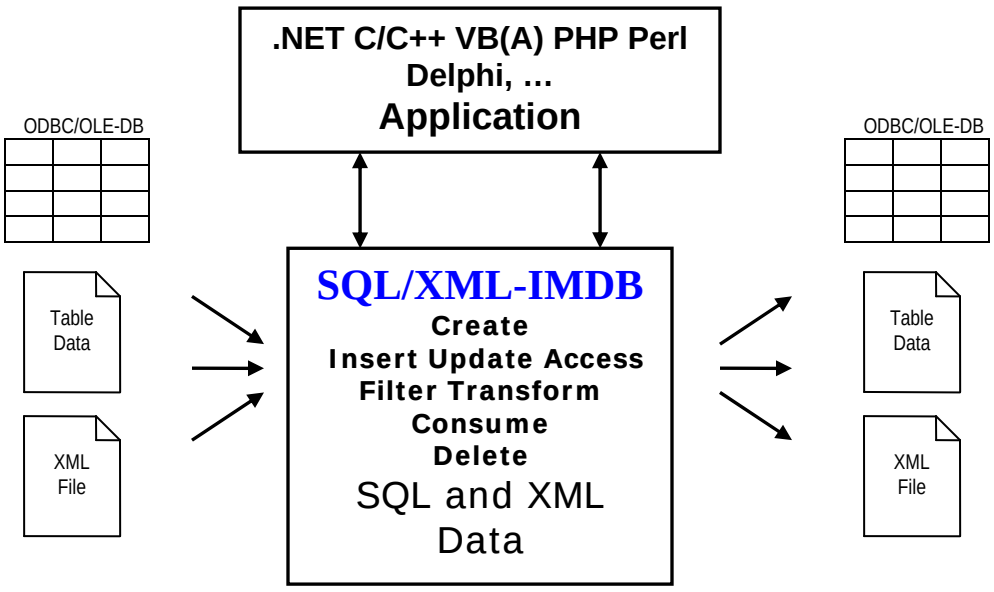
SQL/XML-IMDB

In-Memory SQL / XML Database Component for
Universal Data Management

White Paper

© QuiLogic Inc. 2000-2009

www.QuiLogic.com



QuiLogic has developed SQL/XML-IMDB, a high performance in-memory sql / xml database engine with SQL and XQuery interface, transaction and multi-threading support. SQL/XML IMDB simplifies application development and integration through declarative data management and by providing a seamless integration between SQL and XML data.

SQL/XML-IMDB is part of QuiLogic's information integration strategy to unify structured and unstructured data from sources such as relational databases, XML documents, flat files and Web services (SOAP). With SQL/XML-IMDB, users can access relational data as if it were XML data; exchange complex data structures between different applications; access real-time info and search across XML text documents and/or relational tables.

Features

- Combined relational and XML database engine with SQL and XQuery interface. Supports **.**NET, Visual Basic, VBA, C, C++, Delphi, Perl, and PHP application development environments. Available as NET Assembly, ActiveX, DLL and LIB component including ANSI, as well as UNICODE libraries.

- Supports process local-memory tables for high speed application-internal data management and **shared memory** tables for high performance data sharing and exchange between different applications and development environments.

- Data exchange between different applications will be as simple as executing INSERT/SELECT statements in these applications.

- Ability to create XML views over relational data and to create, manipulate and transfer data between XML and SQL tables.

- Build in XML Data-Binding facility for connecting XML files to in-memory objects and for easy processing of SOAP messages.

- Supports UPDATE, DELETE and INSERT operations on SQL and XML data.

- Capability to store and manage up to 2 billion XML nodes and SQL rows.

- Export /Import content and query results to/from XML string variables and disk files.

- Filter, manipulate and update SQL/XML data with an extreme easy to use API.

- Supports a significant subset of the SQL92 language and the current XQuery draft.

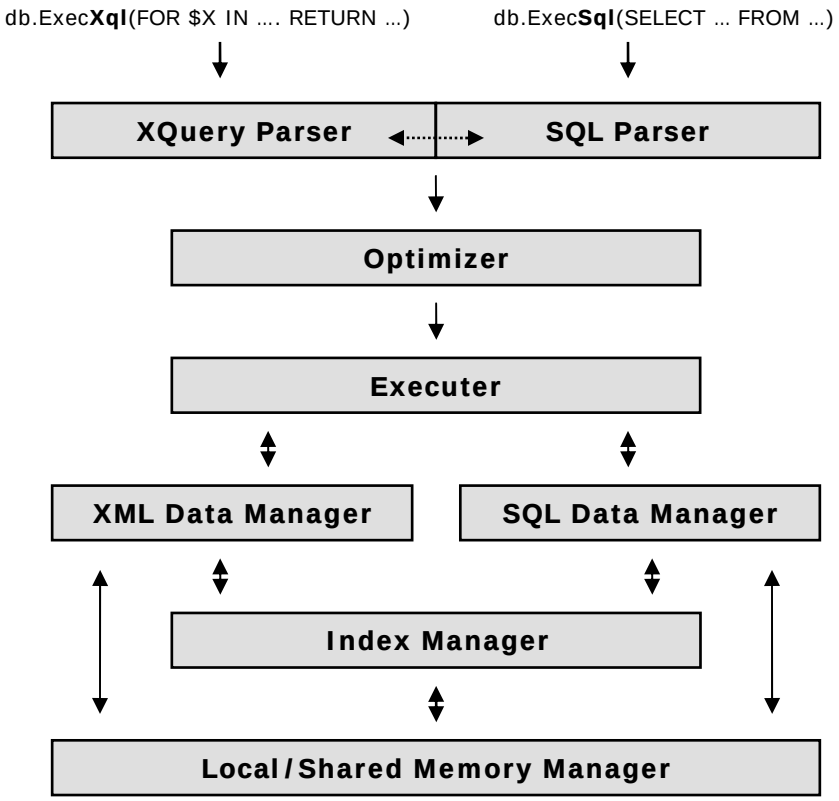
- Supports Prepared SQL statements and Transactions (Begin, Commit, Rollback).

- Special, memory optimized index structures for outstanding search performance.

- **Very small memory footprint**.

Database Architecture

SQL/XML-IMDB stores XML documents and relational data in their native structure, meaning that they remain in their natural form, either as complex, hierarchical XML objects (not broken down into tables and columns), or as collection of rows for relational data. When they are stored or accessed there is **no** conversion of the structures, resulting in unsurpassed store and retrieval speeds.

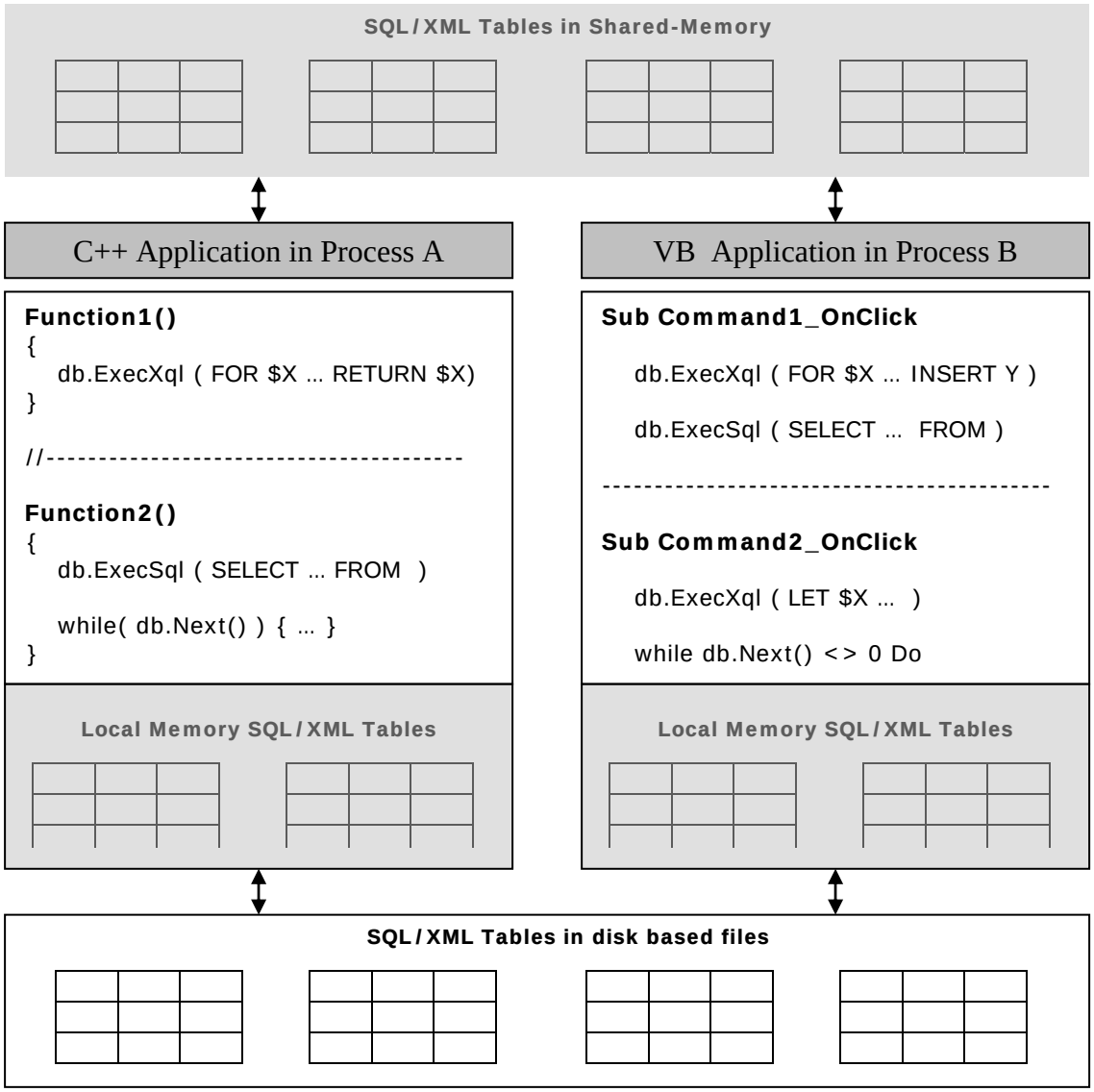


SQL/XML-IMDB's data storage includes compression technology to keep the memory requirement as low as possible. String data is always stored in variable length byte arrays and a bool data type requires only one bit. The optimizer dynamically re-optimizes plans during execution based on the actual size information of intermediate results. Queries can even be further accelerated by the use of prepared statements. Once the result set is ready, data values will be retrieved with speed comparable to linked list operations.

SQL/XML-IMDB uses a TST-tree as the main indexing algorithm. This algorithm combines the speed advantage of a hash table with the ordered access of a binary tree. Additional for XML data, the engine uses Reverse-Lookup indexing and a special Token-Segment-Build-Up indexing mechanism invented by QuiLogic for ultra fast loading and processing of XML files.

Principal Application Architecture using SQL/XML IMDB

SQL/XML-IMDB provides the flexibility to work in different application environments and to share data in an easy and seamless way between different processes. It is a **data management framework** with an extreme easy to use API for sharing and management of complex data structures, internal and across applications. The database is multi-user and multi-threading ready and reduces the implementation effort for custom applications to an absolute minimum and helps to produce scalable and well designed software.



An XML Data-Binding Facility

It would be much easier to write XML enabled programs if we could simply map the components of an XML document to simple in memory objects that represent, in an obvious and useful way, the XML document's intended meaning according to its schema.

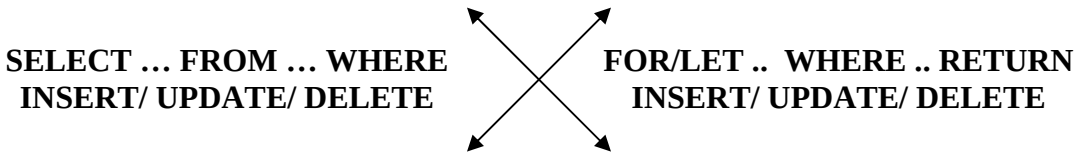
SQL/XML-IMDB supports this, through its cursor based data binding facility, which allows a developer to bind the returned XML query result to specific columns of a relational view. The application developer can then iterate over the set of bound objects, visiting the XML objects one by one. The cursor columns contain either simple typed values (int, float, string…) or xml trees of any complexity. Access the column values with Get/SetXXXValue() functions, walking the cursor set with one of the navigating functions Next(), Previous(), Last(), First().

SQL/XML-IMDB let you **create, insert, update, access, filter, transform** and **consume** xml data either trough declarative statements or by using read only and read/write cursor based access from within your application code with simple to use API calls.

It is possible to create XML views over relational data and to manipulate and transfer the data between SQL and XML tables.

Titel	Author	Year
TCP/IP …	Walter …	1994
Programming…	…	…
…		

```
<book year="1994">
    <title>TCP/IP Illustrated</title>
    <author>
      <last>Stevens</last>
      <first>Walter</first></author>
    <publisher>Addison-Wesley</publisher>
    <price> 65.95</price>
</book>
  . . . .
```



Cursor

→	TCP/IP Illustrated	Walter Stevens	1994	65.95
→	Programming for UNIX	Frank Novak	1992	57.30
	SQL Unleashed	Gordon Brovis	1990	32.00
	…			

Extreme Ease of Use

SQL and XQuery commands will be executed by calling two simple functions, providing a command string as argument. If the executed command returned result rows, a cursor is automatically opened and the data is accessed through cursor based API calls. By using either SQL or XQuery statements you declarative specify what to filter out from any XML structure or relational data table and access it by simple traversing the returned cursor row by row or node by node with almost no or little coding.

SQL	XQuery
db.**ExecSql**("SELECT ... FROM ... ")	db.**ExecXql**("FOR $X IN ... RETURN ...")

SQL and XQuery query results are retrieved through cursor based access:

```
while( db.Next() )
{
   ...
   ...
}
```

SQL/XML-IMDB provides a rich set of API functions:

- **Command execution** (ExecSql, ExecXql, Prepare…)
- **Data manipulation** (Update, Delete, Insert)
- **Cursor movement** (Next, Previous, Last, First, RowCount…)
- **Accessing data** (Get/SetIntVal, Get/SetCharVal, Get/SetBoolVal…)
- **Import/Export SQL+XML** (Export, ExportX, Import, ImportX, ExportToMemory, ImportFromMemory…)
- **Format handling** (SetOutputDateFormat, SetInputDateFormat, SetXmlFormat...)
- **Transaction support** (Commit, Rollback, Begin)

and more …

XML documents contain text and values. Sure, even the value content is in text form, but it can be interpreted as having a value of a certain type. For example, the string "07/12/1999" can be interpreted as being a sequence of characters (text) or being of type date having the value of a given date. To specify the data type of the return value use one of the data type modifiers added to the end of the bound variable separated by a "/".

- **…/text()** Text
- **…/number()** Integer
- **…/real()** Double
- **…/datetime()** Date/Time
- **…/date ()** Date
- **…/time()** Time
- **…/bool()** Bool
- **…/name()** Name of Element as text
- **…/node()** Unique id of xml node as integer
- **…/position()** Position of xml element as integer

If you omit the data type modifier, type text is assumed and the query returns both the element name **and** content of the parent and all contained child elements:

```
<author><first>Stanislav</first> … <author/>
```

For example, to return the text content of an element enclosed between the tag-name, use the following example expression:

```
… RETURN { $X/author/name() + ' ' + $X/author/text() + ' ' + $X/author/name() }
```

which returns a result string like: "author Stanislav Lem author"

Example1 (XQuery)

List book titles published by Addison-Wesley after 1993.

XQuery:

```
for $b in document('bib.xml')/bib/book where $b/@year > 1993 return $b/title/text()
```

Code example in C++

```
-----------------------

CIMDb db;
CString bookTitle;

db.Open();
db.ExecXql(xquery);
while( db.Next() ) { bookTitle = db.GetCharVal(1) }
```

Returns the list of book titles in variable bookTitle:

TCP/IP Illustrated
Data on the Web
….
….

Example2 (XQuery)

Up to 64 cursor columns can be bound to different XML result nodes/tuples by simple using a " , " in the RETURN statement. An ORDER BY can be used to order the result set on different columns in the result cursor.

List book titles published by Addison-Wesley after 1993. Order the result by year descending.

XQuery:

```
for $b in document('bib.xml')/bib/book where $b/@year > 1993
RETURN  $b/title, $b/@year/number() ORDER BY 2 DESC

-----------------------

CIMDb db;
CString bookTitle;
int nYear;

db.Open();
db.ExecXql(xquery);
while( db.Next() ) { bookTitle = db.GetCharVal(1); nYear = db.GetIntVal(2); }
```

Returns the list of book titles in variable bookTitle:

<title>Data on the Web</title>
<title>TCP/IP Illustrated</title>
….

and the publishing year in variable nYear as of type number:

2000
1994
…
…

Example3 (SQL)

Accessing a relational table:

Code example in C++

```
CIMDb db;
CString resultA;
int x;

db.Open();
db.ExecSql("SELECT * FROM A, B WHERE a.A = b.B");

while( db.Next() )
{
    result = db.GetCharVal(1);
    x = db.GetIntVal(2);
}

db.Close();
```

Working with SQL and XQuery together

SQL/XML-IMDB let you seamless work with both, relational and XML data at the same time. The engine uses separated tables for SQL and XML data but you can freely mix the calls to the SQL as well as the XQuery part of the engine at any time and any location in your application. To do so, use one of the two functions below:

- `ExecSql( sql statement )`
- `ExecXql( xquery statement )`

The result of the query is accessed in both cases by iterating over the automatically opened cursor. The values can be retrieved by using one of the `GetXXXVal` functions.

For XQuery, depending on the formulation of your `RETURN` clause, the returned data is either the element content value of the specified type (string, integer, double…) or an entire sub-tree of the XML tree as of type string containing element names and value.

…RETURN $X/author	…RETURN $X/author/last/text()
<author><first>Walter ….</last></author> <author><first>Frank ….</last></author> <author><first>Gordon ….</last></author> … …	Stevens Novak Provis … …

Each bound variable accessed in the `RETURN` part of the XQuery statement creates a new column in the resulting cursor when the variable (or expression) is separated by a "," . This is very similar to the SQL `SELECT` statement with one or more projection columns.

SELECT last ,	First ,	Birthday FROM …
FOR/LET … RETURN $X/last/text() ,	$X/first/text() ,	$X/birthday/date()
Stevens Novak …	Walter Frank …	1960-12-03 1965-02-14 …

To sort the result on a specific column, simply append an ORDER BY statement:

- `SELECT … FROM TR` **`ORDER BY`** `a, b`
- `FOR/LET … RETURN $X/..., $Y/...` **`ORDER BY`** `n1, n2`

For XQuery, the column to sort is specified with position numbers starting from 1…

Mixing XQuery with SQL statements

In an attempt to ease the management and access of relational data in xml based environments, QuilLogic has developed an extension to the XQuery draft specification which allows the use of SQL statements within any part of the XQuery statements where an expression is allowed too.

- `FOR/LET … IN ( SQL SELECT query ) …`
- `… WHERE $X/… = ( SQL SELECT query )`
- `… WHERE $X/… IN ( SQL SELECT query )`
- `… WHERE $X/… = All/Any/Some ( SQL SELECT query )`
- `… RETURN { (SQL INSERT/UPDATE/DELETE statement ) }, …`

Columns in the `WHERE` clause section of the SQL query can be compared against any bound variable from the earlier part of the XQuery query, making it effectively possible to use correlated values between XQuery and the SQL sub-queries.

Example:

```
… FROM TR WHERE TR.price <> $Y/[path]/number() AND …
```

Please note that the other way around, using XQuery expressions within SQL statements (`ExecSql`) is not supported!

When using a SQL statement within the `RETURN` part of XQuery it is possible to manipulate the content of relational tables by using SQL Insert/Update/Delete statements. Applying a SQL `SELECT` statement has no effect (although it is possible).

Bridging Relational (SQL) Technology and XML

One of the main features provided by SQL/XML-IMDB is the ability to create XML views of existing relational data. SQL/XML-IMDB does this by automatically mapping the data of the underlying relational database system to a low-level default XML view. Users can then create **application-specific XML views** on top of the default XML view. These application-specific views are created using XQuery. Moreover, users often need to synthesize and extract data from multiple relational and XML sources. SQL/XML-IMDB allows arbitrarily complex views and queries to be expressed, combining any number of sources, xml documents or relational data in one query.

Default XML view of relational data:

```
<row>
   <col1>xxx</col1>
   <col2>xxx</col2>
   <col3>xxx</col3>
</row>
...
```

In summary, you can:

- **Compose XML Views over Relational Data**

```
FOR $TR IN (SELECT a,b,c FROM A,B WHERE ....) RETURN <> xml </>
```

- **Mix XML Data Sources with Relational Data Sources**

```
FOR $TR IN (SELECT a,b,c FROM A,B WHERE ....)
  FOR $TX IN DOCUMENT('abc.xml')
    WHERE ...
    RETURN ...
```

- **Transfer Relational Data to XML Documents**

```
INSERT INTO TX (SELECT * FROM ... WHERE ...)   (TX as XML table)
```

- **Transfer XML Data to Relational Tables**

```
FOR $TX IN DOCUMENT('abc.xml')
  WHERE ...
  RETURN {(INSERT INTO TR VALUES($TX/title/text(), $TX/year/number()) }
```

- **Use Correlate Variables between XML and SQL Queries**

```
... WHERE $TR/[path]/number() = $TX/column/number() ...
```

Working with SQL and XML Tables together

SQL/XML-IMDB promotes a tight coupling of SQL and XML tables. Not only is the creation and loading of data into the memory tables extremely simple and similar for both data domains, as well as easy will be the exchange of information between both domains.

Creating tables

SQL	XML
db.ExecSql("CREATE TABLE TR(...)")	db.ExecXql("CREATE TABLE TX")

Use the keyword **SHARED** (*CREATE TABLE SHARED T)* to create tables in shared memory for sharing and exchanging data between different processes ! and application environments like C++, .NET, Perl, VBA…

Loading tables from file data:

LOAD 'abc.txt' INTO TR	LOAD 'abc.xml' INTO TX

Loading tables from other tables:

INSERT INTO TR SELECT ... FROM ..	INSERT (SQL/XML SUBSELECT) INTO TX

Saving table data into files:

SAVE TR INTO 'abc.txt'	SAVE TX INTO 'abc.xml'

Relational tables store there data content in row/column format, XML tables as tree.

Deleting table data:

DELETE FROM TR WHERE ...	DELETE FROM TX

Destroying tables:

DROP TABLE TR	DROP TABLE TX

Based on our special indexing technology, the loading and processing of XML as well as relational data is ultra fast and the response times for queries in general are far below compared to traditional file based database systems.

SQL Data Update

To insert/edit relational data you may either use declarative DML statements like `INSERT… / UPDATE… / DELETE…` executed by calling the function `ExecSql()` or use one of the update-cursor based API functions.

Example of DML:

```
db.ExecSql("UPDATE TR SET ab = 'Walter' WHERE isbn = '134-3447-838'")
```

For `SELECT` queries, the cursor is bi-directional and read only by default. If you need an update cursor, append the SQL command with a „**`FOR UPDATE`**“ clause. This opens the cursor in bi-directional read/write mode.

The data edit functions `Insert(), Update(), Delete()` affect the database row(s) at the current cursor position. The Insert function appends a new row on the table. Before inserting new data, provide the values for this new row with one of the `SetXXXVal` functions. The Delete function deletes the row at the current cursor position.

Example of API usage:

```
Db.ExecSql("SELECT first FROM TR FOR UPDATE")

While( db.Next() )
{
    db.SetCharVal("first", "xxxx")
    db.Update();
}
```

Both methods above for updating data may be enclosed by a transaction for safe rollback in case of update failure.

```
db.BeginTransaction()
```

Update data…
…
…

```
db.Commit()  or db.Rollback()
```

XML Data Update

W3C is considering letting XQuery go to recommendation status without UPDATE or DELETE semantics being a part of the recommendation. QuiLogic therefore has designed and implemented an XQuery extension based on the simple INSERT-UPDATE-DELETE semantics that is found by users of relational databases (SQL users). This extension allows the manipulation of XML data in a declarative and very easy to use style, comparable to that found in the SQL manipulation language.

Deleting Data:

For $X in TX where …. DELETE $X/[path] or …. DELETE $X/@attribut

Rename Nodes:

For $X in TX where …. RENAME $X/[path] TO 'NewName'

Update Node Values:

For $X in TX where …. REPLACE $X/[path]/text() WITH value

Update entire Nodes:

For $X in TX where …. REPLACE $X/[path] WITH <> … <>
For $X in TX where …. REPLACE $X/[path] WITH $Y
For $X in TX where …. REPLACE $X/[path] WITH (SQL/XML Query)
. . . .

Update Attribute:

For $X/@attr in TX where …. REPLACE $X WITH ATTRIBUT('abc',value)

Insert new Data:

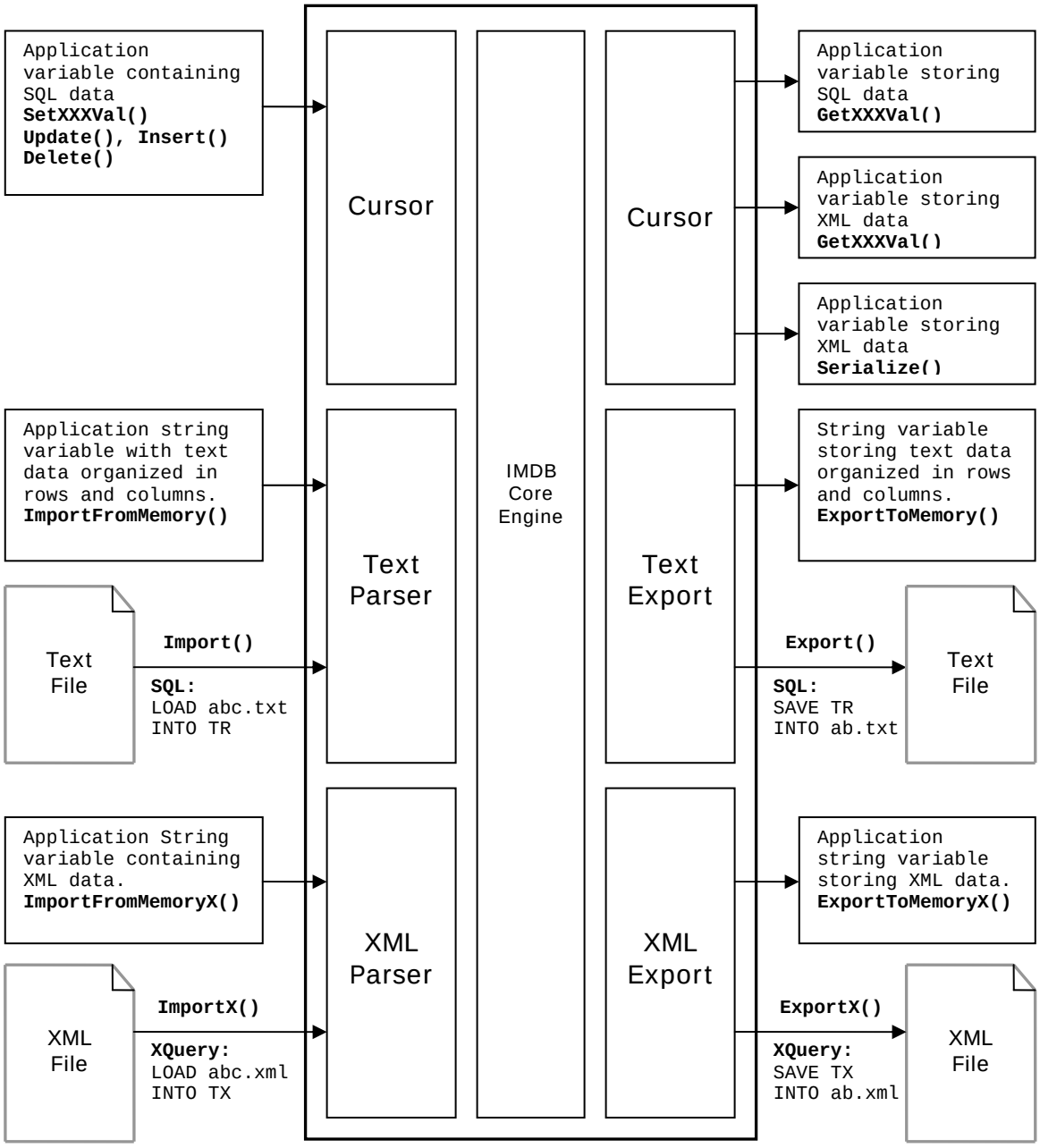
For $X in TX where …. INSERT <> … <> INTO [BEFORE] [AFTER] $X/[path]
For $X in TX where …. INSERT (SQL/XML Query) INTO [BEFORE] [AFTER] $X/[path]
. . . .

…where TX is an in-memory XML table.

Note: Transaction support is currently only available for SQL based data manipulation, for queries like INSERT UPDATE DELETE on relational tables.

Data Import/Export and Persistence

Given the rich set of API functions as well as SQL/XQuery declarative statements, SQL/XML-IMDB provides an enormous flexibility in making the in-memory data content persistent. The graphic below shows the various paths to import/ export data. Import/Export formats are customizable by setting the corresponding attributes.



XQuery Basics

The language is currently being developed by the W3C XML Query Working Group and has working draft status (as of Dec. 2002, see References for details). Even though the current language definition is quite huge based on functional principles and contains at least 7 types of expressions, there is a simple to understand core principle behind all the complexities. It is possible to write really simple constructs which, as you will see, satisfies all your needs for querying xml data.

The data model that XQuery uses is based on that of XPath (see References) and defines each XML document as a tree of nodes. Therefore XPath is heavily used in XQuery to select sub trees out of a larger xml tree just as it is used as the path selection language for XSLT.

The core of the language is based on the FLWR (pronounced "flower") expression, and is very similar to the **SELECT-FROM-WHERE** construction in SQL.

1.) A FLWR expression consists of:

- **FOR**-clause: binds one or more variables ($X…) to a sequence of nodes returned by another expression (usually a path expression, see below) and iterates over the nodes. The variable therefore represents an array of bound nodes.

- **LET**-clause: also binds one or more nodes but without iterating. A single sequence of nodes is therefore bound to the variable.

- **WHERE**-clause: contains one or more predicates that filters or limits the set of nodes as generated by the FOR/LET-clauses.

- **RETURN**-clause: generates the output of the FLWR expression. The RETURN-clause usually contains the references to variables and is executed once for each bound node-reference that was returned by the FOR/LET/WHERE-clauses.

2.) The second important construct are path expressions. As already noted, path expressions are based on the syntax of XPath, the XML standard for specifying "paths" in an XML document. For example:

Find all titles of chapters in document books.xml:

```
document("books.xml")//chapter/title
```

Find all books in document bib.xml published by Addison-Wesley after 1991:

```
document(bib.xml")//book[publisher = "WROX" AND @year > "1991"]
```

3.) Element Constructor.

This type of expression is used when a query needs to create new elements, which is typically found in the return part of queries.

```
… RETURN
 <book year={$X/date()}
      <title>{ $Y/title/text() }</title>
         <author>{$Z/last/text() + ‘ ’ +
      $Z/first/text() }</author>
   </book>
```

During query runtime the return clause iterates over the list of bound variables from the earlier part of the query, creating as many new <book> elements as referenced nodes found in the select/restrict part of the query .

The following example returns the title of all books published by Addison-Wesley:

```
FOR $X IN DISTINCT(document("bib.xml")/book/title)
FOR $Y IN document("bib.xml")/book[title = $X]
    WHERE $Y/publisher = ‘Addison-Wesley’
RETURN $X/text()
```

Although the XQuery draft specifies more constructs (conditional expressions, function expressions…) SQL/XML-IMDB queries are currently restricted to FLWR, Path and Element Constructor expressions, with some SQL stylish **enhancements** to aid in query formulation as described below:

WHERE clause

The WHERE clause accepts XPath expressions and additional predicates very similar to that one found in SQL queries:

Comparison operators (<, >, = …)
[NOT] LIKE ($X/author/text() like ‘Sta%’)
[NOT] BETWEEN ($X/price/number() between 100 and 200)
[NOT] IN ($X/@ISBN/text() IN(‘234-5657’,’234-478’,’463-45’)
Sub-Query ($X/price/number() = (SQL/XQuery query here…))

DISTINCT

Distinct serves the same purpose as found in SQL:

```
FOR $X IN DISTINCT(document('bib.xml')/book/title)…
```

Aggregate Functions

It is possible to create and retrieve query results based on numeric values build up from a set of elements. The aggregate functions can be used in any part of the query, that is; FOR/LET, WHERE, RETURN.

For example to query the price summary of all books published by Addison-Wesley use:

```
FOR $X IN DISTINCT(document('bib.xml')/book)
     WHERE $X/publisher = 'Addison-Wesley'
RETURN { SUM($X/price/real()) }
```

Available functions are:

- COUNT
- SUM
- AVG
- MAX
- MIN

Sub-queries

The purpose and structure of sub-queries is very similar to that found in SQL. Likewise it is possible to use correlated variables and combine it with the ALL, ANY, SOME and EXISTS conditions.

```
FOR $X In document('bib.xml')/book
     WHERE $X/@ISBN = (FOR $Y In FILE/review WHERE
     $X/title = $Y/title RETURN $Y/@ISBN)
Return $X/title/text()
```

Note that for sub queries it is possible to use either XQuery or SQL queries:

```
FOR $X In document('bib.xml')/book
     WHERE $X/@ISBN = (SELECT TR.isbn FROM TR WHERE
     $X/title = TR.title)
Return $X/title/text()
```

SQL Language

SQL/XML-IMDB supports a significant subset of the SQL92 definition of the SQL database language, including support of the following:

SELECT, UPDATE, INSERT, DELETE
CREATE TABLE...[PRIMARY KEY...]
DROP [TABLE]
SELECT qualifiers: DISTINCT, TOP n
SELECT clauses FROM, WHERE, GROUP BY, HAVING, ORDER BY
WHERE expressions: AND, OR, NOT, LIKE, BETWEEN, + - * /, IS [NOT] NULL, <, >, =, < >, <=, >=, Constants, Parameters, Column Names
SELECT list expressions: MAX, MIN, AVG, SUM, COUNT, +, -, *, /, Constants, Parameters, Column Names
Value list qualifiers: ANY, ALL, SOME, IN
UPDATE expressions: +, -, *, /, Constants, Parameters, Column Names
INSERT values expressions: Constants, Parameters
INSERT ... SELECT
Sub-queries within SELECT statements

Supported Data Types

Numeric	4 bytes – 2,147,483,648 to 2,147,483,647 INT, INTEGER, SHORT, LONG, SMALLINT
Decimal	8 bytes (See DOUBLE) REAL, FLOAT, DOUBLE, SINGLE, CURRENCY
Bool	1 bit BOOL, BOOLEAN, YESNO
Counter	4 bytes Auto-Increment value 1 to 2,147,483,647 COUNTER
Character	1 byte per character (2 bytes for UNICODE) Zero to a maximum of 256 MB/row CHAR(n), VARCHAR(n), CHARACTER(n), TEXT
Date/Time	8 bytes 0 to year 20.000, Time 00:00:00.000.000.0 to 24:59:59.999.999.999 DATE, TIME, DATETIME
Binary	Zero to a maximum of 256 MB/row BLOB, LONGBINARY
GUID	Example: 45807EF7-5D36-48CF-BCFE-596E15399DA7

System Defaults and Limits

Maximum data store size	~ 2GB
Maximum number of tables	65,535
Maximum number of columns on table	256
Maximum number of indexes on table	32
Maximum length of table names	64
Maximum length of column names	64
Maximum number of rows in a table	2 Billion
Maximum length for fixed-length column	4096
Maximum length for variable-length column	256 MB
Maximum size for binary columns	256 MB
Maximum number of simultaneously open cursors	Unlimited (memory depending)
Maximum number of columns in an index	1
Maximum number of XML nodes in a table	2 Billion
Maximum of simultaneously active IMDB objects	Unlimited (memory depending)

Supported Environments

- Microsoft .NET
- Microsoft Visual C++ 6, VC2005, VC2008
- Microsoft Visual Basic 6.0, VB-NET
- Microsoft C#
- Microsoft Office 97/2000/XP,2008
- Microsoft IIS/ASP
- Borland C++ 5.0/6.0
- Borland Delphi 5.0/6.0/7.0
- Automation Controller Environments (VBA)
- PHP
- Perl

SQL/XML-IMDB is Linux ready by the end of 2009 ! (both 32 and 64 Bit)

A CE-NET version is available (supporting the Compact Framework) on request.

Availability

SQL/XML-IMDB is available as:

- 1 Developer license.
- 4 Developer license.
- Enterprise edition for an arbitrary number of developers.
- Enterprise edition including full source code in C++.

To order please visit www.quilogic.com

QuiLogic, Inc. is an IT company headquartered in central EU, Austria. Founded in 1995, today QuiLogic creates innovative products and offers exceptional expertise in all sort of data management projects, high performance demanding applications and xml based solutions.

References

XQuery:
www.w3c.org/xml/query.html

XPath:
www.w3c.org/TR/xpath

SQL/XML-IMDB:
www.quilogic.com/whitep.pdf