system.

Therefore the goal of our project is to contribute and develop an XML database benchmark which not only exercises the query processor but all parts of a database, incl. storage, indexing, logging, transaction processing, and concurrency control.

1.2 Objectives

The objective is to develop an XML database benchmark which serves as a vehicle for fair and meaningful performance evaluation of XML database systems. The benchmark should be application-oriented and relevant to database users, database and hardware vendors, researchers and the XML community. The benchmark should stress all key components of a database system, and should be scalable from gigabytes to petabytes and usable on all major computing platforms including Unix, Windows and Linux.

The purpose of the benchmark is to push XML database systems and the underlying hardware to their limits. It's intended to aid in the investigation of performance enhancements and evaluation of design alternatives. The overall goal is to drive technological advancements in hardware and

software to support XML database workloads efficiently. The benchmark will also aid in the performance comparison of alternative technologies and database products.

The current version of TPoX is not complete and not perfect. Therefore, a key objective of this project is to solicit input and contributions from all interested parties, including vendors, research, software developers and database users.

1.3 Overview

Our TPoX benchmark prototype consists of the following 5 parts:

(1) A toolset for XML data generation to efficiently generate millions of XML documents with well-defined value distributions and referential consistency across documents.

(2) XML Schemas for all document types.

(3) A set of transactions to be run on the generated data. This includes queries in XQuery and SQL/XML notation as well as insert, update and delete operations.

(4) A workload driver is implemented in Java. It simulates *n* concurrent database users each of which connects to the database and submits a mix of transactions. The transactions are picked randomly from a set of predefined transaction templates. At run time, parameter markers in the templates are replaced by actual values drawn from configurable random value distributions. The workload driver collects and reports performance metrics, such as min/max/avg response time as well as overall throughput.

(5) Documentation for all of the above mentioned pieces.

We contribute this prototype to the open source community to continuously evolve this benchmark in a collaborative fashion and to help people run XML database performance tests.

The current version of TPoX focuses on data-oriented XML rather than document- or content-oriented XML. Just like the relational database world has TPC-C and TPC-H for OLTP vs. DSS performance evaluations, it's likely that the XML database world needs two benchmarks, one data-oriented and one document-oriented. But, we are also interested in extensions to TPoX to add document-oriented processing.

2 Benchmark Requirements

- Simplicity – The application scenario should be simple and realistic at the same time. Simplicity is important to encourage participation and adoption. The benchmark also needs to use realistic data and operations to be meaningful and relevant.
- Multi-user and single-user tests – The benchmark should include or allow both, with strong emphasis on multi-user tests because single-user databases are extremely rare.
- Response time and throughput – Both should be considered.
- Large collections of small documents – Many XML applications deal with large numbers (millions, billions) of small documents (2K-50K) rather than with one or few very large documents.
- 1-tier or multi-tier – The benchmark should allow 1-tier implementations to make it easy and cheap to set up and execute.

- Scalability – We suggest six scale factors: extra small (~10GB), small (~100GB), medium (~1TB), large (~10TB), extra large (~100TB) and extra-extra large (~1PB). Smaller scale factors can be made possible.
- Read mix – The read component of the workload is a mixture of full document retrieval, fragment retrieval, XML construction, predicate evaluation, joins and aggregations.
- XQuery – The workload is defined in XQuery. All queries are also available in standard-compliant SQL/XML notation. Manual rewrite of the candidate queries is acceptable for the sole purpose of meeting a database system's syntax, but strongly discouraged if done for performance reasons.
- The workload also contains insert, update, and delete operations, as well as XML schema validation.
- Schema evolution – schema evolution is a reality in the XML world. Hence, a future version of the benchmark should include XML schema evolution, with appropriate changes in subsequently inserted documents and submitted queries.

3 The TPoX Application Scenario

XML is very popular in financial applications where numerous XML-based standards have been developed, such as FIXML, FPML, IFX, FinXML, SwiftML and others [16]. From various financial application models, we selected online brokerage & trading because it is an important application area and easily understood by both benchmark participants and database users. The TPoX application scenario has two business entities: customers and the brokerage house (Figure 3.1). Customers have accounts and place orders to buy and sell securities (stocks, bonds and funds), thus changing the holdings in their accounts.

Although the clients would typically interact with an application server, which in turn is backed by a database, we ignore the middleware and focus on the workload that finally arrives at the database system, to focus on database performance only.

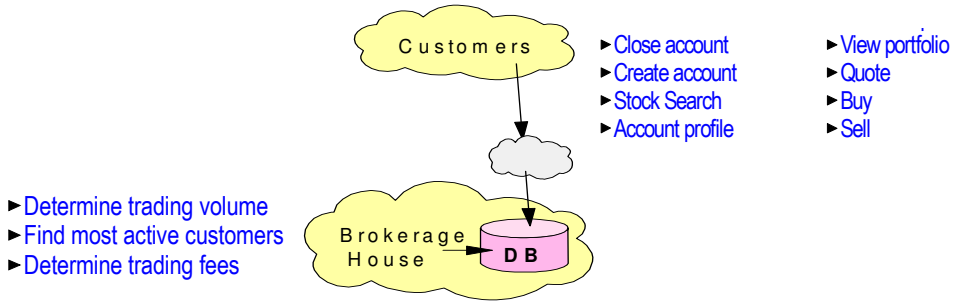


Fig 3.1

This scenario is, purposefully, a *simplification* of a real-world brokerage application. At the same time the goal was to retain the key aspects that influence performance, to keep the benchmark meaningful and realistic. The trade-off between "simple" and "realistic" is an ongoing theme in this benchmark's design. Not everything that is realistic is required to keep the benchmark relevant, e.g. if the impact on performance is negligible.

4 TPoX Data and Schemas

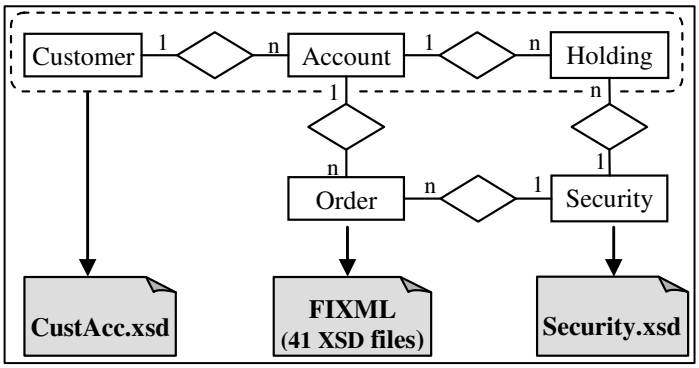
4.1 XML Data

Based on the application scenario, the main logical data entities and their attributes are the following (high level):

Entity	Attributes
Customer	Customer_id, name, address, email, password, date_of_birth,…
Account	account_id, customer_id, account_nr, balance,…
Security	symbol, company, volume, price, sector,…
Holding	symbol, quantity, price,…
Order	order_id, account_id, symbol, order_type, status, date, price, fee, quantity

These can be represented in XML in different ways. For example, each customer and each account could be a separate XML document. But, if we assume a one-to-many relationship between customers and accounts, each customer together with all his or her accounts could be one XML document. We choose this latter approach to make the benchmark more challenging with rewards for technological advances in partial XML updates and sub-document level concurrency control.

TPoX has 3 different types of XML documents[1]: **Order**, **Security**, and **CustAcc** which includes a customer with all her accounts. The information about holdings is included in the account data. Order documents follow the standard FIXML schema. Typical document sizes are 3 to 10 KB for Security, 1 – 2 KB for Order, and 4 - 20 KB for combined Customer/Account documents. To capture the diversity/irregularity often seen in real-world XML data, there are hundreds of optional attributes and elements with typically only a small subset present in any given document instance (such as in FIXML).



4.2 XML Schemas

The XML schemas for "CustAcc" and "Security" have been defined based on our analysis of real-world financial XML applications. The "Order" documents in TPoX are compliant with the FIXML schema Version 4.4 20040109, Revision 1 dated 2006-10-06:
http://www.fixprotocol.org/documents/352/fixml-schema-4-4-20040109rev1.zip
All XML schemas should be used as-is and in full even if the benchmark data does not use all parts of a schema (such as FIXML).

[1] Account documents conforming to the schema for an Account element in the CustaccDocument are also generated for Update 2 when the customer opens a new account (i.e., an account is to be inserted inside of an existing custacc document). See section 5.1.

Efficient handling of schema changes is an important requirement for XML databases. A schema migration should be simulated in a future version of the benchmark using several different types of schema change. Potential schema changes may include:

- add a new element with >1 max_occur indicator
- for an existing element, change the "maxoccur" indicator from 1 to a larger value.
- change the allowed set of data types for an element, e.g. allow both integer and character
- add an optional attribute
- change the allowed set or range of values for an element, e.g. restrict a data type, add values to an enumeration type, change a data type from xs:string to xs:integer

Schema changes could be applied to Orders and/or Accounts since they are continuously inserted/updated and hence the impact of the schema migration is high. Time for schema evolution should be included in the execution time and subsequent transactions should exercise the new schema well, i.e. documents for the new version of the schema are inserted as potential matches for queries. The database should not be prepared for the schema changes in any artificial manner. One way to ensure this might be to have thousands of possible schema changes and selecting a few randomly at runtime. For this to be reasonable, the different schema changes should be of comparable complexity so that benchmark results are comparable. This topic needs further investigation.

5 Workload

The TPoX workload consists of a set of queries, inserts, updates, and deletes, all of which are referred to as "*transactions*". Queries are provided in XQuery and SQL/XML notation. It is a stateless workload, i.e. the transactions are independent of each other, without implied order or think-time. There are 70% queries and 30% insert/update/delete in the workload mix.

We are proposing an initial set of transactions that focus on XML-based financial transaction processing rather than complex analytical queries. Also, our initial transactions intend to represent typical operations in the chosen application scenario without trying to exercise *all* interesting features of the XQuery language. This is a similar approach as in the TPC-C benchmark for relational applications.

The TPoX framework is very extensible and we solicit input to extend the set of transactions. It can be useful to define several different sets of transactions for different purposes. In fact, in [8] we exercised three multi-user workloads on the TPoX database: insert-only to populate the database, query-only, and a mixed workload consisting of inserts, updates, deletes and queries. Another approach is to run a set of complex queries as a single-user workload followed by a multi-user OLTP workload with short read/write transactions (i.e. power test and throughput test).

5.1 Insert, Update, Delete

The following observations were used in defining the update/delete/insert transactions:

- Customer accounts are updated to reflect trades (execution of orders), but not necessarily immediately after *every* order.

- New orders arrive continuously, old orders get pruned from the system eventually and at the same rate (many order inserts, many order deletes).
- Security prices are updated regularly during a business day (updates).
- The turnover of customers is low (few CustAcc inserts, few CustAcc deletes).
- The number of securities remains fixed (no delete or insert of securities).

Some of the updates defined below are complex transactions, consisting of read and write operations and joins. All updates are expressed in the XQuery Update language (http://www.w3.org/TR/xqupdate/).

Insert1: *A customer places a new order to buy or sell a security*
Insert a new Order document in the collection of order documents.

Insert2*: *A new customer signs up for online brokerage*
Insert a new CustAcc document in the collection of CustAcc documents.

Delete1: *An order is cancelled or archived*
For a given order id, delete the corresponding Order document

Delete2: *A customer closes all of his account and terminates business*
For a given customer id, delete the corresponding CustAcc document

Update1: *A customer decides to close one of his/her accounts* [delete subtree]
For a given account number, update the corresponding CustAcc document by removing the account from the CustAcc document, unless it's the customer's last and only account.

Update2*: *A customer opens (another) account* [insert/append subtree]
For a given customer id, update the corresponding CustAcc document by appending a new "Account" subtree to the list of accounts in the CustAcc document, unless this would exceed the maximum of number of accounts per customer (currently seven).

Update3: *The price of a security changes* [simple value update]
For a given security symbol, replace the values of the following elements in the corresponding security document: "LastTrade", "Ask", "Bid".

Update4*: *Processing by the brokerage house updates an order* [value update]
For a given order id, replace the value /FIXML/Order/@SolFlag with "Y" or "N" (choose randomly), and the value of "/FIXML/Order/Instrmt/@Src with a value randomly picked from this list of characters: "1","2",…,"9","A","B","C",…,"J".

Update5: *A previously placed buy order gets executed* [value update, add/replace subtree]
For a given account number, security symbol, and quantity: if the CustAcc document already contains a holding of the given security in the given account, increase the value of the element "quantity". Otherwise add a new "Position" subtree in the given account, which requires a join with Security to obtain the "Name" and "Type" of the Security. In either case also update the values of the following CustAcc elements: "LastUpdated", "OnlineActualBal", "OnlineClearedBal", and "WorkingBalance". Finally, replace the oldest "mValueDate" subtree of the account with a new and updated one. The update needs to abort if the new "Position" exceeds the maximum number of positions per account (currently 10).

Update6: *A previously placed sell order gets executed* [value update, delete/replace subtree]
For a given account number, [security symbol,] and quantity: if the given (sell-) quantity is equal or greater than the "quantity" in the corresponding "Position" in the CustAcc document, delete that "Position" subtree from the given account. Otherwise, just decrease the value of the element "quantity". In either case also update the values of the following CustAcc elements: "LastUpdated", "OnlineActualBal", "OnlineClearedBal", and "WorkingBalance". Finally, replace the last "mValueDate" subtree of the account with a new and updated one. The update needs to abort if it tries to delete the last and only position in the account, which is required by the XML schema.

* Insert2, Update2, and Update4 are executed with schema validation.

For simplicity we decided not to simulate all real-world application logic. For example: the transactions **Insert1** and **Update5/Update6** are decoupled and independent, i.e. when **Insert1** creates a new "sell" order for a specific customer and a specific security, the TPoX workload driver has no logic to subsequently schedule **Update6** for the *same* customer and the *same* security as the previous order insert. However, *some* customer account will eventually be updated with the sale of *some* security. We believe that in the long run this creates the same workload characteristics for the database as if we had implemented the application logic more accurately. We welcome feedback on these design decisions.

5.2 Queries

The workload should use XML data only. In relational database systems there should be XML columns only. No extra relational columns should be used.

We define seven core queries for a transaction processing workload. Their XQuery notation is shown below. The same queries in SQL/XML notation as well as additional candidate queries are provided in the appendix. Upon execution, literal values in predicates are replaced by

Q	Query Name	CustAcc	Security	Order	Characteristic
1	get_order			X	Return full order document without the FIXML root element
2	get_security		X		Return a full security document
3	customer_profile	X			Extract 7 customer elements to construct a new profile document
4	search_securities		X		Extract elements from some securities, based on 4 predicates
5	account_summary	X			Construction of an account statement
6	get_security_price		X		Extract the price of a security
7	customer_max_order	X		X	Join CustAcc & Orders to find the largest order from a certain customer

Table 1: TPoX OLTP queries

Q1: get_order

```
declare namespace o="http://www.fixprotocol.org/FIXML-4-4";
for $ord in db2-fn:xmlcolumn("ORDER.ODOC")/o:FIXML
where $ord/o:Order/@ID="103415"
return $ord/o:Order
```

Q2: get_security

```
declare default element namespace "http://tpox-benchmark.com/security";
for $s in db2-fn:xmlcolumn("SECURITY.SDOC")/Security
where $s/Symbol= "BCIIPRC"
return $s
```

Q3: customer_profile

```
declare default element namespace "http://tpox-benchmark.com/custacc";
for $cust in db2-fn:xmlcolumn("CUSTACC.CADOC")/Customer
where $cust/@id=2009
return  <Customer_Profile CUSTOMERID="{$cust/@id}">
                {$cust/Name}
                {$cust/DateOfBirth}
                {$cust/Gender}
                {$cust/Nationality}
                {$cust/CountryOfResidence}
                {$cust/Languages}
                {$cust/Addresses}
                {$cust/EmailAddresses}
        </Customer_Profile>
```

Q4: search_securities

```
declare default element namespace "http://tpox-benchmark.com/security";
for $sec in db2-fn:xmlcolumn("SECURITY.SDOC")/Security
where
    $sec/SecurityInformation/*/Sector= "Energy" and
    $sec/PE[. >=30 and . <35] and
    $sec/Yield>4.5
return  <Security>
                {$sec/Symbol}
                {$sec/Name}
                {$sec/SecurityType}
                {$sec/SecurityInformation//Sector}
                {$sec/PE}
                {$sec/Yield}
        </Security>
```

Q5: account_summary

```
declare default element namespace "http://tpox-benchmark.com/custacc";
for $cust in db2-fn:xmlcolumn("CUSTACC.CADOC")/Customer[@id=1011]
return  <Customer>{$cust/@id}
                {$cust/Name}
                <Customer_Securities>
                        { for $account in $cust/Accounts/Account
                         return <Account BALANCE="{$account/Balance/OnlineActualBal}"
```

```
                                    ACCOUNT_ID="{$account/@id}">
                                <Securities>
                                    {$account/Holdings/Position/Name}
                                </Securities>
                            </Account>    }
                    </Customer_Securities>
            </Customer>
```

Q6: get_security_price

```
declare namespace s="http://tpox-benchmark.com/security";
for $s in db2-fn:xmlcolumn("SECURITY.SDOC")/s:Security
where $s/s:Symbol= "SFDBX"
return  <print>The open price of the security "{$s/s:Name/text()}" is
                {$s/s:Price/s:PriceToday/s:Open/text()} dollars</print>
```

Q7: customer_max_order

```
declare default element namespace "http://www.fixprotocol.org/FIXML-4-4";
declare namespace c="http://tpox-benchmark.com/custacc";
let $orderprice :=
        for  $cust in db2-fn:xmlcolumn("CUSTACC.CADOC")/c:Customer[@id=1011]
        for $ord in db2-fn:xmlcolumn("ORDER.ODOC")/FIXML/Order[@Acct
                                    =$cust/c:Accounts/c:Account/@id/fn:string(.)]
        return $ord/OrdQty/@Cash
return max($orderprice)
```

6 Workload Execution Rules and Metrics

- The queries must be executed as-is, except when system-specific *syntax* changes are required. Other manual re-writes aimed at gaining performance are not allowed.
- XML documents must be stored in at most 3 tables, or 3 collections.
- Updates should be expressed in the emerging XQuery Update language, (http://www.w3.org/TR/xqupdate/)
- Indexes and materialized views can be defined at liberty. If defined, they must be kept fully consistent with the data at all times. The maintenance cost will prevent excessive use of index and materialized views.
- The time for the initial bulk load of the data base must be measured and reported, including the creation of tables and building of all indexes and materialized views.
- An optional single user test measures the response time of every insert, update, delete, and query, with a single user connected to the system, without concurrent transactions.
- The performance metric of a single user test is the sum of the response times of all transactions. The mandatory multi-user test measures throughput in transactions/second with *n* concurrent users [12]. This test has to use the TPoX workload driver (see 7.2).

- The number of users *n* must be equal or greater than the number of GBs of raw data for the chosen scale factor (see Table 2 below). For example, scale factor S with 100GB of data requires 100 concurrent users.
- All system configurations and parameter settings for hardware, operating system, database system and any other components must be finalized before the initial data population. No configuration or setup changes are allowed after the data population, or between a single and a multi user test, or at any other time.
- The technical details and cost of the benchmark system should be reported if absolute numbers are compared across different hardware.

6.1 Audit Requirements

An independent audit is recommended for the credibility of benchmark results. The following tests should be included in an audit.

- Data population test (proof that all data has been inserted correctly)
- XQuery and Update correctness tests (on smallest scale factor)
- ACID tests
- Must be able to insert and validate any valid FIXML document, not just Orders
- The database must be able to reject invalid XML documents upon insert
- The database must be able to reject invalid document updates (i.e. updates which lead to invalid data w.r.t. the relevant XML schema.)

Maybe a small percentage of invalid XML inserts and updates should be considered as part of the steady-state transaction mix.

7 Benchmark Prototype Implementation

This section describes the current version of the TPoX benchmark prototype. It currently consists of the following components:

- XML schemas for all document types
- A data generation package (see 7.1)
- A concurrent workload driver (see 7.2)
- Documentation for all of the above
- The set of TPoX transactions (inserts, updates, deletes, and queries).

7.1 XML Data Generation and Schema Design

- Large collections of small documents are generated as outlined in section 4.1.
- Toxgene 2.3 (http://www.alphaworks.ibm.com/tech/toxgene) can be used for data generation.
- The XML schemas, data, and use of namespaces reflect what we see in real XML applications. The FIXML industry standard schema is used for Orders (*fixprotocol.org*).
- "Account" data is inlined in the "customer" documents, i.e. there is one XML document per customer that includes all of the customer's accounts. This makes account and customer updates more challenging than having separate documents for customers and

accounts. The intention is to drive and reward technological progress in areas such as sub-document level updates and sub-document level concurrency control.

- TPoX uses a fixed number of "Security" documents (20833), representing the majority of publicly traded stocks, bonds and funds. Real security names and symbols are used, but their prices are fictional.
- TPoX uses a scalable number of "Custacc" and "Order" documents. On average, each customer has 5 orders. Documents are data-centric with some full-text elements.
- The TPoX scale factors are listed in Table 2. The data generator also supports intermediate and smaller scale factors than those shown in Table 1, such as XXS (1GB) and XXXS (100MB) for small test. (Note that the raw data size can vary depending on how it is measured. The exact sum of bytes of all documents at a certain scale factor is lower than the total size reported by a file system, due to internal page fragmentation.)
- It is necessary to generate more documents than just the initial database population, i.e. to feed new inserts during the mixed workload on top of the populated databases.
- Separate documentation describes the data distributions and the use of the data generator.

Scale	Approx raw size		Security	CustAcc	Orders	Actual Total Raw Data Size
XS	10GB	#Docs:	20,833	600,000	3,000,000	3,620,833
		GB:	0.13	3.62	5.79	9.55
S	100GB	#Docs:	20,833	6,000,000	30,000,000	36,020,833
		GB:	0.13	36.24	57.91	94.28
M	1TB	#Docs:	20,833	60,000,000	300,000,000	360,020,833
		GB:	0.13	362.41	579.07	941.61
L	10TB	#Docs:	20,833	600,000,000	3,000,000,000	3,600,020,833
		GB:	0.13	3624.08	5790.71	9414.92
XL	100TB	#Docs:	20,833	6,000,000,000	30,000,000,000	36,000,020,833
		GB:	0.13	36240.77	57907.10	94148.00
XXL	1PB	#Docs:	20,833	60,000,000,000	300,000,000,000	360,000,020,833
		GB:	0.13	362407.7	579071.0	941480.0

Table 2: TPoX Scale Factors and Data Volumes

7.2 Workload Driver

- The TPoX workload driver is implemented in Java and uses JDBC. All interactions with the target database system are in single class "DatabaseOperations" which can be extended/modified to support databases other than DB2.
- The driver spawns 1 to n parallel threads (configurable) each of which simulates a database user that connects to the TPOX database and submits a mix transactions
- The transactions are picked randomly from a set of predefined transaction templates.
    - At run time, parameter markers in the templates are replaced by actual values drawn from configurable random value distributions
    - Each transaction is assigned a *weight* in the workload mix. The total of all transaction weights is 1 (i.e. 100%). When a user picks "the next" transaction to execute, the probability for each transaction is equal to its weight.

- For testing purposes, a benchmark run can be limited either by a time limit or by the number of transactions that each concurrent user executes.
- For multi-user tests, a ramp-up period can be specified which precedes the measurement interval to reach a steady state of transaction throughput.
- Performance metrics are collected and a report is produced (on screen / files):
    - Min, max, avg and total elapsed time for each transaction template
    - Total throughput in transactions per minute
    - Number of completed transactions per user
    - Percentiles and confidence intervals
- During a benchmark run, performance metrics can be emitted every *n* seconds to allow analysis of performance behavior over time.
- The initial version supports DB2, but can be extended to support other systems.
- Separate documentation describes the workload driver in more details

7.3 Current Limitations, "To Do"

The current benchmark implementation has certain limitations which we seek to remove over time. We welcome feedback for prioritization as well as active participation!

- The population of account IDs is not dense. Hence, data selection based on a random account ID is likely to not find any matching data. We're currently improving the data generation to produce dense (consecutive) account IDs to allow additional interesting operations. For example, updates 5 and 6 require dense account IDs.
- On average, each customer has 5 orders. Currently, each customer has *exactly* 5 orders and all of them relate to the first of his accounts. We are prototyping an enhancement to generate *n* orders for *each* account, where *n* is drawn from a normal distribution.
- The element "OnlineActualBal" has a random value and is not the sum of the holdings of the account. We don't think this affects the database performance evaluation. Similarly, other elements in the XML data may have random or fixed data if that has no impact.
- The TPoX data generation was not "resumable". This means you cannot start a data generation where it previously has ended, e.g. to increase your data population from 600K CustAcc and 3M Order documents to 1M Custacc and 5M Orders. We now have a solution for this. It is freely available upon request and will be included in TPoX 1.3.
- All data needs to be pre-generated. It would be interesting to explore if and how data generation on-the-fly is feasible (while preserving all referential integrity in the data population).
- Update6 ("sell"): the TPoX workload driver has no knowledge about which securities are held be a certain customer. Thus, if we specified that customer *x* should sell security *y*, there is a high probability that this customer does not own any shares of that security. Hence, we simply "sell" shares from the first security position in the customer's account.
- Using the workload driver with a database which does not support JDBC will require more additional work than for a JDBC data source.

- Literal values for parameters are currently drawn from uniform integer value distributions or randomly picked from lists of enumerated input values. Non-uniform distributions as well as distributions for date or decimal values may be desirable for additional queries.
- We need to keep improving the simplicity and usability of TPoX. Suggestions & contributions are welcome.
- It will be useful to develop additional workloads for the same data sets, e.g. one that exercises more complex analytical queries, or one that performs update-heavy-batch processing of accounts.

References

[1] L. Afanasiev and M. Marx: "An analysis of the current XQuery benchmarks", *Experimental Evaluation of Data Management Systems* (EXPDB), 2006. http://gemo.futurs.inria.fr/events/EXPDB2006/PAPERS/Afanasiev.pdf

[2] L. Afanasiev and M. Marx: "XCheck – An Automated XQuery Benchmark Tool", 2005, http://ilps.science.uva.nl/Resources/XCheck/index.html

[3] L. Afanasiev, I. Manolescu and P. Michiels: "MemBeR: A Micro-benchmark Repository for XQuery", *XML Symposium (XSym)* 2005. http://ilps.science.uva.nl/Resources/MemBeR/

[4] T. Böhme, E. Rahm: XMach-1: "A Benchmark for XML Data Management", *Proceedings of German database conference BTW2001*, pp 264-273, Springer, Berlin, March 2001, http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html

[5] T. Böhme et al: "Multi-User Evaluation of XML Data Management Systems with XMach-1", LNCS Vol. 2590, 2003. http://www.informatik.uni-leipzig.de/~boehme/paper/xmach1-eextt2002.pdf

[6] S. Bressan, G. Dobbie, Z. Lacroix, M. L. Lee, Y. G. Li, U. Nambiar and B. Wadhwa: "XOO7: Applying OO7 Benchmark to XML Query Processing Tools", *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM),* November 2001, http://www.comp.nus.edu.sg/~ebh/XOO7.html

[7] M. Franceschet: "XPathMark - An XPath benchmark for XMark generated data", *International XML Database Symposium (XSYM)*, 129-143, 2005, http://dare.uva.nl/document/13765 , http://www.dimi.uniud.it/~francesc/xpathmark/index.html

[8] I. Kogan, M. Nicola, B. Schiefer: "DB2 9 XML performance characteristics", http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0606schiefer/index.html

[9] Stefan Manegold: "An Empirical Evaluation of XQuery Processors", in *Experimental Evaluation of Data Management Systems* (EXPDB), 2006.

[10] I. Manolescu, C. Miachon and P. Michiels: "Towards micro-benchmarking XQuery", *Experimental Evaluation of Data Management Systems (EXPDB)*, 2006.

[11] U. Nambiar, Z. Lacroix, S. Bressan, M. L. Lee, and Y. G. Li: "XML Benchmarks put to the test", *3rd International Conference on Information Integration and Web-based Applications & Services* (IIWAS), September, 2001.

[12] Matthias Nicola, Irina Kogan, Berni Schiefer: "An XML Transaction Processing Benchmark", *ACM SIGMOD Conference 2007*. http://tpox.sourceforge.net/Sigmod2007_TPoX.pdf

[13] K. Runapongsa, J. M. Patel, H. V. Jagadish, Y. Chen, and S. Al-Khalifa: "The Michigan Benchmark: Towards XML Query Performance Diagnostics", *Proceedings of the 29th VLDB Conference*, Berlin, Germany, 2003, http://www.eecs.umich.edu/db/mbench

[14] A. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu and R. Busse: "XMark: A Benchmark for XML Data Management", *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pp 974-985, August 2002, http://monetdb.cwi.nl/xml/

[15] A. Schmidt, F. Waas, S. Manegold, M. L. Kersten: „A Look Back on the XML Benchmark Project", in *Intelligent Search on XML*, Volume 2818 of LNCS/LNAI, pp 263-278, 2003.

[16] XML on Wall Street, http://lighthouse-partners.com/xml

[17] B. Yao, M. T. Özsu, and J. Keenleyside: "XBench - A Family of Benchmarks for XML DBMSs", *Proceedings of EEXTT 2002 and DiWeb 2002, LNCS Vol. 2590*, pages 162-164, http://db.uwaterloo.ca/ddbms/projects/xbench/

Appendix: TPoX Transactions

A. The 7 Core Queries in SQL/XML Notation

Q1: get_order

```
SELECT XMLQUERY
        ('declare namespace o="http://www.fixprotocol.org/FIXML-4-4";
        for $ord in $odoc/o:FIXML
        return $ord/o:Order'
        PASSING odoc AS "odoc")
FROM order
WHERE XMLEXISTS
        ('declare namespace o="http://www.fixprotocol.org/FIXML-4-4";
        $odoc/o:FIXML/o:Order[@ID=$id]
        'PASSING odoc AS "odoc", cast (? as varchar(10)) as "id")
```

Q2: get_security

```
SELECT XMLQUERY
        ('declare default element namespace "http://tpox-benchmark.com/security";
        for $sec in $sdoc/Security
        return $sec'
        PASSING sdoc AS "sdoc")
FROM security
WHERE XMLEXISTS
        ('declare default element namespace "http://tpox-benchmark.com/security";
        $sdoc/Security[Symbol=$sym]'
        PASSING sdoc AS "sdoc", cast(? as varchar(10)) as "sym")
```

Q3: customer_profile

```
SELECT XMLQUERY
        ('declare default element namespace "http://tpox-benchmark.com/custacc";
        for $cust in $cadoc/Customer
        return
            <Customer_Profile CUSTOMERID="{$cust/@id}">
                {$cust/Name}
                {$cust/DateOfBirth}
                {$cust/Gender}
                {$cust/CountryOfResidence}
                {$cust/Languages}
                {$cust/Addresses}
                {$cust/EmailAddresses}
            </Customer_Profile>'
        PASSING cadoc AS "cadoc")
FROM custacc
WHERE XMLEXISTS
        ('declare default element namespace "http://tpox-benchmark.com/custacc";
        $cadoc/Customer[@id=$id]'
        PASSING cadoc AS "cadoc", cast (? as double) as "id" )
```

Q4: search_securities

```
SELECT XMLQUERY
        ('declare default element namespace "http://tpox-benchmark.com/security";
        for $sec in $sdoc/Security
        return
            <Security>
                {$sec/Symbol}
                {$sec/Name}
                {$sec/SecurityType}
                {$sec/SecurityInformation//Sector}
                {$sec/PE}
                {$sec/Yield}
            </Security>'
        PASSING sdoc AS "sdoc")
FROM security
WHERE XMLEXISTS
        ('declare default element namespace "http://tpox-benchmark.com/security";
        $sdoc/Security[SecurityInformation/*/Sector=$sector and Yield>$yield]'
        PASSING     sdoc AS "sdoc",
                    cast (? as varchar(25)) as "sector",
                    cast (? as double) as "yield" )
AND XMLEXISTS
        ('declare default element namespace "http://tpox-benchmark.com/security";
        $sdoc/Security/PE[.>=$pe1 and .<$pe2]'
        PASSING  sdoc AS "sdoc", cast (? as double) as "pe1", cast (? as double) as "pe2")
```

Q5: account_summary

```
SELECT XMLQUERY
        ('declare default element namespace "http://tpox-benchmark.com/custacc";
        for $cust in $cadoc/Customer
        return
           <Customer>{$cust/@id}
                {$cust/Name}
                <Customer_Securities>
                        { for $account in $cust/Accounts/Account
                        return
                                <Account BALANCE="{$account/Balance/OnlineActualBal}"
                                        ACCOUNT_ID="{$account/@id}">
                                        <Securities>
                                                {$account/Holdings/Position/Name}
                                        </Securities>
                                </Account>     }
                </Customer_Securities>
        </Customer>'
        PASSING cadoc AS "cadoc")
FROM custacc
WHERE XMLEXISTS
        ('declare default element namespace "http://tpox-benchmark.com/custacc";
        $cadoc/Customer[@id=$id]'
        PASSING cadoc AS "cadoc", cast (? as integer) as "id")
```

Q6: get_security_price

```
SELECT XMLQUERY
        ('declare namespace s="http://tpox-benchmark.com/security";
        for $sec in $sdoc/s:Security
        return
           <print>The open price of the security "{$sec/s:Name/text()}" is
                {$sec/s:Price/s:PriceToday/s:Open/text()} dollars</print>'
        PASSING sdoc AS "sdoc")
FROM security
WHERE XMLEXISTS
        ('declare namespace s="http://tpox-benchmark.com/security";
        $sdoc/s:Security[s:Symbol=$sym]'
        PASSING sdoc AS "sdoc", cast (? as varchar(10)) as "sym")
```

Q7: customer_max_order

```
SELECT DECIMAL(CAST(MAX(price) AS INTEGER), 15, 2) AS maxprice
FROM
(SELECT       XMLCAST(XMLQUERY
        ('declare default element namespace "http://www.fixprotocol.org/FIXML-4-4";
        let $orderprice := $odoc/FIXML/Order/OrdQty/@Cash
```

```
        return $orderprice'
        PASSING odoc AS "odoc")
                AS DOUBLE) AS price
FROM custacc, order
WHERE XMLEXISTS
        ('declare namespace c="http://tpox-benchmark.com/custacc";
        $cadoc/c:Customer[@id=$id]'
        PASSING cadoc AS "cadoc", cast (? as double) as "id")
AND XMLEXISTS
        ('declare default element namespace "http://www.fixprotocol.org/FIXML-4-4";
        declare namespace c="http://tpox-benchmark.com/custacc";
        $odoc/FIXML/Order[@Acct=$cadoc/c:Customer/c:Accounts/c:Account/@id/fn:string(.)]'
        PASSING cadoc AS "cadoc", odoc AS "odoc")) AS T
```

B. Additional Candidate Queries

C1, C2, C3, C4 are additional candidate queries:

	Query Name	CustAcc	Security	Order	Characteristic
Q1	get_order			X	Return order information
Q2	get_security		X		Return a full security document
Q3	customer_profile	X			Construct a new profile document
Q4	search_securities		X		Extract elements, based on 4 predicates
Q5	account_summary	X			Construction of an account statement
Q6	get_security_price		X		Extract the price of a security
Q7	customer_max_order	X		X	Join CustAcc & Orders to find the largest order from a certain customer
C1	**max_stock_order**	**X**	**X**	**X**	Obtain the largest order value for given sets of customers and stocks
C2	**cust_expensive_orders**	**X**		**X**	Find customer in a given state who had orders larger than a certain value
C3	**cust_sold_security**	**X**		**X**	For a given zip code, find the primary phone numbers of customers who sold securities.
C4	**current_order_price**		**X**	**X**	For a given order, get the current open price of the corresponding security.

Table 3: TPoX queries

C1: max_stock_order

```
(: Return the maximum order value for the stocks in a certain industry bought by customers
living in the specified state:)
declare default element namespace "http://www.fixprotocol.org/FIXML-4-4";
declare namespace s="http://tpox-benchmark.com/security";
declare namespace c="http://tpox-benchmark.com/custacc";
let $order  :=
   for $ss in db2-fn:xmlcolumn("SECURITY.SDOC")/s:Security[
                s:SecurityInformation/s:StockInformation/s:Industry ="Energy"]
   for $ord in db2-fn:xmlcolumn("ORDER.ODOC")/FIXML/Order[
```

```
                Instrmt/@Sym= $ss/s:Symbol/fn:string(.)]
    for $cs in db2-fn:xmlcolumn("CUSTACC.CADOC")/c:Customer[
                c:Addresses/c:Address/c:State= "West Virginia"]/c:Accounts/c:Account[
                                                        @id =$ord/@Acct/fn:string(.)]
        return $ord/OrdQty/@Cash
return string(max($order))
```

C2: cust_expensive_orders

```
(: Retrieve the names of the customers in the specified country who have orders higher than a
given value. :)
declare default element namespace "http://www.fixprotocol.org/FIXML-4-4";
declare namespace c="http://tpox-benchmark.com/custacc";
for $cust in db2-fn:xmlcolumn("CUSTACC.CADOC")/c:Customer[
                                                        c:CountryOfResidence="Germany"]
for $ord in db2-fn:xmlcolumn("ORDER.ODOC")/FIXML/Order[
        OrdQty/@Cash>3000]/@Acct[fn:string(.)=$cust/c:Accounts/c:Account/@id/fn:string(.)]
return $cust/c:ShortNames/c:ShortName
```

C3: cust_sold_security

```
(: Get the phone numbers of customers in a specific zip code who have
 sold any security. Sort by customer last name. :)
declare default element namespace "http://www.fixprotocol.org/FIXML-4-4";
declare namespace c="http://tpox-benchmark.com/custacc";
for $cust in db2-fn:xmlcolumn("CUSTACC.CADOC")/c:Customer[
                                                c:Addresses/c:Address/c:PostalCode=95141]
for $ord in db2-fn:xmlcolumn("ORDER.ODOC")/FIXML/Order[
                        @Acct=$cust/c:Accounts/c:Account/@id/fn:string(.) and @Side="2"]
order by $cust/c:Name/c:LastName/text()
return
    <Customer>
        {$cust/c:Name/c:LastName/text()} –
        {$cust/c:Addresses/c:Address[@primary="Yes"]/c:Phones/c:Phone[@primary="Yes"]}
    </Customer>
```

C4: current_order_price

```
(:For a given order, get the current open price of the corresponding security:)
declare default element namespace "http://www.fixprotocol.org/FIXML-4-4";
declare namespace s="http://tpox-benchmark.com/security";
for $ord in db2-fn:xmlcolumn("ORDER.ODOC")/FIXML/Order[@ID="109505"]
for $sec in db2-fn:xmlcolumn("SECURITY.SDOC")/s:Security[
                                                s:Symbol=$ord/Instrmt/@Sym/fn:string(.)]
return
        <Today_Order_Price ORDER_ID="{$ord/@ID}">
                {string($ord/OrdQty/@Qty*$sec/s:Price/s:PriceToday/s:Open)}
        </Today_Order_Price>
```

More queries will be added.

C. Update Statements

U1: close_account

```
-- A customer decides to close one of his/her accounts [delete subtree]
-- For a given account number, update the corresponding CustAcc document by
-- removing the account from the CustAcc document.

-- Removal does not take place if the customer had only one account (minimum
-- allowed).

UPDATE custacc
SET cadoc = XMLQUERY(' declare default element namespace "http://tpox-
benchmark.com/custacc";
  transform
    copy $c := $doc
    modify
      (: do not delete the account if the customer has only one account :)
      if (count($c/Customer/Accounts/Account) >= 2)
      then do delete
            $c/Customer/Accounts/Account[@id="104138966"]
      else ()
  return  $c'
  PASSING cadoc AS "doc")
WHERE XMLEXISTS
('declare default element namespace "http://tpox-benchmark.com/custacc";
$cadoc/Customer/Accounts/Account[@id="104138966"]'
PASSING cadoc AS "cadoc")
```

U2: open_account

```
-- A customer opens (another) account [insert/append subtree]
-- For a given customer id, update the corresponding CustAcc document by
-- appending a new Account subtree to the list of accounts in the CustAcc
-- document.

-- Insert does not take place if the customer already has seven accounts
-- (maximum allowed).

UPDATE custacc
SET cadoc = XMLVALIDATE(XMLQUERY('declare default element namespace
"http://tpox-benchmark.com/custacc";
    copy $c := $doc
    modify
```

```
    (: do not add the account if it exceeds the max of seven accounts :)
    if (count($c/Customer/Accounts/Account) < 7)
    then do insert
            <Account id="104138966">
                <Category>9</Category>
                <AccountTitle>Dr Vineeta Krablin YEN</AccountTitle>
                <ShortTitle>Krablin YEN</ShortTitle>
                <Mnemonic>KrablinYEN</Mnemonic>
                <Currency>YEN</Currency>
                <CurrencyMarket>5</CurrencyMarket>
                <OpeningDate>1993-01-31</OpeningDate>
                <AccountOfficer>Lorraine Bos</AccountOfficer>
                <LastUpdate>2003-12-10T09:07:20</LastUpdate>
                <Balance>
                        <OnlineActualBal>932797</OnlineActualBal>
                        <OnlineClearedBal>852847</OnlineClearedBal>
                        <WorkingBalance>739379</WorkingBalance>
                </Balance>
                <Passbook>Yes</Passbook>
                <gValueDate>
                        <mValueDate>
                                <ValueDate>2001-08-21</ValueDate>
                                <CreditMovement>77452.85</CreditMovement>
                                <ValueDatedBal>210573</ValueDatedBal>
                        </mValueDate>
                </gValueDate>
                <ChargeCcy>YEN</ChargeCcy>
                <InterestCcy>YEN</InterestCcy>
                <AllowNetting>Yes</AllowNetting>
                <gInputter>
                        <Inputter>Mostefa Kruseman</Inputter>
                        <Inputter>Mostefa Kruseman</Inputter>
                </gInputter>
                <Holdings>
                        <Position>
                                <Symbol>OIIM</Symbol>
                                <Name>Cendant Corporation</Name>
                                <Type>Stock</Type>
                                <Quantity>2020.072</Quantity>
                        </Position>
                </Holdings>
            </Account>
    into $c/Customer/Accounts
    else ()
return $c'
PASSING cadoc AS "doc")
```

```
   according to xmlschema id custacc)
WHERE XMLEXISTS
('declare default element namespace "http://tpox-benchmark.com/custacc";
$cadoc/Customer[@id=1011]'
PASSING cadoc AS "cadoc")
```

U3: price_change

```
-- The price of a security changes [simple value update]
-- For a given security symbol, replace the values of the following elements
-- in the corresponding security document: LastTrade, Ask, Bid.

UPDATE security
SET sdoc = XMLQUERY('declare default element namespace "http://tpox-
benchmark.com/security";
   copy $secdoc := $doc
   modify
         let $price := $secdoc/Security/Price
         let $newlasttrade := $price/PriceToday/Open*0.95
         return
         (
         do replace value of
                  $price/LastTrade with $newlasttrade,
         do replace value of
                  $price/Ask with $newlasttrade*1.01,
         do replace value of
                  $price/Bid with $newlasttrade*0.99)
   return  $secdoc'
   PASSING sdoc AS "doc")
WHERE XMLEXISTS
('declare default element namespace
"http://tpox-benchmark.com/security";
$sdoc/Security[Symbol="OIIM"]'
PASSING sdoc AS "sdoc"
)
```

-- U4: order_status

```
-- Processing by the brokerage house updates an order [attr value update]
-- For a given order id, replace the value /FIXML/Order/@SolFlag with "Y" or
-- "N" (choose randomly), and the value of "/FIXML/Order/Instrmt/@Src with a
-- value randomly picked from this list of characters: "1","2",...,"9","A","B",
-- "C",...,"J".

UPDATE order
SET odoc = XMLVALIDATE(XMLQUERY('declare default element namespace
"http://www.fixprotocol.org/FIXML-4-4";
```

```
   copy $o := $doc
   modify (
     do replace value of $o/FIXML/Order/@SolFlag with "N",
     do replace value of $o/FIXML/Order/Instrmt/@Src with "C")
   return  $o'
   PASSING odoc AS "doc")
     according to xmlschema id order)
WHERE XMLEXISTS
('declare default element namespace "http://www.fixprotocol.org/FIXML-4-4";
$doc/FIXML/Order[@ID="103415"]'
PASSING odoc AS "doc")
```

U5: buy_security

```
-- A previously placed buy order gets executed [value update, add & replace subtree]
-- For a given account number, security symbol, and quantity: if the CustAcc
-- document already contains a holding of the given security in the given
-- account, increase the value of the element quantity. Otherwise add a new
-- Position subtree in the given account, which requires a join with Security
-- to obtain the Name and Type of the Security. In either case also update
-- the values of the following CustAcc elements: LastUpdated,
-- OnlineActualBal, OnlineClearedBal, and WorkingBalance. Finally, replace
-- the last mValueDate subtree of the account with a new and updated one.

-- The transaction does not take place if the customer already had holdings of
-- more than 10 securities (maximum allowed) and tries to buy a new security.

UPDATE custacc
SET cadoc = XMLQUERY(
'declare default element namespace "http://tpox-benchmark.com/custacc";
declare namespace s = "http://tpox-benchmark.com/security";
     copy $c := $doc
     modify
          let $acct := $c/Customer/Accounts/Account[@id="104138966"]
          let $actualbalance := $acct/Balance/OnlineActualBal
          let $clearedbalance := $acct/Balance/OnlineClearedBal
          let $workingbalance := $acct/Balance/WorkingBalance
          let $mvaluedate := $acct/gValueDate/mValueDate[last()]
          let $currentdate := fn:current-date()
          (: need to get security information :)
          let $sec := db2-fn:xmlcolumn("SECURITY.SDOC")/s:Security[
          s:Symbol="OIIM"]
          return (
          (: do not buy if it exceeds the maximum of 10 holdings per acc :)
          if (count($acct/Holdings/Position)=10 and
            count($acct/Holdings/Position[Symbol="OIIM"])=0)
          then ()
          else (
                    (: add new position if no shares of this security exist in this acc :)
```

```
                if (count($acct/Holdings/Position[Symbol="OIIM"])=0) then
                        do insert
                                <Position>
                                        <Symbol>{$sec/s:Symbol/text()}</Symbol>
                                        <Name>{$sec/s:Name/text()}</Name>
                                        <Type>{$sec/s:SecurityType/text()}</Type>
                                        <Quantity>50</Quantity>
                                </Position>
                        into $acct/Holdings
                (: if shares of this security existed in this acc, add the new shares:)
                else
                        do replace value of
                                $acct/Holdings/Position[Symbol="OIIM"][1]/Quantity with
                                xs:decimal($acct/Holdings/Position[Symbol="OIIM"][1]/Quantity
                                        + 50),
                        (:  now update the account balances and timestamp :)
                        do replace value of $acct/LastUpdate with fn:current-dateTime(),
                        do replace value of $actualbalance with
                                xs:decimal($actualbalance+(50*$sec/s:Price/s:Ask)),
                        do replace value of $clearedbalance with
                                xs:decimal($clearedbalance+(50*$sec/s:Price/s:Ask))),
                        do replace value of $workingbalance with
                                xs:decimal($workingbalance+(50*$sec/s:Price/s:Ask)),
                        do replace $mvaluedate with
                                <mValueDate>
                                        <ValueDate>{$currentdate} </ValueDate>
                                  <CreditMovement>{xs:decimal(156882.77)}</CreditMovement>
                                  <ValueDatedBal>{xs:decimal(45736.85)}</ValueDatedBal>
                                </mValueDate>  )   )
return  $c'
PASSING cadoc AS "doc")

WHERE XMLEXISTS (
'declare default element namespace "http://tpox-benchmark.com/custacc";
$cadoc/Customer/Accounts/Account[@id="104138966"]'
PASSING cadoc AS "cadoc")
```

U6: sell_security

```
-- A previously placed sell order gets executed [value/delete & replace subtree]
-- For a given account number and quantity: if the given
-- (sell-) quantity is equal to or greater than the "quantity" in the first
-- "Position" in the CustAcc document, delete that "Position" subtree from the
-- given account. Otherwise just decrease the value of the element "quantity"
-- for that security holding.
-- In either case also update the values of the following CustAcc
-- elements: "LastUpdated", "OnlineActualBal", "OnlineClearedBal", and
```

```
-- "WorkingBalance". Finally, replace the last "mValueDate" subtree of the
-- account with a new and updated one.

-- The transaction does not take place if the customer has a holding of
-- only one security (minimum allowed) and tries to sell everything he has.
-- The last position element cannot be removed, as not to violate the custacc schema.

UPDATE custacc
SET cadoc = XMLQUERY (
'declare default element namespace "http://tpox-benchmark.com/custacc";
declare namespace s= "http://tpox-benchmark.com/security";
   copy $c := $doc
   modify
        let $acct := $c/Customer/Accounts/Account[@id="104138966"]
        let $actualbalance := $acct/Balance/OnlineActualBal
        let $clearedbalance := $acct/Balance/OnlineClearedBal
        let $workingbalance := $acct/Balance/WorkingBalance
        let $mvaluedate :=$acct/gValueDate/mValueDate[last()]
        let $currentdate := fn:current-date()
        (: need to get security information :)
        let $sec := db2-fn:xmlcolumn("SECURITY.SDOC")/s:Security[
                s:Symbol=$acct/Holdings/Position[1]/Symbol/fn:string(.)]
  return (
        (: do not sell if it depletes the last position in this account :)
        if (count($acct/Holdings/Position) < 2 and
                $acct/Holdings/Position[1]/Quantity <= 50) then ()
        else (
                (: delete the position if the sell redeems all shares held :)
                if ($acct/Holdings/Position[1]/Quantity <= 50) then
                        do delete $acct/Holdings/Position[1]
                (: otherwise subtract the sold shares :)
                else
                        do replace value of $acct/Holdings/Position[1]/Quantity
                        with xs:decimal($acct/Holdings/Position[1]/Quantity – 50),
                        (:  now update the account balances and timestamp :)
                        do replace value of $acct/LastUpdate with fn:current-dateTime(),
                        do replace value of $actualbalance with
                                xs:decimal($actualbalance - (50*$sec/s:Price/s:Bid)),
                        do replace value of $clearedbalance with
                                xs:decimal($clearedbalance - (50*$sec/s:Price/s:Bid)),
                        do replace value of $workingbalance with
                                xs:decimal($workingbalance - (50*$sec/s:Price/s:Bid)),
                        do replace $mvaluedate with
                        <mValueDate>
                                <ValueDate>{$currentdate} </ValueDate>
                                <CreditMovement>{xs:decimal(156882.77)}</CreditMovement>
```

```
                                <ValueDatedBal>{xs:decimal(45736.85)}</ValueDatedBal>
                        </mValueDate>  )   )
return  $c'
PASSING cadoc AS "doc"
)
WHERE XMLEXISTS
('declare default element namespace "http://tpox-benchmark.com/custacc";
$cadoc/Customer/Accounts/Account[@id="104138966"]'
PASSING cadoc AS "cadoc")
```