

DataDirect XQuery™ 2.0 and Oracle 10g Release 2 Compared

January, 2007

Table of Contents

Table of Contents.....	2
Introduction	3
Product Overview.....	3
DataDirect XQuery.....	4
Oracle 10g Release 2	4
Introduction	4
Oracle XML Developer's Kit.....	5
Oracle XML DB	5
Comparing DataDirect XQuery and Oracle XQuery	7
Data Source Support	7
Using XML Files or Streams with Oracle XQuery	7
Using XML Files or Streams with DataDirect XQuery	9
Querying Traditional Relational Data with Oracle XQuery ...	9
Querying Traditional Relational Data with	
DataDirect XQuery.....	10
Using XMLType Columns with Oracle XQuery.....	11
Using XMLType Columns with DataDirect XQuery	12
Conclusion: Data Source Support Compared	13
Language support.....	13
Oracle XQuery	13
DataDirect XQuery.....	14
Oracle and DataDirect Compared	14
Conclusion: Language Support	16
Proprietary extensions	16
Oracle XQuery	16
DataDirect XQuery.....	17
Application integration	17
Oracle XQuery	18
DataDirect XQuery.....	20
Conclusion: Application Integration	21

Introduction

Both DataDirect and Oracle offer a solution for users who want to leverage XQuery's rich feature set to speed up the development time of applications that integrate, query, and transform traditional relational data and XML information.

To realize the benefits of DataDirect XQuery, you must understand the differences between DataDirect XQuery and Oracle's implementation.

This paper addresses the following questions:

What are the main characteristics of the products?

What types of data can be queried and how is this accomplished?

How well do the products comply with the XQuery standard?

What extra features are available?

How does one integrate the XQuery functionality into an application environment (the APIs)?

These questions are addressed by systematically analyzing the relevant standard specifications and product documentation, and by executing small isolated tests. Additionally, it is recommended you evaluate the products in your own environment.

The comparison is based on DataDirect XQuery 2.0 and Oracle 10g Release 2 (R2).

Product Overview

To set the stage for the comparison of DataDirect XQuery and Oracle XQuery, the relevant product features are highlighted.

Detailed DataDirect XQuery information is available from the [documentation](#) and other sources on our website, XQuery.com. Detailed Oracle information is available in the [Oracle documentation](#) and accompanying [technical documents](#).

DataDirect XQuery

[DataDirect XQuery](#) is a Java implementation of XQuery that does not require any other product or application server, and has no server of its own. It can be used to transparently query and integrate XML information originating from different sources including XML files or streams (for example, through an HTTP connection with a web (service) client or server). Furthermore, through tight integration with the [DataDirect XML Converters](#), a wide range of text and binary non-XML file formats are supported (for example, csv and EDI). Finally, DataDirect XQuery has strong support for data stored in [relational databases](#), currently supporting different versions of Oracle, Sybase, Microsoft SQL Server and DB2.

Besides the pure W3C XQuery features, DataDirect XQuery comes with a number of useful extensions that support the invocation of Java or RDBMS external functions, including RDBMS functions that query natively stored XML information.

To support the data sources listed above, DataDirect XQuery uses standard [W3C XQuery syntax](#). The doc function is used to access XML files or streams and the data sources supported through the [DataDirect XML Converters](#). The collection function is used to access relational data.

Oracle 10g Release 2

Introduction

Oracle started out as a relational database system, and more than 20 years after its first appearance on the market, the main product focus is still the typical features of a high-end relational database management system (RDBMS). Nevertheless, to cope with market trends, a number of non-relational features have been added to the system. Oracle 8 (1997) introduced object relational features and four years later Oracle 9i added XML storage and limited XPath 1.0 query facilities to the list of supported technologies. The latest version, Oracle 10g R2 (2005), introduces XQuery support.

The Oracle XML features are roughly split into two related and integrated groups of tools: the XML Developer's Kit (XDK) and Oracle XML DB, both of which are discussed in more detail below.

There is also a third group of pre-release XML tools. These are available from the [Oracle Technology Network](#) web site and include a StAX (pull) XML parser.

Finally, there is also a component called [Oracle XML Query services](#) that started out as a pre-release XML tool but was upgraded and is now an integral part of Oracle Application Server release 10g Release 3. It is positioned as an XML data integration tool with XQuery support.

Oracle XML Developer's Kit

The XML Developers Kit (XDK) is documented in "[Oracle® XML Developer's Kit Programmer's Guide](#)". It contains a number of XML technologies that can be used outside the database engine.

The group includes a DOM and SAX XML parser, an XSLT engine, an XML Schema processor, a JAXB implementation, and more. The complete list can be found in the Oracle XDK guide.

These tools are not the focus of this document. Except for a number of Oracle legacy tools, equivalent functionality can be found in the Java JDK or in one or more mature open source products (from example, from the Apache Software Foundation).

DataDirect XQuery plays no role in this area.

Oracle XML DB

The Oracle XML DB features are integrated in the database engine and handle XML storage and its XML query and update capabilities. These features are described in detail in the "[Oracle® XML DB Developer's Guide](#)".

To provide context for the product comparison, here is an overview of the relevant XML DB features.

Storing XML

Different options are available to store XML information in the Oracle database:

XMLType data type

- Oracle's XMLType data type is used to store XML information in a database table or column.
- The XMLType can be associated with one or more columns in a table, or alternatively, a complete table can be defined as an XMLType table. In this case, the table has one pseudo column containing the XML data.
- XML storage mode:
 - When an XML Schema is available (stored in the Oracle XML DB repository) and the schema is associated with the XMLType, Oracle uses a structured storage approach. In this case, inserted XML information is shredded in its composing parts and stored in traditional object-relational columns. The user has flexibility in how the shredding algorithm creates the underlying tables and columns.
 - When no XML Schema is associated with the XMLType, Oracle stores the XML as a CLOB (unstructured).
 - The Oracle documentation describes the advantages and disadvantages of each of these storage modes in detail. Here is

a quick summary of the information. The structured approach is (1) somewhat slower when creating XML data, (2) less flexible in what XML it accepts (being XML Schema bound), and (3) opens up more powerful optimizations and data validations. The unstructured approach excels in flexibility but not in query or update performance.

XML DB Repository. Besides the traditional table/column mechanism of storing information in a RDBMS, Oracle XML DB offers a folder/resource (directory/file) based view. Once information, including XML files, is stored in the Oracle XML DB repository, it can be accessed using an URI scheme. When the stored information is XML, the URI can contain XPath 1.0 expressions that select part of the stored XML data.

Updating XML

Oracle supports updating XML data using a number of proprietary SQL functions:

updateXML replaces XML fragments.

insertChildXML inserts XML nodes as children of a given node.

insertXMLbefore adds XML nodes before a given node.

appendChildXML adds XML nodes as the last child nodes of a given node.

deleteXML deletes nodes.

Creating and Querying XML

The number of Oracle tools and features that allow querying and repurposing of relational and XML information is extensive. The many tools and features are the result of Oracle's long history in a fast changing XML world, and its good habit of offering backward compatibility. The overlap between the different features is substantial and often confusing. Starting with Oracle 10g R2 the functionality offered by the SQL/XML and XQuery standards (both partially supported) should be sufficient to cover most needs. There might be scenarios where the legacy tools offer functional or performance benefits, but that situation is sure to disappear as Oracle's implementation of the standards matures.

The SQL/XML publishing constructs (for example, XMLType) create XMLType values from traditional relational data. The resulting values can be queried with Oracle XQuery. Furthermore, Oracle introduces a proprietary XQuery function, ora:view(), that creates a default SQL/XML view of relational data without going through the SQL/XML publishing features. This function can be used inside XQuery expressions to access data stored in tables or views.

Oracle supports XQuery features through the SQL/XML XMLQuery and XMLTable Select statement clauses. These constructs allow integration of XQuery in the Select statement syntax. It is an awkward but necessary approach to bridge the traditional SQL environment with the new XQuery features.

Comparing DataDirect XQuery and Oracle XQuery

Based on the previous background information, the following product features are compared in more detail:

Data source support—what types of data can be queried and how easy is it to use these data sources in a *typical* query

Language support—breadth of XQuery support of the two implementations

Proprietary extensions

Application integration

Data Source Support

Using XML Files or Streams with Oracle XQuery

Oracle XQuery operates on information that is available in the Oracle database server. This implies that if an XML file or stream needs to be queried or has to be joined with data stored in Oracle, first the XML information has to be transferred to the Oracle database server.

The following three options are available to transfer the XML information:

Load the XML file in the Oracle XDB repository.

Insert the XML file in an Oracle XMLType column.

Use a SQL bind parameter in a SQL statement that invokes XQuery on the bind marker value.

The following procedure explains how to load the XML file into the Oracle XDB repository.

1. Create an Oracle directory if one is not available:

```
create directory ddxqxmdir as 'c:\temp\oraxmlfiles';
```

2. Make sure the XML file is in that physical directory, and then, add the XML file to the Oracle DB repository. Using PL/SQL this can be achieved using the following code:

```
declare
  xbfiler BFILE;
  ret      BOOLEAN;
BEGIN
  xbfiler := bfilename('DDXQXMDIR', 't1.xml');
  ret := DBMS_XDB.createResource
        ('/public/t1.xml', xbfiler,0);
END;
```

3. Once the XML file is loaded into the Oracle XDB repository, it can be accessed from within Oracle XQuery using the doc function, for example:

```
select * from xmltable('
  for $id in ("elem1","elem2","elem3")
  return
    doc("/public/t1.xml")//*[id=$id]
')
```

The following procedure explains how to insert the XML file in an Oracle XMLType column:

1. The XML file can be inserted into a table with an XMLType column:

```
create table xmlfile(filename varchar2(255),
  xml xmltype)
```

and insert the xml file information into a row in the table:

```
insert into xmlfile values (
  't1.xml',
  XMLType(bfilename('DDXQXMDIR', 't1.xml'),0))
```

2. Now, the information from the XML file is available in the xml column in the xmlfile table.

```
select xmlquery('
  for $id in ("elem1","elem2","elem3")
  return
    /*[id=$id]
  ' passing xml returning content) from xmlfile
```

Using JDBC, XML information can be queried without storing it in the database by using a bind marker whose value is used in an XQuery expression. For example:

```
// Prepare JDBC statement
String statement = "select xmlquery('//a' passing ? " +
    + "returning content ) from dual";
PreparedStatement ps = con.prepareStatement(statement);
// Parse XML from file using DOM parser
DocumentBuilderFactory dbf=DocumentBuilderFactory.newInstance();
DocumentBuilder db=dbf.newDocumentBuilder();
Document doc=db.parse(new File("c:/temp/t.xml"));
// Create Oracle XMLType object and bind it to the statement
XMLType xmlIn=new XMLType(con, doc);
ps.setObject(1, xmlIn);
// Execute query and get result
ResultSet rs = ps.executeQuery();
rs.next();
Object o = rs.getObject(1);
XMLType xml = (XMLType) o;
```

Using XML Files or Streams with DataDirect XQuery

Using XML files with DataDirect XQuery is simple. The XQuery fn:doc function is used for this purpose.

```
for $id in ("elem1","elem2","elem3")
return
  doc("file:///c:/temp/oraxmlfiles/t1.xml")//*[@id=$id]}
```

In addition to the file: scheme, DataDirect XQuery natively supports ftp: and http: schemes. Furthermore, the user can implement custom URI resolvers in Java or use XQuery external variables. The net result is that any XML information for which the user has the required access privileges can be queried using DataDirect XQuery—it does not matter if it is stored or created dynamically on the local machine, on the company network, or on the internet.

Furthermore, DataDirect XQuery has full support for XQuery external variables. These can be used, similar to Oracle bind variables, to pass XML information from the client application to DataDirect XQuery. (Examples are shipped with the product and documented in the ["Examples" appendix](#) of the [DataDirect XQuery User's Guide](#).)

Querying Traditional Relational Data with Oracle XQuery

Information that is stored in traditional (non-XMLType) columns first needs to be converted to XML before it can be handled by Oracle XQuery. The conversion, typically, happens by creating an implicit or explicit virtual XML view of the relational information.

Two Oracle approaches are available. The first approach uses an explicit XML view of the relational information. This view is typically created using the SQL/XML publishing functions.

Assume a traditional relational table created as:

```
create table t3(c1 int, c2 varchar2(100))
```

The following view could be created:

```
create view t3_v
as
select
  xmlelement("root",
    xmlelement("t",
      xmlelement("x1", c1),
      xmlelement("x2",c2))) t3_c
from t3
```

Once created, the t3_v view can be queried as any other table with an XMLType column:

```
select xt.*
from
  t3_v,
  xmltable('for $v in //t/(x1|x2) return $v' passing
    t3_c ) xt
```

The second solution uses the Oracle proprietary XQuery function, ora:view. This function creates a virtual implicit XML view of any table the user has select privileges for. The structure of the XML view is defined in the SQL/XML specification.

```
select *
from xmltable('
  for $v in ora:view("T3")/(C1|C2)
  return $v') xt
```

The details of the SQL/XML mapping are not described in the Oracle documentation. It is not documented how the different Oracle relational types map into the SQL/XML view and what XQuery types are associated with the mapped information.

Querying Traditional Relational Data with DataDirect XQuery

Where Oracle introduces a proprietary ora:view function to create a SQL/XML view of relational tables, DataDirect XQuery uses the standard XQuery collection function.

```
for $v in collection("T3")/(C1|C2) return $v
```

The details and configuration options of the SQL/XML mapping are discussed in the DataDirect XQuery documentation.

Legacy Data

Oracle XQuery does not support non-XML data. If one wants to execute an XQuery expression against an EDI message, for instance, the message first must be converted to XML and then transferred to the Oracle server before it can be used in XQuery expressions.

DataDirect XQuery, through the separately downloadable [DataDirect XML Converters](#), opens up its powerful XQuery features to a range of text and binary file formats, including tab-delimited files, comma-separated files, and EDI files. Once the adapters are installed, the doc function will recognize URIs starting with the “adapter:” prefix. The documentation of the [DataDirect XML Converters](#) contains the necessary details.

Using XMLType Columns with Oracle XQuery

A few options are available to query XMLType columns or tables with Oracle XQuery. The two most common methods are based on the XMLQuery and XMLTable SQL/XML constructs, passing in the XMLType information through the “passing” argument.

Using the SQL/XML XMLQuery construct:

```
select
  t2.name,
  xmlquery('
    for $id in ("elem1","elem2","elem3")
    return
      $x//*[ @id=$id]
  'passing t2.content as "x" returning content)
from t2
```

Using the SQL/XML XMLTable construct:

```
select
  xmlf.name,xmlt.column_value
from
  t2 xmlf,
  xmltable('
    for $id in ("elem1","elem2","elem3")
    return
      $x//*[ @id=$id]
  'passing xmlf.content as "x") xmlt
```

A third alternative is based on the proprietary Oracle XQuery ora:view function:

```
select xmlquery('
  for $id in ("elem1","elem2","elem3")
  return
    ora:view("t2")//*[ @id=$id]
  ' returning content)
from dual
```

Using XMLType Columns with DataDirect XQuery

Returning the information stored in an Oracle XMLType column is simple:

```
collection("DDXQ.T2")/T2/CONTENT/*
```

When one wants to use XQuery expressions to retrieve or restructure part of the XML data stored in an XMLType column, the required syntax is a bit more extended. Two possible approaches exist.

The first approach relies on the DataDirect XQuery support for Oracle's native XQuery features through a DataDirect XQuery proprietary function, ora-xmlquery:

```
for $x in collection("DDXQ.T2")/T2/CONTENT
return
  ddtek-sql:ora-xmlquery('
    for $id in ("elem1","elem2","elem3")
    return /*[ @id=$id]
  ', $x/node())
```

The second approach relies on the DataDirect proprietary evaluate-in-memory extension expression:

```
for $x in collection("DDXQ.T2")/T2/CONTENT
return
  (# ddtek:evaluate-in-memory #) {
    for $id in ("elem1","elem2","elem3")
    return $x//*[xs:string(@id)=$id]
  }
```

The first approach is preferred when the size of information in the XMLType column is large and the amount to be retrieved is much smaller. In all other cases, the second approach is the better approach because it uses DataDirect XQuery to query the data instead of Oracle XQuery, and DataDirect XQuery is a more reliable and complete XQuery implementation.

Conclusion: Data Source Support Compared

To conclude, here is a summary of the comparisons:

XML files & streams

- Oracle—XML information first must be transferred to the Oracle database server.
- DataDirect—using the XQuery doc function, XML files and streams (http and ftp) are supported.

XMLType columns

- Oracle—all Oracle XQuery operations work on XMLType values. The XMLType values are passed to the SQL/XML XMLQuery or XMLTable constructs using the “passing” clause.
- DataDirect—supports XMLType data through a proprietary function that translates to Oracle’s XQuery functionality or through the evaluate-in-memory extension expression.

Traditional relational data

- Oracle—supported by first creating an XML view or by using the Oracle proprietary ora:view function.
- DataDirect—supported through the XQuery collection function.

Legacy data

- Oracle—not supported
- DataDirect—supported through the XQuery doc function once the [DataDirect XML Converters](#) are installed.

Language support

Oracle XQuery

It is difficult to get a clear picture of what XQuery features Oracle does and does not support. The documentation is vague and not very useful.

To quote the Oracle documentation:

“Oracle XML DB supports the latest definition of the XQuery language. Because the language definition is an ongoing process, some areas of the language that are not yet firmly established are unsupported by Oracle XML DB or supported in a limited manner. This limits any impact on early adopters when the language definition evolves. Oracle participates vigorously in the definition of the XQuery language, is committed to full XQuery support, and will continue to remain at the forefront of XQuery development. For the latest status of the Oracle XML DB XQuery implementation, please consult the current version of Oracle Database Read Me.”

Unfortunately “*latest definition of the XQuery language*” is not very helpful. Making a guess, “latest” probably refers to the July 2004 drafts. These are the last XQuery drafts to describe the must-understand extensions (which are listed as not supported in the Oracle documentation) and the first to mention the prolog ordering mode, which Oracle supports.

The documentation is mostly “by example” and the obvious question is: what if something is not documented. The only options left are asking for advice on the Oracle XDB mailing list¹ or using the trial-and-error approach.

DataDirect XQuery

DataDirect XQuery implements most of the W3C XQuery November 3 2005 Candidate Recommendation. [Appendix A](#) of the [DataDirect XQuery User's Guide](#) explains in detail what is and what is not supported under what conditions.

Furthermore, DataDirect has published the [results of running the W3C XQuery test suite against its implementation](#), making it apparent what is and what is not supported.

Oracle and DataDirect Compared

Given the different versions of the XQuery standard supported by both products comparing them is not always possible.

Details of the more important differences in XQuery support are given in the following sections. The structure and order of these sections is the same as the “XQuery Support” appendix in the [DataDirect XQuery User's Guide](#) (which parallels the structure and order of the W3C XQuery language specification).

“2.1 Expression Context”

Expression Context—Construction mode

The Oracle ora:view function seems to use a mixture of construction mode strip and preserve. DataDirect XQuery uses preserve when constructing the SQL/XML view of relational tables. Preserve provides better type-related behavior. To give just one example, the elements in the DataDirect SQL/XML view for date columns keep their date properties, which means that date comparisons or date arithmetic works as expected, something that is not the case with Oracle XQuery.

Expression Context—Boundary-space policy

For Oracle, this always seems to be strip. DataDirect supports both strip and preserve.

Expression Context—copy-namespaces mode

Oracle seems to behave as if copy-namespaces mode is set to “no-inherit, preserve”. DataDirect always uses “inherit, preserve”.

¹ <http://forums.oracle.com/forums/forum.jspa?forumID=34>

Expression Context—base-uri

DataDirect supports base-uri. Oracle seems to behave inconsistently in this context. There is an ISQLPlus option to set the base-uri and the XQuery declare base-uri syntax is also accepted, but none of those seem to influence the base uri of constructed nodes. On the other hand, relative URIs, for example, as specified in the doc function are resolved against the base-uri property.

“2.2 Processing Model - Data Model Generation (2.2.1)”

When Oracle creates explicit XML views of relational information, type information is lost (construction mode strip-like behavior).

When Oracle inserts XML in an XMLType column with an associated XML Schema, it looks like the type information from the XML Schema is more or less preserved. But in general the behavior seems rather unpredictable.

“2.2 Processing Model - Serialization (2.2.4)”

DataDirect XQuery supports a wide range of serialization options that are based upon the XQJ and XQuery [serialization specification](#).

Oracle XQuery has no specific serialization options available.

“2.4 Concepts - Document order (2.4.1)”

Oracle XQuery is rather unpredictable with respect to document order, node identity, and duplication elimination for information that is stored in a relational database. This holds both for XML data that is stored in an XMLType column and for the implicit XML views created with the ora:view function.

“2.4 Concepts - Input sources (2.4.4)”

Oracle XQuery

- fn:doc refers to XML data stored in its XML DB repository.
- fn:collection refers to an XML DB repository folder and the XML documents contained within.
- ora:view is used to create an SQL/XML view of a relational table or view.
- The initial context item and input variables can be bound to XMLType values or expressions using the passing clause of the SQL/XML XMLQuery or XMLTable constructs.

DataDirect XQuery

- fn:doc refers to XML files or streams. The file, ftp, and http URI schemes are supported. When the [DataDirect XML Converters](#) are installed, the adapter: scheme can be used to refer to non-XML data sources (for example, EDI messages, csv files, etc.)
- fn:collection creates a configurable SQL/XML view of a database table or view.
- XML values can be bound to external variables through the XQJ API.

“3.2 Path Expressions”

Oracle can evaluate path expressions on nodes with constructed names even when the source information comes from the database. Although not explicitly mentioned in the DataDirect documentation, it is easy to work around this limitation using the DataDirect extension expression, ddtek:evaluate-in-memory.

Oracle supports the full axis feature.

“4 Modules and Prologs”

Oracle XQuery does not support modules.

Oracle accepts both construction mode strip and preserve, but it seems to always implement strip.

Conclusion: Language Support

DataDirect XQuery is better documented.

DataDirect XQuery implements a more recent version of the XQuery specification and is in general substantially more complete (for example, it supports modules).

DataDirect XQuery has published its XQTS 1.0 results.

DataDirect XQuery is more robust. Although this statement is not the result of a systematic comparison of quality, all tests and experiments conducted support this conclusion.

Both DataDirect XQuery and Oracle XQuery support integration between traditional relational information and information stored in XMLType columns.

Proprietary extensions

Oracle XQuery

Oracle XQuery supports a number of proprietary functions that are documented in its XML DB guide (“17 Using XQuery with Oracle XML DB”).

A few of these functions relate to string searching and manipulation (ora:match, ora:contains, ora:replace). The ora:sqrt function calculates the

square root of its input. Finally, there is ora:view that exposes a SQL/XML view of any accessible table or view.

Note that the functionality of ora:match, ora:contains, ora:replace, and ora:sqrt can also be accessed using DataDirect XQuery's extensions, for instance, by writing a user defined PL/SQL function that is then called from within a DataDirect XQuery expression. Furthermore, as mentioned earlier, the functionality of ora:view and DataDirect XQuery's collection function are equivalent.

DataDirect XQuery

The DataDirect XQuery extensions serve two purposes:

Extend the feature set, including:

- Call any Java function, SQL function, or user-defined PL/SQL scalar or table function. This also includes PL/SQL functions that invoke the Oracle functions that update XML values.
- Invoke the Oracle XQuery processor from within DataDirect XQuery.
- Parse an arbitrary string as XML so that it can be further processed with DataDirect XQuery.
- Validate XML fragments for their adherence to a given XML Schema.
- Support for custom URI resolvers, a user Java class that can be configured to intercept an fn:doc call and return XML data.

Configure the processing algorithms for optimization purposes.

Application integration

In most cases, XQuery will not be the only tool that is used to develop a solution for a problem. More likely, XQuery execution and result processing will be driven from a more traditional programming environment like Java using an XQuery-capable API.

The feature richness of the API and thus the flexibility with which XQuery can be integrated in the programming environment is a key (and often underestimated) factor determining eventual success or failure.

Another important aspect is the possible environmental restrictions the XQuery engine imposes, for example, a platform-specific client library that has to be installed or a required server architecture. The fewer restrictions the XQuery implementation imposes the better. Even if a given restriction is not a problem in the beginning of a project, application deployment requirements tend to change during the lifetime of an application, making some of the restrictions that were originally not an issue, a real problem.

Oracle XQuery

API

Oracle has a history of incrementally adding XML features, and the Oracle APIs that are available to work with XML reflect that history. In general, it is a challenge to find the right information: for example, there is a PL/SQL package that is called `DBMS_XMLQUERY`, which unlike the XMLQuery SQL/XML clause discussed earlier, has nothing to do with XQuery.

In general, Oracle supports different programming languages: PL/SQL, C++, C# (.NET), and Java, and while each of these programming environments obviously has a separate API, the way an application works with Oracle's XQuery and XML data is equivalent and SQL-centric.

DataDirect XQuery 2.0 only has a Java-based API, therefore, the rest of the discussion focuses on how to work with Oracle XQuery from within a Java application.

To invoke Oracle XQuery from a Java application, JDBC is used. The high-level approach is the same as for any JDBC application:

1. Set up the JDBC environment and connect to the database.
2. Prepare the SQL/XML statement that contains an embedded XQuery statement in an XMLQuery or XMLTable clause.
3. Create the input parameters for the statement, some of which may be XML.
4. Bind input bind markers.
5. Execute the statement.
6. Retrieve the results.
7. Process the results, some of which may be XML.
8. Optionally re-execute with new bind marker values.
9. Clean up.

The details of step 3 and 7 differ from what is done in a SQL-only JDBC application that does not manipulate XML data. These steps rely on the Oracle proprietary `oracle.xdb.XMLType` Java object.

The XMLType object (`oracle.xdb.XMLType`) is documented in the Oracle XML XDK Java doc. Unfortunately, the Java doc is not always as clear as one would like. To give an example, there is an XMLType constructor that builds an XMLType object from a "*pickled image*", but it is not immediately clear what that could be. Google returns a single link for "pickled image", referring to what seems to be an [Oracle patent](#) description.

The XMLType class includes:

More than a dozen constructors (and equivalent static createXML methods) that allow creating an XMLType object from different sources, including a Java String, a Java InputStream, a DOM document, and Oracle CLOB and BLOB objects.

Methods to convert the XMLType object to an oracle.sql.CLOB object or an oracle.sql.BLOB object, and then to an InputStream, a DOM node, or a String.

A method to write the XML to a Java output stream.

Methods to extract information from the XMLType object using XPath (a subset of XPath 1).

A method to transform XMLType objects using XSLT 1.

Here is a code snippet illustrating the simple usage of the XMLType object:

```
String xqString
= "SELECT *" +
  "FROM XMLTable('for $i in ora:view(\"T3\") " +
    "where $i/C1 = $ext/C1 " +
    "return $i/C2 " +
    "PASSING XMLElement(\"C1\", ?) AS \"ext\")";

PreparedStatement stmt=c.prepareStatement(xqString);

String cVal="1";
stmt.setString(1,cVal);

ResultSet rs=stmt.executeQuery();
while(rs.next()) {
  XMLType c2=(XMLType)rs.getObject(1);
  System.out.println("c2="+c2.getStringVal());
}
```

Interesting to note is that while this example only needs to pass an integer to the XQuery expression, it has to be converted to a string and then to an XMLType value before it can be passed to the XQuery expression.

Notes:

Oracle ships both a Type 4 and Type 2 JDBC driver. The type 2 driver has a number of not well documented advantages when working with XMLType data that uses the structured storage method (see [Oracle XML DB forum](#)). On the other hand, a Type 2 driver comes with added deployment complexity.

JDBC 4 (JSR 221) comes with full support for the SQL/XML features, including the XML data type and a range of classes and methods to access the XML information. Oracle 10g R2 and its current JDBC drivers, however, do not support JDBC 4 yet (which will be an integral part of the upcoming Java J2SE 1.6 release).

A number of issues were identified with the Java XMLType API.

Environment

The Oracle XQuery implementation is in the Oracle database server. A Java application that needs to execute an XQuery expression uses JDBC (Type 2 or Type 4) to communicate with the XQuery execution engine that runs in the database server.

Oracle XQuery applications, therefore, require an environment where:

- An Oracle database server is available

- An installed Oracle JDBC driver on each of the client machines

- Installed Oracle client libraries (OCI) when the Type 2 driver is used

DataDirect XQuery

API

DataDirect XQuery implements the XQuery API for Java (XQJ) version 0.2.1, which is being developed within the Java Community Process and is known as "[JSR 225 XQuery API for Java \(XQJ\)](#)".

XQJ is designed with JDBC in mind. The result is that the API immediately feels familiar for those who know JDBC. At the same time, the JSR expert group recognizes the different needs of XQuery programmers, most of which find their origin in the fact that the XML (and XQuery) data model differs from the traditional relational data model. To accommodate this need, tight integration with existing and upcoming XML Java APIs (DOM, SAX, StAX) is available.

Therefore, when needed, XQJ allows fine grained access to information retrieved from an XQuery expression, which makes it possible to build extremely scalable XQuery applications. At the same time, simple needs (for example, execute a query and serialize the result as a string) can be implemented in a very limited amount of code—simple tasks stay simple.

As an example:

```
// The data source connection
DDXQDataSource ds=new DDXQDataSource();
ds.setJdbcUrl("jdbc:xquery:oracle://dbora;SID=test");
XQConnection c=ds.getConnection("myuserid", "mypswwd");

// Create the query string
String xq="declare variable $ext as xs:int external; " +
        "for $i in collection('T3')/T3 " +
        "where $i/C1=$ext " +
        "return $i/C2";

// Create prepared expression
XQPreparedExpression e=c.prepareExpression(xq);

// Bind the variable
e.bindInt(new QName("ext"),1);
```

```
// Execute the query
XQSequence r=e.executeQuery();

// Result
System.out.println(r.getSequenceAsString());

// free all resources
...
```

Note the similarity with the JDBC API. What is not obvious from this example is the richness of features XQJ (and DataDirect XQuery) offer to work with the XQSequence object, allowing either coarse grained, easy to use access to the XML (as in the example) or very fine grained and type safe access to the individual items in the XQSequence.

Environment

A Java Runtime Environment version 1.4 (JRE 1.4) or higher is needed to run applications that use DataDirect XQuery. Any environment where a Java jar file can be deployed is suitable to run DataDirect XQuery. This includes a wide range of hardware and software architectures, from a standalone Java client application running on a desktop machine to a complex distributed enterprise application deployed in a clustered J2EE server environment. Obviously, when an application wants to use XQuery expressions to query information stored in an Oracle database, access to an Oracle database server is required.

Conclusion: Application Integration

DataDirect XQuery implements XQJ (version 0.2.1), which is specifically designed with the XQuery developer in mind. Both coarse grained, easy to use access methods and fine grained, type safe and scalable access methods are available.

Oracle XQuery relies on a proprietary JDBC 3.0 extension to integrate XQuery in a Java application environment. The flexibility of this proprietary extension does not compare to what is offered by XQJ.

A DataDirect XQuery application can be deployed in any environment where a Java Run time environment is available, ranging from web browser applets to complex application server architectures. An Oracle database server is needed only for the data stored in the database, not for the XQuery implementation.

An Oracle XQuery application uses the JDBC API and needs connectivity to an Oracle 10g R2 database server to execute XQuery expressions (even if the XQuery expressions do not access any data stored in Oracle).

We welcome your feedback! Please send any comments concerning documentation, including suggestions for other topics that you would like to see, to:

docgroup@datadirect.com

FOR MORE INFORMATION

800-876-3101

Worldwide Sales

Belgium (French).....	0800 12 045
Belgium (Dutch).....	0800 12 046
France	0800 911 454
Germany	0800 181 78 76
Japan	0120.20.9613
Netherlands	0800 022 0524
United Kingdom	0800 169 19 07
United States	800 876 3101

Copyright © 2007 DataDirect Technologies Corp. All rights reserved. DataDirect Connect is a registered trademark of DataDirect Technologies Corp. in the United States and other countries. DataDirect XQuery is a trademark of DataDirect Technologies Corp. in the U.S. and other countries. Java and all Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. Other company or product names mentioned herein may be trademarks or registered trademarks of their respective companies.



DataDirect Technologies is the software industry's only comprehensive provider of software for connecting the world's most critical business applications to data and services, running on any platform, using proven and emerging standards. Developers worldwide depend on DataDirect® products to connect their applications to an unparalleled range of data sources using standards-based interfaces such as ODBC, JDBC™ and ADO.NET, XQuery and SOAP. More than 300 leading independent software vendors and thousands of enterprises rely on DataDirect Technologies to simplify and streamline data connectivity for distributed systems and to reduce the complexity of mainframe integration. DataDirect Technologies is an operating company of Progress Software Corporation (Nasdaq: PRGS). For more information, visit www.datadirect.com.