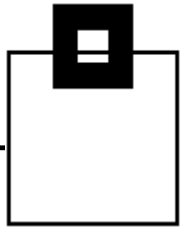
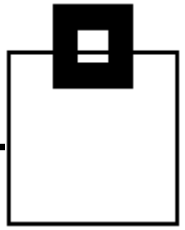


Episode 9

The Return of the Hierarchical Empire

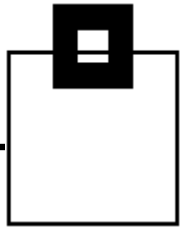
Ulf Heinrich
SEGUS, Inc
u.heinrich@segus.com

Agenda



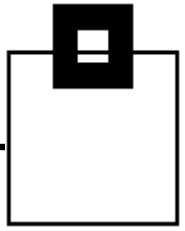
- A few basics about XML in general
- XML integration in DB2 for z/OS 9 and above
- Index design for XML
- DB2 9 APARs and DB2 10/11 enhancements
- Hints and tips

<XML>Basics</XML>



- XML: Extensible Markup Language
- simplified subset of Standard Generalized Markup Language (SGML)
- Simultaneously human and machine-readable format
- Supports Unicode, allowing almost any information in any written human language to be communicated;
- Self-documenting format
- Describes structure and field names as well as specific values
- Platform-independent

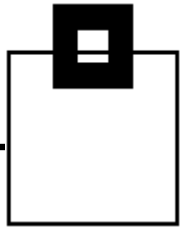
<XML>Basics</XML>



- XML is a good choice for
  - highly changed data models
  - data exchange

- Some XML industry standards:
- SEPA/UNIFI, MIMSO, XBRL, DJXDM, HR-XML, HL7, ARTS, HIPAA, NewsML, XForms, FpML, FIXML, ACORD
- UNIFI ISO standard, (ISO 20022) is the standard for financial industry

<XML>Basics</XML>

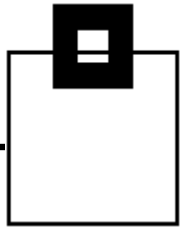


- Elements
- Attributes
- Nesting
- Content / Data

- Document Type Definition (DTD)
- XML schema language

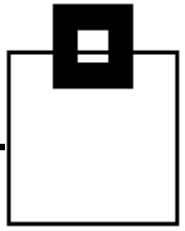
- http://www.w3.org/XML/
- http://www.w3.org/TR/REC-xml/
- Check also http://www.w3schools.com

<XML>Basics</XML>



- XML describes structure and field names as well as specific values
- An XML document can be represented as a tree of nodes
- Optionally a structure can be defined by DTDs or XML schemas
  - This requires schema registration, schema validation, and annotated schema-based decomposition

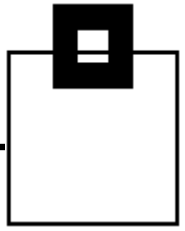
<XML>Basics</XML>



- All elements must have a closing tag
- XML tags are case sensitive
- Elements have to be properly nested
- Attribute values must be in quotes
- Entity references must be used for key characters:

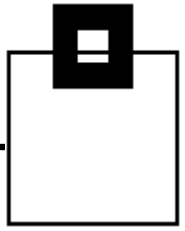
<	→	<
>	→	>
&	→	&
'	→	'
"	→	"

XML example



```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css"
  href="format.css"?>
<salad>
  <name>Tomato Salad</name>
  <dressing oil="70" vinegar="30" salt="3g"
  pepper="2g">
    <name>Vinegar-Oil</name>
  </dressing>
  <greens quantity="97">Tomatoes</greens>
  <greens quantity="3">Onions</greens>
</salad>
```

XML example



XML allows flexible data, but a XML document has to be well formed

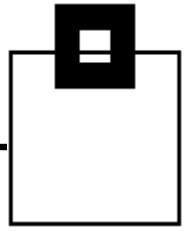
- 1st line: XML declaration
  - version
  - Encoding scheme used in the document

```
<?xml version="1.0" encoding="UTF-8"?>
```

- At least a root element

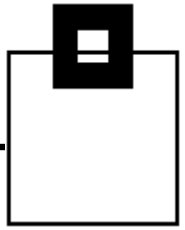
```
<salad>
</salad>
```

<XML>Basics</XML>



- There are six types of nodes in XML:
  - Document or root node
  - Element node
  - Text node
  - Attribute node
  - Processing instruction node
  - Comment node

XML example



Root or Document Node

```
<salad>
   <name>Tomato Salad</name>
   <dressing oil="70" vinegar="30" salt="3g"
    pepper="2g">
      <name>Vinegar-Oil</name>
   </dressing>
   <greens quantity="97">Tomatoes</greens>
   <greens quantity="3">Onions</greens>
</salad>
```

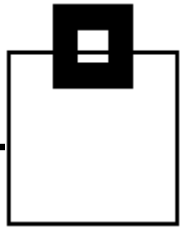
Attribute

Element

Content / Data

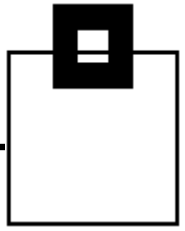
(DTD can be used to define the legal building blocks of an XML document.)

<XML>Basics</XML>



- Element nodes are designed to contain any combination of
  - text
  - attributes
  - other elements
- Letters, numbers and other characters are allowed, but no:
  - Leading numbers, leading punctuation, or leading characters “xml”
  - Spaces are also not allowed

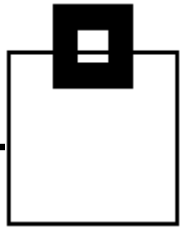
<XML>Basics</XML>



- Attribute nodes are designed to contain metadata for elements that has to be in quotes
- If the value contains quotes, enclose it
  - Use character entities (")
- Attributes can not contain other elements
- Attributes can not contain multiple values

```
oil="70" vinegar="30" salt="3g"
```

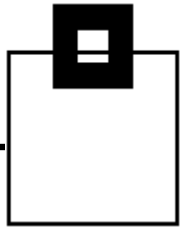
<XML>Basics</XML>



- Comment nodes are designed for additional information, not processed by the XML parser

```
<!-- This is a comment -->
```

XML and the web



XML Doc

```
<?xml version="1.0" encoding="UTF-8"?>
<salad>
  <name>Tomato Salad</name>
  <dressing oil="70" vinegar="30"
    salt="3g" pepper="2g">
    <dressing>Vinegar-Oil</dressi
  </dressing>
  <greens quantity="97">Tomatoes<
  <greens quantity="3">Onions</gr
<name>Fruits Salad</name>
  <dressing oil="80%" water="20%"
    salt="1g" sugar="2ts">
    <dressing>Oil-Water</dressing
    <comment>Walnut Oil</comment>
  </dressing>
  <fruits quantity="20%">Apples</
  <fruits quantity="20%">Oranges<
  <fruits quantity="20%">Bananas<
  <fruits quantity="20%">Grapes</
  <fruits quantity="5%">Walnuts</
  <fruits quantity="15%">Chicory<
</salad>
```

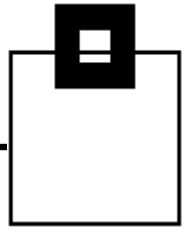
Web- Browser

C:\Users\UHeinrich

C:\Users\UHeinrich.G...

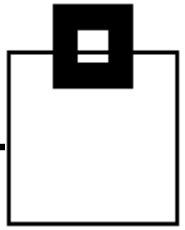
```
<?xml version="1.0" encoding="UTF-8"?>
- <salad>
    <name>Tomato Salad</name>
  - <dressing pepper="2g" salt="3g" vinegar="30" oil="70">
      <dressing>Vinegar-Oil</dressing>
    </dressing>
    <greens quantity="97">Tomatoes</greens>
    <greens quantity="3">Onions</greens>
    <name>Fruits Salad</name>
  - <dressing salt="1g" oil="80%" sugar="2ts" water="20%">
      <dressing>Oil-Water</dressing>
      <comment>Walnut Oil</comment>
    </dressing>
    <fruits quantity="20%">Apples</fruits>
    <fruits quantity="20%">Oranges</fruits>
    <fruits quantity="20%">Bananas</fruits>
    <fruits quantity="20%">Grapes</fruits>
    <fruits quantity="5%">Walnuts</fruits>
    <fruits quantity="15%">Chicory</fruits>
</salad>
```

<XML>Basics</XML>



- If the structure of a XML document should be more than just *well formed*, a Document Type Definnition (DTD), or a schema must be defined
  ```
  <!DOCTYPE note SYSTEM "MyDTD.dtd">
  ```
- The referenced file describes all legal elements of the XML
  ```
  <!DOCTYPE note
  [
  <!ELEMENT salad (name,dressing,greens)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT dressing (#PCDATA)>
  <!ELEMENT greens (#PCDATA)>
  ]>
  ```

<XML>Basics</XML>



- XML schema:

```
<xs:element name="salad">

< xs:complexType>
<xs:sequence>
<xs:element salad="name" type="xs:string"/>
<xs:element salad="dressing"
type="xs:string"/>
<xs:element salad="greens"
type="xs:string"/>
</xs:sequence>
< /xs:complexType>

< /xs:element>
```

XML documents and stylesheets

XML Doc

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css"
    href="css.css"?>
<salad>
  <name>Tomato Salad</name>
  <dressing oil="70" vinegar="30" salt="3g"
    pepper="2g">
    <dressing>Vinegar-Oil</dressing>
  </dressing>
  <greens quantity="97">Tomatoes</greens>
  <greens quantity="3">Onions</greens>
<name>Fruits Salad</name>
  <dressing oil="80%" water="20%" salt="1g"
    sugar="2ts">
    <dressing>Oil-Water</dressing>
    <comment>Walnut Oil</comment>
  </dressing>
  <fruits quantity="20%">Apples</fruits>
  <fruits quantity="20%">Oranges</fruits>
  <fruits quantity="20%">Bananas</fruits>
  <fruits quantity="20%">Grapes</fruits>
  <fruits quantity="5%">Walnuts</fruits>
  <fruits quantity="15%">Chicory</fruits>
</salad>
```

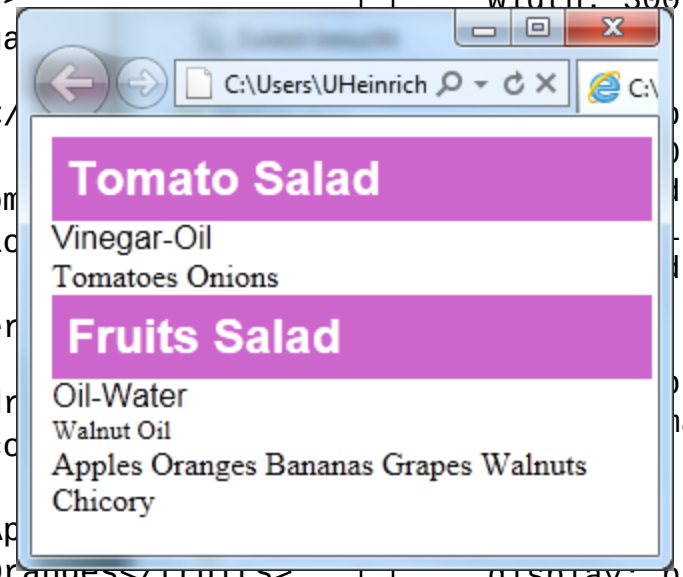
Stylesheet

```
@media screen {
  salad {
    display: block;
    margin: 10px;
    width: 300px;
  }
  name {
    display: block;
    padding: 0.3em;
    font: bold x-large sans-serif;
    color: white;
    background-color: #C6C;
  }
  dressing {
    display: block;
    font: normal medium sans-
serif;
  }
comment {
    display: block;
    font: small medium sans-serif;
  }
  title {
    font-style: italic;
  }
}
```

XML documents and stylesheets

XML Doc

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css"
    href="css.css"?>
<salad>
  <name>Tomato Salad</name>
  <dressing oil="70" vinega
    pepper="2g">
    <dressing>Vinegar-Oil</
  </dressing>
  <greens quantity="97">Tom
  <greens quantity="3">Onic
<name>Fruits Salad</name>
  <dressing oil="80%" water
    sugar="2ts">
    <dressing>Oil-Water</dr
    <comment>Walnut Oil</co
  </dressing>
  <fruits quantity="20%">Ap
  <fruits quantity="20%">Or
  <fruits quantity="20%">Bananas</fruits>
  <fruits quantity="20%">Grapes</fruits>
  <fruits quantity="5%">Walnuts</fruits>
  <fruits quantity="15%">Chicory</fruits>
</salad>
```



Stylesheet

```
@media screen {
  salad {
    display: block;
    margin: 10px;
    width: 300px;
  }
  name {
    display: block;
    font: small medium sans-serif;
  }
  title {
    font-style: italic;
  }
}
```

XML documents and stylesheets

- Besides simple css, eXtensible Stylesheet Language Transformations (XSLT) allows more sophisticated
- XSLT is the recommended stylesheet language

```
<?xml-stylesheet type="text/xsl" href="style.xsl" ?>
```

- XSLT allows to transform XML to HTML

- The XML Extender provides a user defined function
  - XSLTransformToClob()
  - XSLTransformToFile()

DB2 9 and XML

XML Support introduced in DB2 9:

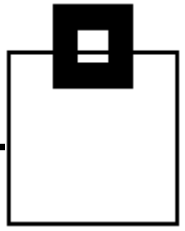
Hierarchical data model: XDM (XQueryData Model)
XML query languages: XQuery, XPath, (XSLT)

pureXML in DB2
„pure“ XML storing
supports hierarchical storing

Support of: XPath (subset of XQuery), SQL/XML

z/OS XMLSS: z/OS 1.8 or z/OS 1.7 with APAR OA16303
XML schemas require IBM 31-bit SDK for z/OS, Java 2 Technology Edition, V5 (5655-N98), SDK5 and Java stored procedure setup.

DB2 9 and XML

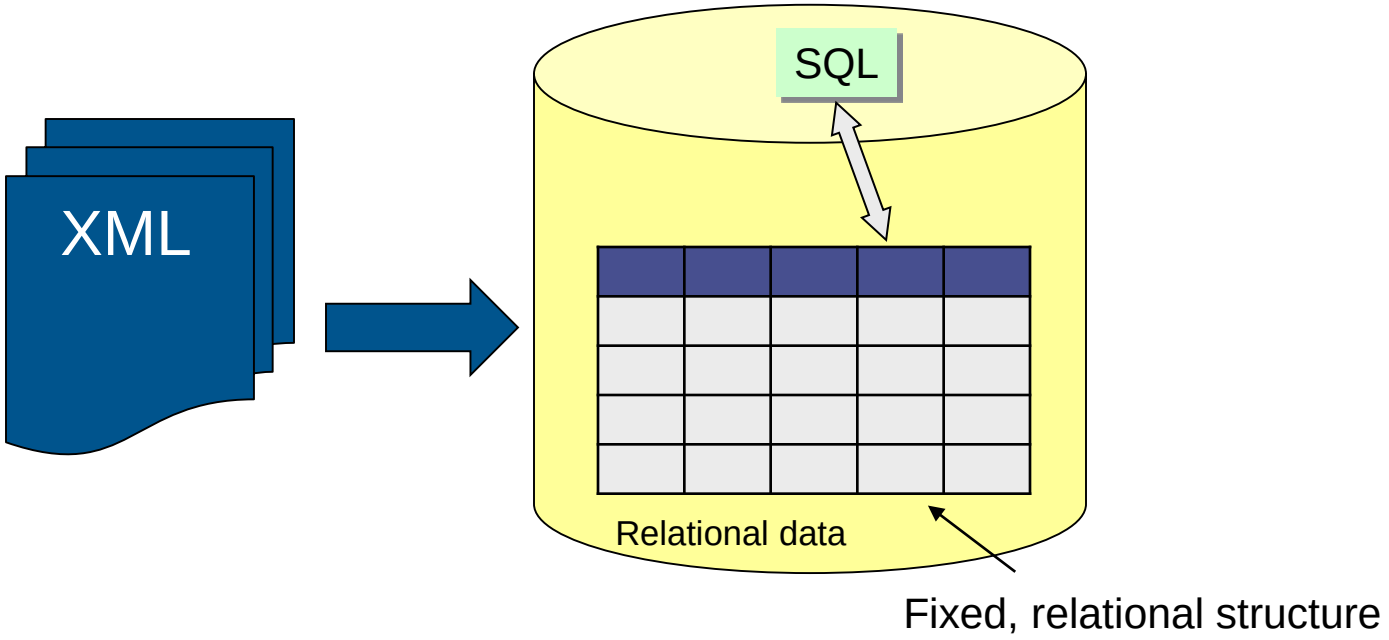


3 possibilities to store XML data in DB2:

- XML as text
- XML shredding
- Native as parstree (parsing at insert time)

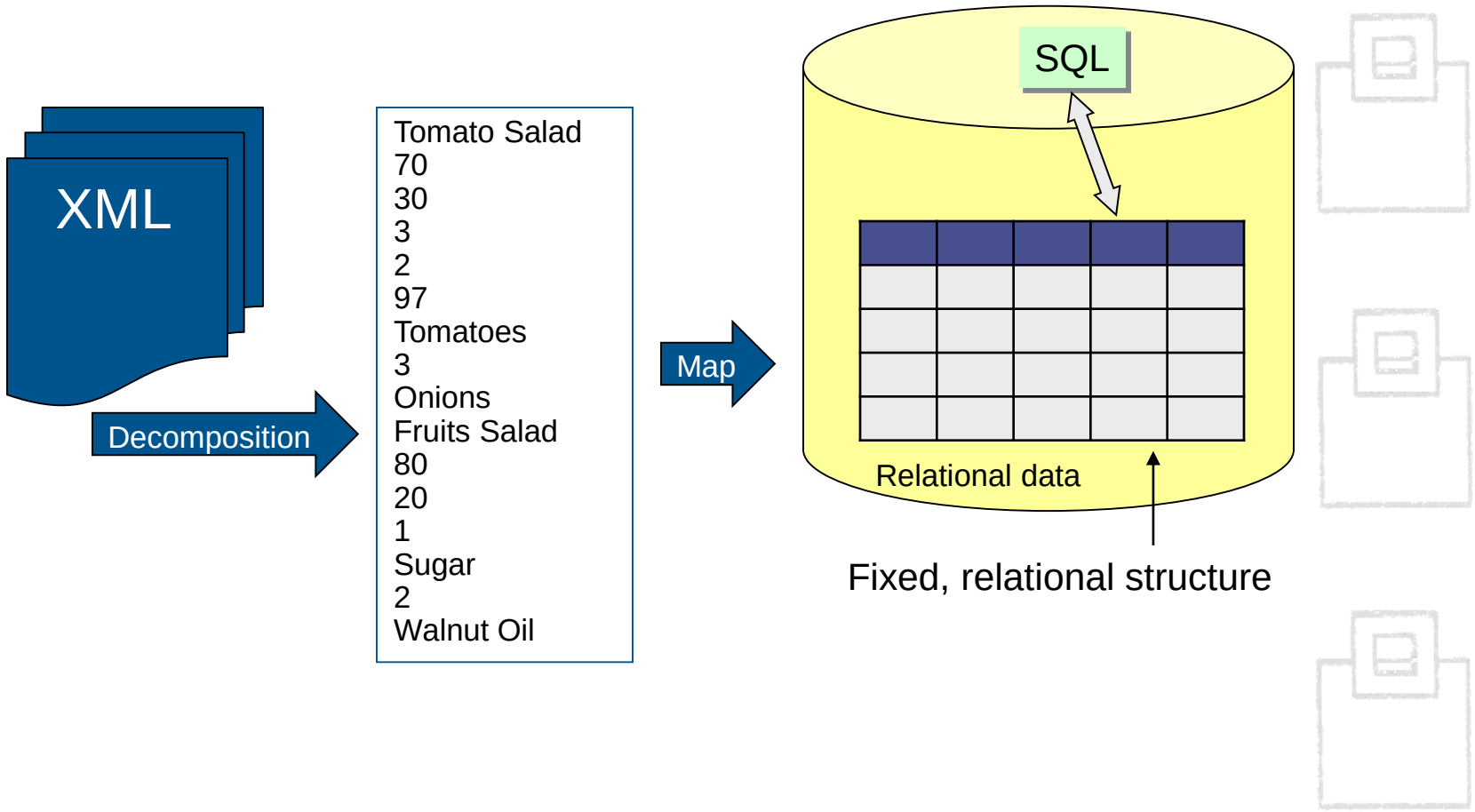
DB2 9 and XML

XML as text in a varchar, or clobs

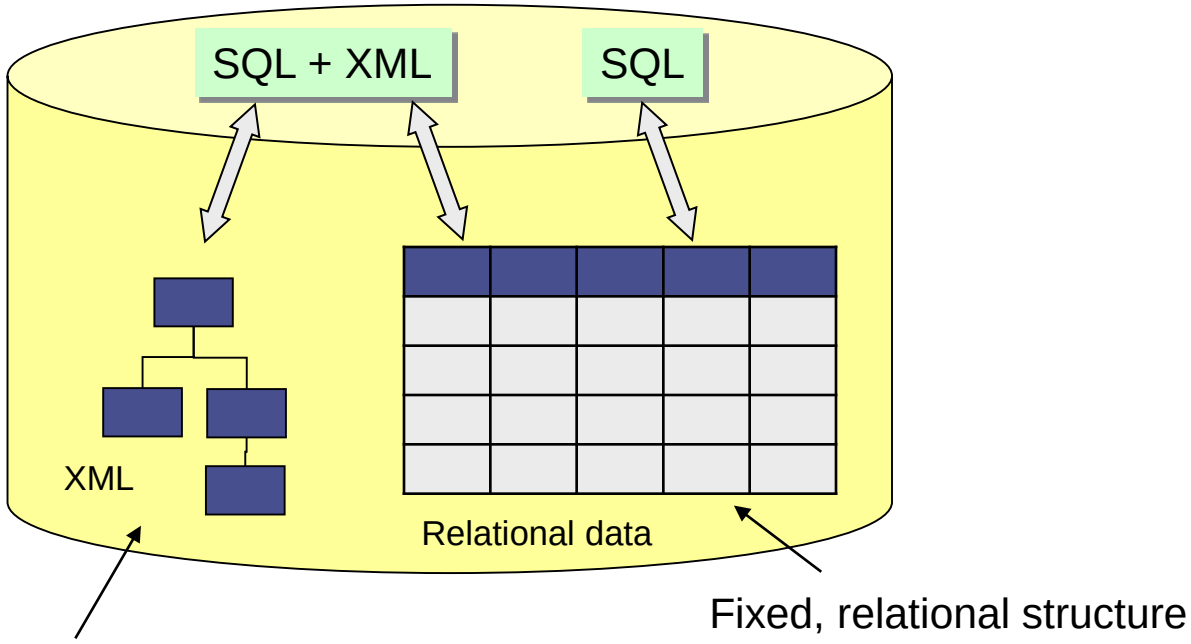


DB2 9 and XML

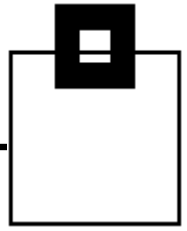
XML shredding



XML in DB2 architecture (z/OS):



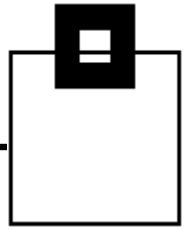
DB2 9 and XML



- Create tables with XML columns
- Insert and optionally validated against schemas
- Create indexes on XML data
- Efficiently search XML data
- Extract XML data
- Decompose XML data into relational data, or create relational view
- Construct XML documents from relational and XML data
- Handle XML with the DB2 utilities

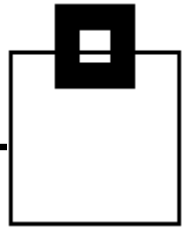
→ Manage XML the same way as relational data

DB2 9 and XML



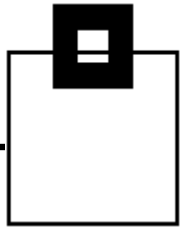
- Complete set of SQL/XML constructor functions
- XML type, native storage of XQuery Data Model
- XMLPARSE, XMLSERIALIZE, XMLCAST
- Other SQL/XML functions with XPath
  - XMLEXISTS, XMLQUERY, XMLTABLE
- XML Schema repository, Validation UDF, and decomposition
- DRDA (distributed support) and application interfaces

DB2 9 and XML



- Update: whole document replacement
- Largest document size: ~2GB
- Type annotation not kept in storage after validation
- XML indexes:
  - Numeric and string types
  - String index key length: 1000 bytes
  - Indexed nodes cannot be longer than 2000 bytes in length
- Triggers: XML columns cannot be transition vars
- Stored procedures: no XML type arguments
- Initial sweet spot: a large number of small documents

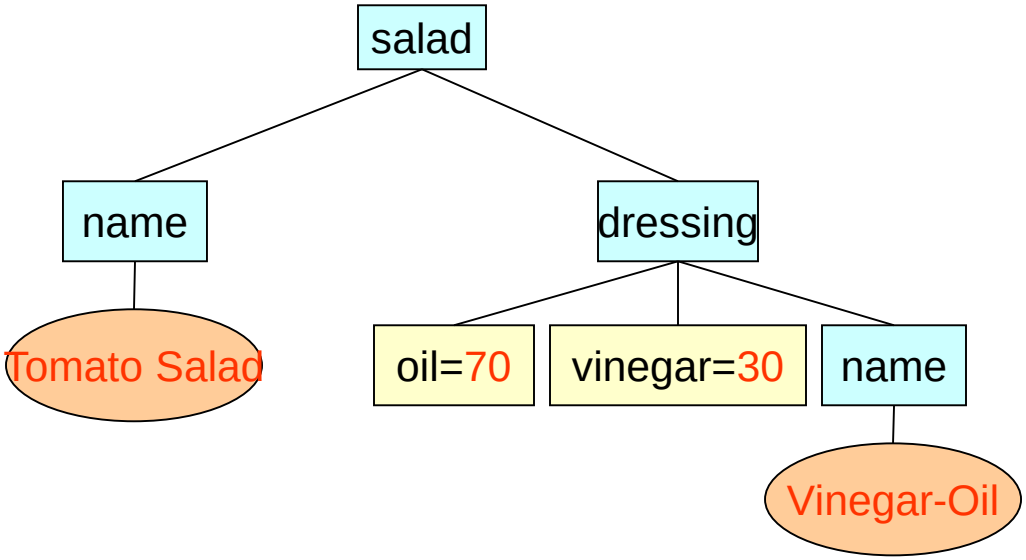
DB2 9 and XML



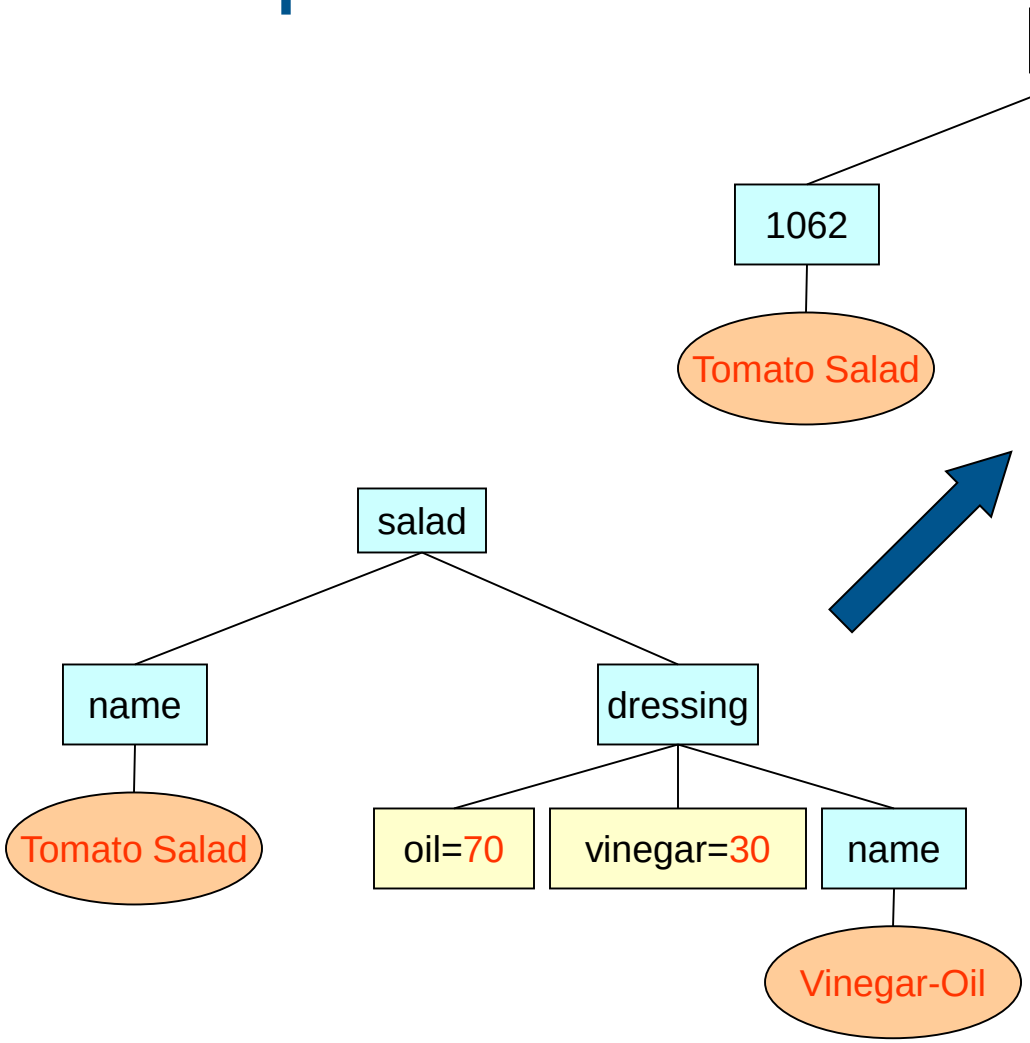
- Original document size can be reduced to 0.3 by compression, otherwise it's about 1.5
- Schema validation exploits XML validating parser (XLXP-C)
- Partitioned table space and data sharing support
- XML DRDA workload is zIIP eligible
- XML parsing by zAAP and zIIP

XML parstree

```
<salad>
  <name>Tomato Salad</name>
  <dressing oil="70" vinegar="30">
    <name>Vinegar-Oil</name>
  </dressing>
</salad>
```



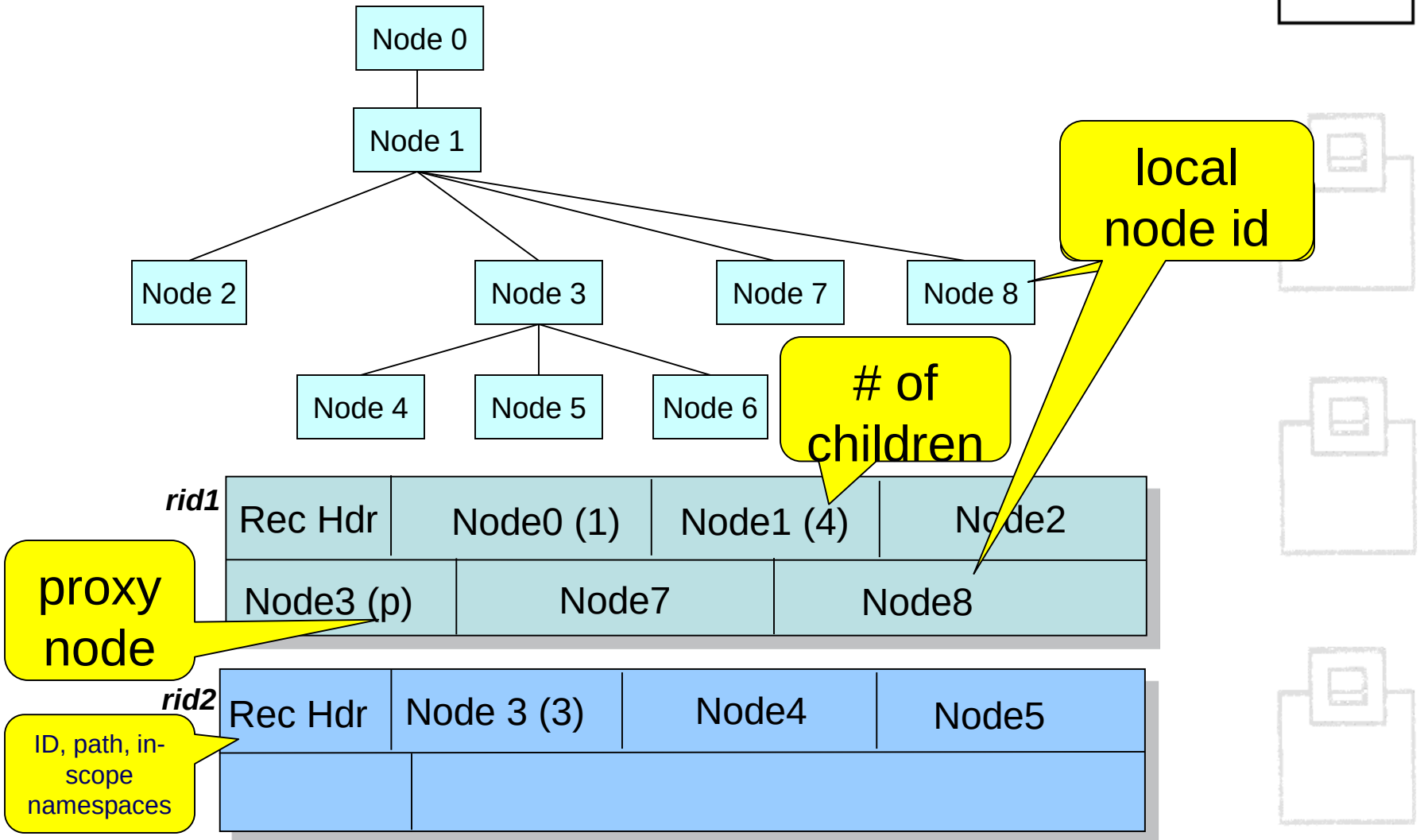
XML parstree and nodes



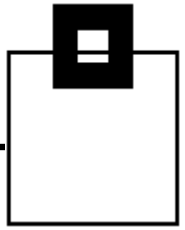
SYSIBM.SYSXMLSTRINGS

STRINGID	STRING
1061	salad
1062	name
1063	dressing
1064	oil
1065	vinegar

XML parstree and nodes



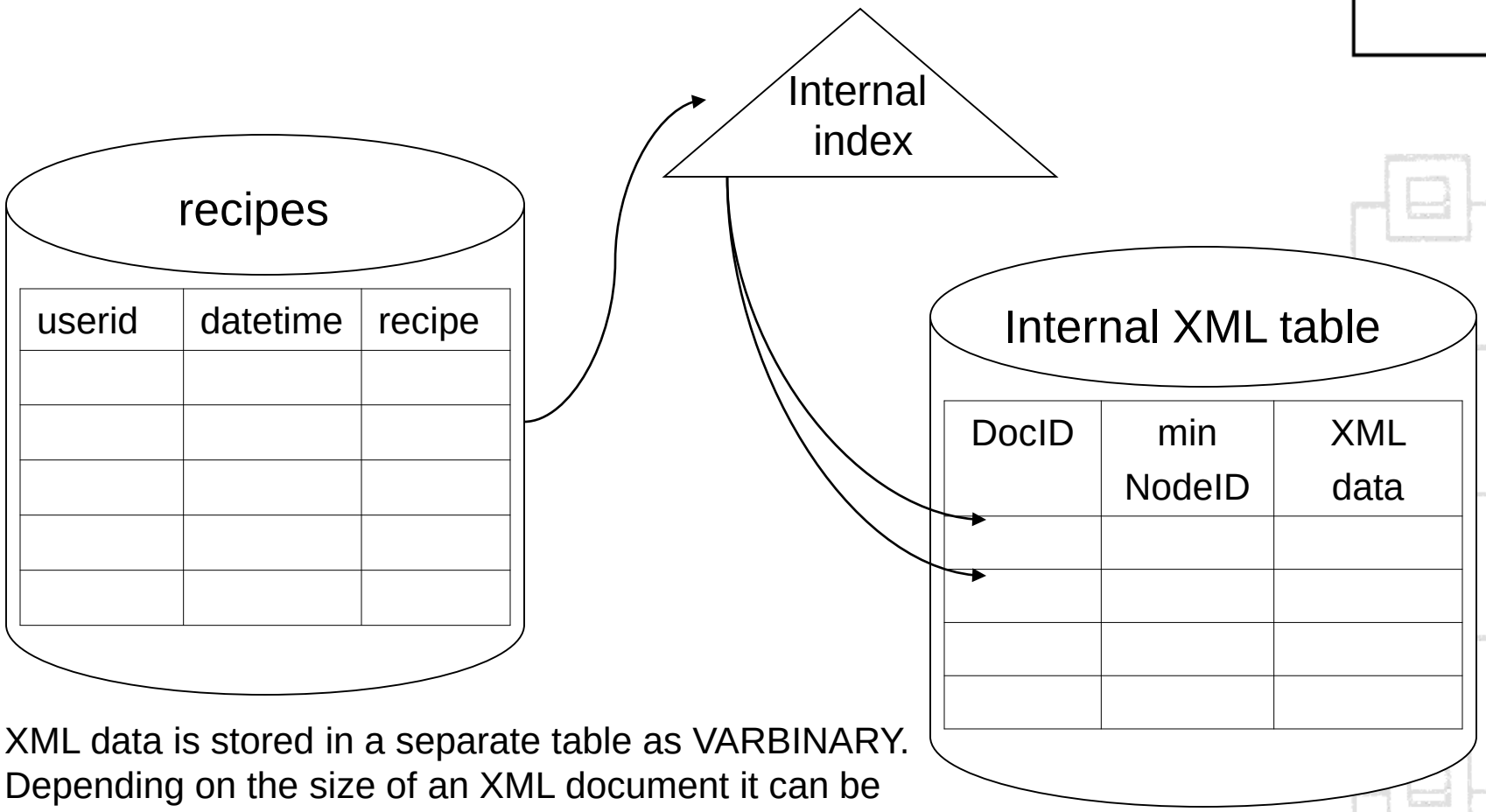
DB2 9 and XML



Column Type XML:

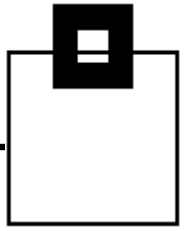
```
CREATE TABLE recipes (
    userid              VARCHAR(30),
    datetime            TIMESTAMP,
    recipe              XML
);
```

XML Storage in DB2



- XML data is stored in a separate table as VARBINARY.
- Depending on the size of an XML document it can be split up into more than one row.
- The XML column in the original table contains a link to the XML table.
- An internal, implicit DocID index is created to access the XML data.

XML Storage in DB2

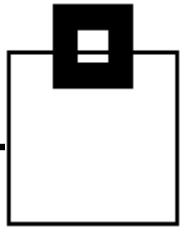


DB2 implicitly creates

- Base table
  - regular columns
  - a hidden DOCID column (BIGINT identity column, NPI if partitioned)
  - A null bit
  - An Invalid bit
  - A few reserved bytes

- internal DOCID index on the DOCID column (NPSI if base is partitioned)

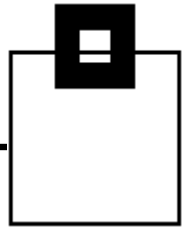
XML Storage in DB2



DB2 implicitly creates

- XML table
  - XML column (16k TS UTS UTF-8 – no CCSID since it's binary data, no limit key)
    - XML column is 2GB max (no size specification possible)
- if base is non-range partitioned – equal partitioning
- If base is range partitioned the partition depends on the # of docs that fit into a XML partition
- LOG, LOCKMAX, DSSIZE, COMPRESS inherited from base
- No check constraints
- No identity columns

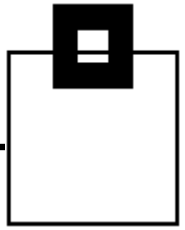
XML Storage in DB2



- XML table and base table are located in separate tablespaces
- RUNSTATS on base table does not collect statistics for XML tablespace nor for XML indexes
- Separate RUNSTATS necessary for XML tablespace

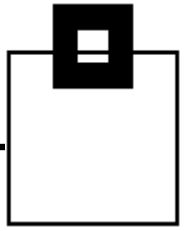
- ZPARMS for storage allocation:
  - User: XMLVALA = 200MB like LOBVALA
  - System: XMLVALS = 10GB like LOBVALS
  - TBSBPXML = BP16K0

XML objects in the DB2 catalog



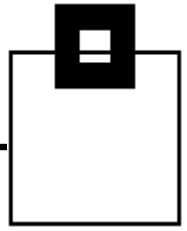
- SYSIBM.SYSXMLRELS
  - Contains one row for each XML table that is created for an XML column.
- SYSIBM.SYSXMLSTRINGS
  - Each row contains a single string and its unique ID that are used to condense XML data. The string can be an element name, attribute name, name space prefix, or a namespace URI.

XML objects in the DB2 catalog



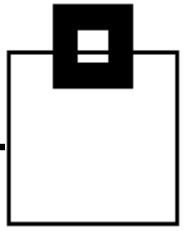
- SYSIBM.SYSKEYTARGETS
  - The SYSIBM.SYSKEYTARGETS table contains one row for each key-target that is participating in an extended index definition.
  - Column: DERIVED_FROM VARCHAR(4000)
    - This is the XML pattern that is used to generate the key-target value.

How to get it in - XMLPARSE



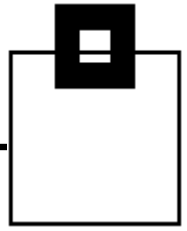
```
INSERT INTO recipes VALUES
('User1'
, current timestamp
, XMLPARSE (DOCUMENT '
  <salad>
  <name>Tomato Salad</name>
  <dressing oil="70%" vinegar="30%" salt="3g" pepper="2g">
  <name>Vinegar-Oil</name>
  </dressing>
  <greens quantity="97%">Tomatoes</greens>
  <greens quantity="3%">Onions</greens>
  </salad>
') )
```

How to get it in - XMLPARSE



- Use UTF-8 if possible, and use binary data type, to avoid encoding conversion overhead
- DB2 LOAD supports XML data up to 32KB - larger documents need “load from file references” (2GB Limit because of LOB MANAGER usage)
- DSN_XMLVALIDATE function can be used to define an XML schema. DB2 provides a XSR XML schema repository (in the catalog) and stored procedures to register schemas (stored binary as a BLOB in the XSR)

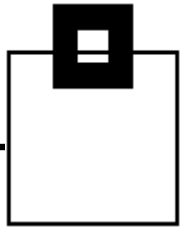
How to get it in - XMLPARSE



XSR (XML Schema Repository) Setup:

- A set of DB2 tables, stored procedures, and user-defined function
  - XSR_REGISTER, XSR_ADDSCHEMADOC, XSR_COMPLETE, XSR_REMOVE
  - XDBDECOMPXML
  - DSN_XMLVALIDATE
- Setup required:
  - Bind package SYSXSR via jobs DSNTIJSG and DSNTIJNX
  - Java 2 SDK V1.5, JCC for DB2 9 for z/OS, and bind the JCC packages to SYSXSR collection ID
  - WLM for stored procedures and functions (default WLMENV3)
  - WLM for Java stored procedure (needed for XSR_COMPLETE)

Accessing XML data with SQL

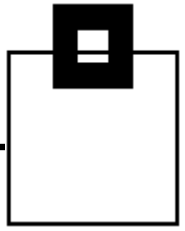


- Use simple SELECT, or use predicates to filter the data.

```
SELECT userid, recipe
  FROM recipe
  WHERE userid = ‘User1‘
```

- With standard SQL, it is possible to receive whole XML documents, but not to filter on elements or extract pieces inside the XML document.

XPath



- Filtering on XML content can be done using XPath expressions imbedded in SQL

- Each node in an XML document can be represented as a path.

- XPATH allows
  - functions (like sum and count: fn:count(/salad/dressing))
  - Cast functions (like integer, double, decimal, string, boolean:

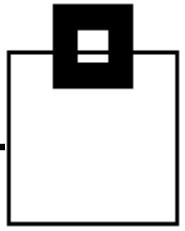
    xs:integer(/salad/dressing/@oil))

XPath

```
<salad>
  <name>Tomato Salad</name>
  <dressing oil="70" vinegar="30">
    <name>Vinegar-Oil</name>
  </dressing>
</salad>
```

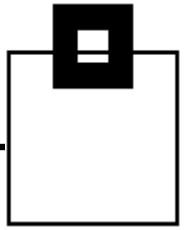
/
/salad
/salad/name
/salad/dressing
/salad/dressing/@oil
/salad/dressing/@vinegar
/salad/dressing/name

Filter on XML using XMLEXISTS



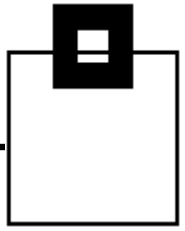
- XMLQUERY and XMLEXISTS allow to include XPath in SQL to select and filter on parts of an XML document
  - XMLQUERY is used in the SELECT clause
  - XMLEXISTS is used in the WHERE clause
- XML data type cannot be compared with any other data type using the regular comparison operators, but IS NULL or IS NOT NULL in XMLEXISTS
- XML data cannot be sorted:
- no distinct, no group by, no order by
- no UNION, but UNION ALL is fine for XML.

Filter on XML using XMLEXISTS



```
SELECT userid, datetime
   FROM recipes
WHERE XMLEXISTS('$c/salad[name="Tomato Salad"]'
            PASSING recipe as "c");
```

Path expressions and wildcards

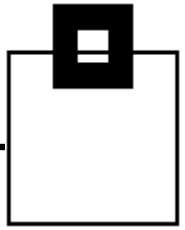


@	specifies an attribute
text()	specifies a node under an element
*	matches any tag name
//	is the “descendent-or-self“ wildcard
.	specifies the current context
..	specifies the parent context

Predicates are enclosed in square brackets [ ]

[*n*] is a shortcut to [fn:position ()*n*]

Path expressions and wildcards



REMINDER:

XML and XPath are case-sensitive

SQLCODE = -16002, ERROR: AN XQUERY EXPRESSION HAS AN UNEXPECTED TOKEN

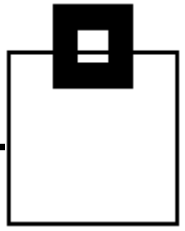
…

CCSID setting has to be consistent for application encoding scheme and 3270 terminal

SQLCODE = -16002, ERROR: AN XQUERY EXPRESSION HAS AN UNEXPECTED TOKEN FOLLOWING

…

XMLQUERY on XML



XMLQUERY returns the selected parts of an XML document in XML format including header and tags.

```
SELECT XMLQUERY ('$c/salad/greens'
            PASSING recipe as "c")
FROM recipes
WHERE XMLEXISTS('$c/salad[name="Tomato Salad"]'
            PASSING recipe as "c");
```

Result:

```
<?xml version="1.0" encoding="IBM01141"?>
    <greens quantity="97">Tomatoes</greens>
    <greens quantity="3">Onions</greens>
```

XMLSERIALIZE and text()

The function XMLSERIALIZE eliminates the xml header, text() returns the data content without the tags.

Truncation of XML data leads to an error

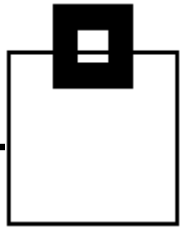
```
SELECT XMLSERIALIZE( XMLQUERY ('$c/salad/greens/text()'
        PASSING recipe as "c") AS CLOB (10k) )
FROM recipes
WHERE XMLEXISTS('$c/salad[name="Tomato Salad"]'
        PASSING recipe as "c");
```

Result:

Tomatoes

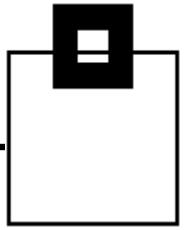
Onions

Parameter Markers



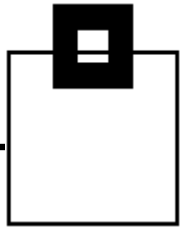
- Parameter markers is supported
- The cast function is needed to match the format of host variable and index
- Since variable names are case sensitive in XPATH, double quotes are needed to delimit
- Truncation of XML data will result in an error. - use UTF-8 to avoid conversion overhead and possible conversion data loss

Parameter Markers



```
SELECT XMLSERIALIZE ( XMLQUERY
   ('$c/salad/greens/text()'
            PASSING recipe as "c") AS CLOB (10k) )
FROM recipes
WHERE XMLEXISTS('$c/salad[name=$h]'
            PASSING recipe as "c"
         , CAST(? AS VARCHAR(128)) as "h"
);
```

Filtering in XMLQUERY vs. XMLEXISTS

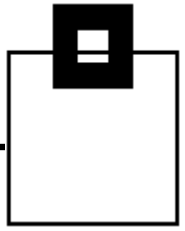


Filtering within XMLQUERY is possible, but has two disadvantages:

1. The filtering is done after retrieval of the row. If the filter does not match an empty row is returned
2. XML index usage is only possible with XMLEXISTS filtering

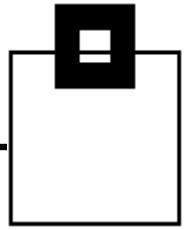
→ To avoid this, use a similar or same XPath in the XMLEXISTS in the WHERE clause

DB2 9 and XML – overview



- XMLEXISTS: is used for filtering in the WHERE clause
- XMLQUERY: is used for filtering in the SELECT clause
- XMLPARSE: is use to parse XML input (preserving whitespaces)
- XMLSERIALIZE: function to convert XML data into a LOB
- XMLTABLE: returns a “repeating group” in an XML document as in a “table”
- XMLCAST: allows casting between XML and non-XML data types
- XMLELEMENT: to construct one element
- XMLATTRIBUTES: to construct one or more attributes for an element.
- XMLNAMESPACES: to declare in-scope namespaces for an element(s)
- XMLFOREST: to construct a sequence of elements omitting null tags completely - trees of nodes from expressions
- XMLCONCAT: like SQL concat function
- XMLAGG: to concatenate multiple XML pieces
- XML2CLOB: converts transient XML data type into a CLOB
- XMLTEXT: Generates an XML text node
- XMLPI: Generates an XML processing instruction node
- XMLCOMMENT: Generates an XML comment node
- XMLDOCUMENT: Generates an XML document node

DB2 9 and XML



XML in DB2 (z/OS):

- SQL/XML functions with XPath
  - XMLEXISTS, XMLQUERY
  - XMLPARSE, XMLSERIALIZE
  - XMLTABLE, XMLCAST
- Update
  - Not yet implemented in Xpath (as of DB2 9)
  - Select -> Delete -> Insert
  - Stored procedure for update delivered with DB2 9
- Functions, to build up XML from relational data:
  - In DB2 V8: XMLElement, XMLAttributes, XMLNamespaces, XMLForest, XMLConcat, XMLAGG, XML2CLOB
  - Since in DB2 9: XMLText, XMLPI, XMLComment, XMLDocument

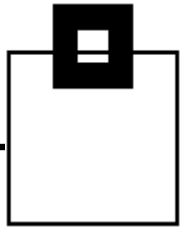
XML indexes

- XML indexes can be defined to optimize filtering
- Indexing is possible on an exact path, or more generic using wildcards
- In general: the more exact the path, the better the performance
- Host variables or constant values should match the format to allow index usage.
- To avoid codepage conversions host variables should be defined in binary format.

```
CREATE INDEX rec_ix1
 ON recipes (recipe)
 GENERATE KEY USING XMLPATTERN
 '/salad/name'
 AS SQL VARCHAR(96)
```

Exact path

XML indexes

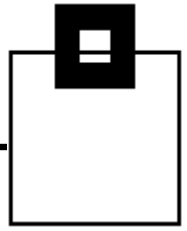


- The format of data inside the XML document is by default undefined.
  - When defining an index on an XML path you must decide between numeric and alphanumeric data (DECFLOAT or VARCHAR).
- An index key has to come from a single XML table row (record).

```
CREATE INDEX rec_ix1
 ON recipes (recipe)
 GENERATE KEY USING XMLPATTERN
 '//name'
 AS SQL VARCHAR(96)
```

Index on all occurrences of ‘name‘

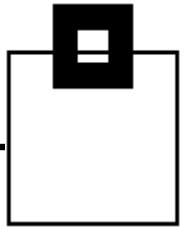
XML indexes



Some specialties for XML Indexes:

- The number of keys for each document (each base row) depends on the document and XMLPattern.
- Only one pattern per index is allowed
- For a numeric index, if a string from a document cannot be converted into a number, it is ignored.
  - <a>X5
  - XMLPattern '/a/b' as SQL Decfloat.
  - Only one entry '5' in the index.
- The uniqueness is checked after the value is cast to a SQL data type!
- For a string (VARCHAR(n)) index, if a key value is longer than the limit, INSERT or CREATE INDEX will fail.

XML Access Path



XML indexes can be used for XMLEXISTS and XMLTABLE functions

If no index exists, if no index matches a query, or if predicate/index data type doesn't match

→ DocScan will apply (ACCESS TYPE = R-scan

If one Index matches (exact or a containment relationship)

1. DB2 gets the DOCID list from the index
2. Sort to remove duplicates
3. DB2 converts the DOCID list to a base table RID list (this is the major usage of the DOCID index)
4. DB2 uses the RID list to fetch the base table rows and the XML data
5. Re-evaluate predicates by DocScan, (like in RID list access)

→ DOCID list access (ACCESS TYPE = DX)

XML Access Path

If multiple Indexes match DB2 generates a multi-index access plan

→ ACCESS TYPE = M + DX, DI, or DU (similar to MI and MU).

For AND predicates:

1. DB2 produces two or more DOCID lists from the indexes
2. Sort, remove duplicates and intersect
3. Continue with single access plan process

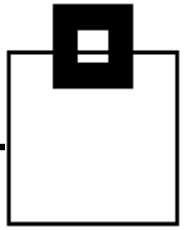
ORing applies if each of the primitive predicates under OR have a matching index:

1. Union DOCID lists from all indexes
2. Produce a unique DOCID list
3. Continue with single access plan process

XML – What was added

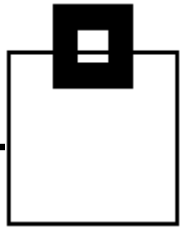
- New XPATH functions
  - PK55585
  - PK55831
- XMLCAST() and XMLTABLE()
  - PK51571
  - PK51572
  - PK51573
- XML Load performance improvements
  - PK47594
  - PK58766

XMLPATH functions



- fn.lower-case
- fn.upper-case
- fn.matches
- fn.position
- fn.replace
- fn.tokenize
- and more – in total 13 new XPATH functions

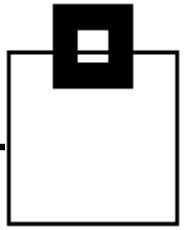
XMLCAST



- XMLCAST() allows casting between XML and non-XML data types
- to get SQL values back from XML documents, use XMLCAST to cast the XMLQUERY singleton result into an SQL type

Remember, XMLQUERY returns a sequence in general

XMLTABLE

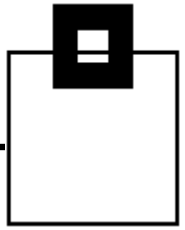


- XMLTABLE() returns a “repeating group” in an XML document as in a “table”

→ This allows using wildcards and column names accessing XML data

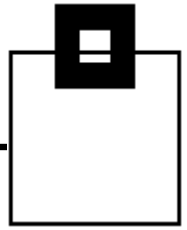
, but it requires materialization of the data BEFORE the predicate can be applied and may lead to a performance gotcha

XML LOAD improvements



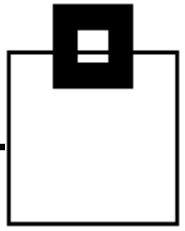
- XML parser is only called once when loading XML data

Changes in DB2 10



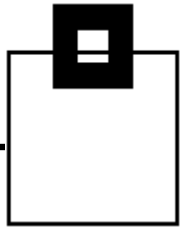
- Enhanced XML index support (also retrofitted to DB2 9 PK80732, PK80735)
- XML column type modifier for automatic schema validation
- Node-level XML updates
- XML support of SPs and UDFs + trigger enhancements
- XML binary encoding
- New multi-versioning concurrency approach
- Enhanced XML XPATH for date and time data types
- CHECK DATA, LOAD, and UNLOAD enhancements
- DEFINE NO for XML columns
- XML parsing and schema validation zIIP and zAAP eligible (also retrofitted to DB2 9 - PK90032, PK90040)

Enhanced XML index support



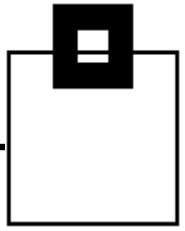
- Case-insensitive search
- Check for the existence, or absence of an element, or attributes (regardless of its value)

XML column type modifier



- Data Definition Language (DDL) option to define an XML column with a type modifier
- Can be altered
- Supports schema evolution
- One or multiple XML Schemas can be assigned
  → schema validation happens automatically for
  - Insert
  - Update
  - load operations
- Validation and XML parsing exploits XML system services (XSS) and runs “inside the engine”

XML objects in the DB2 catalog



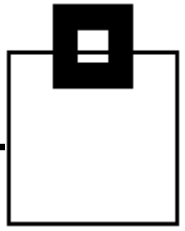
- SYSIBM.SYSXMLTYPMOD

  Contains rows for the XML type modifiers.

- SYSIBM.SYSXMLTYPMSCHEMA

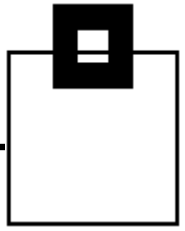
  Contains a row for each XML schema specification for one XML type modifier.

Node-level XML updates



- Insert, delete, or modify individual XML elements or attributes instead of full-document replacements
- Based on the XQuery Update Facility (similar DB2 LUW)

XML support of SPs and UDFs



- Stored procedures and User-Defined Functions can now have parameters and variables of data type XML

  → Significantly increases flexibility for application development

Multi-versioning acc. concurrency

- Only available if UTS and requires new function mode
- Concurrency control for XML now based on a new multi-versioning approach
- Avoids XML locking for readers
- Relieves Real Storage needs

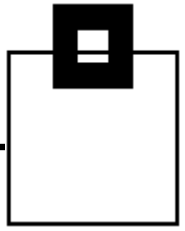
  → allows for higher concurrency and performance

- Length of the XML indicator column is 8 bytes longer (14 instead of 6)
- XML table and index key for the NODEID index will contain two more columns (bin. 8) to identify the version
  - START_TS (RBA/LRSN of the logical creation)
  - END_TS (RBA/LRSN of the logical deletion

XML XPATH for date & time

- Allows to query data matching a given date & time
- XML datetime format uses “T” as a separator indicating time-of-day (2012-01-15T11:00:00)
- xs:dateTime
- xs:yearMonthDuration

Changes in DB2 11



- Binary XML validation, including LOAD (retrofit to DB2 10)
- Cross system loader support for XML columns (LOAD … INCURSOR)
- DB2 implicitly adds a document node when storing data with the following statements:
  - INSERT
  - UPDATE
  - XMLDOCUMENT
- XML insert hotspots eliminated (retrofit to DB2 9 and 10)
- Partial revalidation (retrofit to DB2 10)
- XQUERY can now be used solely instead of SQL/XML
- More expressions (FLOWR, constructors, conditions)

Binary XML validation

- Binary XML can now be validated directly instead of serializing into string XML
- Inserts benefit from this in a range of
  - → 30% - 40% CPU reduction
  - → 15% - 18% CPU reduction compared to string XML
- Loads benefit from this in a range of
  - → 41% CPU reduction
  - → 18% CPU reduction compared to string XML

*This enhancement required z/OS 1.13 with PTF UA63422, or z/OS 1.12 with PTF UA65591

Cross system loader support

- XML crossloader support
  - → reduces virtual storage requirements
  - → avoids DSNU1178i errors via FETCH CONTINUE

Implicit document node

- Adds an implicit document node if missing in the XML document
  - → Avoids -20345 from a SQL instert or update
  - → no XMLDOCUMENT function before the insert or update needed anymore

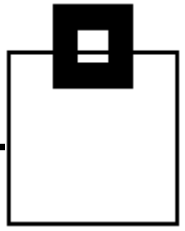
Insert hotspots elimination

- Instead of sequential DOCIDs DB2 now generates them randomly
  - → NPIs on the DOCID/NODEID are no longer a hotspot
  - → You might though see a slight sequential prefetch regression (no more look-aside)

*Requires to set system parameter RANDOMIZE XML DOCID = YES (DSNTIP8)

**Retrofitted into DB2 10 via APARs PM31486, PM31487, PM44216

Partial revalidation



- XML documents no longer require a full revalidation after a partly update
  - → aprox. 92% CPU reduction for 10MB documents
  - → aprox. 60% CPU reduction for 10KB documents

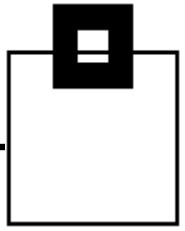
Soley use of XQUERY

- XQUERY continues to be limited compared to LUW, however, you are now able to express queries purely using XQUERY
  - → no more rewriting for supported DB2 syntax
  - → no more mixture of XQUERY and SQL/XML
  - → easier porting of applications to DB2 z/OS

More expressions

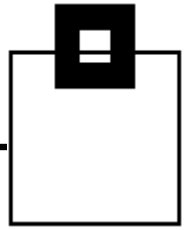
- FLWOR: a loop construct to manipulate XML documents
  - → **F**or loop through a sequence of values
  - → **L**et assign a value to a variable
  - → **W**here defines a criteria
  - → **O**rdere by orders the output
  - → **R**eturn defines what is actually returned
- XQUERY constructors create a XML structures within a query by an XML-like notation to
- Conditional expressions allow IF-THEN-ELSE logic within an XQUERY expression
- New built-in functions to return the average of values in a sequence

Hints and tips



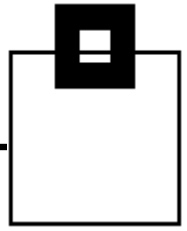
- Smaller documents tend to perform better, but may incur more overhead
- Prefer use of fully specified Xpath expressions rather than wildcards
  - Especially // or * patterns
- XMLQUERY in a SELECT does not filter documents or rows
- XMLQUERY in a SELECT does not use indexes
- If XMLEXISTS returns false no filtering occurs
  - Common pitfall: Missing square brackets in predicates !

Hints and tips



- Use a single XMLEXISTS clause with multiple XML predicates rather than multiple XMLEXISTS clauses whenever possible
- Create Indexes wisely:
  - Indexes add 10-20% of CPU time on top of an insert
- To use an index the index must be equally or less restrictive than the predicate
- Predicate must match the index data type

Where to find more?



- APAR II14426: Info for all the XML support delivery APARs
- Redbook: Extremely pureXML DB2 10 & 9 SG24-7915 (http://www.redbooks.ibm.com/abstracts/sg247915.html)

Ulf Heinrich

SEGUS, Inc

u.heinrich@segus.com