

## **Lesson 15 Transcript: Pure XML – SQL / XML & XQuery**

### **Slide 1: Cover**

Welcome to Lesson 15 of DB2 on Campus lecture series. Today, we are going to talk about Pure XML-SQL and the use of XML and XQuery. My name is Raul Chong, and I'm the DB2 on Campus Program Manager.

### **Slide 2: Agenda**

This is the agenda for today.

### **Slide 3: Agenda (1) Overview**

We will start with a short overview.

### **Slide 4: DB2 The Big Picture**

I have shown this picture before, which provides an overview or a big picture of DB2, and we talked already about the DB2 environment; we talked about the tools, and now we are concentrating on this part, and especially we are concentrating on SQL/XML and XQuery.

### **Slide 5: What is pureXML?**

Now, what is pureXML? I already talked about what pureXML is in the speech number 2, Part 2 of the DB2 on Campus lectures series, which is titled The role of DB2 in Web 2.0 and in the Information on-demand world. So if you don't know what pureXML is, you can pause this presentation, and then start taking a look at that speech number 2 for more details. But there are basically two main characteristics of pureXML. One is that the XML is stored in parsed hierarchical format in the database in a persistent way, and (two) DB2 also has a native XML interface to the engine. So what that means is, we have the XML stored as a tree in hierarchical format and we have an engine that natively processes that format—so the stored format equals to the processing format.

### **Slide 6: Agenda (2) Inserting XML data**

And before moving on to the next section, I am going to show one simple picture that I provided in that speech. So this is what I mean: This is the way we store the XML documents in the database, in parsed hierarchical format, or as a tree. And this is persistent and we have this XML native interface to the engine, and obviously also we have the relational part of the engine. When we issue an Xpath or XQuery it will go to the native XML interface of the engine, and then it will do any processing that it needs, and it will access the tree or the XML document which was stored as a tree. And if you are using SQL as usual, it goes to the relational engine or to the interface of the engine, and then it will go to the relational information which is normally stored in tables. Again, take a look at the speech number 2, part 2, for more details. Now in this presentation, we are actually going to show you how to do things with pureXML. So we will start with inserting XML data into the database.

### Slide 7: Table Definitions with XML columns

So for this what we are going to is to show you a script where we are going to create a database, and we are going to create a couple of tables, and we are going to start loading some information and some XML data. Let's say we have these two tables that we are going to create, the items table and the clients table, and you can see that the last column is defined with the XML data type for the items table. And the same thing for the clients table: the last column which is called "contact" is defined with data type XML. Both tables have one column defined with the type XML, which happens to be the last column on each table; it does not have to be the last one, but in this example we are putting it at the end.

### Slide 8: Contents of C:\DB2workshop\Quicklabs\quicklab14a directory with XML document

I would suggest you to pause right now, and because what we are going to do is to run the script that you already should have if you listened to the getting started lesson of this DB2 on Campus lectures series. We recommended that you download a free online book which is called *Getting started with DB2 Express-C*, and it also comes with a separate file that contains exercises; so in those exercises, basically we are using the same files for those exercises in this presentation. So the recommendation was that you should download that file and unzip it into a directory like this one :

```
C:\DB2workshop\quicklabs\quicklab14a.
```

In this case, we are going to take a look at quicklab #14a. And we are basically going to work with this file that is part of quicklab #14a. If you don't have it, please pause this presentation and go to [www.ibm.com/db2/express](http://www.ibm.com/db2/express), then look for the link of the free online DB2 Express-C book, click on that link, download that book and download also the exercises. Then unzip them into this directory. Then, once you come back after the free pause, we will take a look at this file called table\_creation.txt. If I open that file with Notepad, you will see that, first we **drop database mydb**. This is just to clean up the environment, just in case you already have this database created. So we will clean up the environment, then we again create the database, we are creating it as Unicode. That's what it says, UTF-8. This is not really required to support pureXML. Now, in version 9, it was required, but in version 9.5 it is not required to create as Unicode.

Then we will connect to the database and we will create the tables that I mentioned before: the items table where the last column has an XML data type and the clients table where the last column has an XML data type as well. And then here, this is the way we are talking about inserting XML data into the database. So, there are two ways to do it. One way is, as shown here, which is basically using an SQL insert operation, we are simply using an SQL insert operation, where the XML is passed as a string, as you can see here. So when you pass this to the XML column, then DB2 will say, "ok, this is really an XML document, so I am going to parse it." And basically what it means is to take this XML document and store it in a parsed-hierarchical way as a tree. Now, if for any reason there may be situations where maybe the XML looks very complicated, and DB2 cannot detect that it's XML, then you can add a function called XMLPARSE. For example, if you put XMLPARSE and..., you have to

read the exact syntax, but basically you have to put something like this. I'm just going to give you kind of an overview, something like this, where you invoke XMLPARSE function, and you explicitly are telling DB2 that it needs to parse this string because it is an XML document. I am just not going to make any changes here, ok. That is the first way, it is just to use the simple SQL insert operation and create the XML as a string, and you may have to invoke the XMLPARSE function. The second way is to use the import utility that was discussed in the previous lesson of these eLearning courses.

The difference from what I taught before is that, well first, we provide the source for that import, in this case is file clients.del, which is in the same directory; and then we say of del, so this is file clients of del, which is of delimited format, so it contains delimited ASCII. And then we say XML from and we put "where is the XML". Basically we are saying where are the XML files that we want to insert. So in this case we are using the import utility, and we are going to insert XML, but the XML is stored in separate files, which are in this directory, in the one that I highlighted. And then I will continue with the syntax, insert into clients etc., so that is the syntax of the import. It has an import from where, where is the XML, and then I continue inserting into this file. The same procedure we are going to use for the items table, so we say items.del it's another file, the XML is also on the same directory, and we are going to insert into items.

So let's take a look at that directory in more detail. So we have these same files that I mentioned... sorry, these are clients.del and items.del, they are the ones that were mentioned in the script table\_creation.txt. So let's take a look at clients.del. If I double-click here, this is the content of the clients.del file, and, well, .del means delimited. It's a delimited ASCII file. So you see that the delimiter is a comma. Each of these values would go into a column in the table. So this is for another column, this is for another column, and then, here, what we have, where it says XDS file equals this, is actually like a pointer, from this file to another file. Where is that other file? That other file is in this same directory, as you can see from here, client3227.xml. So this is a pointer to this file client3227.xml, which if I open it, so then this is the file client3227.xml. And as you can see, what we have is the XML document for this particular file. You have the different elements, and this is the root. So that is the XML document that is in this file client3227.xml that I showed you here. So basically what we are doing is that we are going to insert this row where the first, 1, 2, 3 columns are relational, and the last one is to store the XML. The same thing with the other rows; the second row, which is this for the first column of the row, the second column of the row, the third column, and this fourth column of that row, but this is the XML document. So let's continue here. So this is the same information. I will show you items.del, but it is basically the same idea.

### **Slide 9: Insert and Import of XML data**

This I already showed you, right? From the script, you can start using an insert statement; this is an SQL insert, or using an import. And I have highlighted it in blue, the directory where the information is. So the first one in blue is where the file clients.del is, the second one in blue is where the XML documents are; the same thing for the other import.

### Slide 10: Agenda (3) XPath

Ok, that is how that insert works, but let's actually run... let's close this file, and let's actually run the script. So how can you run the script? This is table creation. I'm not going to save the changes. And let's run this script by... let's see... it is opened from the Control Center...

There are two ways to do it; you can open the Command Editor by clicking the fourth button of the Control Center, or typing **db2ce** from the command line. And from here, you could click on this button here to open, and basically you have to go to this directory here to find the file table-creation.txt. You can go here, say ok, and then find the table-creation.txt. I can go here, say ok, and then I have to find table-creation.txt, and then I say ok. And then what happens is that it will open that file. Then I have to click on this button to execute. Note that here this target has to be blank, because that means there is no connection, and there should be no connection because I am dropping a database and I am creating a new database. So that would be one way to do it. By clicking this it will start running.

But I am going to show you another way to do it, which would be by invoking the command window, or the Linux shell, if you are using Linux. And I am going to type **db2cmd** which is already there. I need to open the command window. And from here, it's the same idea. You can go **cd**, and then you change to that directory. So now I am going to list the directory and now here you have table-creation.txt. I also explained this when I was talking about tools and scripting. Anyway, from here you can do **db2 -tvf** which is a way to execute this script. And then I press Enter, and it starts executing. So it first drops the database. In my case I had it before, that's why it says drop was successful. In your case, it may say there is an error because it cannot find the database mydb, but that's normal because we are trying to clean up our environment. Then I was going to create the database mydb, in this case it is creating it as Unicode because it says codeset UTF-8, but again that is optional starting with DB2 version 9.5. Creating a database sometimes takes one or two minutes because, this again, as I explained before when I was talking about the DB2 environment, when you create a database in DB2, you are also creating some metadata information, and there is also an auto-configure utility that is run to take a look at your system, trying to configure DB2 parameters, so that the parameters are set to good values for good performance. Then it connects to the database, starts creating the two tables, and then it finishes with an import and an export.

So let me just take a look up here, after the creation of the database, you see the create database was successful, then it connected to the database successfully, and then it created the first table, successfully, and the second table, the clients table successfully. Then it inserted that XML successfully using the SQL insert; then it did an import, as you can see... sorry going too far... anyway, from here you can see it did an import for the first file. From here, it says that it read 6 rows, and it inserted 6 rows. So you have to pay attention to this, it inserted 6 rows, so that's good. That was for the first import, and this is for the second import, it says that it inserted 4 rows, so that's good. Perfect. So this script ran correctly.

And now we invoke the Control Center, and we can take a look here to the mydb database that we just created. Let's take a look at the clients table, and let's see the contents of this table. And here we can see the same values, you have John Smith, this is the one that we

inserted using... if we go back up here. You see here it says insert into, etc., John Smith, Gold, etc. That is the first record, John Smith. We take a look at the XML, we click on those three buttons, you can click the Expand all button, well, that's expanded already, and that's the address, etc. Texas 00112, that is what we had in the PowerPoint: Texas 00112. The same thing for the other ones. Let's take a look at the other ones now. So the other records will be here. We have Ella Kimpton. Let's take a look at this in the XML, Expand All, and then we can see, for example, here is Client, Address, street, 5401, Julio Avenue, city, San Jose, etc. If we take a look at the actual file which is already closed, let's see, which is here, so if we take a look at this file that I already had open, so, as you can see here the information is the same information, 5401 Julio Avenue, San Jose, California 95116, etc. So, that's just to show you, that the information is now inside the database.

Now what's the big deal about this? Well, now that it is inside the database, I can perform queries on this. Let me close this, actually before I close, you can click on Source View, you can also see the Source View, you can copy/paste from here, you can Collapse All, Expand All, you can do Rind and so on. These are legends of the little graphics that are provided. Anyway, so, as I was saying, the good thing now that it's in the database is that you can perform queries, not only for the XML, but also for sub-documents, for little parts of the XML, you can also combine relational information with XML information, and I am going to show you some examples in the next few slides. Let's just close this now. And again these would be other records, these are all taken from the files of the import and corresponding XML files. So let's close this, and let's go back to the presentation,

### **Slide 11: XPath**

Ok, before I teach you how to query relational data and XML data, I will provide a very quick overview on XPath and XQuery. Let's start with XPath. So XPath is a language that you can use to navigate an XML document. It is a standard, it is not something proprietary of IBM, but it's a standard. It's basically a query language. What we have here on the left is an XML document, serialized for representation, where, each of these are elements, and the first one is the root, and every element has to have a terminator, or terminator tag. So it's like HTML, but the tags have a different meaning, or you can put some meaning to these tags, and they are basically used to describe information. When you have something like this that is inside a tag, so for example, so in this tag dept, or in this element dept, I have this string. This is an attribute. So within the tag, if you have a string, that would be an attribute, in this case the attribute has a value of 101, so anything that you see in red are the tags' node which provides values to the tags.

Now, this representation, which is a serialized representation, can also be seen or represented as a tree or also known as a parsed hierarchical presentation. So these two things are exactly the same thing, only two representations, two different representations, and in this case, you can think of this representation the way that DB2 stores the XML document inside the database in a persistent way. XPath is shown here, these are some XPath statements, and XPath is fairly easy to learn; it is very similar to the **cd** command, the **change directory** command that you used in MS-DOS, Windows, Linux, Unix, right? So 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 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 here to John Doe and all the way here to Peter Pan.

### **Slide 12: XPath Simple Xpath Expressions**

Now as we will see on the slide, 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. So assuming that this is my XML document, if I do a /dept/@bldg, building, the @ symbol means that I want the attributes, that's why it is retrieving 101, because that's the attribute 101. Right? We want the attribute 101. Then /dept, employee, attribute id, the same thing. In this case the attribute will be 901 here. 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, which is the same example I provided in the previous slide. Note that what I am getting as well is the tag, 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.

### **Slide 13: XPath Wildcards**

On this slide, using the same example, if you use /dept/employee, asterisk, asterisk is a wildcard and it's basically any name, any element. Any element but only one in between this, you can put it. So, that is dept, employee, whatever element, and get me the text. It will go dept, employee and all of these: John Doe, 408, 344, and then the same for the other employee. Then here is a similar example, dept, whatever element, and get the attribute id, 901 and 902. Here, two slashes means that it's not just one element, but a number of elements, 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. Here, /dept//phone, so any element between dept and phone. Here is dept, and here is phone. So any element in between, I don't know how many or what they are, just go through that navigation, and that's why I get these two phone numbers.

### **Slide 14: XPath Predicates**

Moving on here, you can also use square brackets, for example, it's like in SQL you have a where clause, So this is sort of like a where clause. You say /dept/employee, where attribute id is 902, and from there it takes the name. So /dept/employee, and then where attribute id is 902. So, this one does not apply. So, we have to choose the next one which is id 902, and from there take the name. That's why it gives Peter Pan. Similarly here, but in this case, we have two conditions, or two square brackets. So the first one, and then the second one. This is similar as well, in this case, we have two slashes. So whatever that is before the employee element, and then where office equals 344, or office equals 216, and from there take the attribute id. This one is saying, when you have a number here, it means I want the second employee. So that's what this means. If it was one, it will be "I want the first employee."

### **Slide 15: XPath The Parent Axis**

This is also the current context and parent context. So it's similar to when you use the change directory command..., right? So that will be going to the current context or to the parent.

### **Slide 16: Agenda (4) XQuery**

Alright, so that was a very quick overview of XPath.

### **Slide 17: What is XQuery**

Now, a very quick overview of XQuery.

### **Slide 18: XQuery**

XQuery is basically a superset of XPath. As you can see from here, this is XQuery, and XPath is inside XQuery. So XQuery is a superset of XPath. So XQuery is used to navigate an XML document, it supports typed and untyped data. When there is no XML value, in XQuery you will not put null values, just blank, and I will give an example of this later on, and it returns a sequence of XML data.

### **23:50 Slide 19: FLWOR and PATH Expressions**

Within XQuery, we have something called the FLWOR expression, F-L-W-O-R, and with that expression you can do more manipulations. FLWOR, you can think of it as analogy, you know, in SQL you have SELECT-FROM-WHERE, in XQuery, an analogy is that you will have FLWOR. FLWOR, F-L-W-O-R. FLWOR stands for: For, Let, Where, Order, Return.

### **Slide 20: XQuery: the FLWOR expressions**

This is an example where I have an XML document, I have this table that I have created, **create table dept** with a relational column. We have one XML column. Then here is my 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, but in this case, if you want to run an XQuery, which is a FLWOR expression, as you can see from here, f-l-w-o-r, then you need to prefix these all by XQuery. And then you go for \$d in, etc. I will explain these functions in more detail, but basically here, I am getting into variable d the XML document that is provided here; and then let \$emp, so I am assigning a new variable called emp, the value of \$d, which was the entire XML document, //employee/name, and then it says where, \$d, so here are some conditions, ordered by this, and 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. So 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.

### **Slide 21: Agenda (5) Querying XML data using SQL/XML**

Let's move on to the next section of this presentation which is Querying XML data using SQL/XML

### Slide 22: Two worlds

In DB2 there are two worlds, not in DB2, but in XML manipulation or query, there are two worlds: one way is to query using the XQuery standard, and the other way is to query using the SQL and XML standards. And in the case of DB2, we support both, SQL/XML and XQuery. And in this part of the presentation, we are going to focus first on the SQL/XML standards.

### Slide 23: What you can and can't do with SQL

So first of all, if I just had SQL, can I manipulate XML? Yes I can, but I would only be able to get the entire XML document. So a very quick example here, let me connect to the mydb database. At the same time that it's connecting, I am going to delete all of these, which I didn't execute from here, but from the command window. And now, let's say, I am going to do **select contact from clients**, if you recall, contact is a column as in the clients table. contact is a column that contains all the XML. Here you can see, so if I execute these, let me clear the result here, what's going to happen when I select the XML column, using just SQL, well I get all the XML, so that's good. But what happens if I just want to get not the entire XML, let's say I only want to get the fax number. How can I do that with SQL? I could not do that just with plain SQL. For that, we need SQL with XML or XQuery which we will cover later.

### Slide 24: SQL/XML Queries

SQL with XML is a bridge between the SQL and XML worlds, and we can embed XPath and XQuery as part of this.

### Slide 25: New SQL/XML functions in SQL 2006

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. 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.

### Slide 26: XMLexists function

So let's start with the XMLexists. So this 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 rows, or lines, and the last, actually the first two lines are just SQL as usual; I want the name from clients. And then the third line that says **XMLexists, \$ c, Client, address, zip**, right? Now, **\$ c** means, anything with a **\$** means it's a variable. Where is the variable defined? This variable is defined on the fourth line, the one that says **passing CLIENTS.CONTACT as c**.

If you recall, let me show you again, **contact** is the column that contains the XML document. So what we are doing here is we are passing, CLIENTS, the name of the table, CONTACT,

the name of the column that contains the XML documents. We are passing that to the variable `c`; so that `c` has the XML document. And what we are doing here is `/Client/Address`, what is this? This is XPath, and then this is also XPATH, where square brackets are used like a where clause in SQL, but it is within the XPath, 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 basically we want to, let's say, we give an example, we want Client, Address, zip, and we test for 95116. 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, now, this is the syntax that was required prior to DB2 9.5, with DB2 9.5, as you can see here in the example below, the syntax is a little simpler, because, we go **select name from client where XMLexist**, and we have **\$ CONTACT** so this is a variable that is automatically created with the same name as the column name that contains the XML, so remember CONTACT is the name of the column that contains the XML document. So in DB2 9.5, we just use **\$ CONTACT**. It's a variable created with the same name as the column.

### **Slide 27: Case sensitivity-Caution!**

So that is the first example. Note here that CONTACT is in uppercase, as in the previous example, CONTACT is uppercase, right? You have to be very careful when you work with SQL/XML and XQuery because they are case sensitive. SQL is not case sensitive, so when you are working with SQL, you may be used to working with SQL, you don't care about the case, whether it's uppercase or lowercase. But now that we are working with XPath and XQuery, you do have to be careful about it. And if you combine, if you work with SQL and you are working with XQuery and XPath, then the SQL part can still be case insensitive, but the XPath and XQuery part must be case sensitive.

So, for example here, this **select name from clients**, you could have written this in any case, and then `XMLexists` could also be in any case, but whatever is inside, you have to pay attention on the case. 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 you cannot find this path, and then you will get an incorrect result.

Now, why is CONTACT in uppercase? Like in this case, this is wrong, because it is in lowercase. It should be in uppercase as shown here. Why should it be in uppercase? What happens is that DB2 objects, that is, tables and columns, have to be put in uppercase. Why? Because when you create a table in DB2, you can issue a command **db2 create table**, I am going to show you very quickly here, you can say **select contact from clients** for example, this is in lowercase, and you can say create table t1 for example, and you can put all the columns, but these tables names and also column names are actually stored inside DB2 in uppercase. If I go to the Control Center, I close this and I list the tables, they are all listed in uppercase. That's the way they are stored. You can see from here the column names of the table, they are also stored in uppercase. This is by default. You could potentially store or create tables, you can force DB2 to create them in lowercase as well as the column. We don't really recommend you to do that, because it will just add some more complexity to handling these tables and columns, but anyway, you could do it but we don't recommend you to do it.

And again, going back to the original topic, basically by default these tables and columns will be created in uppercase. So that's why every time we invoke the name of the column or a table within the XML part, or the XPath or the XQuery part of the query, it needs to be in uppercase. So that's one thing you should be careful about. And the problem is that if you don't make... that's a tricky part, in XQuery and XPath if you use instead, as I said before, a lowercase c instead of uppercase, you don't get an error, you get an incorrect result, and that is more dangerous sometimes.

### **Slide 28: Curly Quotes-Caution!**

Another thing you should be careful about is the use of quote. And I mention this because I am using PowerPoint and MS Word, and they tend to change the quotes, and they start using curly quotes. The one at the top is curly; the one at the bottom is straight. You should be using the straight instead of curly quotes. The same thing here, curly, and then here is straight. You should use, again, the straight. And then, in this example here, where I will be getting an error because I was using a curly quote here, curly quotes here, the same thing. This is a double quote, but it's also using curly, it should not be, it should be just straight quotes. That's a typical problem when you copy/paste from the PowerPoint or MS Word into the Command Editor for example.

### **Slide 29: XMLquery function**

Moving on the next function, we have XMLquery, and this function allows me to perform, and basically retrieve an XML element, like email, as a column. So I'm going to **select XMLquery, \$ CONTACT, Client, email**. So here the email will be treated as a column, but actually it is an element of this XML document. And then you say **from client where status = 'Gold'**. So in this case, I am using relational column to filter the rows that I want, and I am getting an 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.

By the way, I didn't test these two queries. Let me just try them out right now. So the first one is here, I just put them in the Notepad, because it is easy to copy/paste. So this is the same query that you can see, actually from here. So, **select name from clients where** etc. etc. But now I am going to execute it from the Command Editor. So I go to the Command Editor, let me delete these, clear the result here, CTRL-V, I am connected to the mydb database, so that's fine. I execute and get one return back, which is Ella Kimpton. And you can compare that, again, it should have this 95116. I have already showed you this actually because I have been showing the clients table several times, and I have showed you the Ella Kimpton value several times, but let me just quickly do it. And then you have that she is the one with zip 95116, so that's correct.

Ok, going back here to the Command Editor. Again, as I said before the syntax in version 9.5 has changed, so here I just put CONTACT, I don't need to put all of these **passing CLIENTS ... as c**, etc. This can be deleted. I can clear the results here, and you can execute this, and you get the same result. So this is the simpler syntax in 9.5, where you can just use

CONTACT. The other Query that I showed you was using XMLquery. Let me run this as well. But you can, if you are following my presentation, and you paused, and you downloaded the book and the exercises, I hope that you follow the things that I am doing here, because it's always better to follow what I do. It's just better for learning purposes. Here I run this XMLquery to retrieve that email and I get this result. Now note that it says 3 records, but I only see one. I only see one but the first two are blank. Remember I talked about this very briefly when I was talking about XQuery; in XQuery or XPath, you don't put null values. In XML, there are no null values like in relational. So that's why it just appears blank. You just put blank. If there is no email, blank. Otherwise it will provide that email, so that's why it says 3 records.

### **Slide 30: Retrieving XML data using for and return clause of Xquery**

So, moving on, I already talked about XML query. This is another example that you still are using the XMLquery function, but inside the XMLquery 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, but it is using For, in this case it is **For \$ e in, CONTACT, Client, email one, return \$ e**. So, you don't actually need to use FLWOR expression in this example, you could have just returned that 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.

### **Slide 31: Retrieving and transforming XML into HTML**

I want to execute that, just to save some time, the same thing here, using the XMLquery, and you can use the FLWOR expression, you are getting the first email and you are going to get the text from that, and what you are going to return in this case is, as you can see, we have p and /p, that's HTML, and here is XML. We are embedding, or combining HTML and XML together. So let me just show you that as an example very quickly, this one I will skip to save time, now I am going to choose this other one, CTRL-C, go to the Command Editor, let me clear the result so it's just cleaner, and I execute, and then the result that I get is here, you see, it's HTML, p and /p, and here I have the email, again this is 3 records because it is the same query as before, these are blank, just blanks.

### **Slide 32: XMLtable function: from XML to relational**

Ok, moving on, we have here another function called XMLtable, and this one 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. So, potentially maybe 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 path is going to be CommentID; actually it's going to be all of these, right? All of these are the path, Comments, Comment, and then CommentID will be the last element. 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. Now let me execute that query very quickly, CNTL-C, and then from here, clear the results, copy and paste, and then execute, and as you can see, what I get seems to be a table, but actually all of this information was taken from XML documents. So all of these from XML documents, but now it looks like it's a table. That's why you can work with these two systems, one is relational and the other one is XML.

### **Slide 33: XMLelement function: from Rational and XML**

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.

### **Slide 34: Review: XMLtable vs XML element**

Now, basically what we have here is that XMLtable can be used to go from XML to relational, while XMLelement is to go from relational to XML.

### **Slide 35: Agenda (6) Querying XML data using XQuery**

Now we are going to move on to the next section of this presentation, which is Querying XML data using XQuery.

### **Slide 36: Two worlds**

Remember before we were using, or we had these two worlds, we have DB2 to support both, and we have covered this part, SQL with XML, but now we are going to cover XQuery.

### **Slide 37: Simple Xquery to return customer contact data**

In XQuery, everything has to be prefixed with the keyword XQuery. As I said before, if you don't put this keyword, that means that I am issuing SQL. That's why I need to put this XQuery keyword so that I am really issuing XQuery. This particular function is just to retrieve all the content of this XML column, so this actually will be equivalent to issuing a **select CONTACT from clients** if I am just using SQL, **select CONTACT from clients**. But this would be, if I use the XQuery, this would be XQuery: invoke this function, and then I put the name of the table, dot, the name of the column that has the XML. So let me just quickly execute this; I didn't execute that, the previous one, just to save some time, here I have the XQuery, again I am going to clear the results, and from here, I am going to execute the XQuery. And as you can see, I get all the XML documents back. Since we are here, let's see what will happen here if I choose **contact** in lowercase. And I execute, and then I get an error message, because again, I mentioned this, it has to be in uppercase when you are working with XML in DB2.

### **Slide 38: FLWOR expression to retrieve client fax data**

Moving on to the next section, we have the FLWOR expression. We are using XQuery and we are using the FLWOR expression; we have CLIENTS.CONTACT and we want to get the fax. So an example of that would be this output. Another example is this one. It is similar, but

now we are doing some restricting or filtering using zip. This one, we can maybe execute it very quickly. You can execute all of them. I don't want to make it too long; so I am skipping some of these examples. So if I execute this, I will get the zip code, all the way where the zip code is 95116. So that's the only one record I get, only one record because of the restriction that I put here.

### **Slide 39: Path expression with additional filtering predicates**

### **Slide 40: Querying DB2 XML data and returning results as HTML**

This is another example. In this case you are also combining the HTML, ul, for example, li, ul. You are combining HTML with XML. A very quick explanation of this, let's see. Here. CTRL-C. Here clear results, CTRL-V, execute, and then here I have, you see, now it has ul, li embedded within, or combined with the XML. So potentially you can return this to a web browser, and then it will be nicely displayed.

### **Slide 41: Sample HTML**

For this example, this is basically what I just showed you, sample HTML.

### **Slide 42: Embedded SQL within XQuery**

You can also use another function called DB2 function sqlquery (**db2-fn:sqlquery**), and that allows me to embed SQL as shown in red inside XQuery. So before, what I was showing using SQL with XML was that you would start with select, and in there I would use XMLexists, XMLquery etc. Now what I am using is XQuery at the top level and inside there I am using SQL in red. So this would be doing the following: this would be selecting comments from items where srp>100, and from there it will maybe select, let's say we have 100 rows, now it may select 10 rows, and from there, by the way, this column has to be an XML column, the one that I am using in select, and from there, I go /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. So let me quickly execute that XQuery here. And I'm going to... sorry, select this, the Command Editor, clear the results, CTRL-V. I execute, and this is the output I get for running that particular query. I get the ProductID, customers and message; so that's what I get, ProductID, customer and Message, and that's all within the action tag.

### **Slide 43: Joins with SQL/XML**

So moving on, you can also do joins with SQL/XML. So in this example, you have two tables, a department table and a unit table; the department table has an XML column, which is the last one, the unit table does not have any XML columns, so in this example, we are going to join an element of the deptdoc document, XML document, with a relational column which, in this case, is going to be the Manager column. And there are different ways to do it. The first way, for example, would be using this syntax, **select etc., where XMLEXISTS, \$e, employee, name = \$ m**. So here is where the join is happening, **\$e, employee... \$e//employ, name** that means that we want the name element of this deptdoc document, to be joined to \$m, where \$m is defined here, 'u.manager as m', basically this one here, Manager, this

column. That's what I am joining. I am joining the name element of the XML document with the relational column called manager. The same thing on select below; I'm doing... it's the same idea, but look here: manager column is on the left side while the XML element is on the right side; and then I have to change the syntax, I have to do a *casting*. I have to use XMLquery to get the name. Well, you know, this is the old syntax, I have to pass d.deptdoc as e, So I could have put just \$NAME from here... sorry, the name of the column, deptdoc, \$DEPTDOC//employee/name, and then I am doing the casting. Because you know in DB2, whenever you do a =, or whenever you are doing a comparison, the data type of the two sides should be compatible, since data manager is varchar, so it is compatible with char(20), or varchar. The difference between the two, it should be the same output, but in the first case, probably you are going to be using an XML index, while in the second case, because the relational column is on the left side, you are probably going to be using the relational index. Now you can try and test which method or syntax is better in terms of performance, using the XML index or the relational index.

#### **Slide 44: Joins with SQL/XML & XQuery**

Moving on, you can also do a join, where you can join, in this case, you have two XML documents, one is in one table called dept, this is in the dept table, and the other is in the project table, there you have another XML document. So you have two XML documents, what you are going to do here is you are going to join two elements from those two different XML documents, or maybe an attribute and an element. In this case there are two syntaxes; the easier syntax to understand is the second one, the one that uses XQuery. And with XQuery, you just do two FLWOR expressions, basically, one is for \$dept and the other one is for \$proj, and in there you are storing the XML documents, and then you do the equation here. You say, where... or the join here, \$dept, @deptID, equals \$proj, deptID. So I am joining an attribute with an element.

#### **Slide 45: Agenda (7) Update and Delete Operations with XML**

That's the end of the Query part. Now we are going to quickly cover the update and delete operations with XML.

#### **Slide 46: Update and Delete Operations**

An update and delete can be done in two ways, one is to use SQL update, and delete, so it is just a normal update and delete in SQL, or you can use the XQuery TRANSFORM expression.

#### **Slide 47: Update Example using SQL Update**

This is an example of SQL update, so this example, again, is just an update as usual, you have to use the SQL part. But basically, let's say if your original XML document has this fax number, 48... or let's say, 999999, and now what you want to do is to change it to this number, then you do an update, and you use set. And then you have to write the entire XML document here with the change. Ok, that's the way it works if you use an SQL update.

### **Slide 48: Update example using XQuery TRANSFORM**

If you use the other method, which is XQuery, and using that transform clause, then you can just specify what you want to change. So in the previous example, for example, you would just have to specify the fax number. So here we are going to do an update, we are going to set contactinfo to the following, and you have to put all the XML query information, and this is the namespace that has to be put in this case. Here is the transform clause, and you say: **copy \$newinfo to \$c. c**. Basically, we are passing CONTACTINFO as c, so c has the XML document, and then here we are copying the XML document to a variable newinfo. Then you will modify, or insert a new email in this example, as the last one into newinfo, customerinfo. Then you will return newinfo. So basically you are adding one, this tag here, to the existing XML, and that's how you are updating it.

### **Slide 49: Agenda (8) Indexes**

Moving on to XML indexes.

### **Slide 50-51: XML Indexing Examples**

You can define indices. There are XML indexes, and you have to specify the path. If you, for example, access this attribute constantly, then you can create an XML index just to the particular attribute. You will do it using this syntax: **create unique index, etc., generate key using XMLpattern**, and then you put an **Xpath** statement here; the same thing for this other one here. This one in this case would be for... sorry here, //name, so all the names, and then you have this one here, //text means you basically 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 and delete, you will also update the index, and that will cause more performance issues.

### **Slide 52: Agenda (9) Other XML support**

Finally we are going to talk about other XML support.

### **Slide 53: Other XML support**

In DB2 9.5, we also can do some sort of compression with XML, so we have base table inlining and compression of small XML documents. So if the XML document is not big and it can fit into the same table, because internally, the XML is stored in different object, and in the table we have something like a pointer; but when the XML document is small, in 9.5, you can also store the XML document as part of the same object of the table. And when it is stored like that, you can also apply compression that applies... or that row compression that works in DB2. Now, row compression doesn't work with DB2 Express-C, but it works with other editions of DB2. Then you can also use... there is some support for the XSLT functions, so you can transform your XML documents to provide a different format like HTML format. You can perform validation of your XML document, you can use XML schema, you can store your XML schemas in repositories; you can use the XMLVALIDATE function in your insert statement, so you go insert etc., and you put XMLvalidate, parenthesis, and you put all the

values that you want to be validated, against a given schema, XML schema, that is in a XML schema repository that would be basically per row, and that would be done per row. Then you can use a BEFORE trigger, you can define a BEFORE trigger on the table, and that trigger can perform the validation; it will also have a CHECK constraint, using the IS VALIDATED predicate to detect if a given column has been validated using XMLvalidate.

Then we also have in 9.5 compatible XML schema evolution using the UPDATE XMLSCHEMA command. What that means is 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, and can be done using the UPDATE XMLSCHEMA command.

Then pureXML, as I said before, in 9.1 it was supported on Unicode databases, but with 9.5 you can create them either on Unicode or non-Unicode databases; and XML, what we talked about so far is pureXML, but we also provide support using the composition method, which I talked about in speech No.2, which is a different method of manipulating your XML documents. 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.

#### **Slide 54: Quicklab #14a SQL/XML and XQuery**

With this, I suggest you to pause and work quickly on Quicklab #14a. This quicklab is actually a very short quicklab; we should already have created the tables and loaded the information. Now all we are doing here is running a couple of queries using pureXML.

#### **Slide 55: What's Next?**

With this, we have finished this lesson, so congratulations for completing Lesson 15 on pureXML. As to what is next, it will be Lesson 16: on Java, Ruby on Rails, and PHP. Thank you very much and have a good day.