



# DB2 pureXML

IBM Information Management Cloud Computing Center of Competence  
IBM Canada Lab

## Agenda

- Overview
- Inserting XML data
- XPath
- XQuery
- Querying XML data using SQL/XML
- Querying XML data using XQuery
- Update & Delete operations with XML
- XML Indexes
- XML Schema Validation
- Other XML support

05/30/2011

Template Documentation

2

© 2011 IBM Corporation

## Supporting reading material & videos

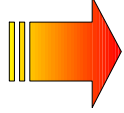
- **Reading materials**

- Getting started with DB2 Express-C eBook
  - Chapter 15: DB2 pureXML
- Getting started with IBM Data Studio for DB2 eBook
  - Chapter 4: Creating SQL and XQuery scripts

- **Videos**

- db2university.com course AA001EN
  - Lesson 11: DB2 pureXML

## Agenda



- Overview
- Inserting XML data
- XPath
- XQuery
- Querying XML data using SQL/XML
- Querying XML data using XQuery
- Update & Delete operations with XML
- XML Indexes
- XML Schema Validation
- Other XML support

05/30/2011

Template Documentation

4

© 2011 IBM Corporation

## What is XML?

- **eXtensible Markup Language**
  - XML is a language designed to describe data
- **A hierarchical data model**

```
<book>
  <authors>
    <author id="47">John Doe</author>
    <author id="58">Peter Pan</author>
  </authors>
  <title>Database systems</title>
</book>
```

### Characteristics of XML



05/30/2011

5

Template Documentation

© 2011 IBM Corporation

XML stands for eXtensible Markup Language. XML's popularity and use has grown exponentially in the past few years. Why is XML important?

- XML is the **foundation of web services**, and web services are the foundation of information on demand. **Information On Demand**, as its name implies, is making information available whenever it is requested. This can be made possible by providing information as a service.
- XML is also at the **core of Web 2.0** technologies.

XML is at the base of these concepts; without it, they would be hard to implement.

XML data uses a **hierarchical model**, which is most appropriate to store unstructured types of information.

XML has a set of specific characteristics:

- **Flexible**: easy to modify or adapt
- **Easy to extend**: you can create your own tags
- **Describes itself**: XML Schema (which itself is an XML document) provides rules and description to tags used in a document
- **Can be transformed to other formats**: e.g. to HTML, XSLT
- **Independent of the platform or vendor**
- **Easy to share**: easy to share with other applications since it can be stored as a text document

## Who uses XML?

### **Banking**

IFX, OFX, SWIFT, SPARCS,  
MISMO +++

### **Life Sciences**

MIAME, MAGE,  
LSID, HL7, DICOM,  
CDIS, LAB, ADaM +++

### **Retail**

IXRetail, UCCNET, EAN-UCC  
ePC Network +++

### **Healthcare**

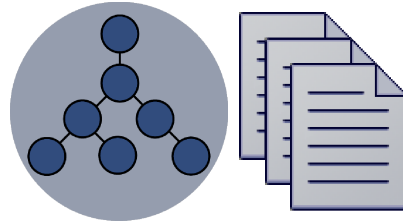
HL7, DICOM, SNOMED,  
LOINC, SCRIPT +++

### **Electronics**

PIPs, RNIF, Business Directory,  
Open Access Standards +++

### **Insurance**

ACORD  
XML for P&C, Life +++



### **Telecommunications**

eTOM, NGOSS, etc.  
Parlay Specification +++

### **Financial Markets**

FIX Protocol, FIXML, MDDL,  
RIXML, FpML +++

### **Automotive**

ebXML,  
other B2B Stds.

### **Energy & Utilities**

IEC Working Group 14  
Multiple Standards  
CIM, Multispeak

### **Cross Industry**

PDES/STEPml  
SMPI Standards  
RFID, DOD XML+++

### **Chemical & Petroleum**

Chemical eStandards  
CyberSecurity  
PDX Standard+++

05/30/2011

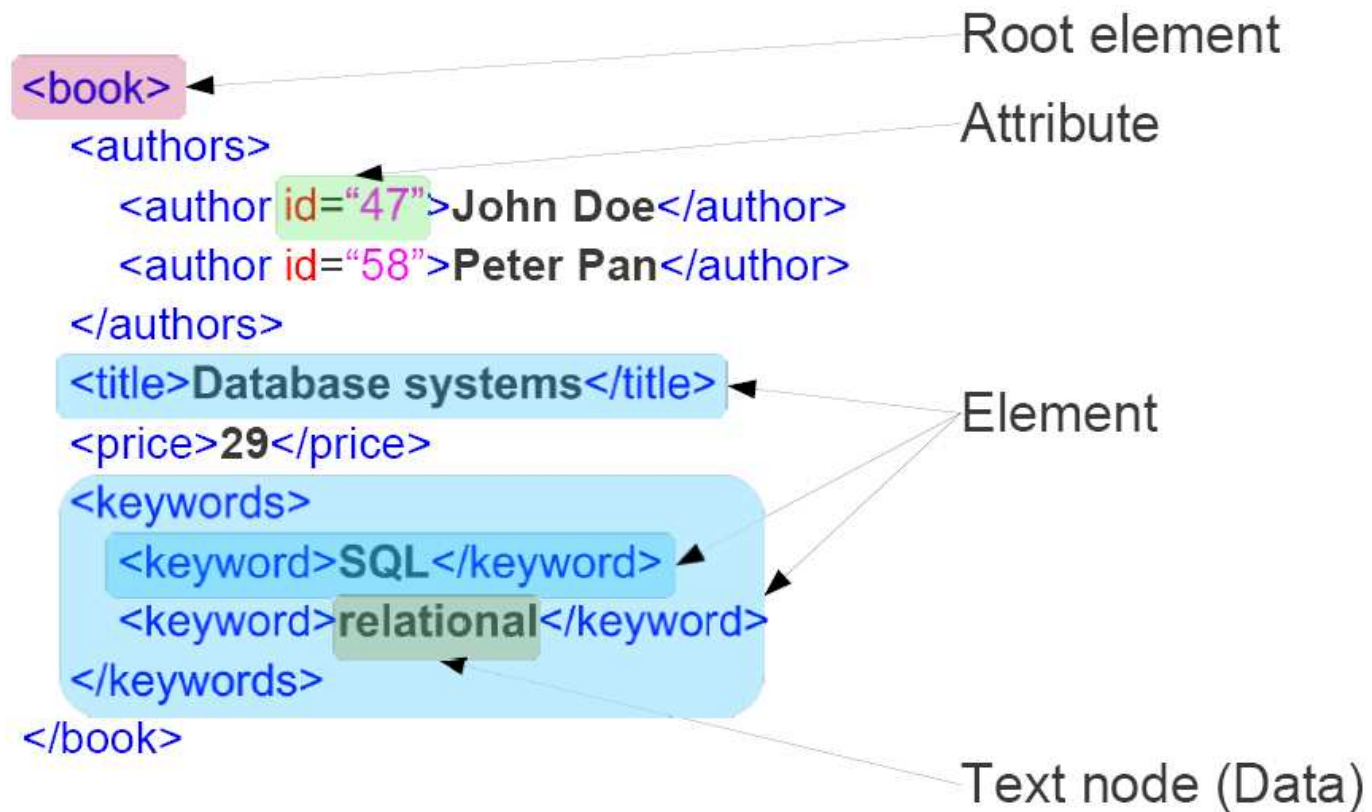
Template Documentation

6

© 2011 IBM Corporation

XML is used by many different industries, such as banking, healthcare, etc.

## XML document: Serialized representation



05/30/2011

Template Documentation

7

© 2011 IBM Corporation

XML is a language designed to describe data. It is comprised of nodes such as elements and attributes.

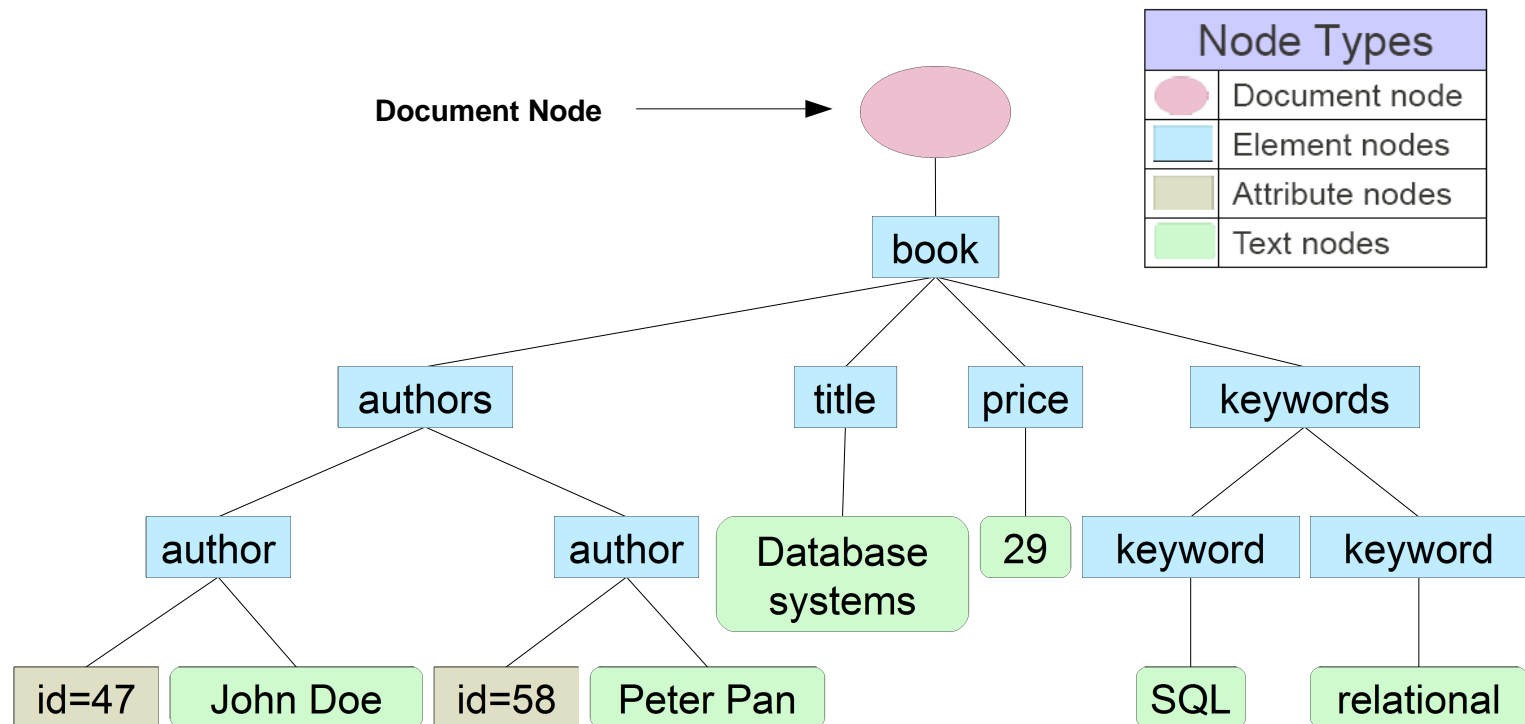
Its components are:

- element
- root element
- attribute
- text nodes

In this example:

- root element: `<book>`
- element: `<authors>`, `<author>`, `<title>`, `<price>`, `<keywords>`, `<keyword>`.
- attribute: `id="47"`, `id="58"`
- text nodes: John Doe, Peter Pan, Database systems, 29, SQL, relational

## XML document: Parsed-hierarchical representation



05/30/2011

Template Documentation

8

© 2011 IBM Corporation

This is just another example showing the different types of nodes that an XML can have:

- root
- element
- attribute
- text



## Well-formed vs. valid XML documents

- A **well-formed XML document** is a document that follows **basic rules**:
  - 1) It must have one and only one root element
  - 2) Each element begins with a start tag and ends with an end tag
  - 3) An element can contain other elements, attributes, or text nodes
  - 4) Attribute values must be enclosed in double quotes. Text nodes, on the other hand, should not.
- A **valid XML document** is **BOTH**:
  - 1) A well-formed XML document
  - 2) A document compliant with the rules defined in an XML schema document or a Document Type Definition (DTD) document.

05/30/2011

Template Documentation

9

© 2011 IBM Corporation

There are two concepts that people often get confused with regarding XML documents.

- well-formed XML documents
- valid XML documents

A **well-formed XML** document is a document that follows these basic rules.

- It must have one and only one root element.
- Each element begins with a start tag and ends with an end tag.
- An element can contain other elements, attributes, or text nodes.
- Attribute values must be enclosed in double quotes. Text nodes, on the other hand, should not.

e.g.:

- missing ending tag:

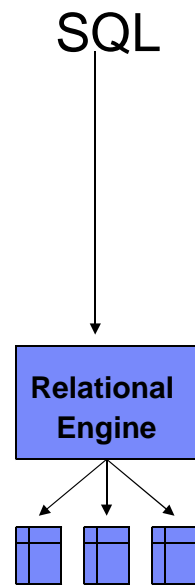
```
<employee>
  <name>John
</employee>
```
- no root node

```
<department></department>
<name>John</name>
```

A **valid XML** document is

- A well-formed XML document.
- A document compliant with the rules defined in an XML schema document or a Document Type Definition (DTD) document.

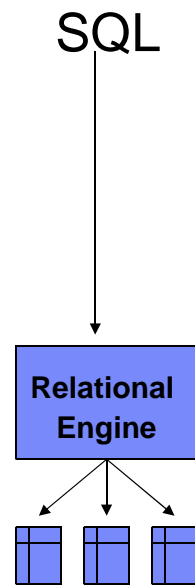
## pureXML overview



First we show a very simple overview of a relational management system on the left, where you issue SQL that is processed by a relational engine which access the data stored in tables.

## pureXML overview

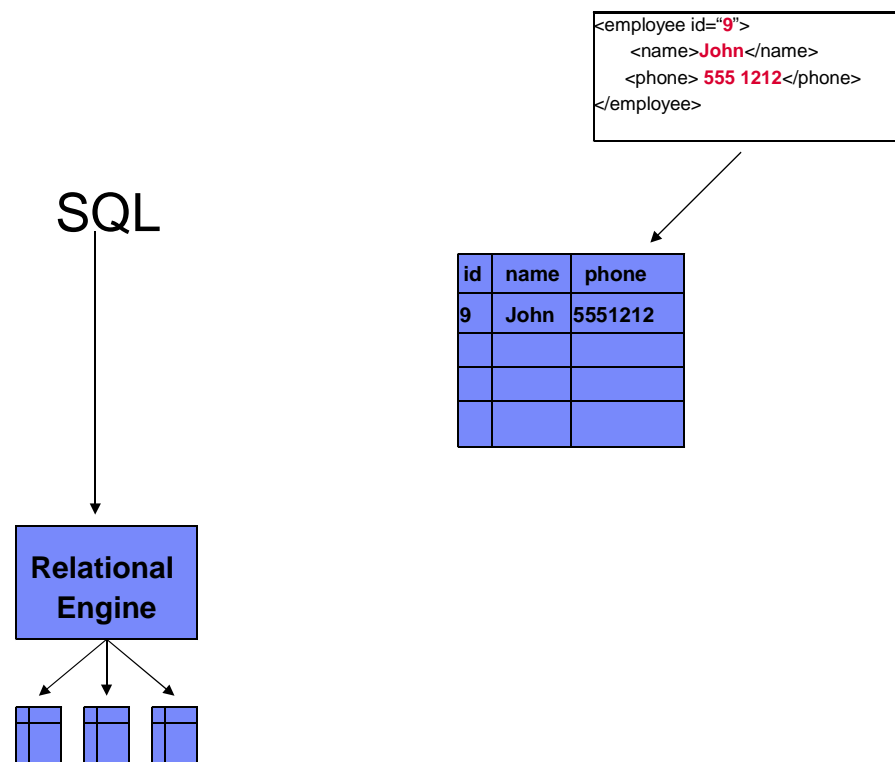
```
<employee id="9">  
  <name>John</name>  
  <phone>555 1212</phone>  
</employee>
```



Now, because of Web 2.0 and SOA we have an exponential growth in the usage of XML.

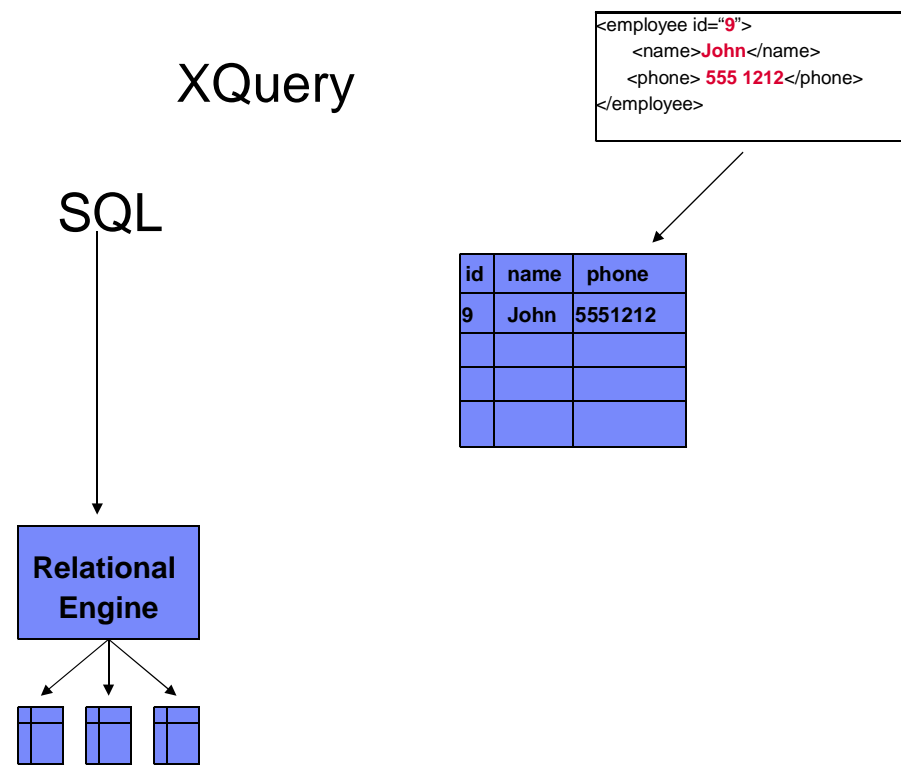
How is XML stored in many database products?

## pureXML overview



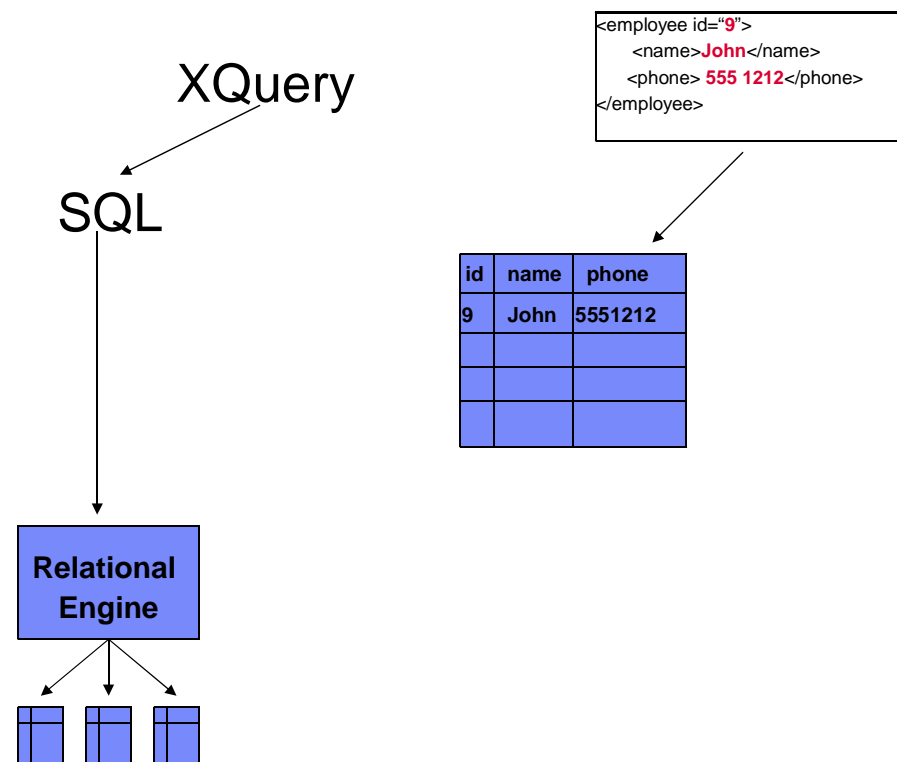
Well, they are mapping the XML document to table format (so converting it from hierarchical to relational format).

## pureXML overview



Next, when you issue an XQuery,

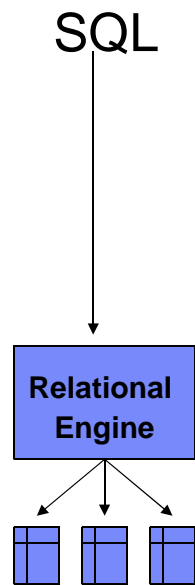
## pureXML overview



since Xquery cannot be understood by the relational engine, it has to be mapped to SQL first, behind the scenes.

## pureXML overview

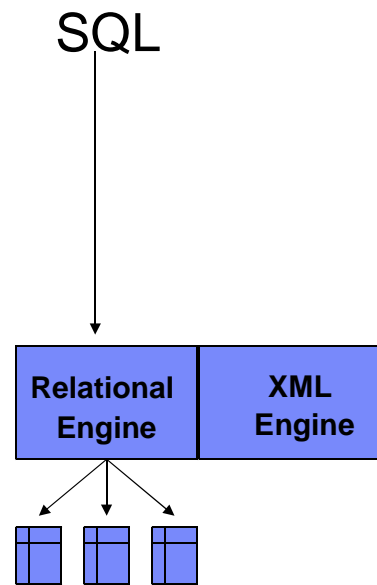
```
<employee id="9">  
  <name>John</name>  
  <phone> 555 1212</phone>  
</employee>
```



With DB2 pureXML technology, we don't do this.

## pureXML overview

```
<employee id="9">  
  <name>John</name>  
  <phone>555 1212</phone>  
</employee>
```



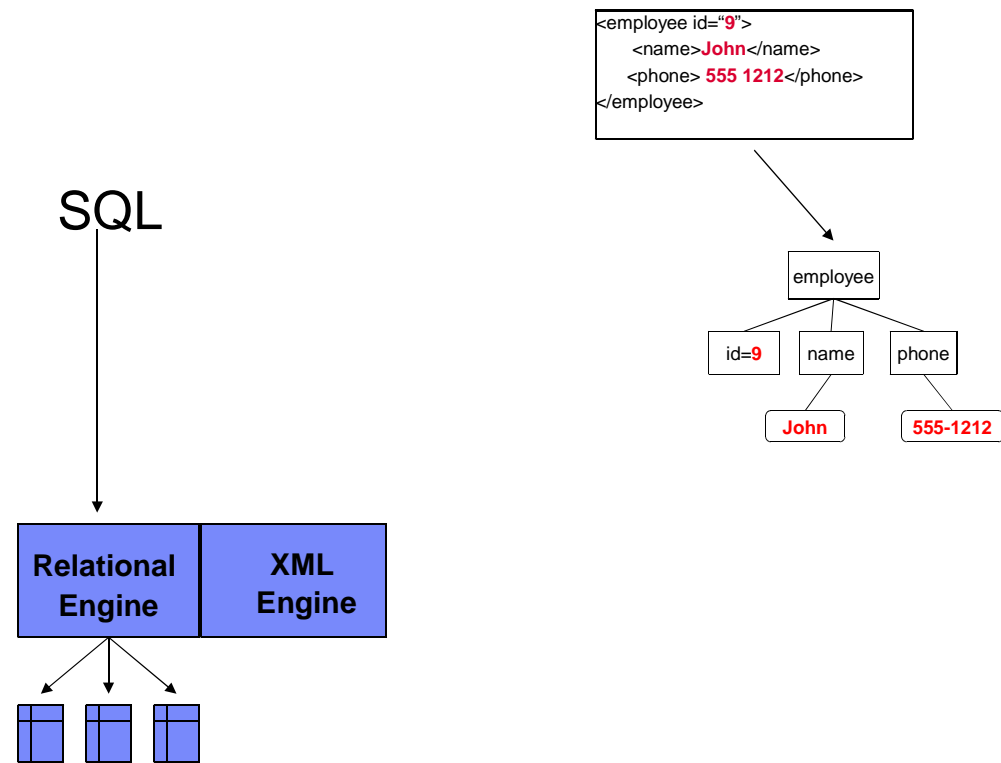
PureXML has two main characteristics:

1) DB2 has a second part for its engine that understands native XML

This means it can understand Xquery, so there's also no need to map to SQL



## pureXML overview

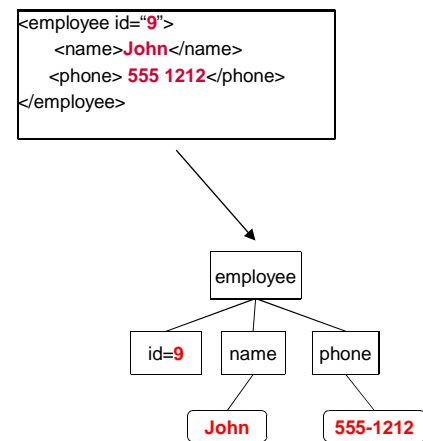
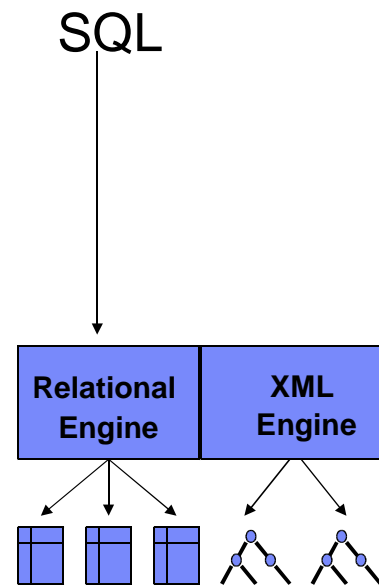


2) XML is stored in parsed-hierarchical format in the database.

So there is no mapping required. There is no conversion from hierarchical to relational. The XML is stored as a tree which is hierarchical in nature.

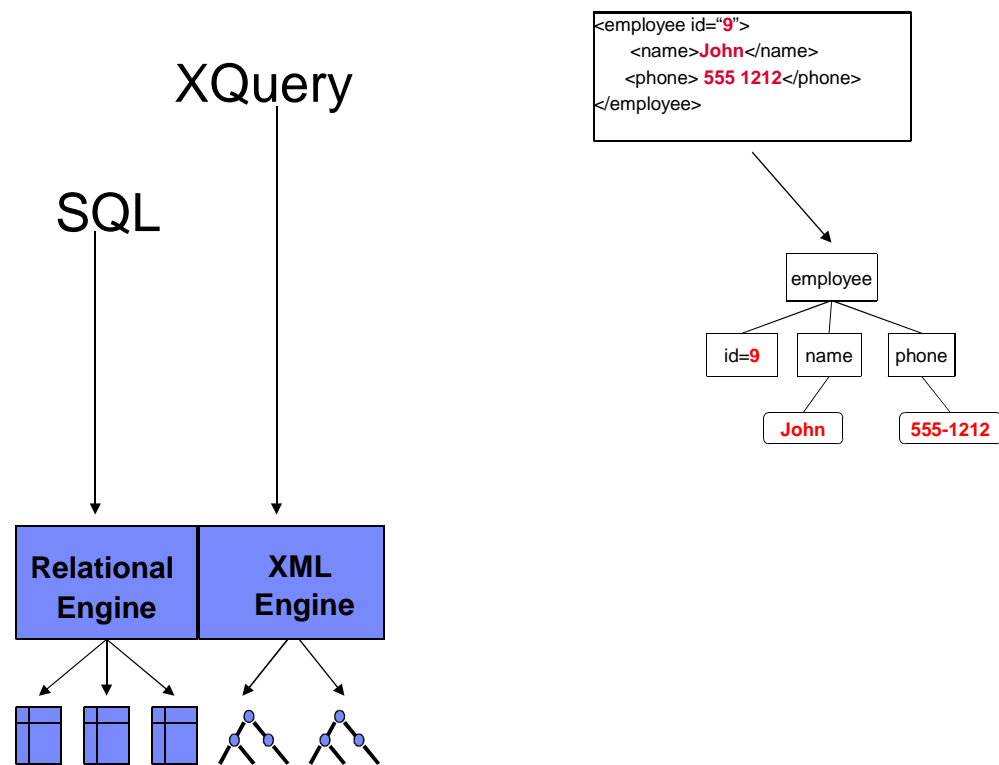
## pureXML overview

## XQuery



So now, if we issue a Xquery statement,

## pureXML overview



19 May 30, 2011 IBM Confidential

© 2011 IBM Corporation

It can be understood by the XML part of the engine without any mapping required.

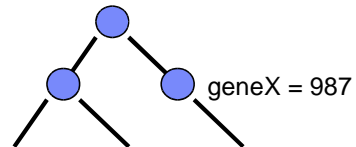
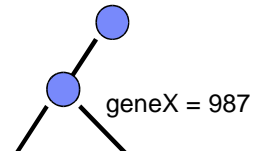
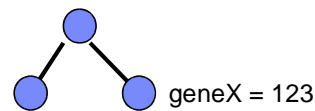
In summary, the storage format = the processing format

Therefore, performance will be better, there will be less code, and easier maintenance.

Note as well that the two parts of the engine (relational and XML) can talk between each other, so I can combine SQL with Xquery as we will see next.

## Integration of XML and relational data

### PATIENTS table

NAME	SEX	AGE	COFFEE	SMOKE	DNA
Elizabeth	F	65	35	15	
Raul	M	47	10	0	
...					
Tina	F	58	4	10	

05/30/2011

Template Documentation

20

© 2011 IBM Corporation

Integrating relational and XML data in the same query can also lead to useful information. For example, let's say I am a medical researcher and I'm trying to find the cure of cancer. Let's say I've been doing this for the past 25 years where I've been storing information about my patients in a table, where I have relational data like the name of the patient, the sex of the patient, the age, how many years this patient has been drinking coffee, how many years this patient has been smoking, and I'm also storing the DNA of each patient on the last column. And I'm storing DNA information as an XML document because XML is good to store unstructured or semi-structured type of information like DNA. I am representing the XML document as a tree because an XML document is hierarchical and a tree is hierarchical in nature. Now let's assume that all cancer patients or most of my cancer patients have a gene X with a value of 987.

So with DB2 I can integrate relational data like coffee and smoking, which are some factors that may or may not cause cancer, and this gene X with a value of 987, which seems to appear in most cancer patients. Let's see how we can do this on the next slide.

## SQL with XQuery/XPath

```
SELECT name from PATIENTS
WHERE
xmlexists('$p/Patient/MedicalDetail[geneX="987"]'
          passing PATIENTS.DNA as "p")
and sex = 'F'
and age > 55
and coffee > 15
and smoke > 10
;
```

05/30/2011

21

Template Documentation

© 2011 IBM Corporation

Running this query we can perform this integration.

The first two lines and the last four lines are just SQL as usual where I'm selecting the name from the patient's table and I'm doing some filtering based on the sex, the age, coffee, and smoking. The third and the fourth line is what are new with DB2. First, we call the XMLEXISTS function which is part of the SQL/XML standard. This function allows me to perform some test to see if a condition is satisfied in the XML document, and I use this as another filter for my query.

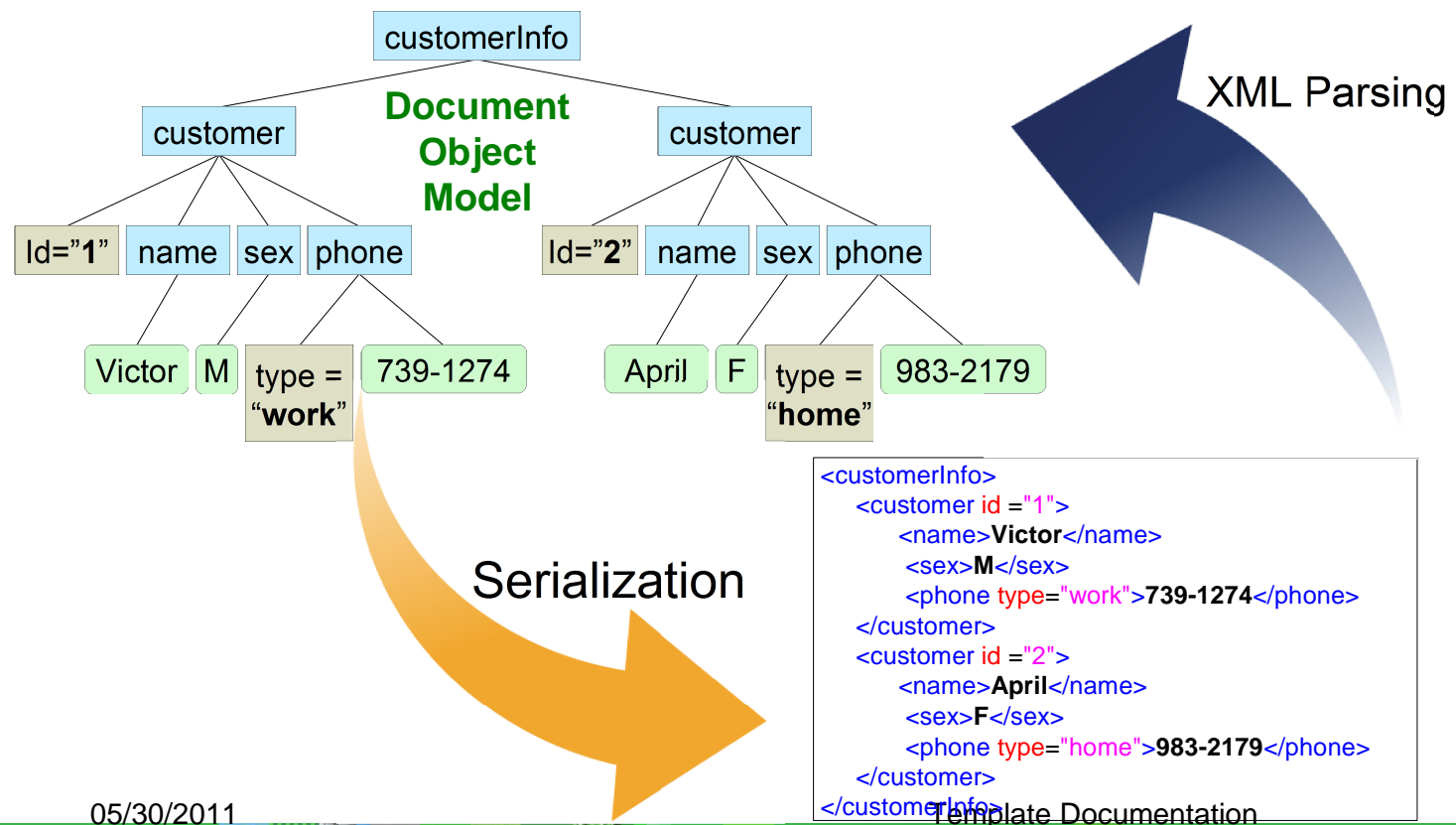
The \$p is a variable (anything with '\$' means it's a variable), and this variable is defined in the 4<sup>th</sup> line, where I'm assigning PATIENTS.DNA to 'p'. PATIENTS.DNA is the XML column, so I'm basically passing to 'p' the XML document. Then going back to line 3, in \$p/Patient/MedicalDetail... I'm using Xpath to traverse the XML document until I test if geneX = '987'.

So I am combining SQL with XPATH or XQUERY, or mixing the relational model with the hierarchal model of XML.

The end results is that I may be able to find some correlation between several factors like drinking coffee or smoking with the appearance of this gene, and this type of queries may help find the cure of cancer.

## Native XML storage

- Documents are stored in parsed representation



05/30/2011

22

© 2011 IBM Corporation

When the XML document is inserted into the database, DB2 parses the XML document and stores it internally as a parsed tree using the XQuery Data Model (XDM). This tree is persistent.

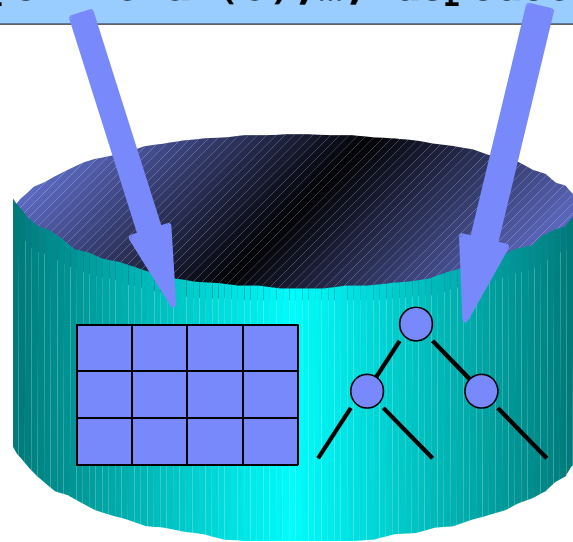
If you want to do the opposite process (going from a parsed-tree format to a serialized format), this is called **serialization**

## Native XML storage

- XML stored in parsed hierarchical format

```
create table dept (deptID char(8),..., deptdoc xml);
```

- Relational columns are stored in relational format (tables)
- XML columns are stored natively
- XML stored in UTF8



05/30/2011

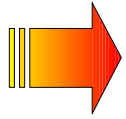
23

Template Documentation

© 2011 IBM Corporation

To store XML documents, you create a table as usual, but for XML documents, you need to create a column defined with the XML data type. So in the example, there is a table “dept” which has a relational column “deptID” defined as char(8); and then we have a column “deptdoc” defined as XML.

## Agenda



- Overview
- Inserting XML data
- XPath
- XQuery
- Querying XML data using SQL/XML
- Querying XML data using XQuery
- Update & Delete operations with XML
- XML Indexes
- XML Schema Validation
- Other XML support

05/30/2011

Template Documentation

24

© 2011 IBM Corporation



## Table definitions with XML columns

```
create table items (  
  id          int primary key not null,  
  brandname   varchar(30),  
  itemname    varchar(30),  
  sku         int,  
  srp         decimal(7,2),  
  comments    xml  
);
```

```
create table clients(  
  id          int primary key not null,  
  name        varchar(50),  
  status      varchar(10),  
  contact     xml  
);
```

05/30/2011

Template Documentation

25

© 2011 IBM Corporation

Let's say we have these two tables that we are going to create, the **items** table and the **clients** table, where the last column is defined with the XML data type for both tables. XML columns do NOT need to be defined on the last column of a table, we just put them there in this particular example.

## XML documents to load to tables “client” and “item”

- Assume these files are in C:\DB2workshop

The screenshot displays a file explorer window with a list of files in the 'C:\DB2workshop' directory. The files are:

- Client3227.xml
- Client4309.xml
- Client5681.xml
- Client8877.xml
- Client9077.xml
- Client9177.xml
- ClientInfo.xsd
- clients.del
- Comment3926.xml
- Comment4023.xml
- Comment4272.xml
- items.del
- table\_creation.txt

Arrows point from the XML files to a Notepad window titled 'clients.del - Notepad'. The Notepad window shows the following XML content:

```
<?xml version="1.0" ?>
- <Client xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
- <Address>
  <street>5401 Julio Ave.</street>
  <city>San Jose</city>
  <state>CA</state>
  <zip>95116</zip>
</Address>
- <phone>
  <work>4084630000</work>
  <home>4081111111</home>
  <cell>4082222222</cell>
</phone>
<fax>4087776666</fax>
<email>love2shop@yahoo.com</email>
</Client>
```

Below the Notepad window, a table lists the files and their sizes:

File Name	Size	Type
Client3227.xml	1 KB	XML Document
clients.del	1 KB	DEL File
table_creation.txt	2 KB	Text Document

05/30/2011

Template Documentation

26

© 2011 IBM Corporation

This figure shows several files under the directory “C:\DB2workshop”.

The file “CLIENT. DEL” is a Delimited ASCII file which has data separated by commas (the delimiter). If we pick the first row in this file, and the last column in that row, we see something like “<XDS FIL='Client3227.xml'>”. This is a pointer to an XML file, and the contents of this file are also shown in the figure.

Similarly there is an items.del file, and corresponding XML files.

Basically we are explaining in this figure what we plan to insert into the tables we created earlier.

## Inserting XML documents

- **1st method: A simple SQL INSERT**

- Can be implicitly or explicitly

- **Implicit XML parsing: XML document entered as a string**

```
INSERT INTO clients VALUES (77, 'John Smith', 'Gold',  
    '<addr>111 Main St., Dallas, TX, 00112</addr>') ;
```

- **Explicit XML parsing with the XMLPARSE function**

- Tell system how to treat whitespaces (strip/preserve)

- Default is 'Strip WHITESPACE'

```
INSERT INTO clients VALUES (77, 'John Smith', 'Gold',  
    xmlparse (document '<addr>111 Main St., Dallas, TX,  
    00112</addr>' preserve whitespace) );
```

05/30/2011

Template Documentation

27

© 2011 IBM Corporation

There are two methods to insert an XML document in a table:

1) Use a simple SQL INSERT.

You can do this implicitly or explicitly.

Implicitly means that you simply pass the XML document as a string, and because it is being inserted into an XML column, DB2 will know it has to parse it.

Explicitly means you must tell DB2 with the XMLPARSE function to parse the XML document. You can also indicate how to treat white spaces.

## Inserting XML documents

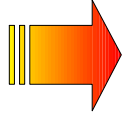
- 2nd method: Use the DB2 IMPORT utility

```
IMPORT from "C:\DB2workshop\clients.del" of del  
xml from "C:\DB2workshop" INSERT INTO  
CLIENTS (ID, NAME, STATUS, CONTACT);
```

```
IMPORT from "C:\DB2workshop\items.del" of del  
xml from "C:\DB2workshop" INSERT INTO  
ITEMS (ID, BRANDNAME, ITEMNAME, SKU, SRP, COMMENTS);
```

The second method is to use the IMPORT utility. Going back two slides we showed the contents of the C:\DB2workshop directory. Now the files in that directory are used in the IMPORT command.

## Agenda



- Overview
- Inserting XML data
- XPath
- XQuery
- Querying XML data using SQL/XML
- Querying XML data using XQuery
- Update & Delete operations with XML
- XML Indexes
- XML Schema Validation
- Other XML support

05/30/2011

Template Documentation

29

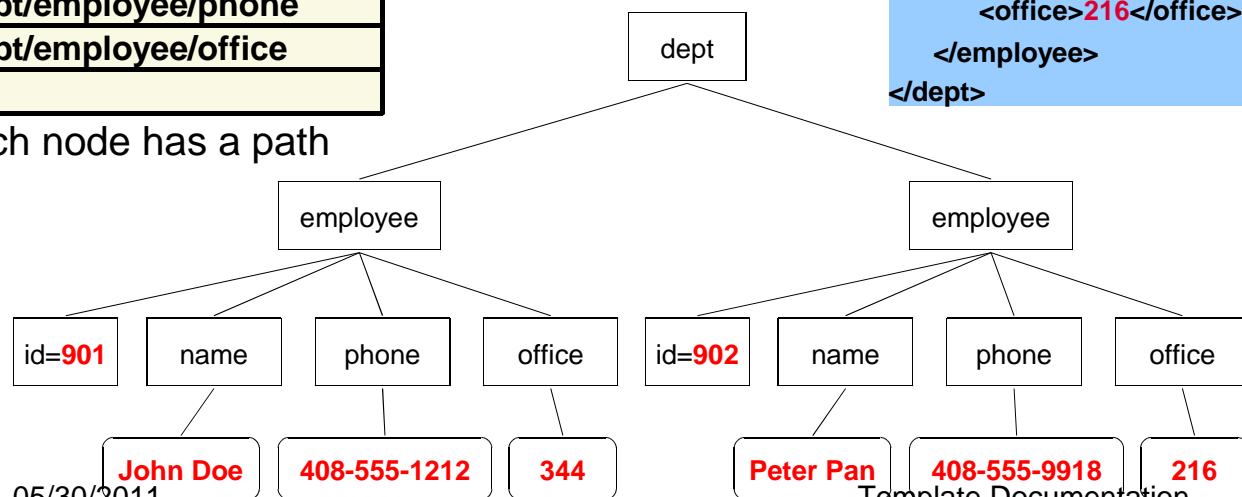
© 2011 IBM Corporation

## XPath

- XML Query Language
- Subset of XQuery & SQL/XML

/
/dept
/dept/employee
/dept/employee/@id
/dept/employee/name
/dept/employee/phone
/dept/employee/office
(...)

Each node has a path



```

<dept bldg="101">
  <employee id="901">
    <name>John Doe</name>
    <phone>408 555 1212</phone>
    <office>344</office>
  </employee>
  <employee id="902">
    <name>Peter Pan</name>
    <phone>408 555 9918</phone>
    <office>216</office>
  </employee>
</dept>
  
```

05/30/2011

30

Template Documentation

© 2011 IBM Corporation

XPath is a language that you can use to navigate an XML document. It is not something proprietary of IBM, but it's a standard. It's basically a query language. What we have here on the right is an XML document in serialized representation.

Below is the same document represented as a tree, also known as a parsed hierarchical representation. So these two representations are exactly the same thing, and in this case, you can think of the parsed-hierarchical representation as the way that DB2 stores the XML document inside the database, in a persistent way.

Some XPath expressions are shown at the top left corner. XPath is fairly easy to learn; it is very similar to the **cd** (**change directory**) command that you used in MS-DOS, Windows, Linux, Unix, etc. For example, you go:

CD /directory/subdirectory/subdirectory, so basically you are using the **cd** command to navigate a tree, which is hierarchical in nature.

The same thing happens with XPath, you go, for example, **/dept/employee/name**; that means you are going to the dept element first, or the root, then you go to employee, then you go to name and you will get this information which is John Doe. Now in this case, there are, as you can see, two employees, so you will get both, you will get all the way to John Doe and all the way to Peter Pan.

## XPath: Simple expressions

- Use fully qualified paths to specify elements/attributes
- “@” is used to specify an attribute
- use “text()” to specify the text node under an element

```
<dept bldg="101">
  <employee id="901">
    <name>John Doe</name>
    <phone>408 555 1212</phone>
    <office>344</office>
  </employee>
  <employee id="902">
    <name>Peter Pan</name>
    <phone>408 555 9918</phone>
    <office>216</office>
  </employee>
</dept>
```

XPath	Result
/dept/@bldg	101
/dept/employee/@id	901 902
/dept/employee/name	<name>John Doe</name> <name>Peter Pan</name>
/dept/employee/name/text()	Peter Pan John Doe

05/30/2011

Template Documentation

31

© 2011 IBM Corporation

There are different other things you can do with XPath. There are several XPath expressions to choose exactly what you want, so we will go through some of them very quickly.

Assuming that the top right XML document is the one I'm working on, if I do a:

**/dept/@bldg**, the @ symbol means that I want the attributes, that's why it is retrieving 101, because that's the attribute 101.

Then for **/dept/employee**, attribute id, the same thing. In this case the attribute will be 901. And because there two employees, there is also 902, so that's why I am getting both attributes , 901 and 902.

A **/dept/employee/name**, is the same example I provided in the previous slide. Note that what I am getting as well is the tag name, and the ending tag, that is just the way it works, the tags will be included.

If you don't want the tags to be included, then you can add the function called **text** at the end of the XPath, so that you only get the values, as you can see in the last example expression.

## XPath: Wildcards

- \* matches any tag name
- // is the “descendent-or-self” wildcard

```
<dept bldg="101">
  <employee id="901">
    <name>John Doe</name>
    <phone>408 555 1212</phone>
    <office>344</office>
  </employee>
  <employee id="902">
    <name>Peter Pan</name>
    <phone>408 555 9918</phone>
    <office>216</office>
  </employee>
</dept>
```

XPath	Result
/dept/employee/*/text()	John Doe 408 555 1212 344 Peter Pan 408 555 9918 216
/dept/*/@id	901 902
//name/text()	John Doe Peter Pan
/dept//phone	<phone>408 555 1212</phone> <phone>408 555 9918</phone>

05/30/2011

Template Documentation

© 2011 IBM Corporation

On this slide, using the same example XML document as before, if you use:

`/dept/employee/*`, asterisk is a wildcard and it's basically any one element. It will go dept, employee and will get all of these: John Doe, 408, 344, and then the same for the other employee.

The second row in the example is similar: Go to dept, whatever element, and get the attribute id, 901 and 902.

For the third row, two slashes means that it's not just one element, but a number of elements up to when you reach name. So for example, here is name, any elements above, I don't care what they are, any elements. And then get me the text: that's Peter Pan, John Doe.

Finally, the last row of examples: `/dept//phone`, means any elements between dept and phone.



## XPath: Predicates

- Predicates are enclosed in square brackets [...]
- Can have multiple predicates in one XPath
- Positional predicates: [n] selects the n-th child

```
<dept bldg="101">
  <employee id="901">
    <name>John Doe</name>
    <phone>408 555 1212</phone>
    <office>344</office>
  </employee>
  <employee id="902">
    <name>Peter Pan</name>
    <phone>408 555 9918</phone>
    <office>216</office>
  </employee>
</dept>
```

XPath	Result
/dept/employee[@id="902"]/name	<name>Peter Pan</name>
/dept[@bldg="101"]/employee[office > "300"]/name	<name>John Doe</name>
//employee[office="344" OR office="216"]/@id	901 902
/dept/employee[2]/@id	902

05/30/2011

Template Documentation

33

© 2011 IBM Corporation

Using square brackets is analogous to the WHERE clause in SQL. For example, for the first row, you say /dept/employee, where attribute id is 902, and from there it takes the name. That's why it gives Peter Pan.

In the expression in the second row you have two conditions

In the expression in the 3<sup>rd</sup> row, we have two slashes, so whatever elements there are before employee, then where office equals 344, or office equals 216, and from there take the attribute id.

The last example in the 4<sup>th</sup> row has a [2]. This means I want the second employee.. If it was one, it will be "I want the first employee"

## XPath: Parent axis

- Current context: “.”
- Parent context: “..”

```
<dept bldg="101">
  <employee id="901">
    <name>John Doe</name>
    <phone>408 555 1212</phone>
    <office>344</office>
  </employee>
  <employee id="902">
    <name>Peter Pan</name>
    <phone>408 555 9918</phone>
    <office>216</office>
  </employee>
</dept>
```

XPath	Result
/dept/employee/name[../@id="902"]	<name>Peter Pan</name>
/dept/employee/office[.>"300"]	<office>344</office>
/dept/employee[office > "300"]/office	<office>344</office>
/dept/employee[name="John Doe"]/../@bldg	101
/dept/employee/name[.="John Doe"]/../../@bldg	101

05/30/2011

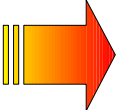
Template Documentation

34

© 2011 IBM Corporation

This is similar to when you use the change directory command where you can use “.” or “..” which indicate the current context or the parent context.

## Agenda

- Overview
- Inserting XML data
- XPath
- • XQuery
- Querying XML data using SQL/XML
- Querying XML data using XQuery
- Update & Delete operations with XML
- XML Indexes
- XML Schema Validation
- Other XML support

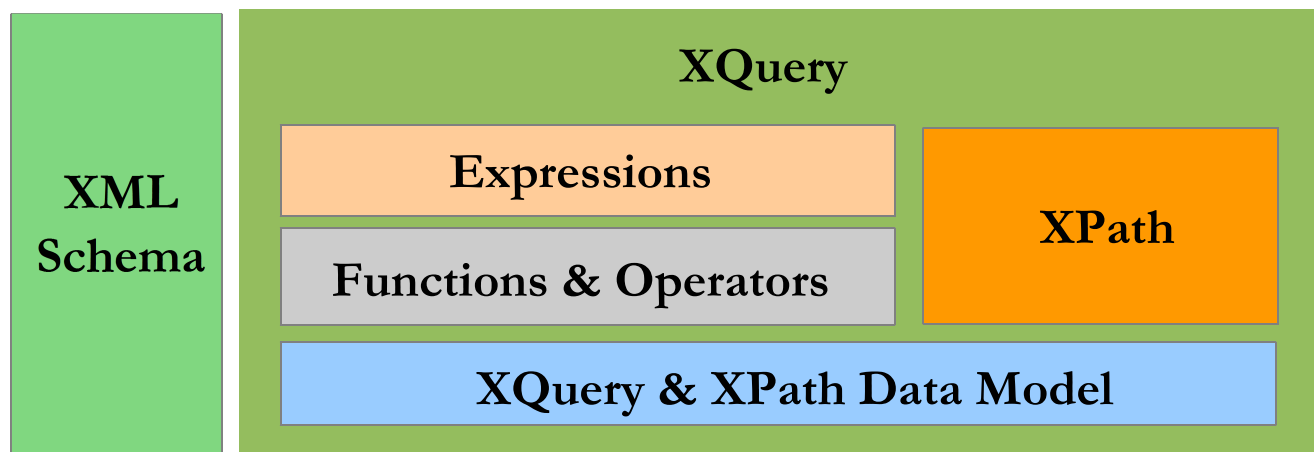
05/30/2011

Template Documentation

35

© 2011 IBM Corporation

## What is XQuery?



- **XQuery supports path expressions to navigate XML**
- **XQuery supports both typed and untyped data**
- **XQuery lacks null values because XML documents omit missing or unknown data**
- **XQuery returns sequences of XML data**

05/30/2011

Template Documentation

36

© 2011 IBM Corporation

XQuery is a superset of XPath. XQuery is used to navigate an XML document. There are other components shown in this chart that are also part of XQuery. Some of them are discussed in more detail later on.

XQuery returns a sequence of XML data. In XQuery there are no NULL values. Instead, blanks are used.

## XQuery: The FLWOR expression

- **FOR:** iterates through a sequence, bind variable to items
- **LET:** binds a variable to a sequence
- **WHERE:** eliminates items of the iteration
- **ORDER:** reorders items of the iteration
- **RETURN:** constructs query results

05/30/2011

37

Template Documentation

© 2011 IBM Corporation

Within XQuery, we have something called the FLWOR expression, F-L-W-O-R, and with that expression you can do more manipulations. FLWOR stands for: For, Let, Where, Order, Return.

FLWOR is to Xquery what in SQL would be a SELECT-FROM-WHERE ORDER BY clause.

## XQuery: The FLWOR expression

```
create table dept
  (deptID char(8),deptdoc xml);

xquery
  for $d in
    db2-fn:xmlcolumn( 'DEPT.DEPTDOC' ) /dept
  let $emp := $d//employee/name
  where $d/@bldg > 95
  order by $d/@bldg
  return
    <EmpList>
      {$d/@bldg, $emp}
    </EmpList>
```

Input:

```
<dept bldg="101">
  <employee id="901">
    <name>John Doe</name>
    <phone>408 555 1212</phone>
    <office>344</office>
  </employee>
  <employee id="902">
    <name>Peter Pan</name>
    <phone>408 555 9918</phone>
    <office>216</office>
  </employee>
</dept>
```

05/30/2011

38

Template Documentation

© 2011 IBM Corporation

Let's look at this example. At the top right corner I have an XML document we will use as input. At the top we have the table definition of the **dept** table with a relational column, and an XML column where we assume have stored the XML document to be used as input for the XQuery.

You can also see the FLWOR expression. I have to always prefix this by XQuery, because if we don't put XQuery at the beginning, then DB2 will assume by default that we are using SQL, or that you are going to run an SQL statement.

Let's look at the FLWOR expression: **"for \$d in, etc"**. here, I am getting into variable "d" the XML document that is provided with the function db2-fn:xmlcolumn;

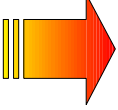
**"let \$emp..."**, I am assigning a new variable called **emp**, the value of \$d, which was the entire XML document, //employee/name,

**"where \$d..."**, here are some conditions which are later **ordered by**

Then what I want to return is **Emplist**, so what exactly will appear is \$ d@bldg, so it will use 101, and then \$emp, whatever name was involved, so John Doe, and Peter Pan as well as part of this, and close with Emplist.

You can use the FLWOR expression to change the format of your XML document. For example, you can make it follow the rules or the syntax of RSS or Atom. Then that way you could create a feed out of this XML.

## Agenda

- Overview
- Inserting XML data
- XPath
- XQuery
-  • Querying XML data using SQL/XML
- Querying XML data using XQuery
- Update & Delete operations with XML
- XML Indexes
- XML Schema Validation
- Other XML support

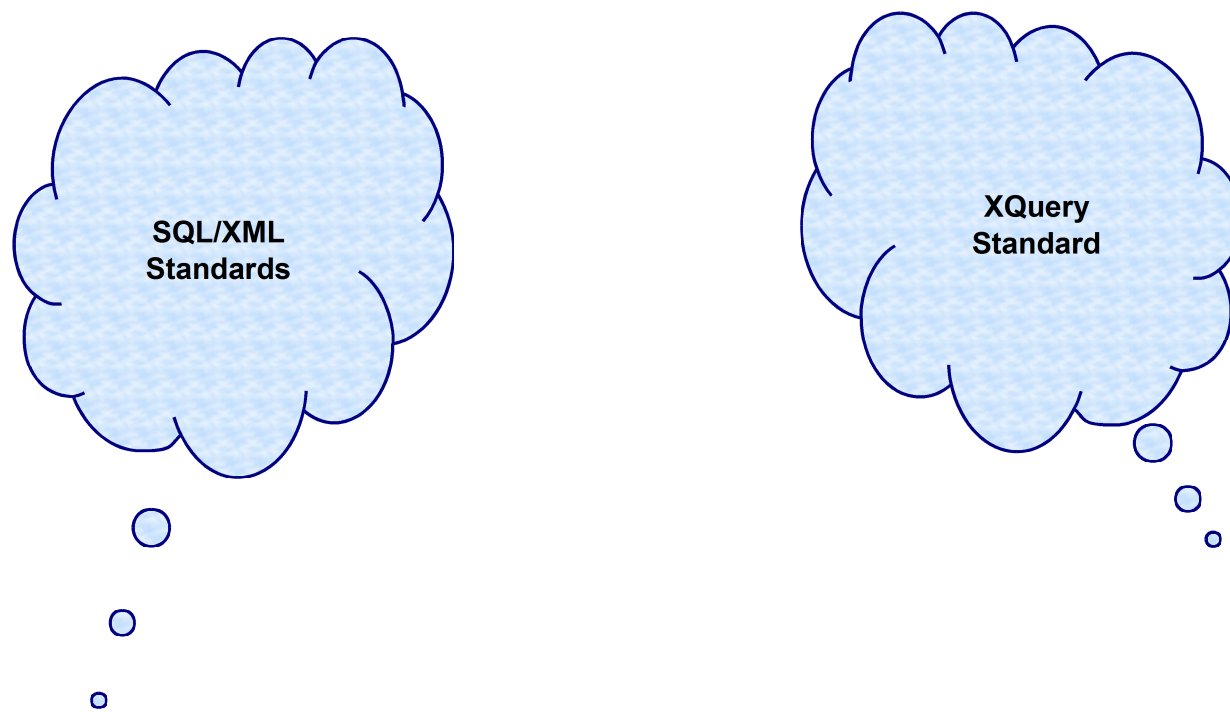
05/30/2011

Template Documentation

39

© 2011 IBM Corporation

## Two worlds



05/30/2011

40

Template Documentation

© 2011 IBM Corporation

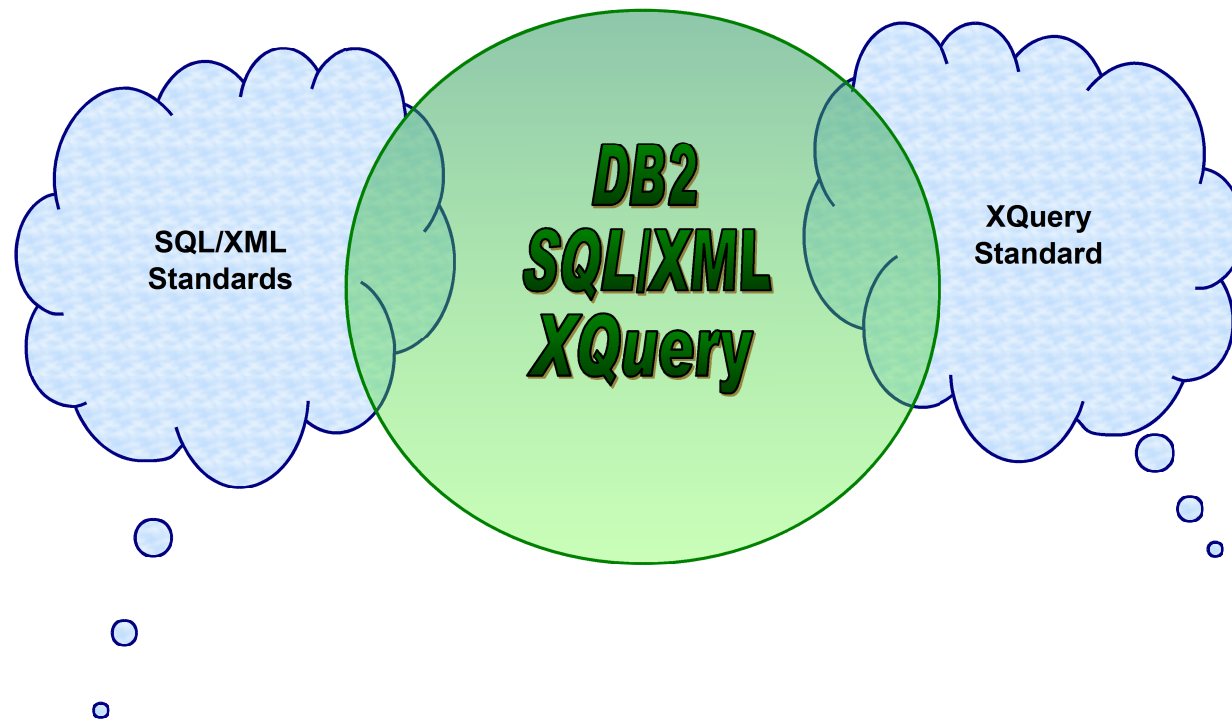
Plain SQL statements enable you to retrieve full XML documents but you cannot specify XML-based query predicates, and you cannot retrieve partial XML documents. For example, if I just had SQL, I would only be able to get the entire XML document:

If you do a “**select deptdoc from dept**” where deptdoc is an XML column, you will retrieve all the XML document in that table, so a simple SQL statement like this can work with XML. However, what if you only want part of the XML document to be retrieved? Or what if you want to use part of the XML document as a condition in the WHERE clause? In those cases you cannot just use SQL, you need other standards:

1. SQL/XML (which is an extension to SQL that includes XML functions so it works as a bridge between the SQL and XML world), or
2. Xquery (A superset of XPath)



## Two worlds



05/30/2011

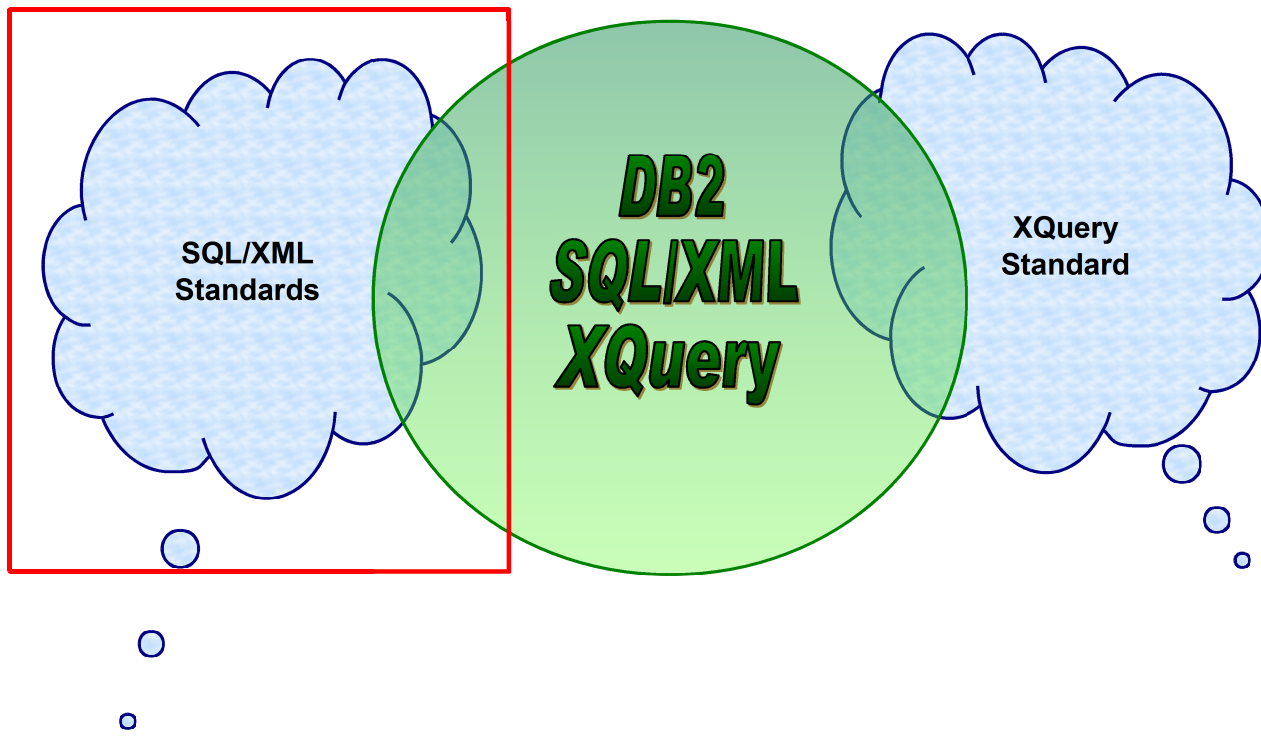
41

Template Documentation

© 2011 IBM Corporation

DB2 supports both, SQL/XML and Xquery

## Two worlds



05/30/2011

42

Template Documentation

© 2011 IBM Corporation

Let's start describing the SQL/XML standard first

## SQL/XML functions

Function	Description
<b>XMLPARSE</b>	Parses character/BLOB data, produces XML value
<b>XMLSERIALIZE</b>	Converts an XML value into character/BLOB data
<b>XMLVALIDATE</b>	Validates XML value against an XML schema and type-annotates the XML value
<b>XMLEXISTS</b>	Determines if an XQuery returns results (i.e. a sequence of one or more items)
<b>XMLQUERY</b>	Executes an XQuery and returns the result sequence
<b>XMLTABLE</b>	Executes an XQuery, returns the result sequence as a relational table (if possible)
<b>XMLCAST</b>	Cast to or from an XML type

05/30/2011

43

Template Documentation

© 2011 IBM Corporation

In SQL/XML, which is a part of the new SQL 2006 standard, that includes XML functions, there are different functions that we can use. Here is a list and description of some popular SQL/XML functions.

XMLPARSE, we talked about XMLPARSE when we were doing that insert;

XMLSERIALIZE is the opposite of XMLPARSE;

XMLVALIDATE for validation of an XML document vs. an XML schema. We will talk a little more about this at the end of this presentation.

XMLEXISTS, XMLQUERY, XMLTABLE etc., we will show some examples of these functions in the next few slides.

## XMLEXISTS function

- Use it on the **WHERE** clause to filter rows based on an XML element value
- Syntax 1: Explicitly create a variable to hold the XML document

```
select name from clients
where
  xmlexists( '$c/Client/Address[zip="95116"] '
             passing CLIENTS.CONTACT as "c" )
```

- Syntax 2: A variable is automatically created with the name of the XML column

```
select name from clients
where
  xmlexists( '$CONTACT/Client/Address[zip="95116"] ' )
```

05/30/2011

Template Documentation

44

© 2011 IBM Corporation

XMLEXISTS function allows me to perform a test based on the XML column, and that way I can restrict some number of rows, for example, here I say **select name from clients, where, XMLexists, \$ c, Client, address, zip, 95116**, passing CLIENTS.CONTACT as c. So basically the first two lines are just SQL as usual; I want the name from clients.

then the third line uses **XMLexists**. The **\$ c** is a **variable**. Anything with a **\$** represents a variable. Where is the variable defined? This variable is defined on the fourth line, the one that says **passing CLIENTS.CONTACT as c**.

**CONTACT** is the column that contains the XML document, and **CLIENT** is the name of the table

So now, the variable **c** has the XML documents. And then the **/Client/Address**, is XPath, where square brackets are used like a where clause in SQL, to test that zip is 95116.

Basically the XMLexists function allows me to filter some rows based on an element value, which in this case is zip.

So I want the all the rows, where in the XML document the zip element is 95116. So that's what we are doing with this example.


With DB2 9.5 the syntax has been simplified. In the example below we go **select name from client where XMLexist**, and we have **\$ CONTACT**. This is a variable that is automatically created with the same name as the column name that contains the XML.. There is no need to assign a value to it in the syntax anymore.

## Case sensitivity - Caution!

- SQL is not case sensitive
- XPath and XQuery are case sensitive!
- DB2 objects (tables/columns) need to be put in upper case.
- Example:


Incorrect:

```
select name from clients
where
xmlexists( '$contact/Client/Address[zip="95116"]' )
```



Correct:

```
select name from clients
where
xmlexists( '$CONTACT/Client/Address[zip="95116"]' )
```



05/30/2011

Template Documentation

45

© 2011 IBM Corporation

SQL is not case sensitive, so when you are working with SQL, you don't care about the case; however, with XPath and XQuery, you do have to be careful about it. And if you combine SQL with XQuery/XPath, then the SQL part can still be in any case, but for the XPath and XQuery part you must take into account the case.

So, for example this `select name from clients` part could have been written in any case, and then `XMLexists` could also be in any case, but for whatever is inside this function, is case sensitive. For example, **Client**, if you had put the **C** in lowercase, then you would have probably received an incorrect result. There would not be an error because the syntax is ok, but DB2 would not be able to find this path, and then you get an incorrect result.

Now, why is **contact** (in lowercase) incorrect, and **CONTACT** (in uppercase) correct? What happens is that DB2 objects, that is, tables, columns, etc. have to be put in uppercase. because when you create an object like a table in DB2, it is normally going to be stored in uppercase regardless of the case used in the `create table` statement. Of course you can use double quotes to force a DB2 table to be created in lowercase, but this is not recommended in general because it will just add complexity to handling the tables.

This is why every time you invoke the name of a column or a table within the Xpath/XQuery part of the query, it needs to be in uppercase.

**Note:** If you work with Data Studio, Data Studio creates objects exactly as how you typed them. We suggest you use uppercase when creating objects in Data Studio.

Another thing to consider is the use of quotes. Ensure they are straight quotes, and not curly quotes. Programs such as MS-Word/Powerpoint tend to change straight quotes to curly quotes automatically, so if you copy/paste a statement from those programs, you may have problems with the quotes.

## XMLQUERY function

- Retrieve one or more element values from the XML document

```
select  
xmlquery( '$CONTACT/Client/email' )  
  from clients  
 where status = 'Gold'
```

05/30/2011

46

Template Documentation

© 2011 IBM Corporation

The XMLQUERY function allows me to retrieve an XML element, like email, as a column. In the example, the email is treated as a column, but actually it is an element of the XML document.

And then you say **from client where status = 'Gold'**. In this case, I am using a relational column to filter the rows that I want, and I am getting an XML element back.

In the previous example, going back here, in the previous example, we were totally doing the opposite, we were using an element zip to do the filtering of the rows, and we were getting a relational column back.

If you try this query out, when you get the result it will say 3 records are retrieved, but you will only see one. This happens because the first two are blank. This was mentioned briefly earlier where it was said that in XML, there is not such thing as a NULL. In XML, you get blanks.

## Retrieve XML data using the FLWOR expression

```
SELECT name,  
       xmlquery ('for $e in $CONTACT/Client/email[1]  
                return $e')  
FROM clients  
WHERE status = 'Gold'
```

This is another example that uses XMLQUERY, but inside the function, we are using the FLWOR expression. We are using For. Any time you see For, it is a FLWOR expression. It does not have to be complete. You don't actually need to use FLWOR expression in this example, you could have just returned the email without the **For** \$ e and return \$ e, but we are just putting this for illustration purposes, so that you can see that you can put a FLWOR expression within the XMLQUERY.



## Retrieving and transforming XML into HTML

```
SELECT
    xmlquery('for $e in $CONTACT/Client/email[1]/text()
              return <p>{$e}</p>')
FROM clients
WHERE status = 'Gold'
```

05/30/2011

48

Template Documentation

© 2011 IBM Corporation

in this example note we have <p> and </p>. This is HTML and \$e is going to return XML. So this example illustrates you can embed/combine HTML and XML together.



## XMLTABLE: From XML to relational

```
select t.comment#,i.itemname,t.customerID,Message
  from items i,
  xmltable ('$COMMENTS/Comments/Comment'
    columns Comment# integer      path 'CommentID',
           CustomerID integer     path 'CustomerID',
           Message  varchar(100) path 'Message') as t
```

05/30/2011

49

Template Documentation

© 2011 IBM Corporation

This function called XMLtable is used when you want to go from XML to relational. So for example, you have your system that works with the relational model, then your company bought a software product from another company that works in XML, and now you want the two systems to talk. Potentially you can convert the XML that comes from the other system to relational by treating it as a table. So here we are invoking the XMLtable function, and we are going to go all the way to the Comment element, and then from there we are going to say the columns that I want to define in this table are Comment#, defined as integer, and the last element in the path is going to be CommentID.

Comments/Comment/CommentID. Then, for CustomerID, the same thing: integer, path and then CustomerID. For message, the same thing; although it's a Message.

And then I give an alias, which is t. So basically we are doing **select, etc. from items**, and **from t** as well, from two tables.

## XMLEMENT: From relational to XML

```
select
  xmlement (name "item", itemname),
  xmlement (name "id", id),
  xmlement (name "brand", brandname),
  xmlement (name "sku", sku)
from items
where srp < 100
```

- Some other functions that work with XMLEMENT are:

**XMLNAMESPACES:** Specifies a namespace for the element being created

**XMLATTRIBUTES:** Specifies an attribute for the element

**XMLCONCAT:** Concatenates elements

05/30/2011

Template Documentation

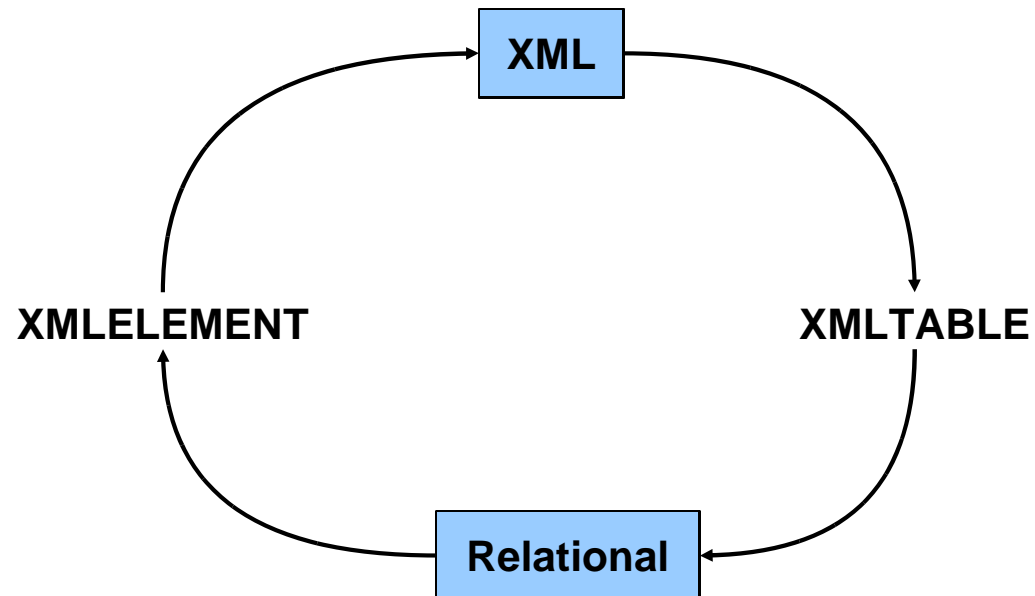
50

© 2011 IBM Corporation

Moving on, we have the XMLelement function which is similar to, or basically does the opposite of the XMLtable. This one goes from relational to XML, so you have all information in tables and you can convert them into XML document. So XMLelement is one of these functions that you can use for XML creation.

There are other functions listed as well like XMLNAMESPACES, XMLATTRIBUTES, etc. that will help build the XML doc.

## XMLTABLE vs XMLELEMENT



05/30/2011

51

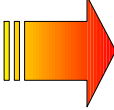
Template Documentation

© 2011 IBM Corporation

This figure summarizes the purpose of the last two functions:

XMLtable can be used to go from XML to relational, while XMLElement is to go from relational to XML.

## Agenda

- Overview
- Inserting XML data
- XPath
- XQuery
- Querying XML data using SQL/XML
-  • Querying XML data using XQuery
- Update & Delete operations with XML
- XML Indexes
- XML Schema Validation
- Other XML support

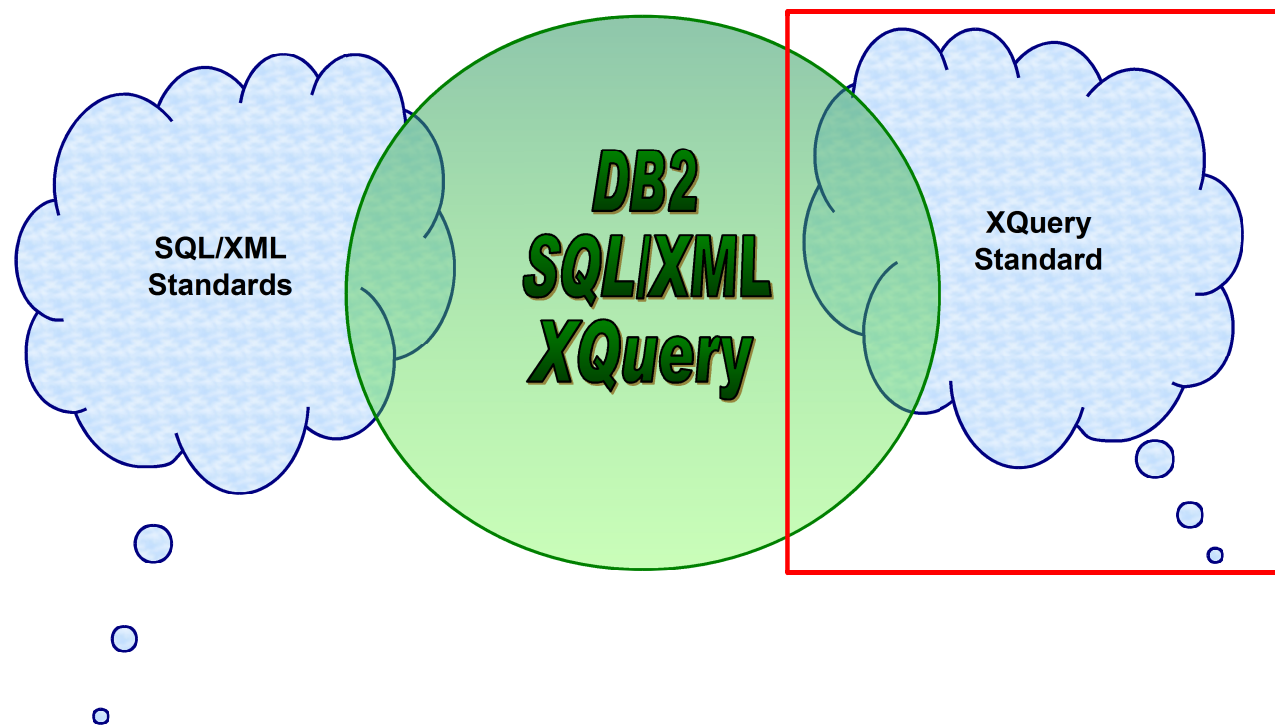
05/30/2011

Template Documentation

52

© 2011 IBM Corporation

## Two worlds



05/30/2011

Template Documentation

53

© 2011 IBM Corporation

Let's now talk about the second part, which is how to query XML data using the XQuery standard

## XMLCOLUMN

- **xmlcolumn** is a function with a parameter that identifies the table name and column name of an XML column.
- Simple XQuery to return customer contact data:

```
xquery
db2-fn:xmlcolumn( 'CLIENTS.CONTACT' )
```

- Adding an additional filtering predicate

```
xquery
db2-fn:xmlcolumn
  ( 'CLIENTS.CONTACT' ) /Client/Address[zip="95116"]
```

05/30/2011

54

Template Documentation

© 2011 IBM Corporation

In XQuery, everything has to be prefixed with the keyword XQuery. As said earlier, if you don't put this keyword, DB2 will assume it's SQL.

The XMLCOLUMN function is used to retrieve all the content of the XML column, so this actually will be equivalent to issuing a **select CONTACT from clients** if I am just using SQL.

However, in SQL that's all I can do with XML, that is, SQL can let me retrieve the entire XML document, but not just part of it. With XMLCOLUMN function, you can retrieve part of the XML document.

## XMLCOLUMN - More examples

- FLWOR expression with XMLCOLUMN to retrieve client fax data:

```
xquery
  for $y in db2-fn:xmlcolumn('CLIENTS.CONTACT')/Client/fax
  return $y
```

- Sample output:

```
<fax>4081112222</fax>
<fax>5559998888</fax>
```

- Querying DB2 XML data and returning results as HTML

```
xquery
<ul> {
  for $y in db2-fn:xmlcolumn('CLIENTS.CONTACT')/Client/Address
  order by $y/zip
  return <li>{$y}</li>
}</ul>
```

05/30/2011

Template Documentation

55

© 2011 IBM Corporation

This is another example using the XMLCOLUMN function inside a FLWOR expression (at the top)

At the bottom you have another FLWOR expression, and the return is using HTML (<ul><li></li></ul>) combined with XML

## XMLCOLUMN - More examples

```
<ul>
<li>
<address>
  <street>9407 Los Gatos Blvd.</street>
  <city>Los Gatos</city>
  <state>ca</state>
  <zip>95302</zip>
</address>
</li>
<Address>
  <street>4209 El Camino Real</street>
  <city>Mountain View</city>
  <state>CA</state>
  <zip>95302</zip>
</address>
</li>
...
</ul>
```

05/30/2011

Template Documentation

56

© 2011 IBM Corporation

This is the sample output of the previous example



## SQLQUERY: Embedded SQL with XQuery

- A function which executes a SQL query and returns only the selected data
- The result set from the query passed to db2-fn:sqlquery must return XML data
- This XML data can then be further processed by XQuery

```
xquery
for $y in
db2-fn:sqlquery('select comments from items where srp > 100')/Comments/Comment
where $y/ResponseRequested='Yes'
return (
  <action>
    {$y//ProductID}
    {$y//CustomerID}
    {$y//Message}
  </action>
)
```

05/30/2011

Template Documentation

57

© 2011 IBM Corporation

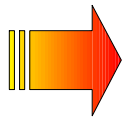
**SQLQUERY** allows you to embed SQL inside XQuery. So before, when working with SQL/XML you would start with a SELECT and within this SELECT you would invoke XMLexists, XMLquery etc, so XML functions were embedded in SQL.

Now what I am using is XQuery at the top level and I'm embedding SQL.

This query is selecting comments from items where srp>100, and from there it will maybe select, let's say we have 200 rows. The comments column must be an XML column. From there we go to /Comments/Comment, so I am going down navigating the elements. Of course these names may not be very good for illustration purposes, but these are different elements, then from there I continue with the XQuery.

## Agenda

- Overview
- Inserting XML data
- XPath
- XQuery
- Querying XML data using SQL/XML
- Querying XML data using XQuery
- Update & Delete operations with XML
- XML Indexes
- XML Schema Validation
- Other XML support



## DELETE operations with XML

- Deletes a row based on a condition that uses an XML element
- Use a SQL DELETE statement
- A DELETE first searches for the document, and then deletes it. The search part can be done using the same SQL/XML functions as when querying data

```
delete from clients
where
  xmlexists( '$c/Client/Address[zip="95116"] '
             passing CLIENTS.CONTACT as "c" )
```

- To delete XML data, see the SQL UPDATE statement next

05/30/2011

Template Documentation

59

© 2011 IBM Corporation

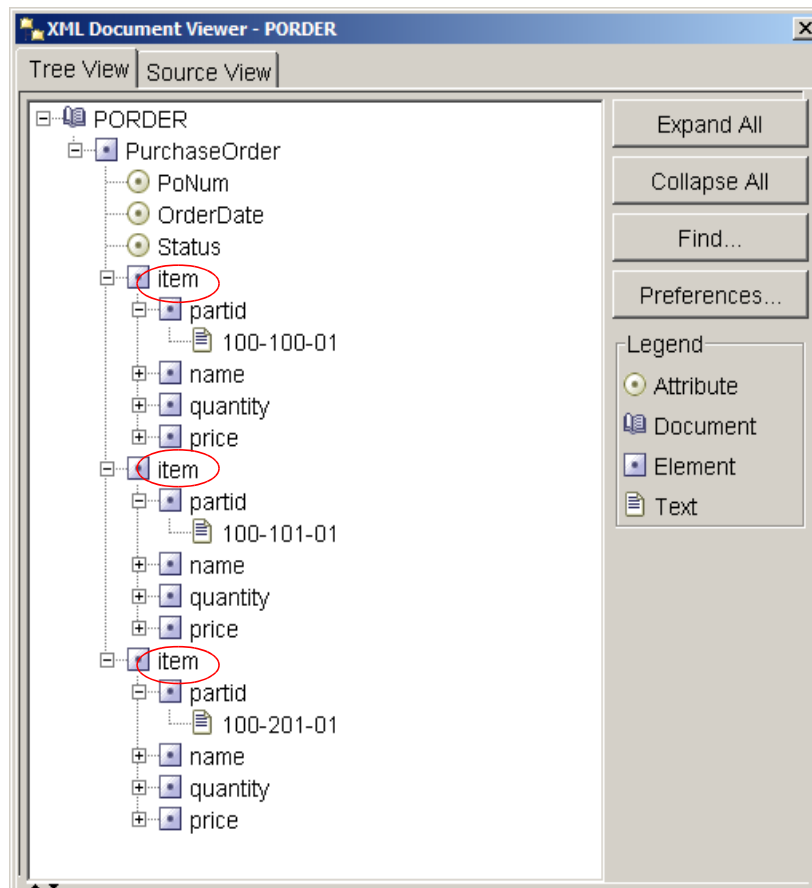
To delete XML data, you can simply use SQL and use an update statement where the XML column is set to NULL..

If you want to delete rows of a table based on a condition found in the XML documents, you can use the SQL/XML functions discussed earlier. An example of this case is shown above.

If you want to delete part of the XML document, what you are actually doing is “updating” the XML document, and this is covered in the “Update” section using the TRANSFORM expression.

## UPDATE operations with XML

- Use the **TRANSFORM** expression
- Let's say you have this XML document (From **SAMPLE** database, purchaseorder table)



05/30/2011

60

© 2011 IBM Corporation

This is an example to show how UPDATE operations work in XML. Here you can see there are three item elements in this XML document.

## UPDATE example

- Adding an element to the end of the document

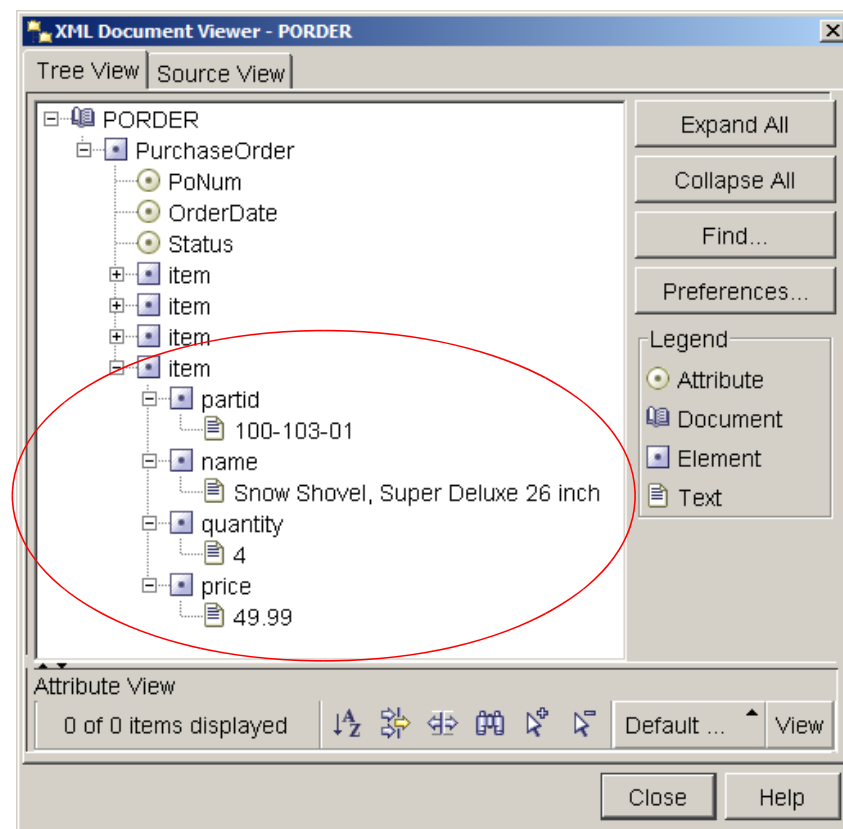
```
UPDATE purchaseorder SET porder =
    xmlquery('transform
              copy $po := $order
              modify do insert

document { <item>
            <partid>100-103-01</partid>
            <name>Snow Shovel, Super Deluxe 26 inch</name>
            <quantity>4</quantity>
            <price>49.99</price>
          </item>
          }
into $po/PurchaseOrder
return $po'
passing purchaseorder.porder as "order")
WHERE poid=5012;
```

Using this example, we are updating the XML document to add one more item.

## UPDATE example

- **After adding the element to the end:**



05/30/2011

Template Documentation

62

© 2011 IBM Corporation

This figure now shows 4 items after executing the update, where the last item added has been expanded, while the other are collapsed.

## UPDATE example

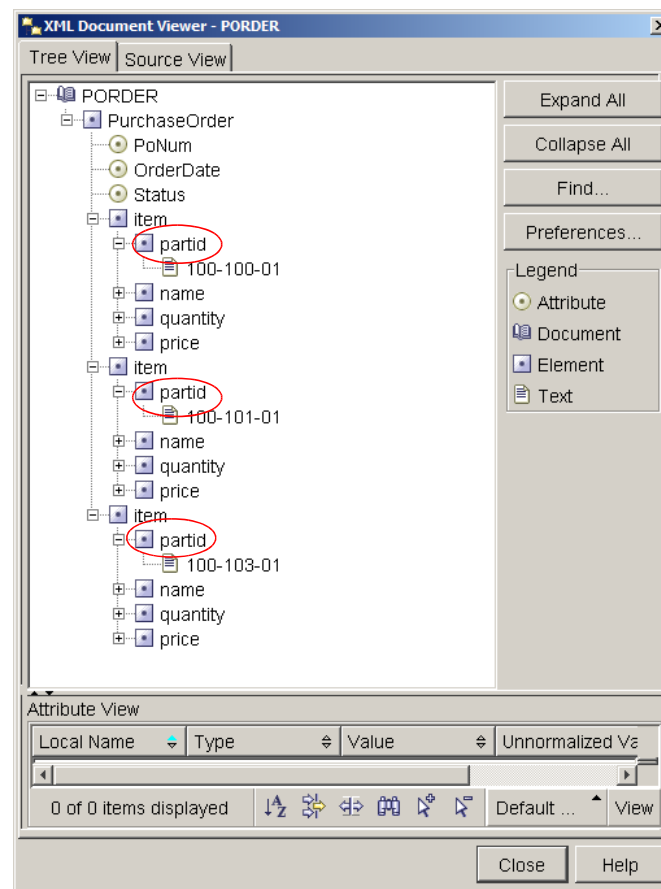
- Deleting an XML element when updating
- For example, deleting element with partid: 100-201-01

```
UPDATE purchaseorder SET porder =  
  xmlquery('transform  
    copy $po := $order  
    modify do delete  
    $po/PurchaseOrder/item[partid = ''100-201-01'']  
    return $po'  
    passing porder as "order")  
WHERE poid=5012;
```

This is another example, but in this case we are deleting an element in the XML document.

## UPDATE example

- After deleting element with partid:  
**100-201-01**



05/30/2011

64

Template Documentation

© 2011 IBM Corporation

This show the element was deleted.



## UPDATE operations with XML

- **Many other things you can do with TRANSFORM:**

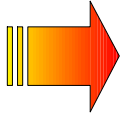
- ▶ Replace the value of an element
- ▶ Replace the value of an attribute
- ▶ Replace an element and attribute
- ▶ Rename an element and attribute
- ▶ etc.

- **Examples were taken from:**

C:\Program Files\IBM\SQLLIB\samples\xml\xquery\clp\xupdate.db2

## Agenda

- Overview
- Inserting XML data
- XPath
- XQuery
- Querying XML data using SQL/XML
- Querying XML data using XQuery
- Update & Delete operations with XML
- XML Indexes
- XML Schema Validation
- Other XML support



05/30/2011

Template Documentation

66

© 2011 IBM Corporation

## XML Indexes

```
create table customer( info XML);  
  
create unique index idx1 on customer  
  (info) generate key using  
    xmlpattern '/customerinfo/@Cid'  
  as sql double;
```

```
<customerinfo Cid="1004">  
  <name>Matt Foreman</name>  
  <addr country="Canada">  
    <street>1596 Baseline</street>  
    <city>Toronto</city>  
    <state>Ontario</state>  
    <pcode>M3Z-5H9</pcode>  
  </addr>  
  <phone type="work">905-555-4789</phone>  
  <phone type="home">416-555-3376</phone>  
  <assistant>  
    <name>Peter Smith</name>  
    <phone type="home">416-555-3426</phone>  
  </assistant>  
</customerinfo>
```

05/30/2011

Template Documentation

67

© 2011 IBM Corporation

You can define XML indexes based on a Xpath expression.

For example, if you access the Cid attribute constantly, then you can create an XML index just to that particular attribute. You will do it using this syntax: **create unique index, etc., generate key using XMLpattern**, and then you put an **Xpath** statement to go to that attribute.

The other examples are similar where the Xpath is what changes.

## XML Indexes

```
create table customer( info XML);
```

```
create index idx2 on customer(info)
generate key using
xmlpattern '/customerinfo/name[1]'
as sql varchar(40);
```

```
<customerinfo Cid="1004">
  <name>Matt Foreman</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Toronto</city>
    <state>Ontario</state>
    <pcode>M3Z-5H9</pcode>
  </addr>
  <phone type="work">905-555-4789</phone>
  <phone type="home">416-555-3376</phone>
  <assistant>
    <name>Peter Smith</name>
    <phone type="home">416-555-3426</phone>
  </assistant>
</customerinfo>
```

## XML Indexes

```
create table customer(info XML);
```

```
create index idx3 on customer(info)
generate key using
xmlpattern '//name'
as sql varchar(40);
```

```
<customerinfo Cid="1004">
  <name>Matt Foreman</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Toronto</city>
    <state>Ontario</state>
    <pcode>M3Z-5H9</pcode>
  </addr>
  <phone type="work">905-555-4789</phone>
  <phone type="home">416-555-3376</phone>
  <assistant>
    <name>Peter Smith</name>
    <phone type="home">416-555-3426</phone>
  </assistant>
</customerinfo>
```

## XML Indexes

```
create index idx4 on customer(info)
generate key using
xmlpattern '//text()'
as sql varchar(40);
```

Don't index everything!  
Too expensive for  
insert, update, delete !



```
<customerinfo Cid="1004">
  <name>Matt Foreman</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Toronto</city>
    <state>Ontario</state>
    <pcode>M3Z-5H9</pcode>
  </addr>
  <phone type="work">905-555-4789</phone>
  <phone type="home">416-555-3376</phone>
  <assistant>
    <name>Peter Smith</name>
    <phone type="home">416-555-3426</phone>
  </assistant>
</customerinfo>
```

05/30/2011

70

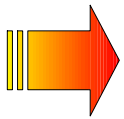
Template Documentation

© 2011 IBM Corporation

Here the **//text** means you want to create an index on all the values, but that's not recommended because you are creating an index that is too big. An analogy would be to this: if you are working on a table and creating a relational index, it's like creating an index on all the columns of the table, and that's not recommended. It would be too big, so when you do insert, update or delete, you will also update the index, and that will cause more performance issues.

## Agenda

- Overview
- Inserting XML data
- XPath
- XQuery
- Querying XML data using SQL/XML
- Querying XML data using XQuery
- Update & Delete operations with XML
- XML Indexes
- XML Schema Validation
- Other XML support



05/30/2011

Template Documentation

71

© 2011 IBM Corporation

## XML Schema validation

- **XML Schemas are supported using “XML Schema repositories”**
- **To validate based on an XML Schema you can:**
  - ▶ Use the XMLVALIDATE function during an INSERT
  - ▶ Use a BEFORE Trigger
- **To test if an XML document has been validated, you can use the “IS VALIDATED” predicate on a CHECK constraint**

05/30/2011

72

Template Documentation

© 2011 IBM Corporation

You can perform validation of your XML document using XML schemas which can be stored in XML Schemas repositories in the database.

You can use the XMLVALIDATE function in your insert statement or a BEFORE trigger to validate against XML Schemas you registered and stored in the XML Schema repository and that would be done per row.

A CHECK constraint, using the IS VALIDATED predicate can be used to detect if a given column has been validated using XMLvalidate.



## Validating with an XML schema (example)

```
INSERT INTO t1 VALUES(xmlvalidate(xmlparse(document('<?xml
version="1.0" encoding="UTF-8"?>
  <po:PurchaseOrder xmlns:po="http://www.test.com/po">
    <Header>
      <Id>1</Id>
      ...
    </Header>
    <Items>
      ...
      <Item>
        ...
      </Item>
    </Items>
    <Customer type="regular">
      ...
    </Customer>
  </po:PurchaseOrder>' ) ) ACCORDING TO XMLSCHEMA ID order ) );
```

05/30/2011

Template Documentation

73

© 2011 IBM Corporation

DROP TABLE t1;

CREATE TABLE t1 (po xml);

INSERT INTO t1 VALUES(xmlvalidate(xmlparse(document('<?xml version="1.0" encoding="UTF-8"?>

<po:PurchaseOrder xmlns:po="http://www.test.com/po">

<Header>

<Id>1</Id>

<date>2004-01-29</date>

<description>purchase order</description>

<value>20</value>

<status>shipped</status>

</Header>

<Items>

<Item>

<ItemDescription color="red" weight="5">

<Name>Widget C</Name>

<SKU>1</SKU>

<Price>30</Price>

<Comment>no comment</Comment>

</ItemDescription>

<NumberOrdered>1</NumberOrdered>

</Item>

</Items>

<Customer type="regular">

<Name>Manoj K Sardana</Name>

<Address>ring road, bangalore</Address>

<Phone>918051055109</Phone>

<email>msardana@in.ibm.com</email>

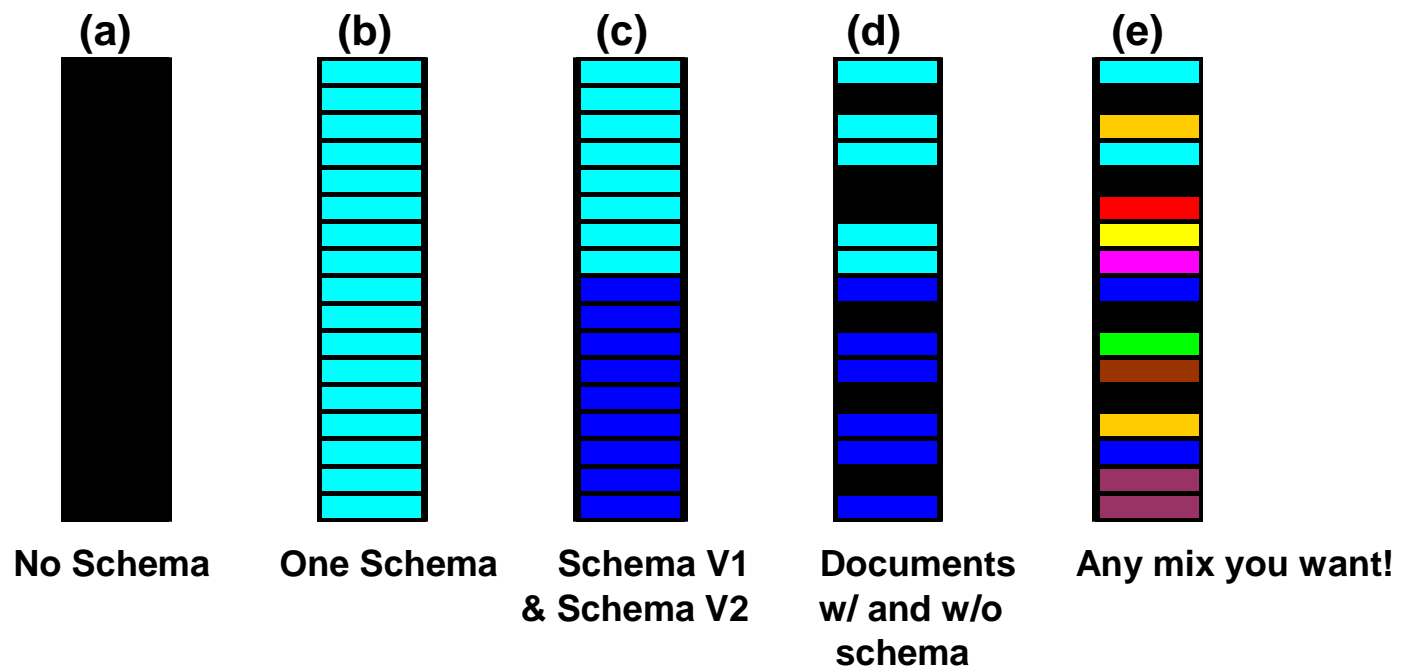
</Customer>

</po:PurchaseOrder>')) **ACCORDING TO XMLSCHEMA ID order**));

## pureXML schema flexibility

- Document validation for zero, one, or many schemas per XML column:

Always **Well Formed XML**



Most databases only support (a) and (b). DB2 allows (a) through (e).

05/30/2011

Template Documentation

74

© 2011 IBM Corporation

This shows you can validate different rows per XML column using different XML schemas

## Agenda

- Overview
- Inserting XML data
- XPath
- XQuery
- Querying XML data using SQL/XML
- Querying XML data using XQuery
- Update & Delete operations with XML
- XML Indexes
- XML Schema Validation
- Other XML support



05/30/2011

75

Template Documentation

© 2011 IBM Corporation

## Other XML support

- **Based-table inlining and compression of small XML documents**
- **Can transform XML documents using XSLT functions**
- **Compatible XML Schema evolution using the UPDATE XMLSCHEMA command**
- **pureXML supported for UNICODE or non-UNICODE databases**
- **Annotated XML Schema Decomposition**

05/30/2011

Template Documentation

76

© 2011 IBM Corporation

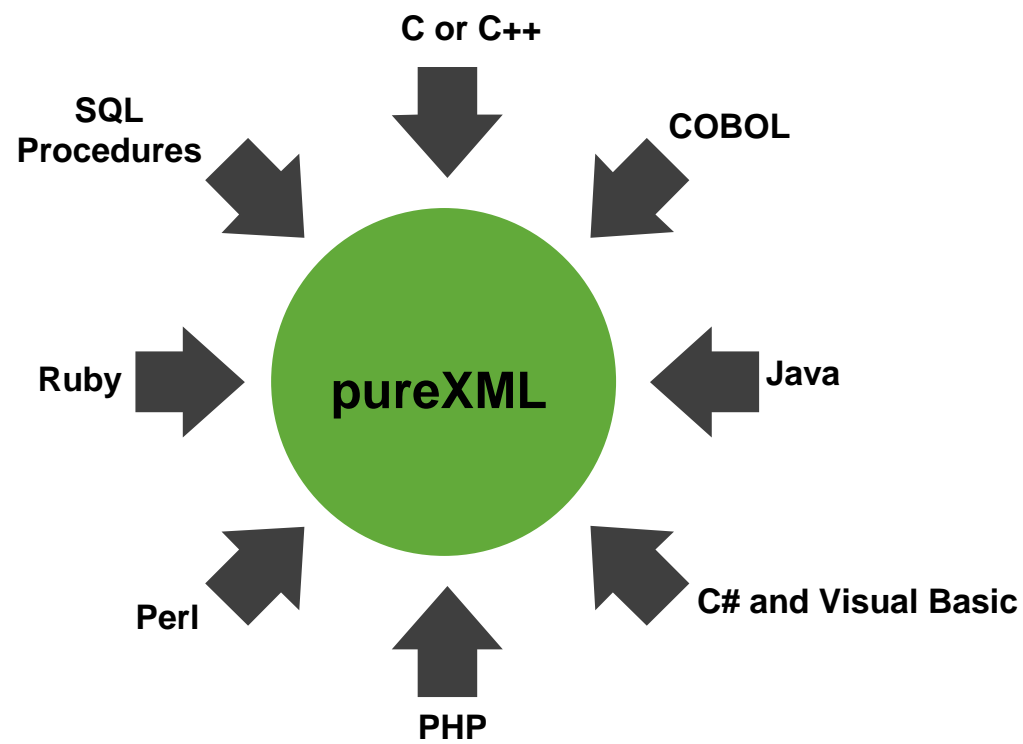
Normally, XML is stored in different object, and in the table we have a pointer to this object. Base table inlining and compression of small XML documents means that if the XML document is not big and it can fit into the same table then DB2 will store it with the base table and row compression can apply also to XML documents.

There is support for XSLT functions, so you can transform your XML documents to provide a different format like HTML format. You can use CSS (Cascading style sheets) so the data is displayed nicely.

Compatible XML schema evolution using the UPDATE XMLSCHEMA command means that your schema can be changing constantly through time. Maybe a string had a given size, now it has changed; now it is larger, so if you run this update XML schema, as long as it is not a completely different schema, then the schema can evolve.

We still support XML schema decomposition, if you want to store XML not using pureXML, but actually decomposing the XML into small pieces and storing each piece in tables.

## Development support for XML data



05/30/2011

Template Documentation

77

© 2011 IBM Corporation

pureXML is supported with any of these languages. All drivers have been modified/updated for this support



# Thank you!

Use the forum in the [db2university.com](http://db2university.com) course AA001EN if you have technical questions about the materials covered in this course. Fellow students, faculty and IBMers can help you!