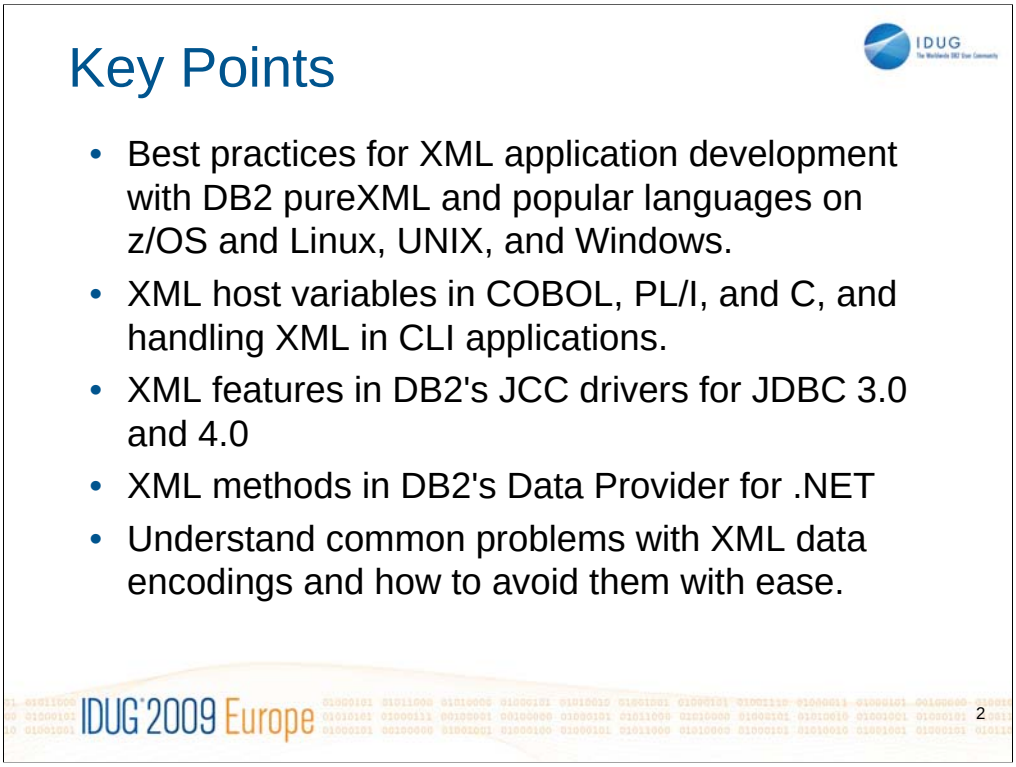
Abstract:

DB2 9 for z/OS and DB2 9 for Linux, UNIX, and Windows have introduced pureXML - a collection of features to store, index, query, update, and maintain XML data alongside your relational data. XML has emerged as a new data type in DB2. In addition to the server side features, all of the major database APIs have been extended with XML capabilities. You can now define XML type variables for embedded SQL, CLI, Java, and .NET applications. These APIs also offer XML-specific functions that simplify application development and help you avoid common encoding problems. This session walks you through "the other side" of pureXML, focusing on application development, APIs, coding guidelines, and performance tips for the XML application developer.

Speaker:

Matthias Nicola is a senior software engineer at IBM's Silicon Valley Lab, in San Jose, CA, USA. He focuses on all aspects of XML in DB2. Matthias works closely with the DB2 pureXML development teams as well as with customers and business partners to help them design, optimize and implement XML solutions. Previously Matthias worked on data warehouse performance with Informix Software. (www.matthiasnicola.de)



•Learn best practices for XML application development with DB2 pureXML and popular languages on z/OS and Linux, UNIX, and Windows.

•Learn how to use the six new data types for XML host variables in COBOL, PL/I, and C. Also learn how best to handle XML in CLI applications.

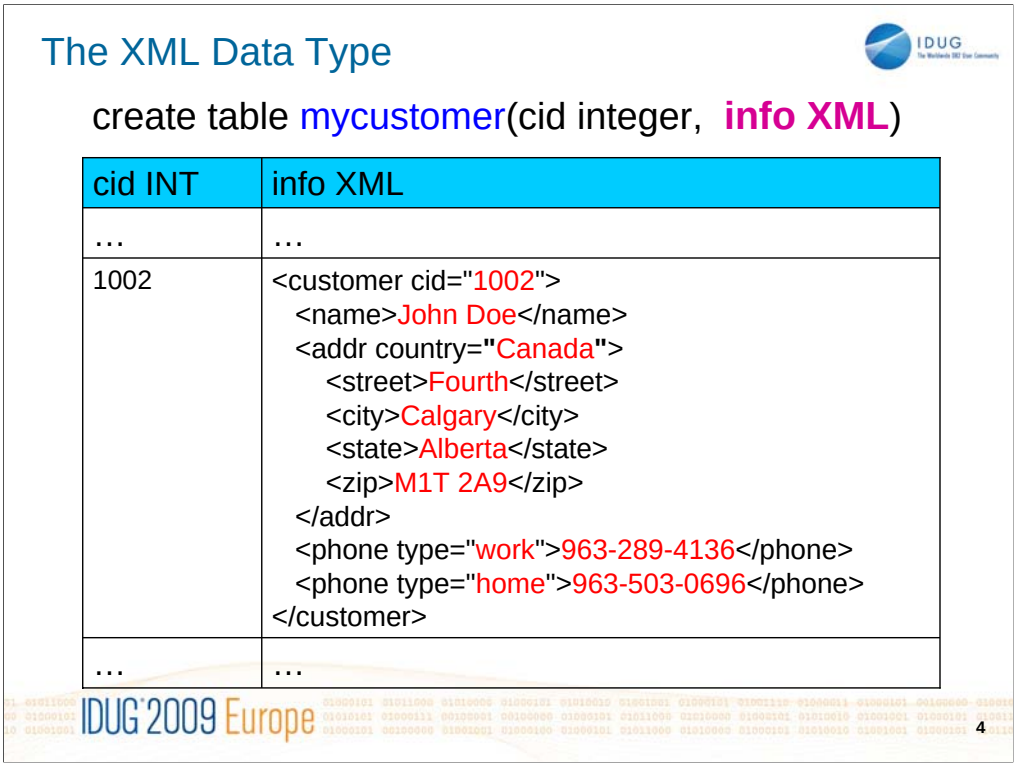
•Understand the XML features in DB2's JCC drivers for JDBC 3.0 and 4.0, and how to use them to develop efficient DB2 pureXML applications in Java.

•Learn about the XML-specific methods in DB2's Data Provider for .NET

•Understand common problems with XML data encodings and how to avoid them with ease.

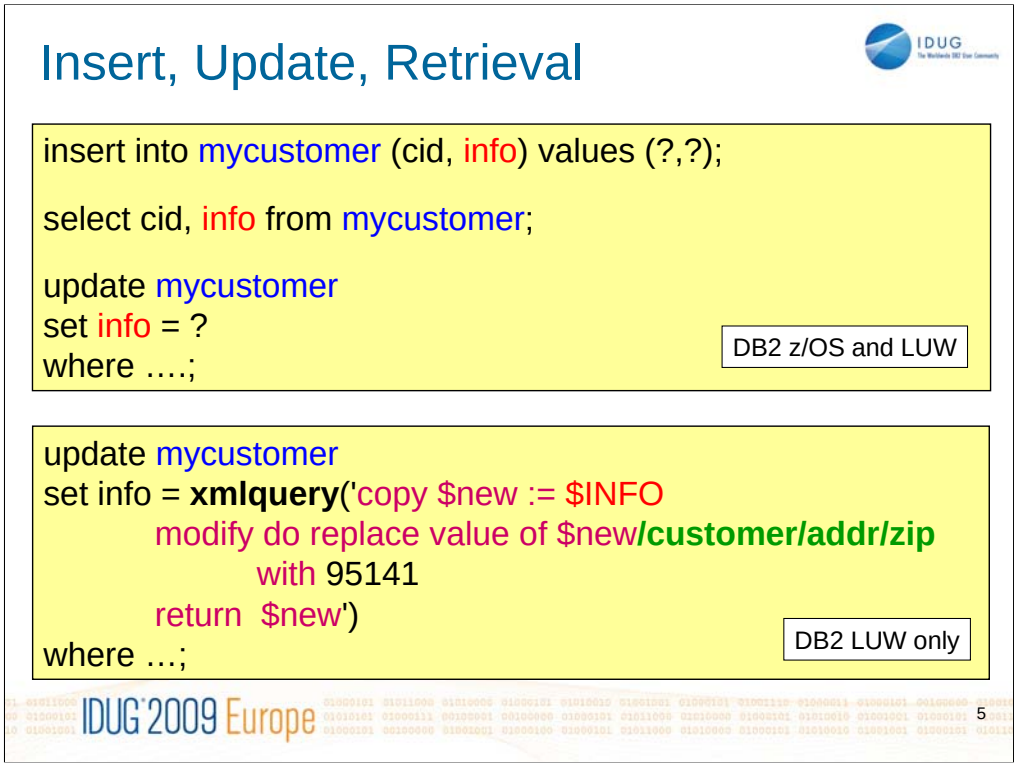
Agenda

- **DB2 pureXML Overview for Developers**
- **Internal and External Encoding of XML Documents**
- **XML in Java Applications**
- **XML in .NET Applications**
- **XML in Embedded SQL Applications, COBOL, PL/I, and C**
- **XML in CLI Applications**
- **Summary**



At the core of the pureXML support is the XML data type that can be used to define one or multiple columns in a tables.

We use this sample table throughout the entire presentation. The table name "mycustomer" and the XML column name "info" will appear many times.



Here are the basics of inserting, updating, and retrieving XML data.

Both DB2 for z/OS and DB2 for Linux, UNIX, and Windows support update statements that perform a full-document replacement.

DB2 9.5 for Linux, UNIX, and Windows also supports updates to individual XML elements and attributes within an XML document (bottom part of this slide).

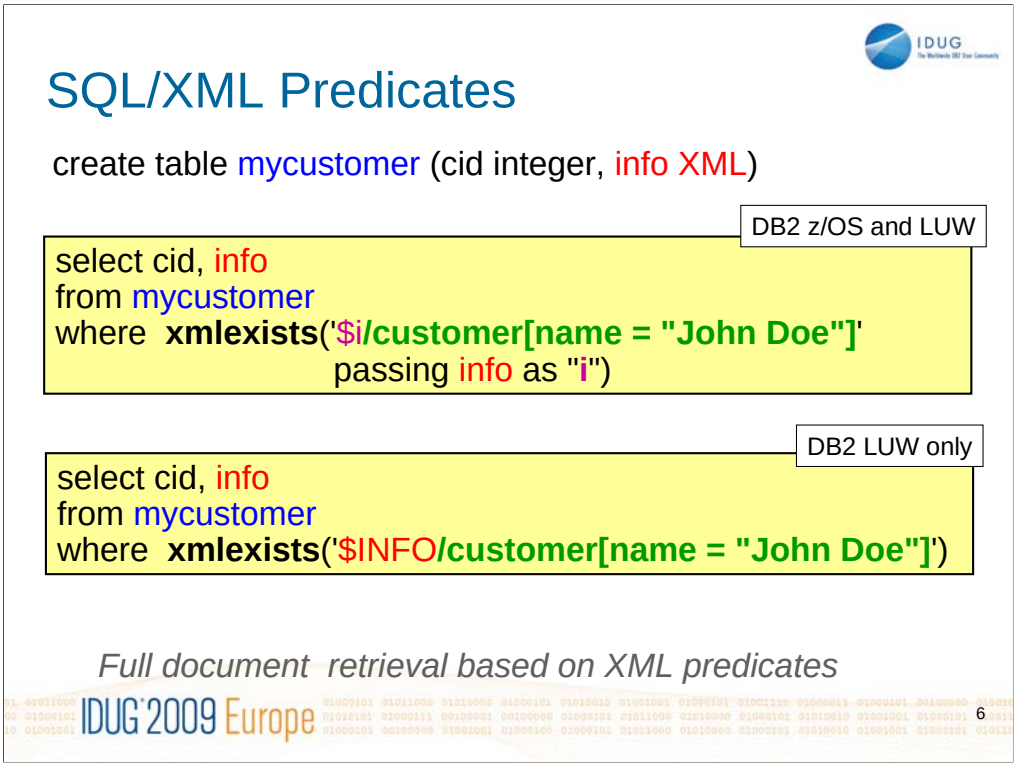
For more details see:

http://www.tinyurl.com/pureXML

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0710nicola/

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0708nicola/

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0709nicola/

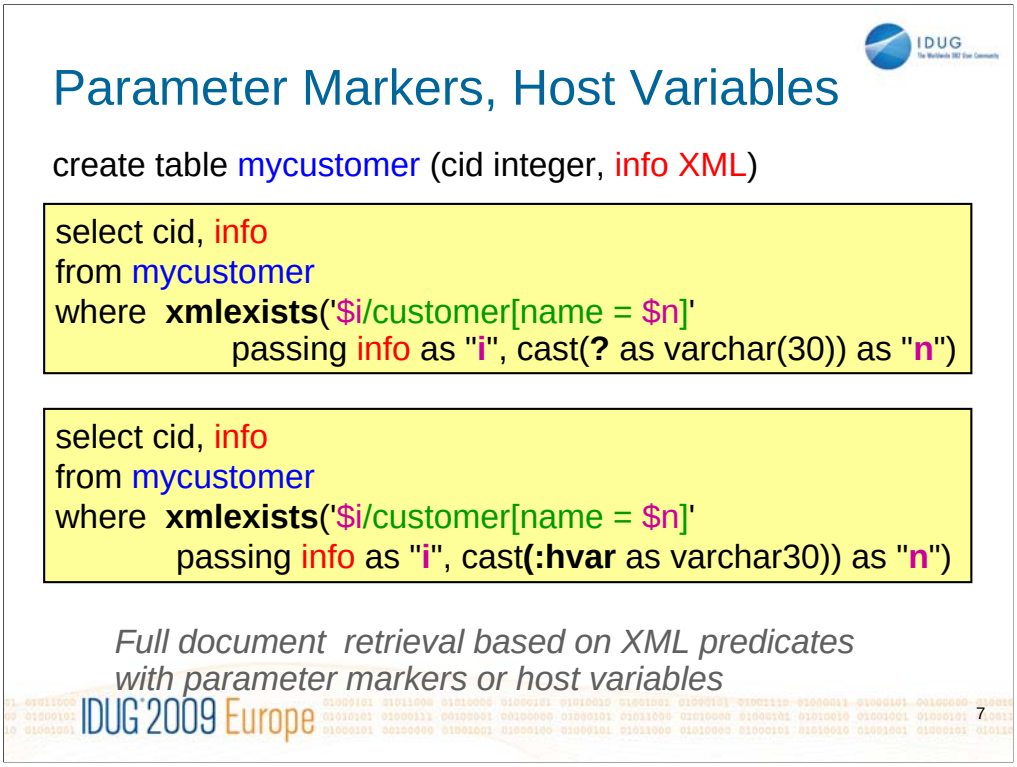


XMLEXISTS is part of the SQL standard and allows you to express predicates on XML columns. Within an XMLEXISTS predicate you need to specify *which* XML column to look at. This is typically done with the "passing" clause. The "passing" clause assigns the XML column "info" to the alias "i". This alias can then be used as a variable ($i) that defines the starting coint(context) of the XPath expression.

In DB2 9.5 for Linux, UNIX, and Windows, the "passing" can be omitted and you can use the XML column name (in uppercase!) directly as the context for the XPathe expression.

The following white paper contains more detailes on SQL/XML queries:

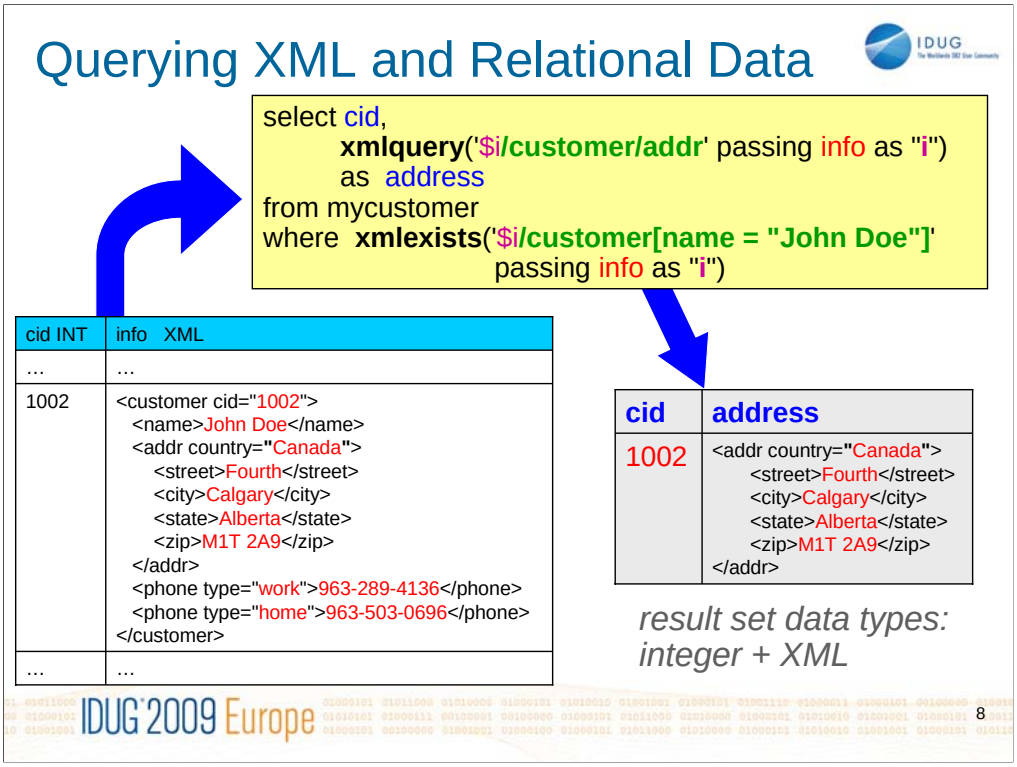
http://www.ibm.com/developerworks/db2/library/techarticle/dm-0606nicola/



The "passing" clause is always required if you want to use parameter markers or host variables in an XMLEXISTS predicate. Since XPath itself does not know or understand SQL-style parameter, nor host variables, the "passing" clause is used to assign a parameter marker or host variable to a variable, such as $n in this example.

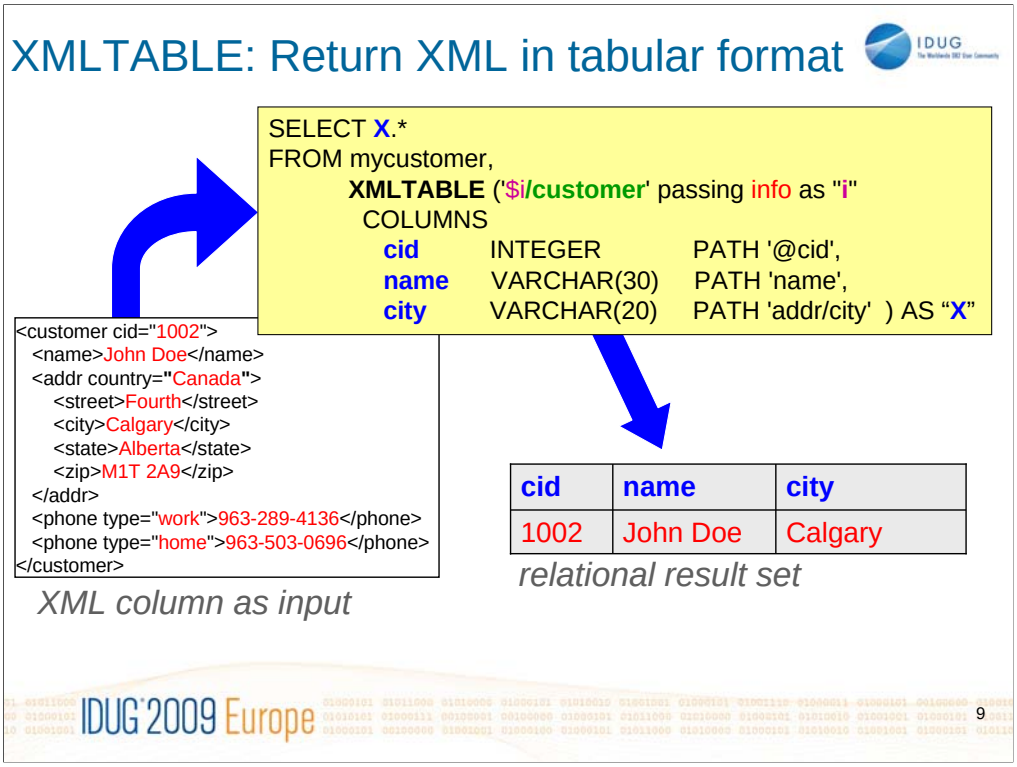
The casting is required as per the SQL standard. It determines the type of comparison that should be performed (string comparison vs. numeric comparison).

Tip: Don't cast a parameter to type CHAR. The reason is that a CHAR may have trailing blanks which would be included in the comparison and typically lead to an empty result set. Use VARCHAR instead of CHAR.

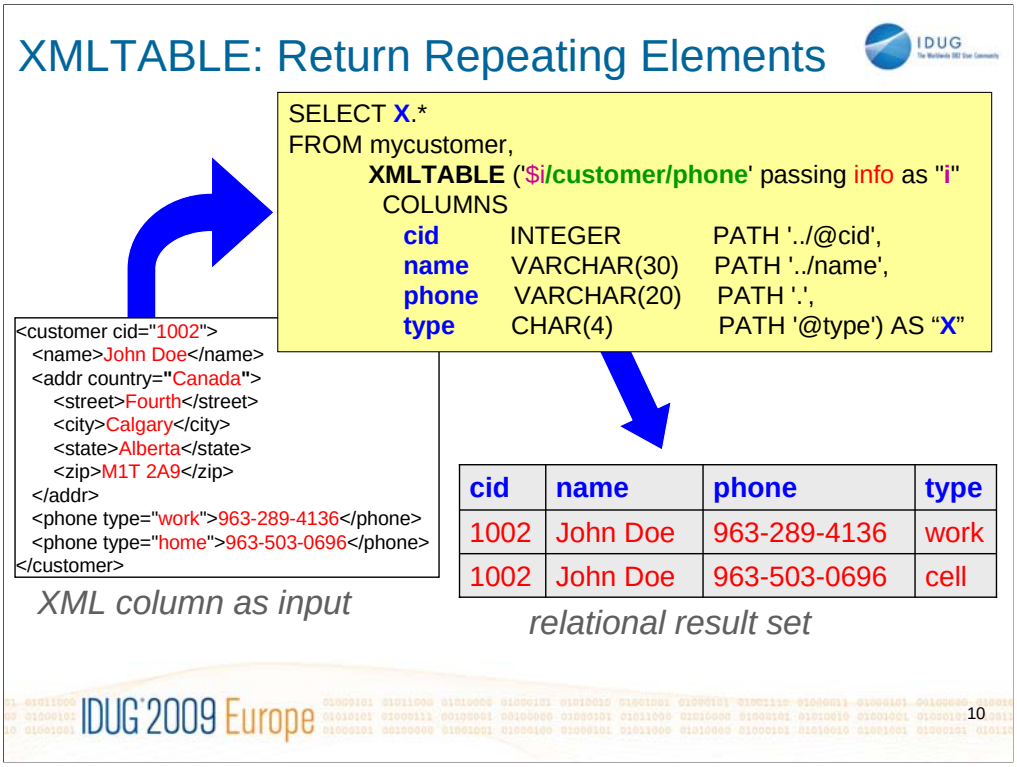


The previous query examples have always returned full documents from the XML column. Here, the XMLQUERY function is used to extract just a fragment of a document.

Note: the scalar function XMLQUERY always produces a single output column of data type XML.

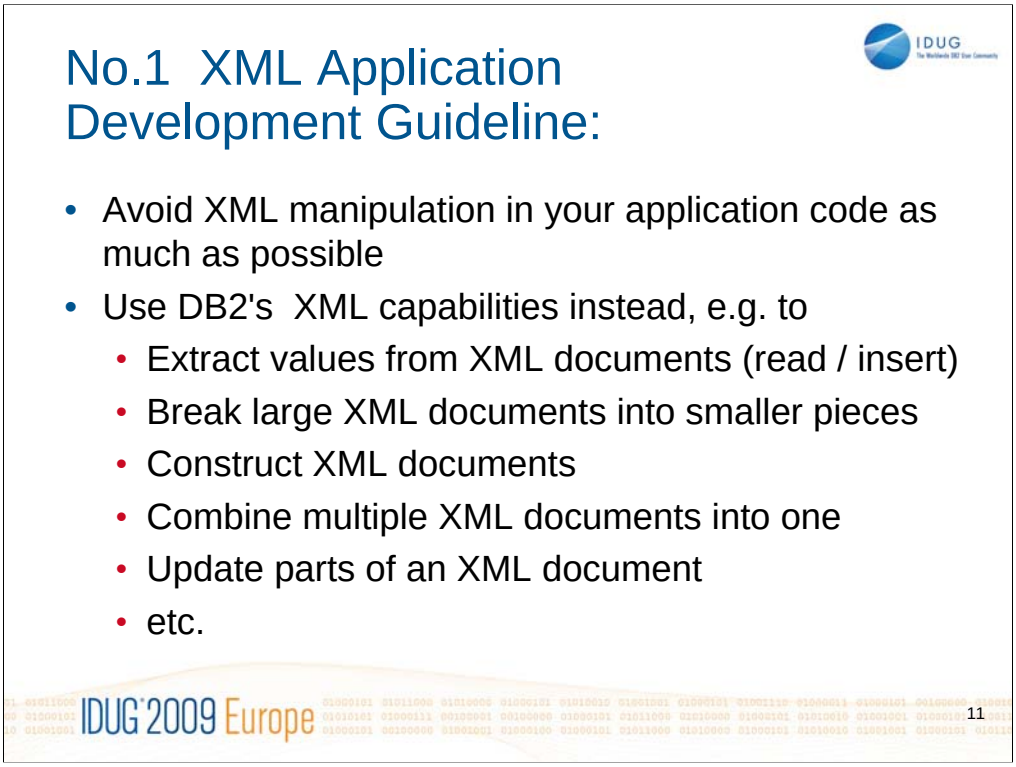


The XMLTABLE function is used in the *FROM* clause of the *SELECT* statement together with the table "mycustomer" that it operates on. The XMLTABLE function is implicitly joined with the table and applied to each of its rows. The COLUMNS clause of the XMLTABLE function defines which columns to produce.

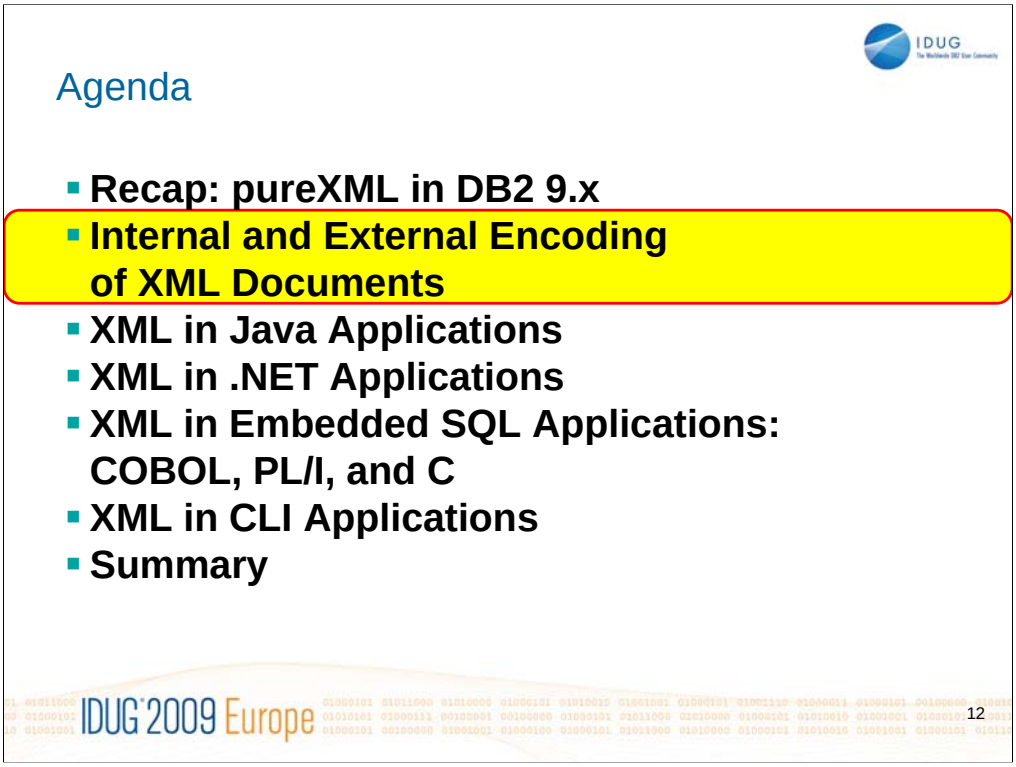


The XMLTABLE function is used in the *FROM* clause of the *SELECT* statement together with the table "mycustomer" that it operates on. The XMLTABLE function is implicitly joined with the table and applied to each of its rows. The COLUMNS clause of the XMLTABLE function defines which columns to produce.

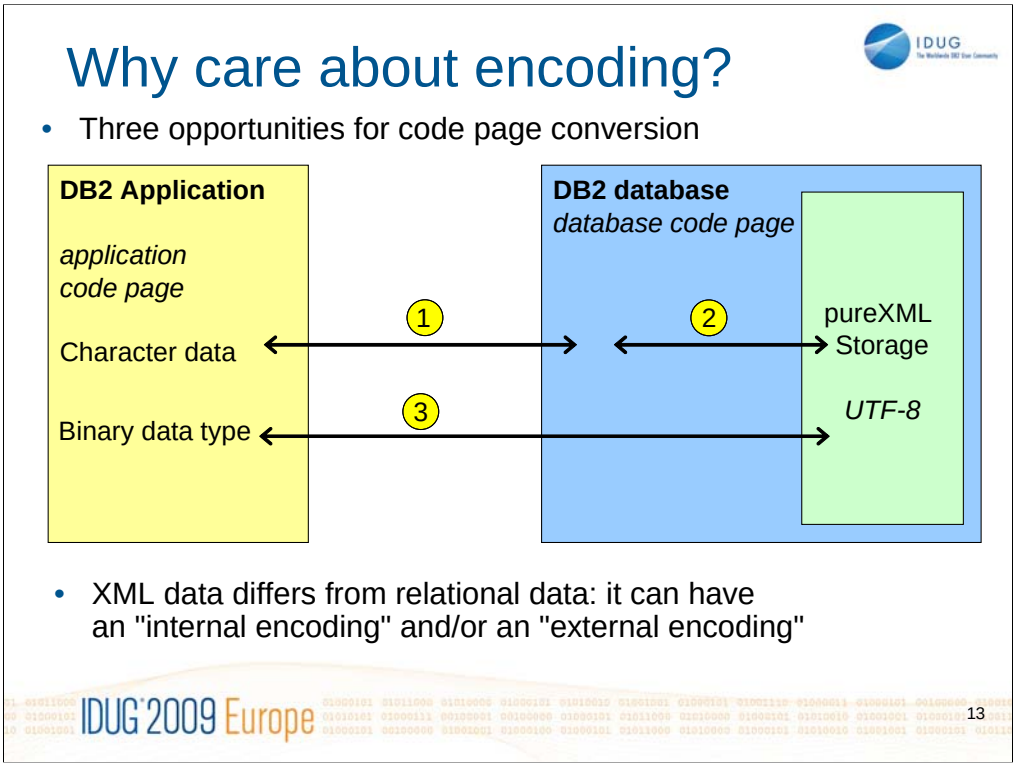
In this example the XMLTABLE function returns repeating elements, such as the <phone> element which can occur multiple ties per document in this example. Note that the row-generating expression is **/customer/phone**, which means that XMLTABLE will produce one row for each phone element. The PATH expressions in the COLUMNS clause are *relative* to /customer/phone. For example, the PATH ../@Cid includes a parent step ("..") to navigate from any phone element to the customer element, and then to Cid attribute (/customer/phone/../@cid).



The examples on the previous slides have shown that many common types of XML manipulation can (and should) be done in DB2. This is easier and more efficient (faster) than doing the same things in application code. A key benefit of performing XML manipulation in DB2, is that DB2 stores XML in a parsed format. Hence, many types of XML manipulation can be done in DB2 **without** any additional XML parsing – a key performance benefit over applications that read/write only full documents but use XML parsing at the application level to perform XML manipulation. This benefit is what pureXML is all about !



XML encoding might not be the most exciting topic, but it is important so we need to cover at least the basics….



Developing XML applications for DB2 is all about moving XML data between the application and the DB2 server. This has encoding implications.

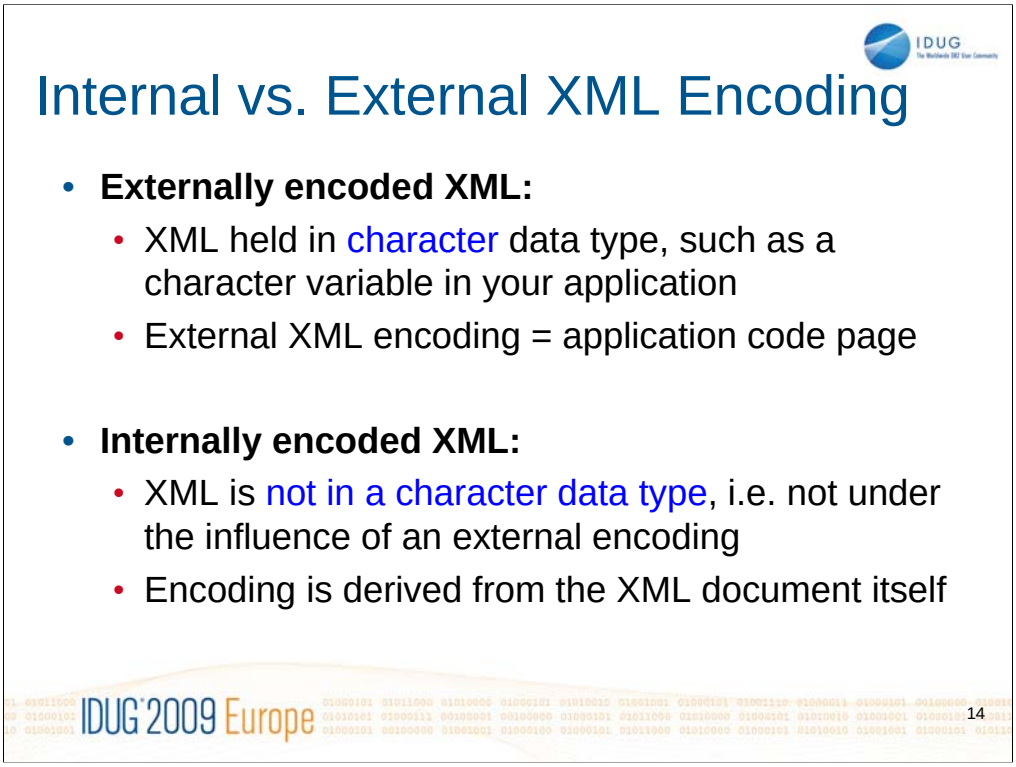
This picture in this chart applies to all DB2 application that use XML data, not matter which API or programming language you use. DB2 for z/OS and DB2 for Linux, UNIX, Windows always store XML in UTF-8 format – regardless of the original encoding. This may cause code page conversion upon insert. When you retrieve XML data from DB2, you can retrieve it in UTF-8 or in the application code. Some APIs (such as JDBC) even allow you to specific a custom target encoding when you retrieve the XML data.

1. Code page conversion from the application code page to the database code page (if the code pages differ)
2. Code page conversion from the database code page to UTF-8 for XML storage (if the database code page isn't UTF-8)
3. Code page conversion on insert, and only if the so-called "internal encoding" of the XML data is not UTF-8. No code page conversion on retrieval of XML data from DB2 into the application.

Code page conversion is undesirable because

(a) it uses extra CPU cycles that you may wish to save

(b) it can lead to data loss in certain cases. For example, if certain characters in one code page cannot be represented in another code page, then those characters get lost and are replaced by substitution characters. Unicode, such as UTF-8 and UTF-16, can represent all characters. However, certain non-Unicode code pages cannot. Hence, using Unicode is recommend. Java and .NET application always use Unicode (UTF-16 / UCS-2). In DB2 9.5 for Linux, UNIX, Windows and higher, UTF-8 is the default database codepage for new databases.



DB2 for z/OS does not enforce consistency of the internal and external encoding. If the internal and external encoding information are different, the external encoding takes precedence although character conversion might have occurred on the data and there might be data loss. Hence, it is strongly recommended to avoid a mismatch between internal and external encoding.

To understand exactly how the internal encoding of an XML document is determined, we need to understand the following:

-XML declaration

-BOM (Unicode Bye Order mark)

-encoding declaration

These three items are explained next..

What is a BOM

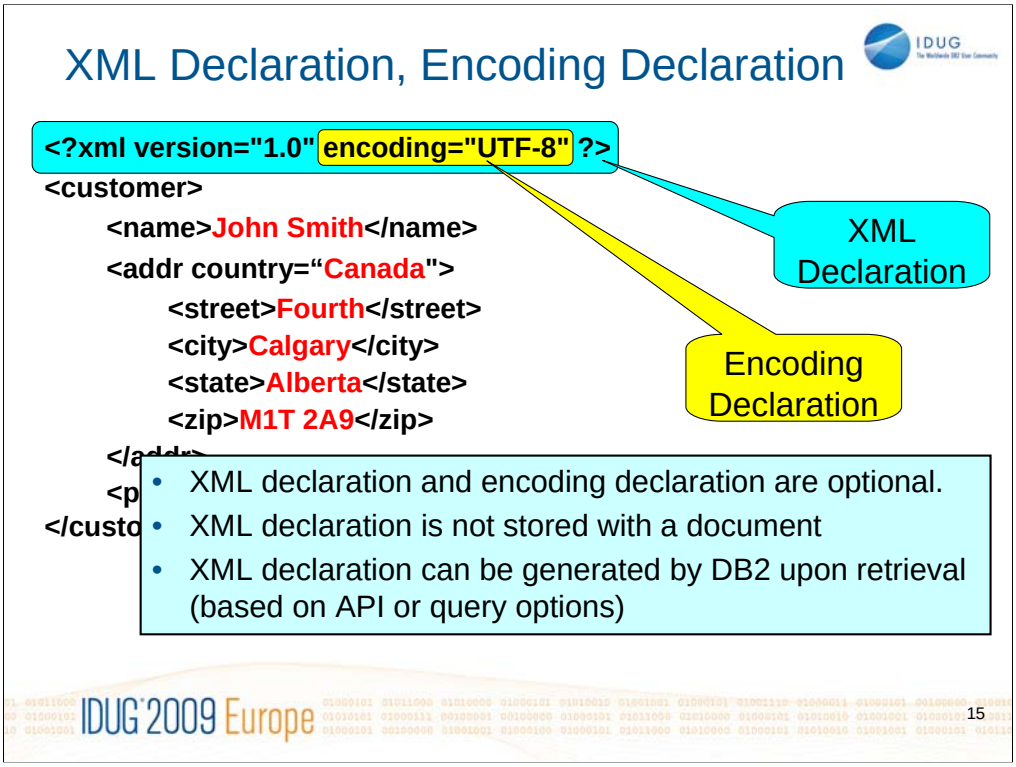
•BOM = Unicode Byte Order Mark

•Special Unicode character U+EFFF ("zero-width no-break space")

•Can exist at the very beginning of an XML document

•Looks different in UTF-8, UTF-16, or UTF-32

•Allows an XML parser (such as in DB2) to detect the encoding of the document



Some XML basics, independent from DB2:

-An XML can have (optionally!) an XML declaration.

-The XML declaration can have (optionally!) an encoding declaration (aka encoding attribute).

-An encoding declaration is always part of an XML declaration cannot exist by itself.

You will see later in this presentation when and how the encoding declaration may be significant to DB2, and how DB2 handles your XML data if the encoding declaration does or does not exist.

What is a BOM?

- BOM = Unicode Byte Order Mark
- Special Unicode character U+EFFF ("zero-width no-break space")
- Can exist at the very beginning of an XML document
- Looks different in UTF-8, UTF-16, or UTF-32
- Allows an XML parser (such as in DB2) to detect the encoding of the document

Optional slide. It's useful background information but maybe not needed during the session.

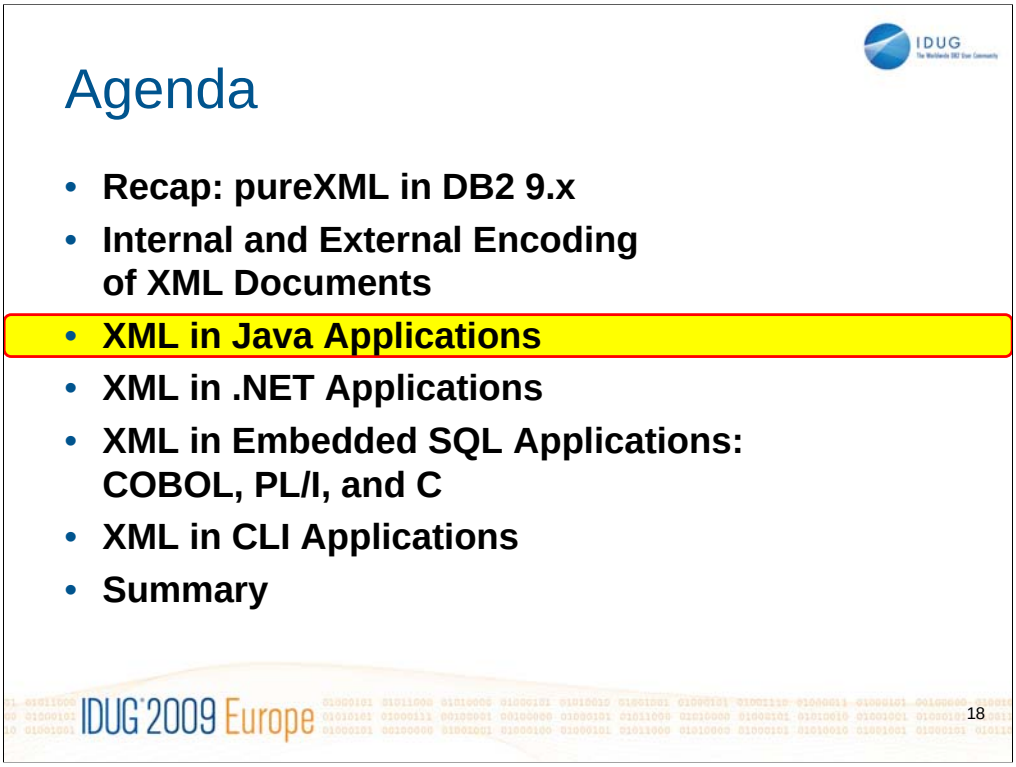
The Internal XML Ecoding

How is the internal XML encoding determined?

	…an XML decl. w/ encoding decl.	…an XML decl. w/o encoding decl.	…no XML declaration
BOM does not exist	Encoding derived from encoding declaration.	XML parser deduces UTF-8 or UTF-16 from the XML decl.	XML is assumed to be in UTF-8. And it better be!
BOM exists	Encoding derived from BOM. Encoding decl. and BOM must match !	Encoding derived from BOM.	Encoding derived from BOM

Optional slide. It's useful background information but maybe not needed during the session.

If the XML document has a BOM and an encoding declaration, they must match. If the BOM indicates a different encoding than the encoding declaration, DB2 LUW rejects the XML data with the following error: **SQL16168N XML document contains an invalid XML declaration. Reason code = "7".**



Now let's look at handling XML in JDBC.

This is not an introduction to JDBC in general but geared towards people who know at least the basics of using JDBC to read/write data from/to a DB2 server

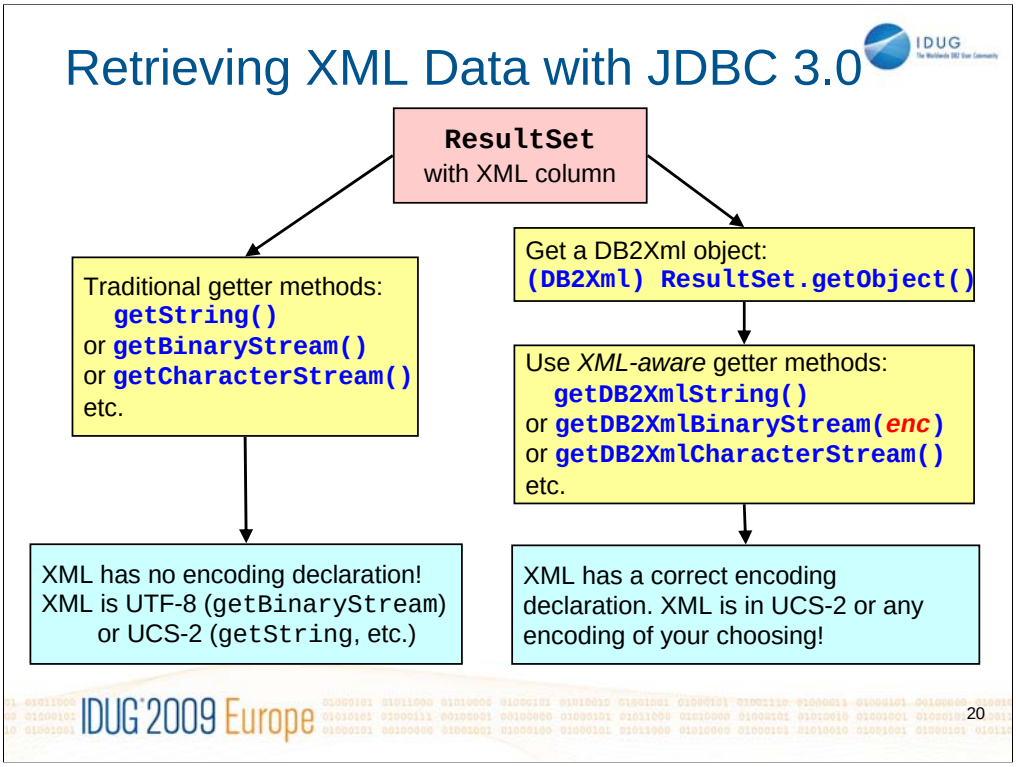
Java Common Client (JCC)

	JCC 3	JCC 4
JDBC Supported	JDBC 3.0	JDBC 3.0 and 4.0
Min. Java Level Required	1.4	6.0
JAR File	db2jcc.jar	db2jcc4.jar
Java Interface for XML Data	com.ibm.db2.jcc.DB2Xml	java.sql.SQLXML
Java Constant for the XML Data Type	java.sql.Types.OTHER	java.sql.Types.SQLXML

- The JDBC 4.0 standard introduces the data type **`SQLXML`**
- DB2's driver for JDBC 3.0 supports the proprietary Java data type **`DB2Xml`**

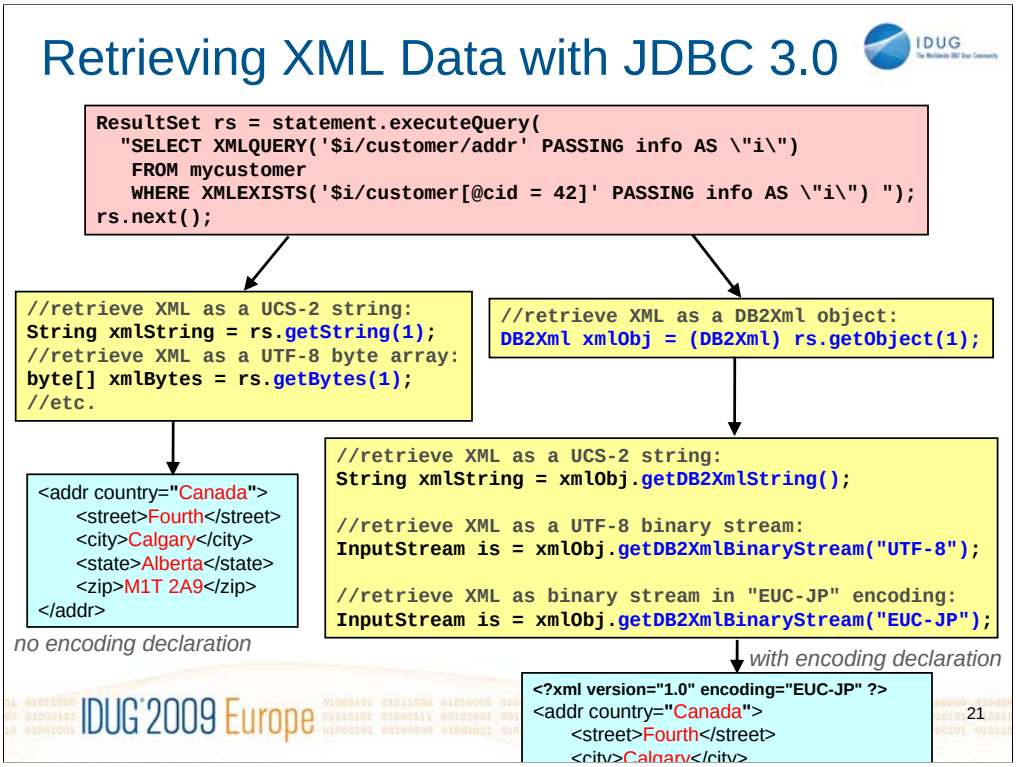
Java 6 and JDBC 4.0 are becoming more and more predominant these days.

Everything that IBM supports in the JCC Version 3 driver, such as the com.ibm.db2.jcc.DB2Xml interface, is still supported in JCC Version 4.



You can use JDBC 3.0 to retrieve XML data in two ways. Either use traditional getter methods on your ResultSet object (left), or get a DB2Xml object first and then apply XML-aware getter methods to that DB2Xml object (right). All of this is also still supported in DB2's JCC4 driver.

Retrieving XML data into binary variables (via getBinaryStream, getBytes, etc.) is preferred, because it avoids code page conversion.

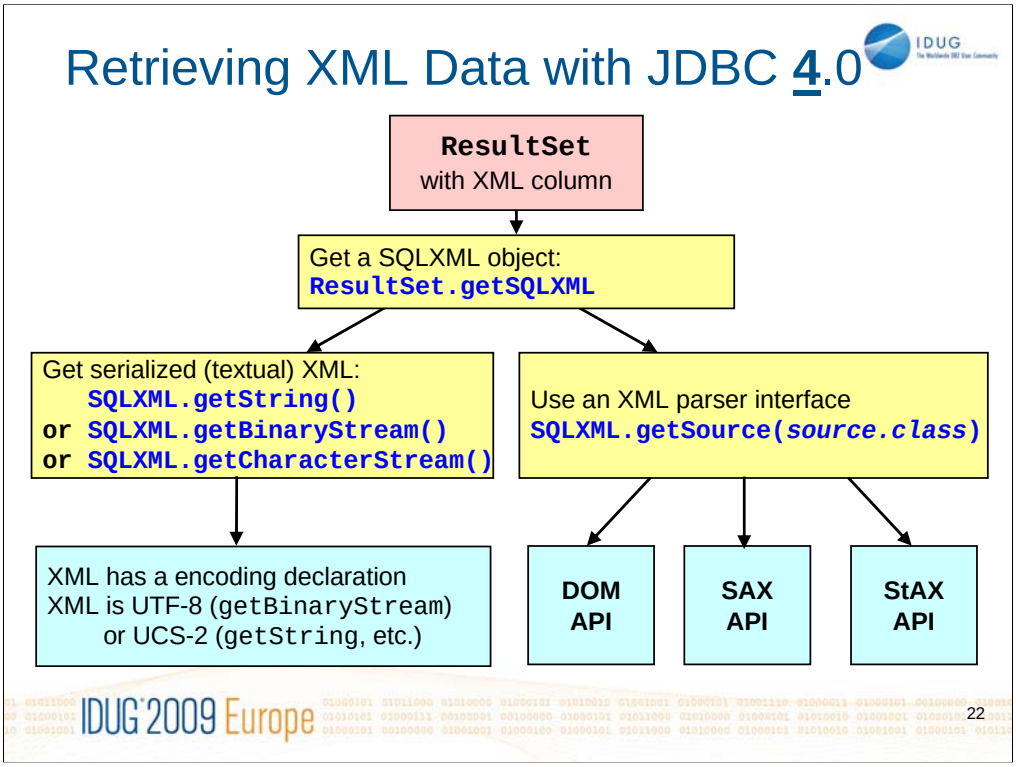


The slide shows the same concepts as on the previous slide, but this time with actual Java code, for a greater level of detail. This code works with JDBC 3.0 and JDBC 4.0.

In the lower right-hand side of the slide you see the first few lines of the XML document that your application receives from the call **xmlObj.getDB2XmlBinaryStream("EUC-JP");**

The parameter "EUC-JP" specifies the desired encoding, and this encoding is shown in the encoding declaration at the very top of the returned XML document.

You can certainly prepare SQL/XML queries before executing them, by using the connection.prepareStatement method. This is omitted on this slide only for brevity.



With JDBC 4.0 you can still do all the things that JDBC 3.0 supports. In other words, the concepts explained on the previous 2 slides also apply to JCC4 and JDBC 4. Additionally, JDBC 4.0 has standardized the SQLXML interface to access XML columns in relational databases. It still allows you to retrieve XML in a serialized (textual) format (left), but also you to access reteieved XML data directly via a DOM, SAX, StAX API (right). The actual Java code for the right branch of this diagram, leading to "DOM API", is shown on the next slide.

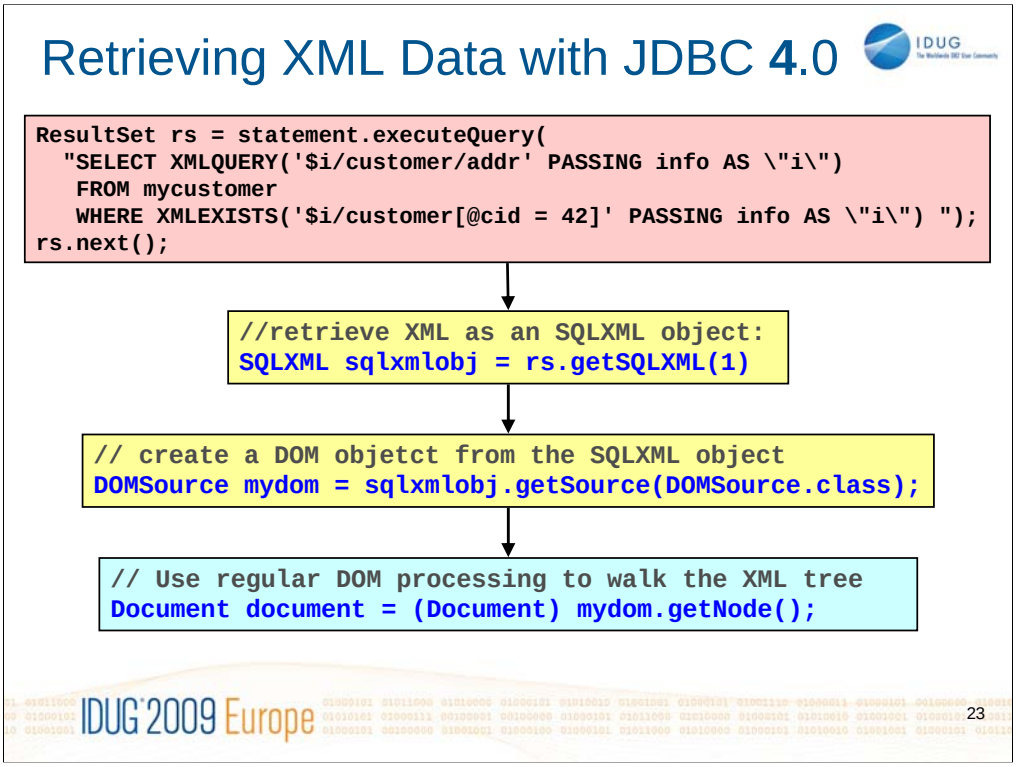
For details on DOM, SAX, and StAX (which are not DB2-specific things), see:
http://java.sun.com/webservices/docs/1.6/tutorial/doc/SJSXP2.html

For details on the SQLXML interface in JDNC 4.0, see:

http://java.sun.com/javase/6/docs/api/java/sql/SQLXML.html

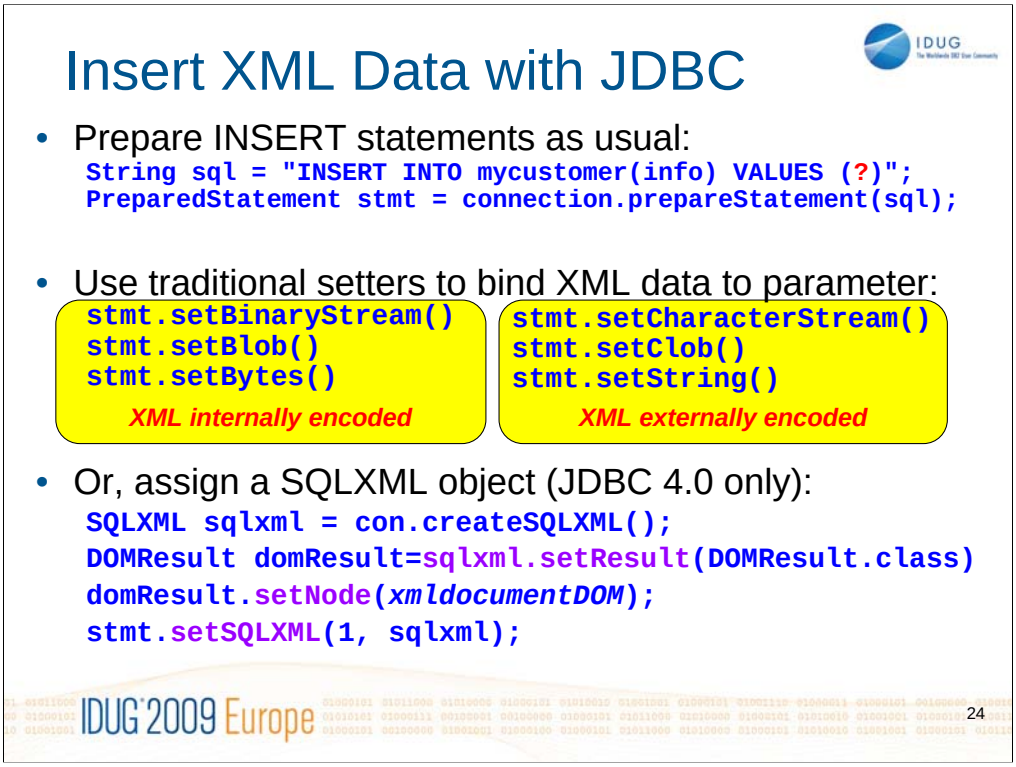
and

http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/topic/com.ibm.db2.luw.apdv.java.doc/doc/c0021817.html



The slide shows actual Java code using the SQLXML object and setting up a DOM to read or manipulate the document.

This code works with JDBC 4.0 and Java 6, or higher. It does not work with Java 1.5 or JDBC 3.0 (JCC 3).



It is recommended to use binary data types to bind in XML data (i.e. setBinaryStream, setBytes, setBlob) because it avoids external encoding and reduces the need for code page conversion.

If your application is already using DOM or SAX today, you use JDBC 4.0 to directly assign a DOM or SAX object to the parameter marker for insert into DB2.

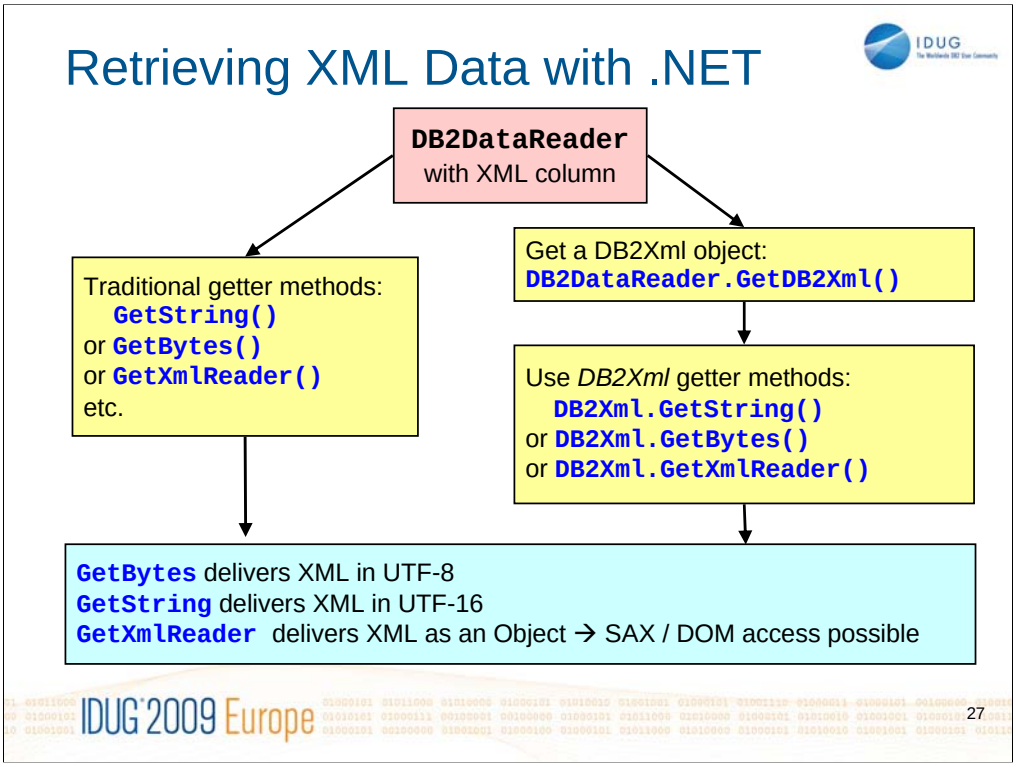
DB2 pureXML with mapping frameworks…

….is beyond the scope of this session, but you can find more information here:

- *"Handle pureXML data in Java applications with pureQuery"*
  http://www.ibm.com/developerworks/data/library/techarticle/dm-0901rodrigues/
- *"Mapping DB2 pureXML data using Hibernate User Types"*
  https://www.hibernate.org/466.html

Agenda

- Recap: pureXML in DB2 9.x
- Internal and External Encoding of XML Documents
- XML in Java Applications
- XML in .NET Applications
- XML in Embedded SQL Applications: COBOL, PL/I, and C
- XML in CLI Applications
- Summary



The XML support in .NET is conceptually similar to the XML support in JDBC 4.0. Either you use traditional getter methods on your *DB2DataReader* object (left), or you first get a *DB2Xml* object and then use the getter methods of the *DB2Xml* class (right). The end result is the same, i.e. you get XML data in UTF-8 (GetBytes) or in UTF-16 (GetString) or as an object (GetXmlReader). The *XMLReader* object, which by the way is not DB2-specific and part of the .NET standard, allows to consume the XML data through a SAX-like API. Once you have an *XMLReader* object you can also use other .NET capbilities for reading and manipulating XML data. For example, you can obtain an *XMLDocument* or *XPathDocument* object from the *XMLReader* object.

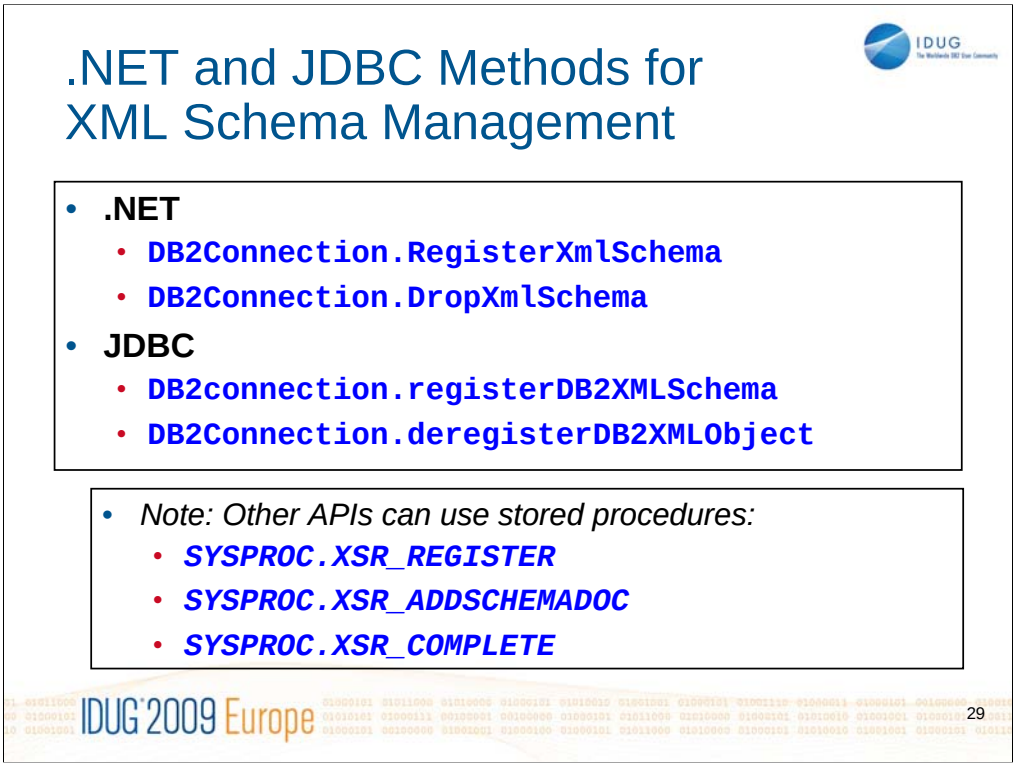
Insert XML Data with .NET

```
DB2Command cmd = DB2Connection.CreateCommand();
cmd.CommandText = "INSERT INTO mycustomer(info)
                   VALUES (?)";

DB2Parameter p1 = cmd.CreateParameter();
p1.DB2Type = DB2Type.XML;
p1.Value = File.OpenRead(filename);

cmd.Parameters.Add(p1);
cmd.Prepare();
cmd.ExecuteNonQuery();
```

Alternatively, **`DB2Parameter.Value`** can also be assigned a value of type **`String`**, **`Byte[]`**, **`XmlReader`**, or **`DB2Xml`**.

Inserting XML data is straight forward.

Be sure to set the "DB2Type" of the parameter to DB2Type.XML.



For details on XML Schema management from your Java or .NET application, see:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.apdv.java.doc/doc/c0021681.html

and

http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.swg.im.dbclient.adonet.ref.doc/doc/DB2ConnectionClassRegisterXmlSchemaMethods.html

For details on registering XML Schemas via stored procedure calls, see:

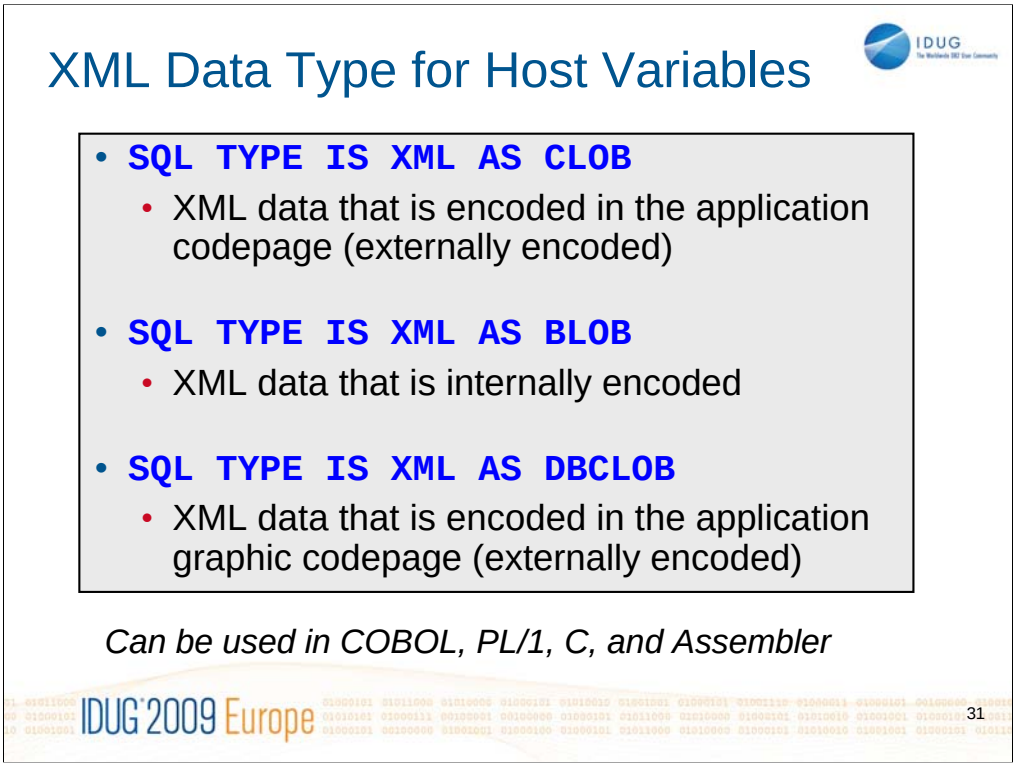
http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.xml.doc/doc/c0022712.html (Linux, UNIX, Windows)

and

http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.xml/db2z_xmldb2storedprocs.htm (z/OS)

Agenda

- **Recap: pureXML in DB2 9.x**
- **Internal and External Encoding of XML Documents**
- **XML in Java Applications**
- **XML in .NET Applications**
- **XML in Embedded SQL Applications: COBOL, PL/I, and C**
- **XML in CLI Applications**
- **Summary**



Similar to using variables of type "DB2Xml" in JDBC or .NET applications, you can and should use these XML data types for host variables in embedded SQL applications.

See also:

http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.xml/db2z_xmlembeddedsqldata.htm

XML Host Variables in COBOL

Declaration:

```
01  MYDOCUMENT USAGE IS SQL TYPE IS XML AS BLOB(1M).
```

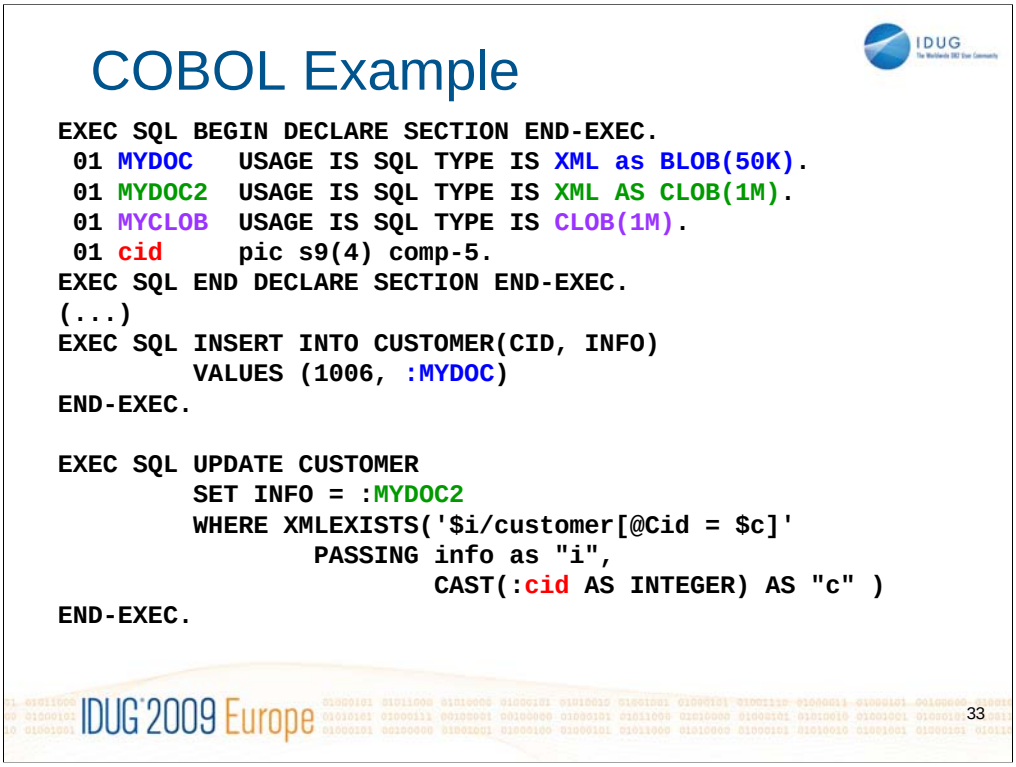
The precompiler produces:

```
01  MYDOCUMENT.
  02  MYDOCUMENT-LENGTH
      PIC 9(9) COMP.
  02  MYDOCUMENT-DATA.
      49  FILLER PIC X(32767).
      49  FILLER PIC X(32767).
      ⋮
      49  FILLER PIC X(32).
```

32 x 32767 bytes + 32 bytes = 1 MB

This is similar in PL/1, see

http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.xml/db2z_xmlembeddedsqldata.htm



This example shows:

1. Insert a document from an XML AS BLOB variable into an XML column.
2. Update an XML column with an XML AS CLOB variable. Uses host variable in the XMLEXISTS in the WHERE clause to select the row to update.

Both the INSERT and UPDATE statement could use any of the first 3 host variables (MYDOC, MYDOC2, MYCLOB) to provide the XML document that is being sent to the DB2 server. But, XML as XML AS BLOB is typically preferred.

COBOL Example *(continued)*


```
EXEC SQL SELECT XMLQUERY('$i/customer/addr' PASSING info AS "i")
         INTO :MYDOC
         FROM CUSTOMER
         WHERE XMLEXISTS('$i/customer[@Cid = 47]' PASSING info AS "i")
END-EXEC.
```

```
EXEC SQL SELECT XMLQUERY('$i/customer/addr' PASSING info AS "i")
         INTO :MYDOC2
         FROM CUSTOMER
         WHERE XMLEXISTS('$i/customer[@Cid = 47]' PASSING info AS "i")
END-EXEC.
```

```
EXEC SQL SELECT XMLSERIALIZE(XMLQUERY('$i/customer/addr'
                             PASSING info AS "i") AS CLOB(10K))
         INTO :MYCLOB
         FROM CUSTOMER
         WHERE XMLEXISTS('$i/customer[@Cid = 47]' PASSING info AS "i")
END-EXEC.
```


1. Retrieve the addr fragment of an XML document into an XML AS BLOB host variable. The XML data is in UTF-8 and has an XML declaration with encoding attribute. This is often the most desirable approach.

2. Retrieve the addr fragment of an XML document into an XML AS CLOB host variable. The XML data is converted to the application code page and has an XML declaration with encoding attribute.

3. Retrieve the addr fragment of an XML document into an CLOB host variable. The XML data is converted to the application code page but has no XML declaration!

The same concepts apply to embedded SQL in PL/1 or C applications.

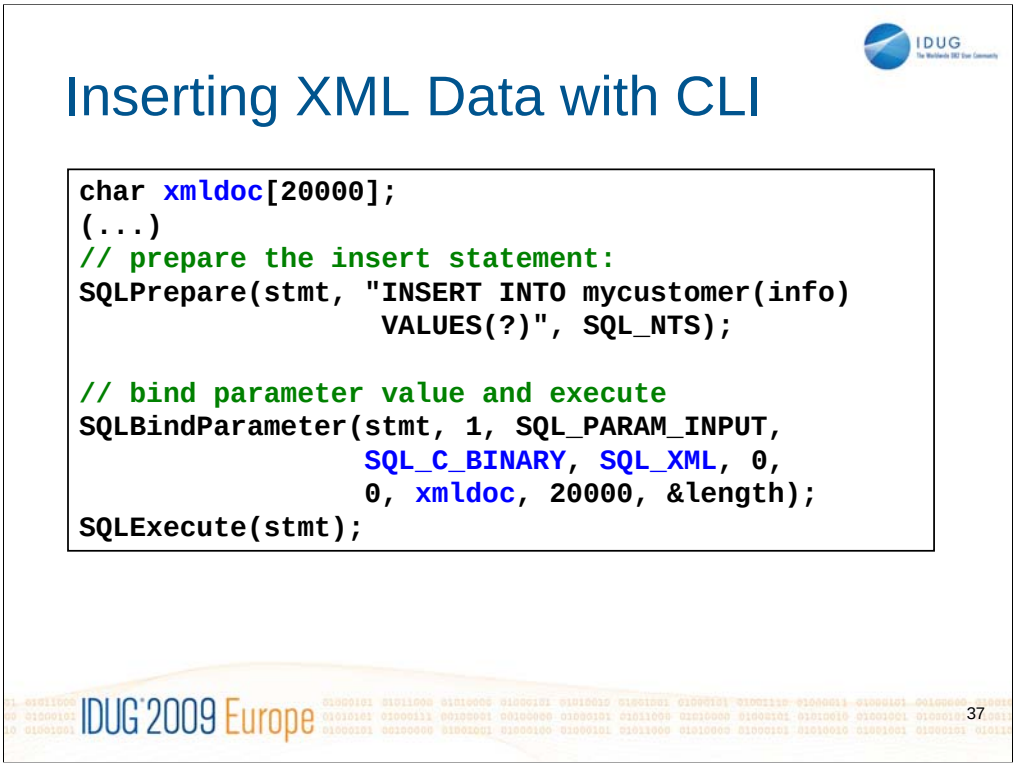
Agenda

- **Recap: pureXML in DB2 9.x**
- **Internal and External Encoding of XML Documents**
- **XML in Java Applications**
- **XML in .NET Applications**
- **XML in Embedded SQL Applications: COBOL, PL/I, and C**
- **XML in CLI Applications**
- **Summary**

XML in CLI Applications

- New CLI SQL type **SQL_XML**
- Insert/Retrieval: bind **SQL_XML** type to…
  - …binary C type **SQL_C_BINARY** (recommended)
  - …character types: **SQL_C_CHAR**, **SQL_C_DBCHAR**, etc.
- By default, retrieved XML data has XML declaration with encoding attribute
- To omit the XML declaration:
  - statement attribute **SQL_ATTR_XML_DECLARATION**
  - connection attribute **SQL_ATTR_XML_DECLARATION**
  - CLI/ODBC configuration keyword **XMLDeclaration** in the db2cli.ini

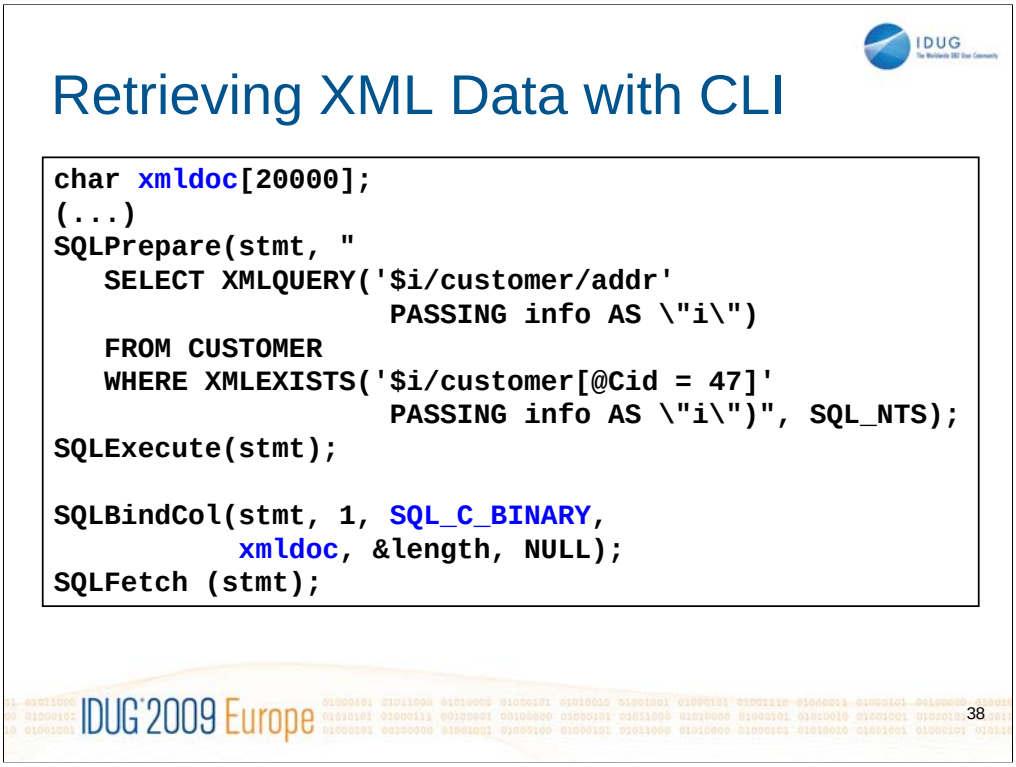
Again, as for the other languages and APIs, if you use binary data types then your XML data is internally encoded (recommended), and if you use character data types then your XML data is externally encoded and subject to possible code page conversion. Code page conversion can lead to problems if characters from one code page cannot be represented in another code page.



For more details, see:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/topic/com.ibm.db2.luw.apdv.cli.doc/doc/c0023367.html

http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/topic/com.ibm.db2.luw.apdv.cli.doc/doc/c0023369.html



For more details, see:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/topic/com.ibm.db2.luw.apdv.cli.doc/doc/c0023367.html

http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/topic/com.ibm.db2.luw.apdv.cli.doc/doc/c0023369.html

Agenda

- **Recap: pureXML in DB2 9.x**
- **Internal and External Encoding of XML Documents**
- **XML in Java Applications**
- **XML in .NET Applications**
- **XML in Embedded SQL Applications: COBOL, PL/I, and C**
- **XML in CLI Applications**
- **Summary**

Summary

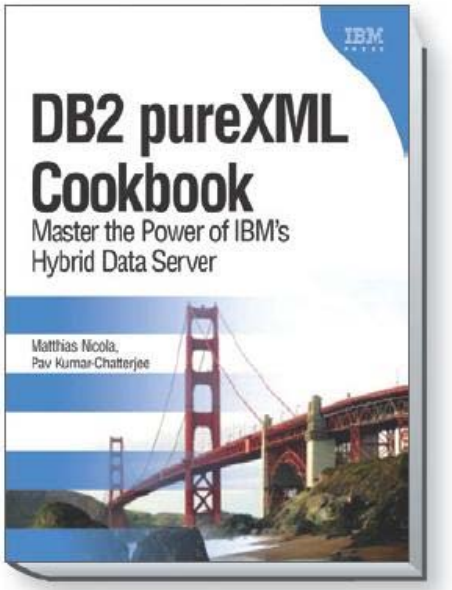
- Avoid XML parsing and manipulation in the application, as far as possible
- Let DB2 manipulate XML for you!
  → Performance, Simplicity, Maintainability
- All major DB2 APIs have been extended for XML
- Beware of external vs. internal encoding of XML data
- In your application, prefer binary data types rather than characters data types for XML

Also see:

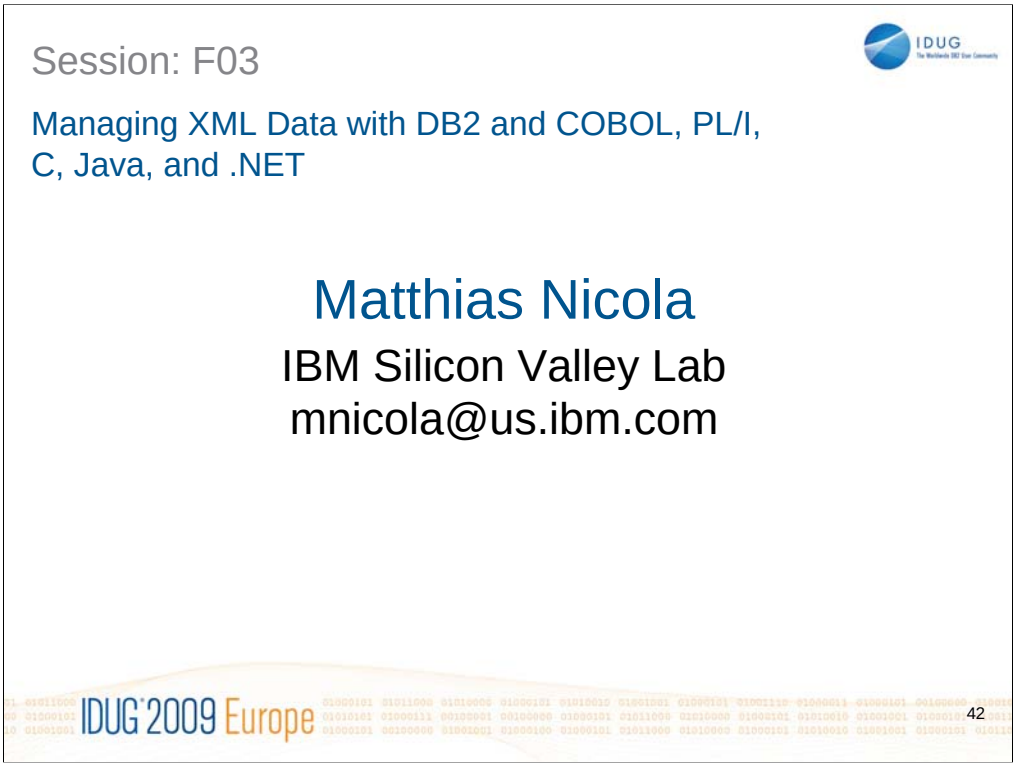
http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.xml/db2z_xmlappdev.htm

Further Reading

- More details on XML application development with
  DB2 for Linux, UNIX, Windows
  and
  DB2 for z/OS
- **http://tinyurl.com/pureXML**



The speaker notes throughout this slide deck contain links to specific related pages in the DB2 Information Center. Beyond the official documentation, the largest chapter in the "DB2 pureXML Cookbook" is the one on "Developing XML Applications with DB2" which contains more details and application code samples for Java, .NET, C, COBOL, PL/1, PHP, and Perl programmers.



Speaker:

Matthias Nicola is a senior software engineer at IBM's Silicon Valley Lab, in San Jose, CA, USA. He focuses on all aspects of XML in DB2. Matthias works closely with the DB2 pureXML development teams as well as with customers and business partners to help them design, optimize and implement XML solutions. Previously Matthias worked on data warehouse performance with Informix Software. (www.matthiasnicola.de)