Using DB2 with XML and JSON

- Rob Bestgen (bestgen@us.ibm.com)

Session ID: ▪ 26AD

Agenda Key: ▪ Database (DB2)
XML and JSON

What are we talking about?

• XML and JSON are self describing data formats
  • A document can ‘stand alone’

• They are commonly used for transferring data across the internet and between businesses
  • Web Services use them

• They are most often represented in Unicode (universal) encoding

• XML is the more traditional form of data interchange
  • But can be quite bloated

• JSON is a newer form
  • More terse. Partially came about as a response to XML bloat
XML Overview
XML

- **Extensible Markup Language**
- A set of rules for encoding documents electronically
- A textual data format
- Widely used for the representation of arbitrary data structures
- Designed to transport and store data
- [http://www.w3.org/standards/xml/](http://www.w3.org/standards/xml/)
- [http://www.w3schools.com/xml/default.asp](http://www.w3schools.com/xml/default.asp)
What is XML?

XML is "self-describing data"

```xml
<book>
  <authors>
    <author id="47">John Doe</author>
    <author id="58">Peter Pan</author>
  </authors>
  <title>Database systems</title>
  <price>29</price>
  <keywords>
    <keyword>SQL</keyword>
    <keyword>relational</keyword>
  </keywords>
</book>
```

XML: Describes data
HTML: Describes display
What is XML?

XML: Describes data

HTML: Describes display
Attributes v Elements

<book>
  <authors>
    <author id="47">John Doe</author>
    <author id="58">Peter Pan</author>
  </authors>
  <title>Database systems</title>
  <price>29</price>
  <keywords>
    <keyword>SQL</keyword>
    <keyword>relational</keyword>
  </keywords>
</book>

Design Choices
- Elements can be repeated (e.g. “keyword”, “author”) attributes can not
- Elements can be extended (made deeper), e.g. “author”
- Attributes are shorter, can sometimes be stored/processed more efficiently
- When in doubt, use elements rather than attributes
XML Text vs. Tree

```xml
<book>
  <authors>
    <author id="47">John Doe</author>
    <author id="58">Peter Pan</author>
  </authors>
  <title>Database systems</title>
  <price>29</price>
  <keywords>
    <keyword>SQL</keyword>
    <keyword>relational</keyword>
  </keywords>
</book>
```

XML Parsing

Serialization
XML v Relational

| Set oriented | Sequences (ordered!) |
| Structure    | Semi-structured   |
| Strong schema| Schema-variability |
| Strongly typed| Optionally typed |
| Tabular data model | XML data model |
| Flat | Nested, hierarchical |
| "Null" | Not there at all |

**Relational**

<table>
<thead>
<tr>
<th>DEPTID</th>
<th>DEPTNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Sales</td>
</tr>
</tbody>
</table>

**Employee**

<table>
<thead>
<tr>
<th>DEPTID</th>
<th>EMPNO</th>
<th>FIRSTNAME</th>
<th>LASTNAME</th>
<th>PHONE</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>27</td>
<td>MICHAEL</td>
<td>THOMPSON</td>
<td>NULL</td>
<td>41250</td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td>CHRISTINE</td>
<td>SMITH</td>
<td>408-463-4963</td>
<td>52750</td>
</tr>
</tbody>
</table>
Schema Evolution

“Employees are now allowed to have multiple phone numbers…”

Requires:
• Normalization of existing data
• Change of applications

Department

<table>
<thead>
<tr>
<th>DEPTID</th>
<th>DEPTNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Sales</td>
</tr>
</tbody>
</table>

Employee

<table>
<thead>
<tr>
<th>DEPTID</th>
<th>EMPNO</th>
<th>FIRSTNAME</th>
<th>LASTNAME</th>
<th>PHONE</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>27</td>
<td>MICHAEL</td>
<td>THOMPSON</td>
<td>406-463-1234</td>
<td>41250</td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td>CHRISTINE</td>
<td>SMITH</td>
<td>408-463-4963</td>
<td>52750</td>
</tr>
</tbody>
</table>

Phone

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>PHONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>406-463-1234</td>
</tr>
<tr>
<td>10</td>
<td>415-010-1234</td>
</tr>
<tr>
<td>10</td>
<td>408-463-4963</td>
</tr>
</tbody>
</table>
What is an XML Schema?

- Defines structure, content, data types for XML documents
  - Which elements & attributes are allowed?
  - How do they have be nested, how often can they occur?
  - What are the permissible values for each elements/attribute?
  - Which elements are optional? Which are mandatory?
- Consists of 1 or more schema documents
- Defines a namespace to avoid element name conflicts
- Defines an agreed-upon vocabulary of XML tags for a specific application domain, e.g.
  - HL7 schema in healthcare
  - FpML, FIXML, etc. in finance.
  - etc.
XML Schema example

<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
JSON Overview
What is JSON?

JSON is a simplified format for describing information

- **JavaScript Object Notation**
- A textual data format
- Designed to transport and store data
- Uses key/value pairs
- Can be arbitrarily complex: key/values within values (nested)
- Curly braces `{ }` indicate object start and end
**JSON Value Pairs**

- **JSON Keys**
  - Must be a string – a sequence of Unicode characters surrounded by double quotation marks
  - For example: “id”, “name”, “phone”...

```json
{
  "id": 901,
  "name": "John",
  "phone": "555-3762"
}
```
**JSON Value Pairs...**

- **JSON Values**
  - Can be arbitrarily complex: from simple values to key/values within values
  - Colon (:) separates key and value
  - Comma (,) separates pairs
  - Braces ({}) and [] identify lists and 'complex' sets

```json
{
  "id": 901,
  "name": {
    "first": "John",
    "last": "Doe"
  },
  "phones": [
    {
      "type": "home",
      "number": "555-3762"
    },
    {
      "type": "work",
      "number": "555-7242"
    }
  ]
}
```
JSON in action

- Study the JSON document →
  https://www.commerce.gov/sites/commerce.gov/files/costsavings.json

```json
{
  "strategies": [
    {
      "strategyId": 1,
      "strategyTitle": "Adobe Licenses",
      "decisionDate": "08/31/2012",
      "ombInitiative": "Commodity IT",
      "amountType": "Cost-Savings",
      "fy2012": {
        "amount": 0.012,
        "netOrGross": "Gross"
      },
      "fy2013": {
"
DB2 Integration…
XML
XML Integration

- Rich XML Support within DB2 for i (since v7.1)

- Integrated solution
  - XML data type to simplify storage and retrieval of XML documents
    - XML data access protected with rock-solid DB2 security
    - XML covered by database backup and recovery processes
  - Annotated decomposition of XML documents into DB2 columns
  - Generate XML document with SQL-XML publishing functions
  - Can replace (no longer supported) DB2 XML Extender product
XML Data Type

- XML data type
  - Supports XML documents up to 2 GB
  - Type can be used for column, parameter, and host variable values

```
CREATE TABLE Reservations
(  res_ID INTEGER
    GENERATED ALWAYS AS IDENTITY,
  res_Doc XML,
  res_TimeStamp TIMESTAMP
    NOT NULL
    IMPLICITLY HIDDEN
    FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP
)
```
XML Data Type

- XML Schema-based validation supported

```sql
INSERT INTO reservations(res_doc)
VALUES (XMLVALIDATE(
    XMLPARSE(DOCUMENT
        GET_XML_FILE('/dir1/r6.xml'))
    ACCORDING TO XMLSCHEMA
    ID mylib.resschema) )
```

- XML File Reference variables enable simple export of XML documents to IFS

```sql
D MY_XMLFILE S SQLTYPE(XML_CLOB_FILE)
/free
    MY_XMLFILE_NAME= '/out1.xml';
    MY_XMLFILE_NL = 9;
    MY_XMLFILE_FO = SQFCRT;
    exec sql SELECT res_DOC INTO :MY_XMLFILE
        FROM reservations WHERE res_ID=1;
/end-free
```
## Integrated XML Utilities

### Built-in Functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_XML_FILE</td>
<td>Returns the contents of an IFS file or source physical file member as a LOB Locator value</td>
</tr>
<tr>
<td>XMLVALIDATE</td>
<td>Validates XML value against an XML schema</td>
</tr>
<tr>
<td>XMLPARSE</td>
<td>Parses Character/LOB data to produce XML value</td>
</tr>
<tr>
<td>XMLSERIALIZE</td>
<td>Converts XML value into Character/LOB data</td>
</tr>
<tr>
<td>XSLTRANSFORM</td>
<td>Convert XML data into other XML, HTML, and plain text formats using the XSLT processor (<em>requires 5733-XT2</em>)</td>
</tr>
</tbody>
</table>

### System Stored Procedures (SYSPROC library):

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSR_REGISTER</td>
<td>Add an XML Schema document into the DB2 XML Schema Repository (XSR) for Validation / Decomposition</td>
</tr>
<tr>
<td>XSR_ADDSCHEMADOC</td>
<td>Merge an XML Schema within an existing XML Schema</td>
</tr>
<tr>
<td>XSR_COMPLETE</td>
<td>Complete the registration of XML Schema(s) within DB2 XSR</td>
</tr>
<tr>
<td>XSR_REMOVE</td>
<td>Remove a registered XML Schema document</td>
</tr>
<tr>
<td>XDBDECOMPXXML</td>
<td>Decompose an XML document into specified DB2 objects using annotated decomposition</td>
</tr>
</tbody>
</table>
Annotated XML Document Decomposition (Schema based)

1) Map the DB2 and XML document relationships

2) Define mapping in XSD file

3) Register and stored XSD mapping within DB2 XML Schema Repository (XSR)

4) Decompose/Shred the XML document
Decomposition Example

```xml
<shipment>
  <author id="22" >
    <name>Tony Dungy</name>
    <book isbn="1414318014" >
      <title>Quiet Strength</title>
      <publisher>Tyndale</publisher>
    </book>
    <book isbn="1414326815" >
      <title>Uncommon</title>
      <publisher>Tyndale</publisher>
    </book>
  </author>
</shipment>
```

XSD

```xml
<xs:element name="shipment">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="author" type="authorType"
        maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:complexType name="authorType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"
      db2-xdb:rowSet="AUTHORS" db2-xdb:column="NAME" />
    <xs:element name="book" type="bookType"
      maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="id" type="xs:integer">
    <xs:annotation>
      <xs:appinfo>
        <db2-xdb:rowSetMapping>
          <db2-xdb:rowSet>AUTHORS</db2-xdb:rowSet>
          <db2-xdb:column>ID</db2-xdb:column>
        </db2-xdb:rowSetMapping>
        <db2-xdb:rowSetMapping>
          <db2-xdb:rowSet>BOOKS</db2-xdb:rowSet>
          <db2-xdb:column.AUTHID</db2-xdb:column>
        </db2-xdb:rowSetMapping>
      </xs:appinfo>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
<xs:complexType name="bookType">
  <xs:sequence>
    <xs:element name="title" type="xs:string"
      db2-xdb:rowSet="BOOKS" db2-xdb:column="BTITLE" />
    <xs:element name="publisher" type="xs:string" />
  </xs:sequence>
  <xs:attribute name="isbn" type="xs:integer"
    db2-xdb:rowSet="BOOKS" db2-xdb:column="ISBN" />
</xs:complexType>
```
Decomposition Example (Schema based)

- XML Decomposition Steps:
  1) Create XSD file with DB2 to XML mapping

  2) Store and register XSD file within DB2 Schema Repository (XSR)
     
     ```
     CALL SYSPROC.XSR_REGISTER('MYLIB', 'BOOKSCHEM', null, 
     GET_XML_FILE('/dir/authbooks.xsd'), null)
     CALL SYSPROC.XSR_COMPLETE('MYLIB','BOOKSCHEM',null,1)
     ```

  3) Decompose XML Document
     
     ```
     CALL SYSPROC.XDBDECOMPXML
     ('MYLIB','BOOKSCHEM', GET_XML_FILE('/mydir/ship1.xml'),null)
     ```

- **Decomposition Generated Statements:**

  ```
  INSERT INTO authors 
  VALUES(22, 'Tony Dungy')

  INSERT INTO books 
  VALUES(22, 'Quiet Strength', 1414318014), 
  (22, 'Uncommon', 1414326815)
  ```
Alternative - SQL XMLTABLE table function

- XMLTABLE table function extracts relational data from XML, dynamically

- Function uses an XPath expression to define and extract columns from xml elements and attributes

XMLTABLE (<XPath Expr> COLUMNS
<col1> PATH <path ref>,
<col2> PATH <path ref>,
... )
XMLTABLE Example

SQL table `emp` has a column named `doc` that has an XML data type. The table contains two rows (xml documents):

Row 1:
```xml
<dept bldg="101">
  <employee id="901">
    <name>
      <first>John</first> <last>Doe</last>
    </name>
    <office>344</office>
  </employee>
  <employee id="902">
    <name>
      <first>Peter</first> <last>Pan</last>
    </name>
    <office>216</office> <phone>905-416-5004</phone>
  </employee>
</dept>
```

Row 2:
```xml
<dept bldg="114">
  <employee id="903">
    <name>
      <first>Mary</first> <last>Jones</last>
    </name>
    <office>415</office> <phone>905-403-6112</phone>
  </employee>
</dept>
```

Using XMLTABLE to extract certain elements and attributes into relational columns:

```sql
SELECT X.*
FROM emp,
XMLTABLE ('$d/dept/employee' passing emp.doc as "d"
  COLUMNS
  EMPID INTEGER PATH '@id',
  FIRSTNAME VARCHAR(20) PATH 'name/first',
  LASTNAME VARCHAR(25) PATH 'name/last') AS X
```

Result

<table>
<thead>
<tr>
<th>EMPID</th>
<th>FIRSTNAME</th>
<th>LASTNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>901</td>
<td>John</td>
<td>Doe</td>
</tr>
<tr>
<td>902</td>
<td>Peter</td>
<td>Pan</td>
</tr>
<tr>
<td>903</td>
<td>Mary</td>
<td>Jones</td>
</tr>
</tbody>
</table>
## Generating XML - SQL XML Publishing Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMLATTRIBUTES</td>
<td>Returns XML sequence that contains an attribute node for each non-null argument</td>
</tr>
<tr>
<td>XMLCOMMENT</td>
<td>Returns XML value with a single comment node from a string</td>
</tr>
<tr>
<td>XMLCONCAT</td>
<td>Returns XML value that represents a forest of XML elements generated by concatenating a variable number of arguments</td>
</tr>
<tr>
<td>XMLDOCUMENT</td>
<td>Returns XML value with a single document node and zero or more nodes as its children</td>
</tr>
<tr>
<td>XMLELEMENT</td>
<td>Returns XML value that represents an XML element</td>
</tr>
<tr>
<td>XMLFOREST</td>
<td>Returns XML value that represents a forest (sequence) of XML elements that all share a specific pattern</td>
</tr>
<tr>
<td>XMLPI</td>
<td>Returns XML value with a single processing instruction node</td>
</tr>
<tr>
<td>XMLNAMESPACES</td>
<td>Returns the declaration of one or more XML namespaces</td>
</tr>
<tr>
<td>XMLROW</td>
<td>Returns XML value with a single document node containing one top-level element node</td>
</tr>
<tr>
<td>XMLTEXT</td>
<td>Returns XML value with single text node that contains value of argument</td>
</tr>
<tr>
<td>XMLAGG</td>
<td>Returns an XML sequence that contains an item for each non-value in set of XML values</td>
</tr>
<tr>
<td>XMLGROUP</td>
<td>Returns XML value with a single document node containing one top-level element node from a group of rows</td>
</tr>
</tbody>
</table>
SQL XML Publishing Example – XMLELEMENT & XMLATTRIBUTE

- Generate XML values for employees celebrating 25th anniversary:

```
SELECT XMLSERIALIZE(
    XMLELEMENT(NAME "employee", XMLATTRIBUTES(e.empno as "id"),
    XMLELEMENT(NAME "Name", e.firstnme || ' ' || e.lastname),
    XMLELEMENT(NAME "Extension", e.phoneno),
    XMLELEMENT(NAME "DeptNo", d.deptno)) AS CLOB(100) ) as "XMLResult"
FROM employee e, department d
WHERE e.workdept = d.deptno AND
    YEAR(CURRENT DATE) –
    YEAR(hiredate) = 25
```

Output for XMLResult:

```
<employee id="000010">
    <Name>JENNA HAAS</Name>
    <Extension>0420</Extension>
    <DeptNo>A00</DeptNo>
</employee>

<employee id="000050">
    <Name>JOSH GEYER</Name>
    <Extension>1103</Extension>
    <DeptNo>E01</DeptNo>
</employee>
```
**SQL XML Publishing Example - XMLAGG**

- Generate count and XML value for parts with specified type:

```sql
SELECT COUNT(*) AS PartCnt,
XMLSERIALIZE(
  XMLELEMENT(NAME "Parts", XMLATTRIBUTES(parttype AS "type"),
    XMLAGG(
      XMLELEMENT(NAME "pid", partid) ORDER BY partid
    )
  ) AS CLOB(130)) AS PartList
FROM parts WHERE parttype IN ('C01', 'E21')
GROUP BY parttype
```

<table>
<thead>
<tr>
<th>PartCnt</th>
<th>PartList</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td><code>&lt;Parts type=&quot;C01&quot;&gt;&lt;pid&gt;000130&lt;/pid&gt;&lt;pid&gt;200140&lt;/pid&gt; &lt;/Parts&gt;</code></td>
</tr>
<tr>
<td>3</td>
<td><code>&lt;Parts type=&quot;E21&quot;&gt;&lt;pid&gt;000320&lt;/pid&gt;&lt;pid&gt;100330&lt;/pid&gt;&lt;pid&gt;200340&lt;/pid&gt; &lt;/Parts&gt;</code></td>
</tr>
</tbody>
</table>
SQL XML Publishing Example - XMLGROUP

- Generate count and XML value for parts with specified type:

```sql
SELECT COUNT(*) AS PartCnt,
      XMLGROUP( parttype AS "type", partid AS "pid"
                ORDER BY parttype, partid
                OPTION ROW "Parts" ROOT "PartList") AS partlist
FROM parts
WHERE parttype IN ('C01', 'E21')
GROUP BY parttype
```

<table>
<thead>
<tr>
<th>PartCnt</th>
<th>PartList</th>
</tr>
</thead>
</table>
| 2       | `<PartList><Parts><type>C01</type><pid>000130</pid>
         </Parts><Parts><type>C01</type><pid>200140</pid>
         </Parts></PartList>` |
| 3       | `<PartList><Parts><type>E21</type><pid>000320</pid>
         </Parts><Parts><type>E21</type><pid>100330</pid>
         </Parts><Parts><type>E21</type><pid>200340</pid>
         </Parts></PartList>` |
DB2 Integration…

JSON
JSON support in DB2 for i

- **DB2 JSON Store**: June 2015
  - IBM i 7.1, 7.2, & 7.3

- **JSON_TABLE()**: November 2016
  - IBM i 7.2 & 7.3

- **JSON Query support**: March 2017
  - Predicates:
    - IS JSON
    - JSON_EXISTS
  - IBM i 7.2 & 7.3
What is the JSON Store?

**JSON DB2 Store**

- Java applications can use the DB2 JSON APIs to store and retrieve JSON documents
  - Stored as BLOB data in DB2 for i tables under the covers

- **Features**
  - Create JSON collections (single BLOB column table)
  - Insert JSON documents into a JSON collection
  - Retrieve JSON documents
  - Interfaces
    - DB2 JSON command line
    - DB2 JSON Java API
  - Convert JSON documents from BLOB to character data with the SYSTOOLS.BSON2JSON() UDF

JSON_TABLE

- Converts a JSON expression into relational data

**JSON-expressions** can be:
  - Character or Graphic (use FORMAT JSON)
  - Binary (use FORMAT BSON)

**JSON path expression**
  - **lax**
    - Structural problems are tolerated
    - Arrays are automatically unnested
  - **strict**
    - Structural problems result in an error
    - Arrays are not automatically unnested

- Can leverage HTTP Functions

```
SELECT * FROM JSON_TABLE(
  <JSON-expression> ,
  <JSON-path-expression>
  COLUMNS (
    <column-definitions>
  )
  <error-option>
)
```
JSON_TABLE example

- JSON objects are represented using a list of key-value pairs
  - Curly braces {} indicate object start and end
  - Value Pairs are separated by commas
  - Keys and values separated by colons

```
SELECT * FROM
json_table('{"first":"John","last":"Doe"}', '$
columns("first" VARCHAR(40), "last" VARCHAR(40))) x;
```
More JSON – IS JSON query predicate

- **IS or IS NOT JSON**
  - The IS JSON predicate determines whether a value is in a JSON format with the specified attributes
  - VALUE, ARRAY, OBJECT, or SCALAR checking

**Example: Use a trigger to validate JSON**

```sql
CREATE TRIGGER VALIDATE_JSON
BEFORE INSERT OR UPDATE OF JSON_DATA ON MYTABLE
REFERENCING NEW AS N
FOR EACH ROW MODE DB2ROW
IF N.JSON_DATA IS NOT JSON OBJECT THEN
  IF INSERTING THEN
    SIGNAL SQLSTATE '75007'
    ('INSERT failed due to invalid JSON');
  ELSE
    SIGNAL SQLSTATE '75007'
    ('UPDATE failed due to invalid JSON');
  END IF;
END IF;
```
More JSON – JSON_EXISTS query predicate

- **JSON_EXISTS**
  - JSON_EXISTS predicate is true if at least one value can be located
  - COALESCE or IFNULL can be used to identify/process NULL values
  - FALSE ON ERROR, TRUE ON ERROR, UNKNOWN ON ERROR, or ERROR ON ERROR processing options

```
-- Identify the employees who do not have
-- at least one emergency contact.
SELECT empno, lastname FROM employee
WHERE NOT JSON_EXISTS(COALESCE(json_data, ''), 'strict $.emergency' FALSE ON ERROR);
```
Leverage built-in Function example

- **LISTAGG** – Aggregates a set of string values for the group into one string by appending the string-expression
  - Can be used to help create JSON document

```sql
-- Produce a list of names
SELECT workdept,
    '{"names": ['
        interact
            LISTAGG(  
                '{' \concat
                "first":"" \concat
                firstname \concat
                ", "last":"" \concat
                lastname, ""},',
            WITHIN GROUP(ORDER BY lastname)
            \concat
        ]}' AS employees
FROM employee GROUP BY workdept
```

<table>
<thead>
<tr>
<th>WORKDEPT</th>
<th>EMPLOYEES</th>
</tr>
</thead>
<tbody>
<tr>
<td>A00</td>
<td>{&quot;names&quot;: [{&quot;first&quot;:&quot;CHRISTINE&quot;, &quot;last&quot;:&quot;HAAS&quot;},{&quot;first&quot;:&quot;DIAN&quot;, &quot;last&quot;:&quot;HEMMINGER&quot;}],{&quot;names&quot;: [{&quot;first&quot;:&quot;MICHAEL&quot;, &quot;last&quot;:&quot;THOMPSON&quot;}]}]</td>
</tr>
<tr>
<td>B01</td>
<td>{&quot;names&quot;: [{&quot;first&quot;:&quot;SALLY&quot;, &quot;last&quot;:&quot;KWAN&quot;},{&quot;first&quot;:&quot;KIM&quot;, &quot;last&quot;:&quot;NATZ&quot;},{&quot;first&quot;:&quot;HEAT&quot;}, {&quot;names&quot;: [{&quot;first&quot;:&quot;BRUCE&quot;, &quot;last&quot;:&quot;ADAMSON&quot;},{&quot;first&quot;:&quot;DAVID&quot;, &quot;last&quot;:&quot;BROWN&quot;},{&quot;first&quot;:&quot;JAMES&quot;, &quot;last&quot;:&quot;JEFFERSON&quot;},{&quot;first&quot;:&quot;SYBILL&quot;, &quot;last&quot;:&quot;JOHNSON&quot;}],{&quot;names&quot;: [{&quot;first&quot;:&quot;JOHN&quot;, &quot;last&quot;:&quot;GEYER&quot;}]}]</td>
</tr>
<tr>
<td>C01</td>
<td>{&quot;names&quot;: [{&quot;first&quot;:&quot;EILEEN&quot;, &quot;last&quot;:&quot;HENDerson&quot;},{&quot;first&quot;:&quot;JOHN&quot;, &quot;last&quot;:&quot;PARKER&quot;}],{-&quot;names&quot;: [{&quot;first&quot;:&quot;ROY&quot;, &quot;last&quot;:&quot;ALONZO&quot;},{&quot;first&quot;:&quot;JASON&quot;, &quot;last&quot;:&quot;GOUNOT&quot;},{&quot;first&quot;}</td>
</tr>
<tr>
<td>D11</td>
<td>{&quot;names&quot;: [{&quot;first&quot;:&quot;EILEEN&quot;, &quot;last&quot;:&quot;HENDerson&quot;},{&quot;first&quot;:&quot;JOHN&quot;, &quot;last&quot;:&quot;PARKER&quot;}],{-&quot;names&quot;: [{&quot;first&quot;:&quot;ROY&quot;, &quot;last&quot;:&quot;ALONZO&quot;},{&quot;first&quot;:&quot;JASON&quot;, &quot;last&quot;:&quot;GOUNOT&quot;},{&quot;first&quot;}</td>
</tr>
<tr>
<td>D21</td>
<td>{&quot;names&quot;: [{&quot;first&quot;:&quot;EILEEN&quot;, &quot;last&quot;:&quot;HENDerson&quot;},{&quot;first&quot;:&quot;JOHN&quot;, &quot;last&quot;:&quot;PARKER&quot;}],{-&quot;names&quot;: [{&quot;first&quot;:&quot;ROY&quot;, &quot;last&quot;:&quot;ALONZO&quot;},{&quot;first&quot;:&quot;JASON&quot;, &quot;last&quot;:&quot;GOUNOT&quot;},{&quot;first&quot;}</td>
</tr>
<tr>
<td>E01</td>
<td>{&quot;names&quot;: [{&quot;first&quot;:&quot;EILEEN&quot;, &quot;last&quot;:&quot;HENDerson&quot;},{&quot;first&quot;:&quot;JOHN&quot;, &quot;last&quot;:&quot;PARKER&quot;}],{-&quot;names&quot;: [{&quot;first&quot;:&quot;ROY&quot;, &quot;last&quot;:&quot;ALONZO&quot;},{&quot;first&quot;:&quot;JASON&quot;, &quot;last&quot;:&quot;GOUNOT&quot;},{&quot;first&quot;}</td>
</tr>
<tr>
<td>E11</td>
<td>{&quot;names&quot;: [{&quot;first&quot;:&quot;EILEEN&quot;, &quot;last&quot;:&quot;HENDerson&quot;},{&quot;first&quot;:&quot;JOHN&quot;, &quot;last&quot;:&quot;PARKER&quot;}],{-&quot;names&quot;: [{&quot;first&quot;:&quot;ROY&quot;, &quot;last&quot;:&quot;ALONZO&quot;},{&quot;first&quot;:&quot;JASON&quot;, &quot;last&quot;:&quot;GOUNOT&quot;},{&quot;first&quot;}</td>
</tr>
<tr>
<td>E21</td>
<td>{&quot;names&quot;: [{&quot;first&quot;:&quot;EILEEN&quot;, &quot;last&quot;:&quot;HENDerson&quot;},{&quot;first&quot;:&quot;JOHN&quot;, &quot;last&quot;:&quot;PARKER&quot;}],{-&quot;names&quot;: [{&quot;first&quot;:&quot;ROY&quot;, &quot;last&quot;:&quot;ALONZO&quot;},{&quot;first&quot;:&quot;JASON&quot;, &quot;last&quot;:&quot;GOUNOT&quot;},{&quot;first&quot;}</td>
</tr>
</tbody>
</table>
Accessing the Web
New HTTP functions available in SYSTOOLS

- HTTP is the preferred way for communicating on the Web
- RESTful services provide access to information addressable via a URL and accessed using HTTP
  - XML and JSON are part and parcel of the data flows
- DB2 for i has shipped HTTP functions in the SYSTOOLS schema
  - SYSTOOLS contain ‘as is’ tools and examples from DB2

HTTP functions:

- httpGetBlob
- httpGetClob
- httpPutBlob
- httpPutClob
- httpPostBlob
- httpPostClob
- httpDeleteBlob
- httpDeleteClob
- httpBlob
- httpClob
- httpHead
- URLencode
- URLdecode
- Base64encode
- Base64decode
HTTP functions - Query the web example - XML

-- Blog Posts for the last 6 months, order by reader responses
SELECT published, updated, author, title, responses, url, author_bio, html_content, url_atom
FROM
XMLTABLE(
XMLNAMESPACE(DEFAULT 'http://www.w3.org/2005/Atom',
'http://purl.org/syndication/thread/1.0' AS "thr"),
'feed/entry'
PASSING XMLPARSE(DOCUMENT
SYSTOOLS.HTTPGETBLOB(
-- URL --
'http://db2fori.blogspot.com/feeds/posts/default?published-min=' ||
SYSTOOLS.URLENCODE(QGPL.RFC339_DATE_FORMAT(CURRENT_TIMESTAMP - 6 MONTHS), 'UTF-8') ||
'&published-max=' || SYSTOOLS.URLENCODE(QGPL.RFC339_DATE_FORMAT(CURRENT_TIMESTAMP + 1 DAYS), 'UTF-8'),
-- header --
'<!--httpHeader> <header name="Accept" value="application/atom+xml"/> </httpHeader>-->
)
COLUMNS ... (see developerWorks for details)

© 2017 IBM Corporation
JSON in action (revisited)

- Study the JSON document →
  https://www.commerce.gov/sites/commerce.gov/files/costsavings.json

```json
{
  "strategies": [
    {
      "strategyId": 1,
      "strategyTitle": "Adobe Licenses",
      "decisionDate": "08/31/2012",
      "ombInitiative": "Commodity IT",
      "amountType": "Cost-Savings",
      "fy2012": {
        "amount": 0.012,
        "netOrGross": "Gross"
      },
      "fy2013": {
      
    }
  }
}
```
JSON in action

- Build the query...

  -- Which IT strategy provides the greatest ROI?
  -- JSON Feed Source: U.S. Department of Commerce

  ```sql
  select * from JSON_TABLE(
    SYSTOOLS.HTTPGETCLOB('https://www.commerce.gov/sites/commerce.gov/files/costsavings.json', null),
    'lax $.strategies[*]' COLUMNS (
      "strategyTitle" VARCHAR(100),
      "amountType" VARCHAR(100),
      amount_saved_2012 DECIMAL(10,3) PATH 'lax $.fy2012.amount',
      amount_saved_2013 DECIMAL(10,3) PATH 'lax $.fy2013.amount',
      amount_saved_2014 DECIMAL(10,3) PATH 'lax $.fy2014.amount',
      amount_saved_2015 DECIMAL(10,3) PATH 'lax $.fy2015.amount',
      amount_saved_2016 DECIMAL(10,3) PATH 'lax $.fy2016.amount'
    )) x
  order by amount_saved_2016 desc
  limit 10;
  ```
**JSON in action**

- Review the results…

<table>
<thead>
<tr>
<th>strategyTitle</th>
<th>amountType</th>
<th>AMOUNT_SAVED_2012</th>
<th>AMOUNT_SAVED_2013</th>
<th>AMOUNT_SAVED_2014</th>
<th>AMOUNT_SAVED_2015</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud Infrastructure as a service (…</td>
<td>Cost-Avoidance</td>
<td>0.000</td>
<td>0.000</td>
<td>128.000</td>
<td>128.000</td>
</tr>
<tr>
<td>Data Center Consolidation</td>
<td>Both</td>
<td>0.070</td>
<td>45.600</td>
<td>78.700</td>
<td>110.980</td>
</tr>
<tr>
<td>Cloud Email</td>
<td>Cost-Avoidance</td>
<td>0.000</td>
<td>45.000</td>
<td>35.000</td>
<td>15.000</td>
</tr>
<tr>
<td>Desktop and Laptop Management</td>
<td>Cost-Avoidance</td>
<td>0.000</td>
<td>1.200</td>
<td>2.899</td>
<td>10.788</td>
</tr>
<tr>
<td>Server Virtualization</td>
<td>Both</td>
<td>2.120</td>
<td>4.509</td>
<td>5.150</td>
<td>1.350</td>
</tr>
<tr>
<td>Data Center Consolidation by Census</td>
<td>Cost-Avoidance</td>
<td>1.960</td>
<td>1.899</td>
<td>2.000</td>
<td>0.650</td>
</tr>
<tr>
<td>Server Consolidation and Improvements</td>
<td>Both</td>
<td>0.830</td>
<td>0.424</td>
<td>0.315</td>
<td>0.357</td>
</tr>
<tr>
<td>Personal Computer Purchasing Agreement</td>
<td>Cost-Savings</td>
<td>0.059</td>
<td>0.020</td>
<td>0.135</td>
<td>0.100</td>
</tr>
<tr>
<td>Server Virtualization Optimization</td>
<td>Cost-Avoidance</td>
<td>0.000</td>
<td>0.315</td>
<td>0.555</td>
<td>0.089</td>
</tr>
<tr>
<td>Adobe Licenses</td>
<td>Cost-Savings</td>
<td>0.012</td>
<td>0.012</td>
<td>0.010</td>
<td>0.010</td>
</tr>
</tbody>
</table>

**Consume JSON Feeds:**
- Government (local, state, national, …)
- Social Media (Twitter, Facebook, …)
- Intranet
- Etc…
Contrasting
## Compare and contrast current DB2 for i support

<table>
<thead>
<tr>
<th>Feature</th>
<th>XML</th>
<th>JSON</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Storing</strong></td>
<td>XML data type</td>
<td>LOB, JSON store</td>
</tr>
<tr>
<td><strong>Parsing (shred)/ Annotation</strong></td>
<td>XMLTABLE UDTF, Schema decomposition</td>
<td>JSON_TABLE UDTF</td>
</tr>
<tr>
<td><strong>Schema support</strong></td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Publishing functions</strong></td>
<td>XML specific functions</td>
<td>Leverage general purpose functions</td>
</tr>
<tr>
<td><strong>Validation</strong></td>
<td>Datatype, Schema validation, UDTF</td>
<td>UDTF, IS_JSON, JSON_EXISTS</td>
</tr>
<tr>
<td><strong>Search (doc itself)</strong></td>
<td>LIKE, CONTAINS(Omnifind)</td>
<td>LIKE</td>
</tr>
<tr>
<td><strong>Status in DB2</strong></td>
<td>Mature</td>
<td>Active development</td>
</tr>
<tr>
<td><strong>First Available</strong></td>
<td>i7.1</td>
<td>i7.2 - as ptfs (i7.1 as json store)</td>
</tr>
</tbody>
</table>
XML/JSON v Relational Database

Factors influencing data model choice

- When you need maximum flexibility ✓ XML/JSON
- When you need maximum performance for data retrieval ✓ RDB
- When data is processed later as relational data ✓ RDB+XML/JSON
- When data components have meaning outside a hierarchy ✓ RDB+XML/JSON
- When data attributes vary, or apply to only a small subset of the data ✓ XML/JSON
- When referential integrity is required ✓ RDB
- When the data needs to be updated often ✓ RDB+XML/JSON
More Information
XML on DB2 for i - Whitepaper

Use this 100+ page resource as an aid to deploying XML on DB2 for i

“Replacing DB2 XML Extender with integrated DB2 for i XML Capabilities”

Abstract
This paper explores using the integrated XML features in DB2 for i 7.1 as a replacement for the XML-related functions and data types provided by the priced DB2 XML Extender option, which is part of DB2 Extenders Version 9.5 for i licensed product (5761DE1 and 5770 DE1).

The paper will review the differences between the DB2 XML Extender and the integrated XML support. A fictional company’s application is utilized as a mechanism to compare the integrated XML functionality with the capabilities provided by XML Extender.

www.ibm.com/systems/i/db2/awp.html
“The powerful JSON_TABLE function”

Who is the DB2 for i Center of Excellence? A highly skilled and experienced IBM Team delivering:

- Product Offerings Developed through Engagements
- Briefings, consulting and guidance on demand
- IBM i for Business Intelligence Installation Services
- DB2 Web Query Getting Started Services
- Query/400 Modernization Services
- DB2 for i Modernization Workshop(s)
- DB2 for i SQL Performance Workshop
- DB2 for i SQL Performance Health Check
- DB2 for i Very Large Database (VLDB) Assessment
- DB2 for i remote database administration and engineer services

For more information, contact Mike Cain (cain@us.ibm.com) or Doug Mack (mackd@us.ibm.com)
Thank You!
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.

Revised September 26, 2006