DB2 for i HTTP Functions and XML

Nick Lawrence
Advisory Software Engineer
ntl@us.ibm.com
Agenda

- Resources
- Scalar and Verbose HTTP Functions
- Three motivating examples
  - Implement business processes via Web-Services (HTTP POST)
  - Retrieve PTF list from IBM (HTTP GET)
  - Retrieve Postal Codes (HTTP GET VERBOSE)
- DB2 XML Overview
  - Publishing Functions
  - XMLTABLE
Overview - Integration
Resources

- Announcement

- i Can
  - http://ibmsystemsmag.com/blogs/i-can/archive/integrate-web-services-with-db2-for-i-71/

- DeveloperWorks Article

- Whitepaper on HTTP Functions
  - https://www-304.ibm.com/partnerworld/wps/servlet/ContentHandler/stg_ast_sys_wp_access_web_service_db2_i_udf

- SQL/XML Programming
    - Publishing functions
    - XMLTABLE
  - XML Whitepaper
    - https://www-304.ibm.com/partnerworld/wps/servlet/ContentHandler?contentId=K$63TzTFkZwiPCA$cnt&roadMapId=IbOtoNRuYNU4MDADrdm&roadMapName=Education+resources+for+IBM+i+systems&locale=en_US
IBM Lab Services - DB2 for i Center of Excellence

- Database performance and scalability
- Advanced SQL knowledge and skills transfer
- Business intelligence and analytics
- DB2 Web Query
- Query/400 modernization for better reporting and analysis capabilities
- Database modernization and re-engineering
- Data-centric architecture and design
- Extremely large database and overcoming limits to growth
- ISV education and enablement

http://www-03.ibm.com/systems/services/labservices/
IBM Lab Services - Advanced SQL class

- Offered Online
  - http://ibm.biz/AdvSQLOnline
- **DB2 for i forum**

- **Technology Updates**
HTTP Functions

- SQL UDFs and UDTFs
  - Defined in the SYSTOOLS schema
    - Not in default SQL Search Path
    - Home of DB2 for i supplied tools and examples
    - Not officially part of a product, or eligible for support

- Ready for use
  - Very stable functions
  - Make sure you load the latest fix pack

- Fast-Start to building your own applications

- Requires Java 1.6 (5761-JV1) or later

- Functions visible with IBM Navigator for i
HTTP and DB2

- Rational
  - Implement businesses processes using remote services
    - Send Data FROM DB2 \(\text{(POST)}\)
    - Retrieve External data
      - Load Data into DB2 \(\text{(GET)}\)

- Invoke HTTP methods
  - Works for REST
  - Works for SOA/SOAP
  - The "manual" for the web service defines which requests to make
HTTP Functions

- **Scalar functions**
  - Send both headers and the message
  - Retrieve “Just the response message”
    - Does not return response headers (except for HTTPHEAD)

- **Verbose Table Functions**
  - Send both headers and the message
  - Retrieves the server’s response HTTP header and response message
    - Two columns

- Client supplies and interprets HTTP headers as serialized XML
  - HTTP Functions will convert to/from XML to HTTP format

- Character (CLOB) and Binary (BLOB) versions of the functions
Scalar Function GET Example

- Get request
- HTTP Header parameter says
  - Redirects should be followed
  - HTML should be returned

DECLARE returned_blob BLOB(1G);
SET returned_blob = SYSTOOLS.HTTP_GETBLOB(
    'http://www.example.com',
    '<httpHeader followRedirects="true">
        <header name="Accept" value="text/html" />
    </httpHeader>'
);

Returned data is in a BLOB

http method (GET)
http address (URL)
http request header
  This header says "The server should return the response in html format"

Returned data
3C21646F63747970652068746D6C3E0A3C68746D6C3E0A3C68561643E0A2020203
Scalar Function GET Example

- Get request
- HTTP Header parameter says
  - Redirects should be followed
  - HTML should be returned

```
DECLARE returned_Clob CLOB(1G);
SET returned_clob = SYSTOOLS.HTTP_GET_CLOB('http://www.example.com',
  '<httpHeader followRedirects="true">
    <header name="Accept" value="text/html" />
  </httpHeader>',
);
```

Returned data is in a CLOB

http method (GET)

http address (URL)

http request header
This header says “The server should return the response in html format”

```
<doctype html><html><head><title>Example Domain</title><meta charset=...>
```
Scalar Function POST Example

- POST request
- Default HTTP Header
- Send and receive CLOB data
  - Note: GET_CLOB_FROM_FILE requires commit

```sql
VALUES
SYSTOOLS.HTTP.POST_CLOB('http://www.example.com',
                           '',
                           GET_CLOB_FROM_FILE('/home/ntl/clobdata.txt')
);
```

http method (POST) → http address (URL) → http request header → Returned data is in a CLOB → Sent Data is a CLOB → http response message

00001
```html
<!doctype html><html><head>  <title>Example Domain</title>  <meta charset=...>
```
### Verbose Table Functions

- **Table Result with two columns**
  - HTTP Response Headers
  - HTTP Response message

```
SELECT *
FROM TABLE (SYSTOOLS.HTTPGETCLOBVERBOSE('http://www.example.com', ''))
) result_set;
```

**http method (GET)**

**Returned data is in a CLOB**

**Table Function**

**Response Message Body**

<table>
<thead>
<tr>
<th>RESPONSES</th>
<th>RESPONSE-HTTPHEADER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example Domain</td>
<td>text/html &quot;Content-Type&quot;/</td>
</tr>
<tr>
<td>Example Domain</td>
<td>text/html &quot;Accept-Ranges&quot;/</td>
</tr>
<tr>
<td>Example Domain</td>
<td>text/html &quot;X-Cache&quot;/</td>
</tr>
<tr>
<td>Example Domain</td>
<td>text/html &quot;Date&quot;/</td>
</tr>
</tbody>
</table>

**Response Header**

```
<?xml version="1.0" encoding="UTF-8"?>
<httpHeader responseCode="200">
  <responseMessage>OK</responseMessage>
  <header value="HTTP/1.1 200 OK" name="HTTP_RESPONSE_CODE"/>
  <header value="Fri, 09 Aug 2013 23:54:35 GMT" name="Last-Modified"/>
  <header value="max-age=604800" name="Cache-Control"/>
  <header value="ECS (mdw/1275)" name="Server"/>
  <header value="Wed, 04 Feb 2015 19:52:10 GMT" name="Expires"/>
  <header value="359670651" name="ETag"/>
  <header value="" name="x-ec-custom-error"/>
  <header value="1270" name="Content-Length"/>
  <header value="bytes" name="Accept-Ranges"/>
  <header value="HIT" name="X-Cache"/>
  <header value="text/html" name="Content-Type"/>
  <header value="Wed, 20 Jan 2015 19:52:10 GMT" name="Date"/>
</httpHeader>
```
SQLSTATE

- Verbose (Table) Functions
  - If the response from the server does not indicate success
    - This is a warning
  - If the server cannot be contacted

- non-Verbose (Scalar) functions
  - If the response from the server does not indicate success,
  - If the server cannot be contacted,

- Code should be prepared to handle both connection and server errors.
Utility Functions

- See the whitepaper a list of other "related" functions
  - [https://www-304.ibm.com/partnerworld/wps/servlet/ContentHandler/stg_ast_sys_wp_access_web_service_db2_i_udf](https://www-304.ibm.com/partnerworld/wps/servlet/ContentHandler/stg_ast_sys_wp_access_web_service_db2_i_udf)

- Encode/Decode a URL
- Encode/Decode data as base64

- Useful for
  - building URLs
  - handling authorizations
    - User id & password are often base64 encoded
The SQL message text in the diagnostic area often times contains better error information
– Easier to work with from SQL interfaces, not as easy in "green screen"

The Job CCSID should NOT be 65535 (hex/binary)
– Java and 65535 don't mix

For a POST, the data to send must be of the same type as the function name.
– e.g. HTTPPOSTCLOB must be provided a CLOB
– XML is NOT a subtype of CLOB or BLOB, an XML value must be serialized
– Errors here might appear as "Function Not Found" errors

Java 1.6 (or later) must be installed

Be sure to read the whitepaper!
Example – web API to send email

- Implement a **business process**
  - Send an email to the customer if
    * Address is changed
    * Phone number is changed

- Would like to (re)use a PHP script
  - Available via a local web server
  - Takes XML as input

```
<?php

/** load and parse the XML document **/
$xmldoc = file_get_contents('php://input');

/** Notify the client we got the message (Async) **/
sendOKResponse();

/** Build Email Headers */
$headers = "MIME-Version: 1.0" . "\n\n" .
   "Content-type:text/html;charset=UTF-8" . "\n\n" .
   "From: <webmaster@chapt3.rch.stqlabs.ibm.com>" . "\n\n";

foreach ($xmldoc[0]->current_data as $curdata) {

    /** Build Email Message Body **/
    $htmldata = <<<EOS
<html><body>Your user information has been changed.<br/><br/>
<h3><font color="red"> New User Information: </font></h3>
<table border="1" style="width:50%">
<tr><th>Name</th><th>Phone</th><th>Address</th></tr>
<tr><td>{$curdata->userid }</td><td>{$curdata->phone}</td><td>{$curdata->address}</td></tr>
</table></body></html>
EOS;

// send email
mail($curdata->email, "User Profile Changed", $htmldata, $headers);
}
```
CREATE OR REPLACE TRIGGER ntl.changed_trigger
AFTER UPDATE OF user_address, user_phone ON ntl.userinfo
REFERENCING NEW AS newr
FOR EACH ROW MODE DB2ROW
BEGIN
DECLARE email_result BLOB DEFAULT NULL;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET email_result =
SYSTOOLS.HTTPPOSTBLOB(
'http://localhost:10088/example_php_db2/send_email/index.php',
'',
XMLSERIALIZE( XMLEMENT(NAME "user_info",
    XMLEMENT(NAME "current_data",
        XMLFOREST(newr.user_name AS "userid",
                newr.user_email AS "email",
                newr.user_address AS "address",
                newr.user_phone AS "phone"
        )
    )
) AS BLOB)
);
END;

Scalar HTTP Function

XML Message to send
Update

UPDATE ntl.userinfo SET user_address = 'Rochester, MN' WHERE pk = 1;
Same idea:
Using a Statement Trigger and Query of the transition table

```
CREATE OR REPLACE TRIGGER ntl.changed_trigger
AFTER UPDATE OF user_address, user_phone ON ntl.userinfo
REFERENCING NEW TABLE AS newt FOR EACH STATEMENT MODE DB2SQL
BEGIN
  DECLARE email_result BLOB DEFAULT NULL;
  SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
  SET email_result =
      SYSTOOLS.HTTPPOSTBLOB('http://localhost:10088/example_php_db2/send_email/index.php',
        '',
        (SELECT
          XMLSERIALIZE(
            XMLGROUP(
              newt.user_name AS "userid",
              newt.user_email AS "email",
              newt.user_address AS "address",
              newt.user_phone AS "phone"
            )
          AS BLOB(2G) )
        FROM newt GROUP BY ()
      );
END;
```

Nested Query – Result is a single XML document!

One XML document/request for all users that need to be notified
Example – Retrieve latest PTF levels from IBM

- IBM has a web service that contains the latest and greatest PTF numbers

- Challenge: Retrieve this information and compare with PTF levels on a system
Example – Retrieve latest PTF levels from IBM

```sql
SELECT xt.* FROM
XMLTABLE('ibm_data/all_psp/psp'
PASSING
) AS "ibm_data"
COLUMNS
  release CHAR(10) PATH 'release',
  number CHAR(10) PATH 'number',
  title VARCHAR(1000) PATH 'title',
  level INTEGER PATH 'level'
) xt;
```

Note: IBM supplies the SYSTOOLS.GROUP_PTF_CURRENCY view that implements a more powerful version of this query........
SYSTOOLS.GROUP_PTF_CURRENCY View

SELECT * from SYSTOOLS.GROUP_PTF_CURRENCY
WHERE PTF_GROUP_RELEASE = 'R720'
ORDER BY ptf_group_level_available -
ptf_group_level_installed DESC

Current or behind on service?
PTF Group Info
Level installed on this partition
Level available from IBM
Date that IBM last updated this group
Example – Retrieve Postal Code Information

Check HTTP response Codes

- Non-IBM Service (for example purposes)
  - http://www.geonames.org/

- Need to also get the HTTP header information
  - Can do this with the VERBOSE table functions

Response Headers

```xml
<httpHeader responseCode="200">
  <responseMessage>OK</responseMessage>
  <header name="HTTP_RESPONSE_CODE" value="HTTP/1.1 200 OK"/>
  <header name="Cache-Control" value="no-cache"/>
  <header name="Server" value="Apache/2.4.6 (Linux/SUSE)"/>
  <header name="Access-Control-Allow-Origin" value="*"/>
  <header name="Keep-Alive" value="timeout=15, max=100"/>
  <header name="Transfer-Encoding" value="chunked"/>
  <header name="Date" value="Wed, 17 Dec 2014 21:18:25 GMT"/>
  <header name="Connection" value="Keep-Alive"/>
  <header name="Content-Type" value="text/xml; charset=UTF-8"/>
</httpHeader>
```

Response Message

```xml
<geonames>
  <totalResultsCount>1</totalResultsCount>
  <code>
    <postalcode>55901</postalcode>
    <name>Rochester</name>
    <countryCode>US</countryCode>
    <lat>44.04957</lat>
    <lng>-92.48962</lng>
    <adminCode1>MN</adminCode1>
    <adminName1>Minnesota</adminName1>
    <adminCode2>109</adminCode2>
    <adminName2>Olmsted</adminName2>
    <adminCode3/>
    <adminName3/>
  </code>
</geonames>
```
Retrieve Postal Code Information (Non-IBM API) (Including HTTP server response code)

With postal_codes_service AS ( 
SELECT responsehttpheader, responsemsg 
FROM TABLE( SYSTOOLS.HTTPGETBLOBVERBOSE( 
'http://api.geonames.org/postalCodeSearch?postalcode=55901&maxRows=10&username=demo', 
'<HttpHeader> <header name="Accept" value="application/xml"/> </HttpHeader>' 
) pc 
) 
SELECT server_http_code, server_http_message, Postal_Code, City, County, State, Country, lat, lng 
FROM 
( postal_codes_service 
CROSS JOIN 
XMLTABLE('httpresp/httpHeader' 
PASSING XMLPARSE(DOCUMENT postal_codes_service.responsehttpheader) AS "httpresp" 
COLUMNS 
server_http_code INTEGER PATH '@responseCode', 
server_http_message VARCHAR(1000) PATH 'responseMessage' 
) headers 
) LEFT OUTER JOIN 
XMLTABLE('$postalInfo/geonames/code' 
PASSING XMLPARSE(DOCUMENT postal_codes_service.responsemsg) AS "postalInfo" 
COLUMNS 
postal_code INTEGER PATH 'postalcode', 
city VARCHAR(100) PATH 'name', 
county VARCHAR(100) PATH 'adminName2', 
state CHAR(2) PATH 'adminCode1', 
country CHAR(25) PATH 'countryCode', 
lng DOUBLE PATH 'lng' 
) postal_data 
ON (1=1);
Retrieve Postal Code Information

WITH postal_codes_service AS (  
SELECT responsehttpheader, responsemsg  
FROM TABLE(  
SYSTOOLS.HTTPGETBLOBVERBOSE(  
'http://api.geonames.org/postalCodeSearch?postalcode=55901&maxRows=10&username=demo',  
'<HttpHeader> <header name="Accept" value="application/xml"/> </HttpHeader>'  
) pc  
)  
) pc  
SELECT server_http_code, server_http_message, Postal_Code, City, County, State, Country, lat, lng  
FROM  
( postal_codes_service  
CROSS JOIN  
XMLTABLE('$httpresp/httpHeader'  
PASSING XMLPARSE(DOCUMENT postal_codes_service.responsehttpheader) AS "httpresp"  
COLUMNS  
server_http_code INTEGER PATH '@responseCode',  
server_http_message VARCHAR(1000) PATH 'responseMessage'  
) headers  
)  
LEFT OUTER JOIN  
XMLTABLE('postalInfo/geonames/code'  
PASSING XMLPARSE(DOCUMENT postal_codes_service.responsemsg) AS "postalInfo"  
COLUMNS  
postal_code INTEGER PATH 'postalcode',  
state CHAR(2) PATH 'adminCode1',  
city VARCHAR(100) PATH 'name',  
latitude DOUBLE PATH 'lat',  
county VARCHAR(100) PATH 'adminName2',  
lng DOUBLE PATH 'lng',  
country CHAR(25) PATH 'countryCode'  
) postal_data  
ON (1=1);
Left Outer Join (Retrieve Postal Code Information)

With postal_codes_service AS (  
SELECT responsehttpheader, responsemsg  
FROM TABLE(  
SYSTOOLS.HTTPGETBLOBVERBOSE(  
'<!HttpHeader> <header name="Accept" value="application/xml"/> </HttpHeader>  
)  
) pc  
)  
SELECT server_http_code, server_http_message, Postal_Code, City, County, State, Country, lat, lng  
FROM  
(postal_codes_service  
CROSS JOIN  
XMLTABLE('$$httpresp/httpHeader'  
PASSING XMLPARSE(DOCUMENT postal_codes_service.responsehttpheader) AS "httpresp"  
COLUMNS  
server_http_code INTEGER PATH '@responseCode',  
server_http_message VARCHAR(1000) PATH 'responseMessage'  
) headers  
) LEFT OUTER JOIN  
XMLTABLE('$$postalInfo/geonames/code'  
PASSING XMLPARSE(DOCUMENT postal_codes_service.responsemsg) AS "postalInfo"  
COLUMNS  
postal_code INTEGER PATH 'postalcode',  
city VARCHAR(100) PATH 'name',  
county VARCHAR(100) PATH 'adminName2',  
state CHAR(2) PATH 'adminCode1',  
country CHAR(25) PATH 'countryCode',  
lat DOUBLE PATH 'lat',  
lng DOUBLE PATH 'lng'  
) postal_data  
ON (1=1);  

Trivial Join  
To return a row (for headers) even IF the response message doesn't return any rows
SQL/XML Increases the usability of the HTTP Functions
XML Overview

- Easiest to send/receive XML data
  - Many web-services can produce & understand XML

- Necessitates the need to
  - Publish relational data as XML
  - Shred XML data to a result-set

- Part-14 ISO SQL/XML Standard
  - Built into DB2 for i 7.1
**XML Handling 101**

- **Approach #1:**
  - parse/process XML as events (Think SAX parser, XML-SAX, XML-INTO)
    
    ```
    <A> text data </A>
    ```

    - Create XML by producing tags & content (start tags, end tags, XML data)

- **Approach #2 - The DB2 approach:**
  - An XML document is an instance of the XPath 2.0 Data model
    - Think of an XML document as a tree of nodes
  - Easy to **query** with XPath `/A/text()`
    - Model describes relationships between data
    - Query capabilities important to databases (DB2)

  - Nodes and trees are constructed and published (rather than tags and content)
Data Models

XML Document Tree (XPath Logical model)

SQL/Relational (Rows & Columns)

<table>
<thead>
<tr>
<th>User name</th>
<th>email</th>
<th>Address</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
</tr>
</tbody>
</table>

Serialized XML

```
<user_info>
  <current_data>
    <userid>Nick Lawrence</userid>
    <email>ntl@us.ibm.com</email>
    <address>Rochester, MN</address>
    <phone>507-222-1234</phone>
  </current_data>
</user_info>
```
**Data Models**

### SQL/Relational (Rows & Columns)

<table>
<thead>
<tr>
<th>User name</th>
<th>email</th>
<th>Address</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
</tr>
</tbody>
</table>

### XML Document Tree

- **Document**
  - **Element user_info**
    - **Element current_data**
      - **Element userid**
      - **Element email**
      - **Element address**
      - **Element phone**

- **Serialized XML**

```xml
<user_info>
  <current_data>
    <userid>Nick Lawrence</userid>
    <email>ntl@us.ibm.com</email>
    <address>Rochester, MN</address>
    <phone>507-222-1234</phone>
  </current_data>
</user_info>
```

**Serialized XML** is CLOB, BLOB, VARCHAR, etc... no consideration of XML features or Structure
IBM Data Studio representation

Serialized XML (aka source or character)

Working with XML in DB2 is sort of like constructing XML in "design" mode

Parsed XML (Design/Tree of nodes)
CLOB vs XML (unusual example)

- Consider the String (which is also legal XML)
  
  - `<String> hello world </String>`

- `XMLDOCUMENT(XMLELEMENT(NAME "root", XMLPARSE(DOCUMENT char_string)))`

  ![Diagram](image)

  Serialize

  `<root><String> hello world </String></root>`

- `XMLDOCUMENT(XMLELEMENT(NAME "root", char_string))`

  ![Diagram](image)

  Serialize

  `<root>&lt;String&gt; hello World &lt;/String&gt;&lt;/root>`
CLOB vs XML (unusual example)

- Consider the String (which is also legal XML)
  \(<String> hello world </String>\)

- \(\text{XMLDOCUMENT} (\text{XMLELEMENT} (\text{NAME} "root", \text{XMLPARSE} (\text{DOCUMENT} \text{char_string})))\)

- \(\text{XMLDOCUMENT} (\text{XMLELEMENT} (\text{NAME} "root", \text{char_string})\))

XMLPARSE interpret the Char/Binary Data as XML structure and create an instance of the XML type (XPath model)

Other conversions of SQL string -> XML just assume an "atomic" string value.
CLOB vs XML (unusual example)

- Consider the String (which is also legal XML)
  – `<String> hello world </String>`

- `XMLDOCUMENT(XMLELEMENT(NAME "root", XMLPARSE(DOCUMENT char_string)))`

  Serialize
  ```
  <root><String> hello world </String></root>
  ```

- `XMLDOCUMENT(XMLELEMENT(NAME "root", char_string))`

  Serialize
  ```
  <root>&lt;String&gt; hello World &lt;/String&gt;</root>
  ```

**NO need for DB2 literally build a tree of nodes implementation inside DB2…this is merely how the XML data type is defined!!!**
Data Models

SQL/Relational (Rows & Columns)

<table>
<thead>
<tr>
<th>User name</th>
<th>email</th>
<th>Address</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
</tr>
</tbody>
</table>

XML Document Tree (Logical model)

- Document
  - Element current_data
    - Element user_info
      - Element userid
      - Element email
      - Element address
      - Element phone

Serialized XML (Character version)

```
<user_info>
  <current_data>
    <userid>Nick Lawrence</userid>
    <email>ntl@us.ibm.com</email>
    <address>Rochester, MN</address>
    <phone>507-222-1234</phone>
  </current_data>
</user_info>
```
**Data Models**

**XML Document Tree**
(Logical model)

- **Document**
  - **Element user_info**
    - **Element current_data**
      - **Element userid**
      - **Element email**
      - **Element address**
      - **Element phone**

**User Information**

<table>
<thead>
<tr>
<th>User name</th>
<th>Email</th>
<th>Address</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
</tr>
</tbody>
</table>

**Serialized XML (Character version)**

```xml
<user_info>
  <current_data>
    <userid>Nick Lawrence</userid>
    <email>ntl@us.ibm.com</email>
    <address>Rochester, MN</address>
    <phone>507-222-1234</phone>
  </current_data>
</user_info>
```

**SQL/Relational (Rows & Columns)**

- XML TABLE
- XMLPARSE

**Data Models Diagram**
Publishing Examples – relational model

INNER JOIN

<table>
<thead>
<tr>
<th>PK</th>
<th>USER_NAME</th>
<th>USER_EMAIL</th>
<th>USER_ADDRESS</th>
<th>USER_PHONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
</tr>
<tr>
<td>2</td>
<td>George</td>
<td><a href="mailto:george@xyz.ibm.com">george@xyz.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-5678</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USER_ID</th>
<th>TRANSID</th>
<th>PTIME</th>
<th>ITEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2014-12-03 11:48:57</td>
<td>TV</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2014-12-04 11:48:57</td>
<td>Computer</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2014-12-05 11:48:57</td>
<td>Phone</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2015-01-02 11:48:57</td>
<td>Coffee</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>2015-02-02 11:48:57</td>
<td>Apple</td>
</tr>
</tbody>
</table>
Publishing example - 1

```sql
SELECT PK,
    XMLSERIALIZE( 
        XMLDOCUMENT( 
                XMLELEMENT(NAME "User_Info", 
                        XMLFOREST(USER_NAME AS "Name", 
                                    USER_EMAIL AS "Email", 
                                    USER_ADDRESS AS "Address", 
                                    USER_PHONE AS "Phone") 
                ) 
            ) AS CLOB ) AS XML_DOC
FROM USERINFO;
```
SELECT PK,
XMLSERIALIZE(
XMLDOCUMENT(
    XMLELEMENT(NAME "User_Info",
    XMLFOREST(USER_NAME AS "Name",
    USER_EMAIL AS "Email",
    USER_ADDRESS AS "Address",
    USER_PHONE AS "Phone")
    )
) AS CLOB
AS XML_DOC
FROM USERINFO;

XML Serialization can be implicit when XML is retrieved into a host variable. Sometimes examples take advantage of this!

BLOB data type is usually better for applications.

CLOB is easier for humans to read.

XML is serialized for applications….not “pretty printed”

“Pretty printed” with a web-browser

<table>
<thead>
<tr>
<th>PK</th>
<th>XML_DOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;User_Info&gt;&lt;Name&gt;Nick Lawrence&lt;/Name&gt;&lt;Email&gt;<a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a>&lt;/Email&gt;&lt;A...</td>
</tr>
<tr>
<td>2</td>
<td>&lt;User_Info&gt;&lt;Name&gt;George&lt;/Name&gt;&lt;Email&gt;<a href="mailto:george@wxyz.ibm.com">george@wxyz.ibm.com</a>&lt;/Email&gt;&lt;Ad...</td>
</tr>
</tbody>
</table>
Publishing functions create XML nodes

XMLDOCUMENT(

  XMLELEMENT(NAME "User_Info",

    XMLFOREST(USER_NAME AS "Name",
    USER_EMAIL AS "Email",
    USER_ADDRESS AS "Address",
    USER_PHONE AS "Phone")

  )

)

document

Element "User_Info"

Element "Name"
  Nick Lawrence

Element "Email"
  ntl@us.ibm.com

Element "Address"
  Rochester, MN

Element "Phone"
  507-222-1234
Some Functions simplify creating groups of nodes

XMLFOREST is a function that combines XMLELEMENT, and XMLCONCAT

Could write XMLFOREST as

XMLDOCUMENT (XMLFOREST (XMLELEMENT (NAME "User_Info", USER_NAME AS "Name", USER_EMAIL AS "Email", USER_ADDRESS AS "Address", USER_PHONE AS "Phone")))

Nick Lawrence ntl@us.ibm.com Rochester, MN 507-222-1234

George george@woyz.ibm.com Rochester, MN 507-222-5978
The XMLDOCUMENT function is "Optional".

The XMLDOCUMENT function exists for compatibility with other platforms - A document node is always published on DB2 for i.

(The document node defines the "root" of the tree in the XPath model - it does not have an impact on the serialized document.)
Publishing example – Aggregation

```
SELECT
    XMLSERIALIZE(
        XMLDOCUMENT(
            XMLELEMENT(NAME "All_Users",
                XMLAGG(
                    XMLELEMENT(NAME "User_Info",
                        XMLFOREST(USER_NAME AS "Name",
                            USER_EMAIL AS "Email",
                            USER_ADDRESS AS "Address",
                            USER_PHONE AS "Phone")
                    ORDER BY USER_NAME DESC
                )
            ORDER BY USER_NAME DESC
        )
    )
AS XML_DOC
FROM USERINFO
GROUP BY ();
```

XMLAGG - Aggregate multiple rows into a single XML value

Ordering of aggregated rows

“Normal” row Grouping and Selection chooses which rows are aggregated
SELECT XMLSERIALIZE(
    XMLDOCUMENT(
        XMLELEMENT(NAME "All_Users",
            XMLELEMENT(NAME "User_Info",
                XMLFOREST(USER_NAME AS "Name",
                        USER_EMAIL AS "Email",
                        USER_ADDRESS AS "Address",
                        USER_PHONE AS "Phone")
            ORDER BY USER_NAME DESC)
    )
) AS CLOB
FROM USERINFO
GROUP BY ()

DB2 Optimizer does its job.....
Repeating Elements and a Join

- Repeating purchases element, single `user_info` element.
  - `USERINFO` has a 1 to many relationship with `PURCHASES`

```xml
<user_purchases>
  <user_info>
    <name>Nick Lawrence</name>
    <email>ntl@us.ibm.com</email>
  </user_info>
  <purchases>
    <purchase>
      <item>TV</item>
      <time>2014-12-03T11:48:57.973347</time>
    </purchase>
    <purchase>
      <item>Computer</item>
      <time>2014-12-04T11:48:57.973347</time>
    </purchase>
    <purchase>
      <item>Phone</item>
      <time>2014-12-05T11:48:57.973347</time>
    </purchase>
  </purchases>
</user_purchases>
```

<table>
<thead>
<tr>
<th>PK</th>
<th>USER_NAME</th>
<th>USER_EMAIL</th>
<th>USER_ADDRESS</th>
<th>USER_PHONE</th>
<th>USER_ID</th>
<th>TRANSID</th>
<th>PTIME</th>
<th>ITEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
<td>1</td>
<td>1</td>
<td>2014-12-03 11:48...</td>
<td>TV</td>
</tr>
<tr>
<td>1</td>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
<td>1</td>
<td>2</td>
<td>2014-12-04 11:48...</td>
<td>Computer</td>
</tr>
<tr>
<td>1</td>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
<td>1</td>
<td>3</td>
<td>2014-12-05 11:48...</td>
<td>Phone</td>
</tr>
<tr>
<td>2</td>
<td>George</td>
<td><a href="mailto:george@woyz.ibm.com">george@woyz.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-5678</td>
<td>2</td>
<td>4</td>
<td>2015-01-02 11:48...</td>
<td>Coffee</td>
</tr>
<tr>
<td>2</td>
<td>George</td>
<td><a href="mailto:george@woyz.ibm.com">george@woyz.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-5678</td>
<td>2</td>
<td>5</td>
<td>2015-02-02 11:48...</td>
<td>Apple</td>
</tr>
</tbody>
</table>
Repeating Purchases Element, Single user_info Element.
- USERINFO table has a 1 to many relationship with PURCHASES table.

Assume one XML doc per PK.

One to many...

One to one...

<table>
<thead>
<tr>
<th>PK</th>
<th>USER_NAME</th>
<th>USER_EMAIL</th>
<th>USER_ADDRESS</th>
<th>USER_PHONE</th>
<th>USER_ID</th>
<th>TRANSD</th>
<th>PTIME</th>
<th>ITEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
<td>1</td>
<td></td>
<td></td>
<td>TV</td>
</tr>
<tr>
<td>1</td>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
<td>1</td>
<td></td>
<td></td>
<td>Computer</td>
</tr>
<tr>
<td>1</td>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
<td>1</td>
<td></td>
<td></td>
<td>Phone</td>
</tr>
<tr>
<td>2</td>
<td>George</td>
<td><a href="mailto:george@woyz.ibm.com">george@woyz.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-5678</td>
<td>2</td>
<td>42014-12-05T11:48:57.973347</td>
<td></td>
<td>Coffee</td>
</tr>
<tr>
<td>2</td>
<td>George</td>
<td><a href="mailto:george@woyz.ibm.com">george@woyz.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-5678</td>
<td>2</td>
<td>42015-01-02T11:48:57.973347</td>
<td></td>
<td>Apple</td>
</tr>
</tbody>
</table>
Repeating Elements and a Join

- Repeating purchases element, single user_info element.
  - USERINFO has a 1 to many relationship with PURCHASES

- Strategy - Build the document "in pieces"
  - Get purchase elements for each user_id and aggregate those into a single XML value
  - Attach value to purchases element of user_purchases (Involves a Join)

```
  <user_purchases>
    <user_info>
      <name>Nick Lawrence</name>
      <email>ntl@us.ibm.com</email>
    </user_info>
    <purchases>
      <purchase>
        <item>TV</item>
        <time>2014-12-03T11:48:57.973347</time>
      </purchase>
      <purchase>
        <item>Computer</item>
        <time>2014-12-04T11:48:57.973347</time>
      </purchase>
      <purchase>
        <item>Phone</item>
        <time>2014-12-05T11:48:57.973347</time>
      </purchase>
    </purchases>
  </user_purchases>
```

```
<table>
<thead>
<tr>
<th>PK</th>
<th>USER_NAME</th>
<th>USER_EMAIL</th>
<th>USER_ADDRESS</th>
<th>USER_PHONE</th>
<th>USER_ID</th>
<th>TRANSID</th>
<th>PTIME</th>
<th>ITEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
<td>1</td>
<td></td>
<td>2014-12-03 11:48...</td>
<td>TV</td>
</tr>
<tr>
<td>1</td>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
<td>1</td>
<td></td>
<td>2014-12-04 11:48...</td>
<td>Computer</td>
</tr>
<tr>
<td>1</td>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
<td>1</td>
<td></td>
<td>2014-12-05 11:48...</td>
<td>Phone</td>
</tr>
<tr>
<td>2</td>
<td>George</td>
<td><a href="mailto:george@woz.ibm.com">george@woz.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-5678</td>
<td>2</td>
<td>4</td>
<td>2015-01-02 11:48...</td>
<td>Coffee</td>
</tr>
<tr>
<td>2</td>
<td>George</td>
<td><a href="mailto:george@woz.ibm.com">george@woz.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-5678</td>
<td>2</td>
<td>5</td>
<td>2015-02-02 11:48...</td>
<td>Apple</td>
</tr>
</tbody>
</table>
```
Join using a Common Table Expression (CTE)

- Sometimes it’s easier to build an XML document “in pieces”

```
WITH user_purchases AS (
    SELECT user_id PK,
            XMLAGG(
                XMLFOREST(ITEM AS "item", PTIME AS "time")
            ) AS XMLDOC
    FROM PURCHASES
    GROUP BY user_id
)

SELECT XMLELEMENT(NAME "user_purchases",
                    XMLELEMENT(NAME "user_info",
                                XMLFOREST(USER_NAME AS "name", USER_EMAIL AS "email")),
                    XMLELEMENT(NAME "purchases", up.xmldoc))
FROM NICK.USERINFO ui INNER JOIN user_purchases up ON (ui.pk = up.pk)
WHERE ui.user_name = 'Nick Lawrence';
```

Include purchase elements (from the join)
Common Table Expression CTE – another look

WITH user_purchases AS (  
  SELECT user_id PK,  
      XMLAGG(  
          XMLELEMENT(NAME "purchase",  
            XMLFOREST(ITEM AS "item", PTIME AS "time")  
          )  
      ) AS XMLDOC  
  FROM PURCHASES  
  GROUP BY user_id  
)  

SELECT *  
FROM USERINFO ui INNER JOIN user_purchases up ON (ui.pk = up.pk)  
WHERE ui.user_name = 'Nick Lawrence';

User_purchases

<table>
<thead>
<tr>
<th>PK</th>
<th>XMLDOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;purchase&gt;&lt;item&gt;TV&lt;/item&gt;&lt;time&gt;2014-12-03T11:48:57.973347&lt;/time&gt;&lt;/purchase&gt;&lt;purchase&gt;&lt;item&gt;Computer&lt;/item&gt;&lt;time&gt;...&lt;/time&gt;&lt;/purchase&gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt;purchase&gt;&lt;item&gt;Coffee&lt;/item&gt;&lt;time&gt;2015-01-02T11:48:57.973347&lt;/time&gt;&lt;/purchase&gt;&lt;purchase&gt;&lt;item&gt;Apple&lt;/item&gt;&lt;time&gt;...&lt;/time&gt;&lt;/purchase&gt;</td>
</tr>
</tbody>
</table>

User_info

<table>
<thead>
<tr>
<th>PK</th>
<th>USER_NAME</th>
<th>USER_EMAIL</th>
<th>USER_ADDRESS</th>
<th>USER_PHONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
</tr>
<tr>
<td>2</td>
<td>George</td>
<td><a href="mailto:george@usz.ibm.com">george@usz.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-5678</td>
</tr>
</tbody>
</table>
Common Table Expression CTE - Optimization

- The CTE is optimized into the rest of the query

```sql
WITH user_purchases AS (
    SELECT user_id PK,
        XMLAGG(
            XMLELEMENT(NAME "purchase",
                XMLFOREST(ITEM AS "item", PTIME AS "time")
            )
        ) AS XMLDOC
    FROM PURCHASES
    GROUP BY user_id
)

SELECT XMLELEMENT(NAME "user_purchases",
    XMLELEMENT(NAME "user_info",
        XMLFOREST(USER_NAME AS "name",
            USER_EMAIL AS "email")),
    XMLELEMENT(NAME "purchases", up.xmldoc)
)
FROM USERINFO ui INNER JOIN user_purchases up ON (ui.pk = up.pk)
WHERE ui.user_name = 'Nick Lawrence';
```
Views – with XMLROW and XMLGROUP

- XML syntax does not have to be bulky in all cases

```sql
CREATE OR REPLACE VIEW user_purchases AS (
    SELECT user_id PK,
    XMLGROUP( ITEM AS "item", PTIME AS "time"
               OPTION ROOT "purchases" ROW "purchase") partial_doc
    FROM PURCHASES
) GROUP BY user_id;
```

```sql
SELECT XMLELEMENT (Name "user_purchases",
    XMLROW(USER_NAME AS "name", USER_EMAIL AS "email"
            OPTION ROW "user_info"),
    up.partial_doc
) FROM userinfo ui INNER JOIN
    user_purchases up ON (ui.pk = up.pk)
WHERE ui.user_name = 'Nick Lawrence';
```

**XMLGROUP - Shorthand for:**

```
XMLELEMENT (NAME "root",
    XMLAGG (XMLROW(....))
)
```

**XMLROW - Shorthand for:**

```
XMLELEMENT (NAME "row",
    XMLFOREST (...)
)
```
Publishing Functions – Data type conversion

- SQL Data types converted to XML equivalents
  - DATE, TIME, TIMESTAMP -> xs:date, xs:time, xs:dateTime
  - Binary to base64 or hex

- Character data escaped as it is serialized

```xml
VALUES XMLELEMENT(NAME "Root", CURRENT TIMESTAMP);
```

```xml
<XMLElement Name="Root">
  Value A is < Value B & Value C is <? Value D'>
</XMLElement>
```

```xml
VALUES XMLELEMENT(NAME "Root", CURRENT TIMESTAMP);
```

```xml
<XMLElement Name="Root">
  2015-01-05T12:04:17.358679
</XMLElement>
```
Be Careful with Data Types

- Suppose the column in the view is serialized as a CLOB…

```sql
CREATE OR REPLACE VIEW user_purchases AS (
    SELECT user_id PK,
    XMLSERIALIZE(
        XMLGROUP( ITEM AS "item", PTIME AS "time"
            OPTION ROOT "purchases" ROW "purchase")
            AS CLOB(2G) CCSID 1208
        ) partial_doc
    FROM PURCHASES
    GROUP BY user_id
);
```

```sql
SELECT XMLELEMENT(Name "user_purchases",
    XMLROW(USER_NAME AS "name", USER_EMAIL AS "email" OPTION ROW "user_info"),
    up.partial_doc
) FROM userinfo ui INNER JOIN user_purchases up ON (ui.pk = up.pk)
WHERE ui.user_name = 'Nick Lawrence';
```

The character data is escaped so that it has the correct representation.
Be Careful with Data Types (2)

- Binary data is (by default) encoded as base64

```sql
CREATE VARIABLE nick.mystring VARCHAR(1000) CCSID 65535 DEFAULT 'hello world';
VALUES
  XMLELEMENT(NAME "root",
              nick.mystring
           );
```

The binary data is encoded so that it has the correct representation.
Key Points

- Publish Nodes, not tags
- SQL data types are very important
  - XML includes structure and is not a synonym for character or binary
  - SQL types have corresponding XML data types
    - SQL TIMESTAMP -> xs:dateTime
    - SQL binary -> base64 encoding

See also
- SQL/XML Tutorial
Nick Lawrence  ntl@us.ibm.com  Rochester, MN  507-222-1234

User name  email  Address  phone
Nick Lawrence  ntl@us.ibm.com  Rochester, MN  507-222-1234

Serialized XML (Character version)

<user_info>
  <current_data>
    <userid>Nick Lawrence</userid>
    <email>ntl@us.ibm.com</email>
    <address>Rochester, MN</address>
    <phone>507-222-1234</phone>
  </current_data>
</user_info>
XMLTABLE

SELECT xt.*
FROM XMLTABLE(''$d/user_info/current_data'
        PASSING XMLPARSE(DOCUMENT serialized_doc) AS "d"
        COLUMNS
        "User Name" VARCHAR(1024) PATH 'userid',
        "Email" VARCHAR(1024) PATH 'email',
        "Address" VARCHAR(1024) PATH 'address',
        "Phone" VARCHAR(30) PATH 'phone'
    ) xt;

<table>
<thead>
<tr>
<th>User Name</th>
<th>Email</th>
<th>Address</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
</tr>
</tbody>
</table>
XMLTABLE

- XML document is assigned to “d”

```
SELECT xt.*
FROM
  XMLTABLE('d/user_info/current_data'
    PASSING XMLPARSE(DOCUMENT serialized_doc) AS "d"
    COLUMNS
    "User Name" VARCHAR(1024) PATH 'userid',
    "Email" VARCHAR(1024) PATH 'email',
    "Address" VARCHAR(1024) PATH 'address',
    "Phone" VARCHAR(30) PATH 'phone'
) xt;
```

<table>
<thead>
<tr>
<th>User Name</th>
<th>Email</th>
<th>Address</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
</tr>
</tbody>
</table>
XMLTABLE

- One row for each node determined by the row expression

```sql
SELECT xt.*
FROM
XMLTABLE('d/user_info/current_data'
PASSING XMLPARSE(DOCUMENT serialized_doc) AS "d"
COLUMNS
"User Name" VARCHAR(1024) PATH 'userid',
"Email" VARCHAR(1024) PATH 'email',
"Address" VARCHAR(1024) PATH 'address',
"Phone" VARCHAR(30) PATH 'phone'
) xt;
```

<table>
<thead>
<tr>
<th>User Name</th>
<th>Email</th>
<th>Address</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
</tr>
</tbody>
</table>
XMLTABLE

- Column paths relative to node for current row

```
SELECT xt.*
FROM
XMLTABLE('d/user_info/current_data'
PASSING XMLPARSE(DOCUMENT serialized_doc) AS "d"
COLUMNS
"User Name" VARCHAR(1024) PATH 'userid',
"Email" VARCHAR(1024) PATH 'email',
"Address" VARCHAR(1024) PATH 'address',
"Phone" VARCHAR(30) PATH 'phone'
) xt;
```

XPATH types are converted to SQL TYPES

<table>
<thead>
<tr>
<th>User Name</th>
<th>Email</th>
<th>Address</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nick Lawrence</td>
<td><a href="mailto:ntl@us.ibm.com">ntl@us.ibm.com</a></td>
<td>Rochester, MN</td>
<td>507-222-1234</td>
</tr>
</tbody>
</table>
Multiple nodes from row expression

XPath variable “d”

```sql
SELECT xt.*
FROM XMLTABLE('d/user_info/current_data'
PASSING XMLPARSE(DOCUMENT serialized_doc) AS "d"
COLUMNS "User Name" VARCHAR(1024) PATH 'userid',
"Email" VARCHAR(1024) PATH 'email',
"Address" VARCHAR(1024) PATH 'address',
"Phone" VARCHAR(30) PATH 'phone'
) xt;
```
XMLTABLE Key Points

- Use XMLTABLE to extract rows and columns from XML data
- Input value must be XML in order to use step expressions
  - Doesn't make sense to look for a child node of a CLOB (xs:string)

- See also
  - SQL/XML Tutorial
Summary - Integrated Technologies
Bonus - Advanced XMLTABLE

- Simplifications
- Type conversion & functions
- Namespaces & predicates
Simplification

- Usually do not need an Xpath variable explicitly named
- Row expression paths are by default relative to a passed argument with no variable name.
  - This argument is called the 'default initial context'
- Leading lone('/') means the root document node.
  - Since the default context is the root document node, we don't need a leading '/' either

```sql
SELECT xt.*
FROM
XMLTABLE('$/user_info/current_data'
  PASSING XMLPARSE(DOCUMENT serialized_doc) AS "d"
  COLUMNS
    "User Name" VARCHAR(1024) PATH 'userid',
    "Email" VARCHAR(1024) PATH 'email',
    "Address" VARCHAR(1024) PATH 'address',
    "Phone" VARCHAR(30) PATH 'phone'
) xt;
```
Simplification (2)

- If a columns does not specify a PATH the path is the name of the column
  - Caution: XML is case sensitive…SQL column names are uppercase unless delimited.

```sql
SELECT xt.*
FROM
XMLTABLE('user_info/current_data'
  PASSING XMLPARSE(DOCUMENT serialized_doc)
COLUMNS
  "userid"   VARCHAR(1024),
  "email"   VARCHAR(1024),
  "address" VARCHAR(1024),
  "phone"   VARCHAR(30)
) xt;
```
Type Conversion

- SQL type determines the XML type (and conversion)

```sql
SELECT * FROM XMLTABLE(
    'doc'
    PASSING XMLPARSE(DOCUMENT
        '<doc>
            <val1>2015-01-01T00:00:00.000000</val1>
            <val2>&lt;not_xml&gt;not xml data&lt;/not_xml&gt;</val2>
            <val3>13.0e5</val3>
        </doc>
    )

COLUMNS
    "val1" TIMESTAMP,
    "val2" VARCHAR(200),
    "val3" DOUBLE
) xt;
```

<table>
<thead>
<tr>
<th>val1</th>
<th>val2</th>
<th>val3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015-01-01 00:00:00.000000</td>
<td>&lt;not_xml&gt;not xml data&lt;/not_xml&gt;</td>
<td>1300000.0</td>
</tr>
</tbody>
</table>
XML Explicit Types

- XPath expressions are powerful and Typed

```
SELECT * FROM XMLTABLE('doc/val'
PASSING XMLPARSE(DOCUMENT '
  <doc>
    <val>true</val>
    <val>false</val>
  </doc>')
COLUMNS
  val1 INTEGER PATH 'xs:boolean(.)'
) xt;
```

```
SELECT * FROM XMLTABLE('doc/val'
PASSING XMLPARSE(DOCUMENT '
  <doc>
    <val>1</val>
    <val>0</val>
  </doc>')
COLUMNS
  val1 CHAR(10) PATH 'xs:boolean(.)'
) xt;
```

xs:boolean -> xs:integer
  -> SQL INTEGER

xs:boolean -> xs:xstring
  -> SQL CHAR
XPath math expressions

- XPath supports very complex expressions
- Number of days between "startDate" and "endDate"?

```
SELECT * FROM XMLTABLE(
  'doc'
PASSING XMLPARSE(DOCUMENT
  '<doc>
    <startDate>2015-01-01</startDate>
    <endDate>2015-02-01</endDate>
  </doc>
')
COLUMNS
  days INTEGER PATH 'days-from-duration(xs:date(endDate) - xs:date(startDate))'
) xt;
```
XMLTABLE- Other topics

- Namespaces
- Avoiding errors due to multiple values in a single column
  - Using a predicate select first value
  - Modifying the row expression
  - Using a parent (..) step
- Handling missing elements

- These topics and more are in the SQL/XML Programming guide
Questions?
Special notices

This document was developed for IBM offerings in the United States as of the date of publication. IBM may not make these offerings available in other countries, and the information is subject to change without notice. Consult your local IBM business contact for information on the IBM offerings available in your area.

Information in this document concerning non-IBM products was obtained from the suppliers of these products or other public sources. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. Send license inquiries, in writing, to IBM Director of Licensing, IBM Corporation, New Castle Drive, Armonk, NY 10504-1785 USA.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

The information contained in this document has not been submitted to any formal IBM test and is provided "AS IS" with no warranties or guarantees either expressed or implied.

All examples cited or described in this document are presented as illustrations of the manner in which some IBM products can be used and the results that may be achieved. Actual environmental costs and performance characteristics will vary depending on individual client configurations and conditions.

IBM Global Financing offerings are provided through IBM Credit Corporation in the United States and other IBM subsidiaries and divisions worldwide to qualified commercial and government clients. Rates are based on a client's credit rating, financing terms, offering type, equipment type and options, and may vary by country. Other restrictions may apply. Rates and offerings are subject to change, extension or withdrawal without notice.

IBM is not responsible for printing errors in this document that result in pricing or information inaccuracies.

All prices shown are IBM's United States suggested list prices and are subject to change without notice; reseller prices may vary.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

Any performance data contained in this document was determined in a controlled environment. Actual results may vary significantly and are dependent on many factors including system hardware configuration and software design and configuration. Some measurements quoted in this document may have been made on development-level systems. There is no guarantee these measurements will be the same on generally-available systems. Some measurements quoted in this document may have been estimated through extrapolation. Users of this document should verify the applicable data for their specific environment.
Special notices (cont.)

IBM, the IBM logo, ibm.com AIX, AIX (logo), AIX 5L, AIX 6 (logo), AS/400, BladeCenter, Blue Gene, ClusterProven, DB2, ESCON, i5/OS, i5/OS (logo), IBM Business Partner (logo), IntelliStation, LoadLeveler, Lotus, Lotus Notes, Notes, Operating System/400, OS/400, PartnerLink, PartnerWorld, PowerPC, pSeries, Rational, RISC System/6000, RS/6000, THINK, Tivoli, Tivoli (logo), Tivoli Management Environment, WebSphere, xSeries, z/OS, zSeries, Active Memory, Balanced Warehouse, CacheFlow, Cool Blue, IBM Systems Director VmControl, pureScale, TurboCore, Chipshopper, Cloudscape, DB2 Universal Database, DS4000, DS6000, DS8000, EnergyScale, Enterprise Workload Manager, General Parallel File System, GPFS, HACMP, HACMP/6000, HASM, IBM Systems Director Active Energy Manager, iSeries, Micro-Partitioning, POWER, PowerExecutive, PowerVM, PowerVM (logo), PowerHA, Power Architecture, Power Everywhere, Power Family, Power Hypervisor, Power Systems, Power Systems (logo), Power Systems Software, Power Systems Software (logo), POWER2, POWER3, POWER4, POWER4+, POWER5, POWER5+, POWER6, POWER6+, POWER7, System i, System p, System p6, System Storage, System z, TME 10, Workload Partitions Manager and X-Architecture are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries.

A full list of U.S. trademarks owned by IBM may be found at: http://www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

AltVec is a trademark of Freescale Semiconductor, Inc.

AMD Opteron is a trademark of Advanced Micro Devices, Inc.

InfiniBand, InfiniBand Trade Association and the InfiniBand design marks are trademarks and/or service marks of the InfiniBand Trade Association.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce. Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linear Tape-Open, LTO, the LTO Logo, Ulittrum, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries or both.

Microsoft, Windows and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries or both.

NetBench is a registered trademark of Zif Davis Media in the United States, other countries or both.

SPECint, SPECfp, SPECbb, SPECweb, SPECAppServer, SPEComp, SPECviewperf, SPECapc, SPECchpc, SPECjvm, SPECmail, SPECmap and SPECsfs are trademarks of the Standard Performance Evaluation Corp (SPEC).

The Power Architecture and Power.org wordmarks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

TPC-C and TPC-H are trademarks of the Transaction Performance Processing Council (TPPC).

UNIX is a registered trademark of The Open Group in the United States, other countries or both.

Other company, product and service names may be trademarks or service marks of others.

Revised December 2, 2010
Notes on performance estimates

rPerf for AIX

rPerf (Relative Performance) is an estimate of commercial processing performance relative to other IBM UNIX systems. It is derived from an IBM analytical model which uses characteristics from IBM internal workloads, TPC and SPEC benchmarks. The rPerf model is not intended to represent any specific public benchmark results and should not be reasonably used in that way. The model simulates some of the system operations such as CPU, cache and memory. However, the model does not simulate disk or network I/O operations.

• rPerf estimates are calculated based on systems with the latest levels of AIX and other pertinent software at the time of system announcement. Actual performance will vary based on application and configuration specifics. The IBM eServer pSeries 640 is the baseline reference system and has a value of 1.0. Although rPerf may be used to approximate relative IBM UNIX commercial processing performance, actual system performance may vary and is dependent upon many factors including system hardware configuration and software design and configuration. Note that the rPerf methodology used for the POWER6 systems is identical to that used for the POWER5 systems. Variations in incremental system performance may be observed in commercial workloads due to changes in the underlying system architecture.

All performance estimates are provided “AS IS” and no warranties or guarantees are expressed or implied by IBM. Buyers should consult other sources of information, including system benchmarks, and application sizing guides to evaluate the performance of a system they are considering buying. For additional information about rPerf, contact your local IBM office or IBM authorized reseller.

CPW for IBM i

Commercial Processing Workload (CPW) is a relative measure of performance of processors running the IBM i operating system. Performance in customer environments may vary. The value is based on maximum configurations. More performance information is available in the Performance Capabilities Reference at: www.ibm.com/systems/i/solutions/perfmgmt/resource.html

Revised April 2, 2007