XQuark Bridge 1.1

Api Tutorial

XQUARK BRIDGE 1.1

API TUTORIAL

Document version 1.0

Table of contents

Part

Abstract

This document is an introduction to the XQuark Bridge API and to XML/DBC, the XQuark XQuery API.

Part

2

Introduction

Overview

This tutorial describes in detail a single complete example of the use of XQuark Bridge. The intended audience is a programmer that is learning the API for the product. A simple auction provides a scenario that consists of users, items and bids of users on items. The product provides a powerful mechanism to map XML data into a relational database and extract XML from a relational database.

As the tutorial shows, only a few lines of code are required to perform these operations where as a "hard-coded" implementation would require hundreds of lines of code. In addition, since XQuark Bridge provides a mapping file, modifications to the schema are now independent of modifications to the XML. The remainder of this tutorial describes the XML data, the relational schema, the mapping file, a view file and example code for insertion of documents and generation of XML from the schema.

Part

3

XML Data

The current state of the auction is expressed as an XML file. The file consists of five pieces: the header, the list of users, the list of items, the bids on items by users, and the close of the record. For simplicity we do not show all the items, users and bids that are subsequently used in this tutorial.

The heading announces the XML version, the character set encoding, the namespace used, and the location of the schema. The parser automatically uses the schema location to parse the XML. For simplicity we skip the presentation of the schema.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<AUCTION
  xmlns="http://www.xquark.org/Auction"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

xsi:schemaLocation="http://www.xquark.org/Auction
                      auction.xsd">
```

A unique identifier, a string name and a rating (a measure of the trust that can be placed in this user to deliver on a promised bid) describe a user.

```
    <USERS>
        <USERID>U01</USERID>
        <NAME>Tom Jones</NAME>
        <RATING>B</RATING>
    </USERS>
    <USERS>
        <USERID>U02</USERID>
        <NAME>Mary Doe</NAME>
        <RATING>A</RATING>
    </USERS>
```

Items are a little more complex. A unique identifier, a description, the user identifier of the user that offered the item for auction, the start and end date of the auction and the reserved price (the minimum price that wins at the auction) describe an item.

```
    <ITEMS>
        <ITEMNO>1001</ITEMNO>
        <DESCRIPTION>Red Bicycle</DESCRIPTION>
        <OFFERED_BY>U01</OFFERED_BY>
        <START_DATE>
          2002-03-05T00:00:00
```

```
        </START_DATE>
        <END_DATE>2002-03-20T00:00:00</END_DATE>
        <RESERVE_PRICE>40</RESERVE_PRICE>
    </ITEMS>
    <ITEMS>
        <ITEMNO>1002</ITEMNO>
        <DESCRIPTION>Motorcycle</DESCRIPTION>
        <OFFERED_BY>U02</OFFERED_BY>
        <START_DATE>
          2002-04-11T00:00:00
        </START_DATE>
        <END_DATE>2002-05-15T00:00:00</END_DATE>
        <RESERVE_PRICE>500</RESERVE_PRICE>
    </ITEMS>
```

Each user makes a bid on a particular item at a particular time. For simplicity we simply consider bids in terms of units without any cash equivalent.

```
    <BIDS>
        <USERID>U01</USERID>
        <ITEMNO>1002</ITEMNO>
        <BID>400</BID>
        <BID_DATE>2002-04-14T00:00:00</BID_DATE>
    </BIDS>
    <BIDS>
        <USERID>U02</USERID>
        <ITEMNO>1001</ITEMNO>
        <BID>35</BID>
        <BID_DATE>2002-03-07T00:00:00</BID_DATE>
    </BIDS>
    <BIDS>
        <USERID>U02</USERID>
        <ITEMNO>1001</ITEMNO>
        <BID>45</BID>
        <BID_DATE>2002-03-11T00:00:00</BID_DATE>
    </BIDS>
```

Finally, providing the matching element to the opening element closes the XML document.

```
</AUCTION>
```

Part

4

Relational Schema

In order to map an XML document into a relational schema, we first need a target schema. In this tutorial we describe a simple schema that permits an almost one-to-one mapping between the XML and the schema, as described in the next section.

The schema consists of three tables, a USER table, an ITEMS table and a BID table. The SQL to generate these tables is straightforward.

```
DROP TABLE BIDS;
DROP TABLE ITEMS;
DROP TABLE USERS;

CREATE TABLE USERS (
 USERID         CHAR(3) PRIMARY KEY,
 NAME           VARCHAR(20) UNIQUE,
 RATING         CHAR(1)
);
CREATE TABLE ITEMS (
 ITEMNO         CHAR(4) PRIMARY KEY,
 DESCRIPTION    VARCHAR(30),
 OFFERED_BY     CHAR(3) NOT NULL,
 START_DATE     DATE,
 END_DATE       DATE,
 RESERVE_PRICE  NUMBER(10)
);
CREATE TABLE BIDS (
 USERID         CHAR(3) NOT NULL,
 ITEMNO         CHAR(4) NOT NULL,
 BID            NUMBER(10) NOT NULL,
 BID_DATE       DATE
);
```

Part

5

Mapping

Next, the XML document is mapped to the schema. In this case, an almost one-to-one mapping is provided. The mapping consists of the heading, the mapping for the entire auction record, the mapping for each child element of the auction record, and the closing of the mapping.

Since the mapping is an XML document itself, it has a typical heading that declares the XML version. Then the namespaces and XML schema location are declared. The mapping loader uses this information to locate the XML schema to interpret the mapping.

```
<?xml version="1.0"?>
<xrm:mapping
  xmlns:xrm=
    "http://www.xquark.org/Bridge/1.0/Mapping"
  xmlns:a="http://www.xquark.org/Auction"
  schemaLocation=
    "http://www.xquark.org/Auction auction.xsd">
```

Next the mapping declares the mapping of the AUCTION element of the XML document. The mapping of the auction element consists of the mapping of its child elements described below.

```
  <xrm:element name="a:AUCTION">
```

The USERS child element of AUCTION is mapped to the USERS table in the schema. In addition, the USERID child element is mapped to the USERID column of the USERS table; similarly for the NAME and RATING child elements. This key “twist” is the extraction of the inner most XML elements for each type of data and mapping them to columns in the relational table.

```
    <xrm:element name="a:USERS">
      <xrm:map table="USERS">
         <xrm:element name="a:USERID"
                      column="USERS.USERID"/>
         <xrm:element name="a:NAME"
                      column="USERS.NAME"/>
         <xrm:element name="a:RATING"
                      column="USERS.RATING"/>
      </xrm:map>
    </xrm:element>
```

The same key twist is used for the ITEMS and BIDS elements, mapping them to the corresponding relational tables.

```
    <xrm:element name="a:ITEMS">
      <xrm:map table="ITEMS">
         <xrm:element name="a:ITEMNO"
            column="ITEMS.ITEMNO"/>
         <xrm:element name="a:DESCRIPTION"
            column="ITEMS.DESCRIPTION"/>
         <xrm:element name="a:OFFERED_BY"
            column="ITEMS.OFFERED_BY"/>
         <xrm:element name="a:START_DATE"
            column="ITEMS.START_DATE"/>
         <xrm:element name="a:END_DATE"
            column="ITEMS.END_DATE"/>
         <xrm:element name="a:RESERVE_PRICE"
            column="ITEMS.RESERVE_PRICE"/>
      </xrm:map>
    </xrm:element>
    <xrm:element name="a:BIDS">
      <xrm:map table="BIDS">
         <xrm:element name="a:USERID"
            column="BIDS.USERID"/>
         <xrm:element name="a:ITEMNO"
            column="BIDS.ITEMNO"/>
         <xrm:element name="a:BID"
            column="BIDS.BID"/>
         <xrm:element name="a:BID_DATE"
            column="BIDS.BID_DATE"/>
      </xrm:map>
    </xrm:element>
  </xrm:element>
</xrm:mapping>
```

Part 6

Insertion of Documents

Now that the set-up is complete, insertion of a document is straightforward with XQuark Bridge. In this section, code fragments will be described, but the complete code for this example is available in Appendix A. (Note that in the appendix, only 8 lines of code are critical, the remaining code is declarations, set-up and error handling.) The coding required to do insertion consists of three parts, the initialization of an XQBridge object, the construction of a Mapper and the mapping of a document.

XQBridge setup uses JDBC connectivity to access a database.

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection jdbc = DriverManager.getConnection(
    "jdbc:oracle:thin:@localhost:1521:oracle",
    "demo","demo");
XQBridge bridge = new XQBridge(jdbc);
```

The `bridge` XQBridge object provides the interface to the XQuark Bridge product. Next a Mapper is constructed. This object permits the caching of the initialization of mapping, giving very high performance for multiple insertions of documents. Loading a mapping and then requesting a mapper constructs a Mapper object.

```
InputSource map = new InputSource("auction.map");
Mapping mapping = bridge.getMapping(map);
Mapper mapper = mapping.getMapper();
```

Finally, the insertion of a document is accomplished with a single line of code.

```
mapper.insertDocument(new
InputSource("auction.xml"));
```

The result of this insertion is a batch transaction on the relational database. The transaction consists of the table rows (tuples) constructed from the mapping file.

XML Views using XQuery

XQuark Bridge provides an implementation of XQuery. The schema of the relational database defines the data model for XQuery expressions. Each relation is defined as a collection. Each row is an element and each value is a leaf of the element. For example, a row of the ITEMS relation as follows

USERID	NAME	RATING
U01	Tom Jones	B

Corresponds to the "virtual" XML fragment

```
…
  <ITEMS>
    <USERID>U01</USERID>
    <NAME>Tom Jones</NAME>
    <RATING>B</RATING>
..</ITEMS>
…
```

Note that this XML fragment is *not* the same as the input XML document. In fact, the relationship between the two is defined purely by the mapping document. The XQuery data model does *not* have any knowledge of the mapping. In particular, the AUCTION element no longer exists and is not recorded at all in the relational database.

For many applications, these data model semantics are required. For other applications, the data model of the *documents loaded* is required as the data model for XQuery expressions.

XQuark Bridge also offers the ability to define views over the XQuery data model. Views are implemented with functions in XQuery. They allow the user to define virtual collections of documents that can be further queried with XQuery, while hiding the flat data model of relational tables. Complex results can thus be obtained with simple queries. Functions can be defined in files, called *modules*. We will do so in our example. Just suppose that our view is defined in a module file (`view.mod`):

```
module namespace ns =
"http://www.xquark.org/Bridge/API/Tutorial";

declare function ns:itemSummary(){

(: For all items, list the item number,
   description, and highest bid
   (if any), ordered by item number. :)
for $i in collection("ITEMS")/ITEMS
let $b:=collection("BIDS")/BIDS[ITEMNO=$i/ITEMNO]
order by $i/ITEMNO
return
  <item>
    { $i/ITEMNO }
    { $i/DESCRIPTION }
    <high_bid>{ max($b/BID) }</high_bid>
  </item>
};
```

Here we show a relatively simple XQuery expression that computes the highest bid for every item. (Some formatting has been added for legibility.) The XQuery uses a FLWR expression and computes an aggregate for each item in the collection. The aggregate is reported in the result expression.

8

Constructing XML Documents via a Query

Appendix B and C list a complete Java program that queries the database using XQuery. To issue an XQuery, five steps are involved. The first step connects to the database and sets up the XQBridge object. The second step obtains a statement object. The third step sets a base URI for locating the imported modules or schemas. The fourth step issues a query and the fifth iterates over the result. The first step, connection to the database, is identical to the mapping example, and we skip it.

The second step obtains a connect object in a manner similar to JDBC.

```
XMLConnection xc = bridge.getXMLConnection();
XMLStatement xs = xc.createStatement();
```

The third step sets the base URI, that will be used to resolve relative URI that may be specified to locate imported modules or schemas.

```
xs.setBaseURI("file://myComputer/myDir");
```

The fourth step issues an XQuery. Here we show a very simple XQuery expression since most of the job is performed by the previously defined view. We just retrieve summaries for bicycles. Just notice how the view is referenced in the main query. The view file will be retrieved at the following URI: `file://myComputer/myDir/view.mod`. The boolean `result` is true if the query generated a result.

```
boolean result = xs.execute(
" import module namespace isn =
\"http://www.xquark.org/Bridge/API/Tutorial\" at
\"view.mod\";
<result>
  {
    for $i in isn:itemSummary()
    where contains($i/DESCRIPTION, \"Bicycle\")
    return $i
  }
</result> ");
```

Finally, iteration over the result set of the query produces the XML result generated by the XQuery. In this example, there is only a single result, as the query starts with an enclosing `result` tag.

```
if (result) {
  XMLResultSet xrs = xs.getResultSet();
  while (xrs.hasNext()) {
    System.out.println(xrs.nextAsString());
  }
}
```

The XML document produced by this query for the entire data set is the following:

```
<result>
    <item>
        <ITEMNO>1001</ITEMNO>
        <DESCRIPTION>Red Bicycle</DESCRIPTION>
        <high_bid>55</high_bid>
    </item>
    <item>
        <ITEMNO>1003</ITEMNO>
        <DESCRIPTION>Old Bicycle</DESCRIPTION>
        <high_bid>20</high_bid>
    </item>
    <item>
        <ITEMNO>1007</ITEMNO>
        <DESCRIPTION>Racing Bicycle</DESCRIPTION>
        <high_bid>225</high_bid>
    </item>
    <item>
        <ITEMNO>1008</ITEMNO>
        <DESCRIPTION>Broken Bicycle</DESCRIPTION>
        <high_bid/>
    </item>
</result>
```

Appendix A - Example 1

```
import org.xquark.xml.xdbc.*;
import org.xquark.bridge.*;
import oracle.jdbc.driver.*;
import java.sql.*;
import org.xml.sax.*;
import org.w3c.dom.*;

public class Four {
  public static void main(String args[]) {
    try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      Connection jdbc = DriverManager.getConnection(
        "jdbc:oracle:thin:@localhost:1521:oracle","demo","demo");
      XQBridge bridge = new XQBridge(jdbc);

      InputSource source = new InputSource("auction.map");
      Mapping mapping = bridge.getMapping(source);
      Mapper mapper = mapping.getMapper();
      mapper.insertDocument(new InputSource("auction.xml"));
    } catch (Exception e) {
      e.printStackTrace();
    }
    System.out.println("done");
  }
}
```

Appendix B - Example 2

```
import org.xquark.xml.xdbc.*;
import org.xquark.bridge.*;
import java.sql.*;

public class Eight {
  public static void main(String args[]) {
    try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      Connection jdbc = DriverManager.getConnection(
        "jdbc:oracle:thin:@localhost:1521:oracle","demo","demo");
      XQBridge bridge = new XQBridge(jdbc);

      XMLConnection xc = bridge.getXMLConnection();
      XMLStatement xs = xc.createStatement();
      xs.setBaseURI("file://myComputer/myDir");
      boolean result = xs.execute(
      " import module namespace isn =
\"http://www.xquark.org/Bridge/API/Tutorial\" at \"view.mod\";
      <result>
      {
       for $i in isn:itemSummary()
       where contains($i/DESCRIPTION, \"Bicycle\")
       return $i
      }
      </result> ");
      if (result) {
        XMLResultSet xrs = xs.getResultSet();
        while (xrs.hasNext()) {
          System.out.println(xrs.nextAsString());
        }
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
    System.out.println("done");
  }
}
```

Appendix C - Example 2 (Alternate connection method using pure XML/DBC)

Note that this method is lighter for the connection process since JDBC is totally encapsulated. But it can only be used for extraction.

```
import org.xquark.xml.xdbc.*;

public class Eight {
  public static void main(String args[]) {
    try {
      Class.forName("org.xquark.extractor.ExtractorDriver");
      XMLConnection xc =
XMLDriverManager.getConnection("xdbc:xquark:extractor:jdbc:oracle:
thin:@localhost:1521:oracle","demo","demo");
      XMLStatement xs = xc.createStatement();
      xs.setBaseURI("file://myComputer/myDir");
      XMLResultSet xrs = xs.executeQuery(
      " import module namespace isn =
\"http://www.xquark.org/Bridge/API/Tutorial\" at \"view.mod\";
      <result>
      {
       for $i in isn:itemSummary()
       where contains($i/DESCRIPTION, \"Bicycle\")
       return $i
      }
      </result> ");
      while (xrs.hasNext()) {
        System.out.println(xrs.nextAsString());
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
    System.out.println("done");
  }
}
```