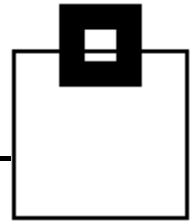
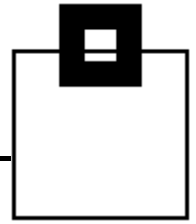


Episode 9

The Return of the Hierarchical Empire

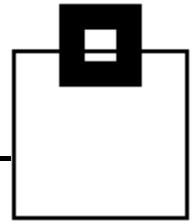
Ulf Heinrich
SEGUS, Inc
u.heinrich@segus.com

Agenda



- A few basics about XML in general
- XML integration in DB2 for z/OS V9
- Index design for XML
- Hints and tips

DB2 9 and XML



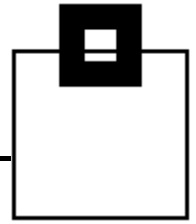
XML Support in DB2 9:

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

pureXML in DB2
„pure“ XML storing
supports hierarchical storing

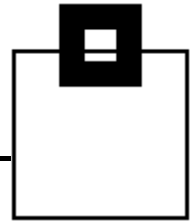
Support of: XPath, SQL/XML (z/OS)
XQuery (LUW)

<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>

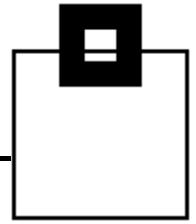


- Elements
- Attributes
- Nesting
- Content / Data

- Document Type Definition (DTD)
- XML schema language

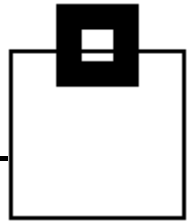
- http://www.w3.org/XML/
- http://www.w3.org/TR/REC-xml/
- Check also http://www.w3schools.com

XML 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



```
<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>
```

Root or Document Node

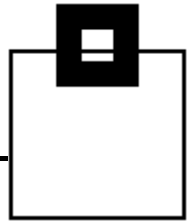
Attribute

Element

Content / Data

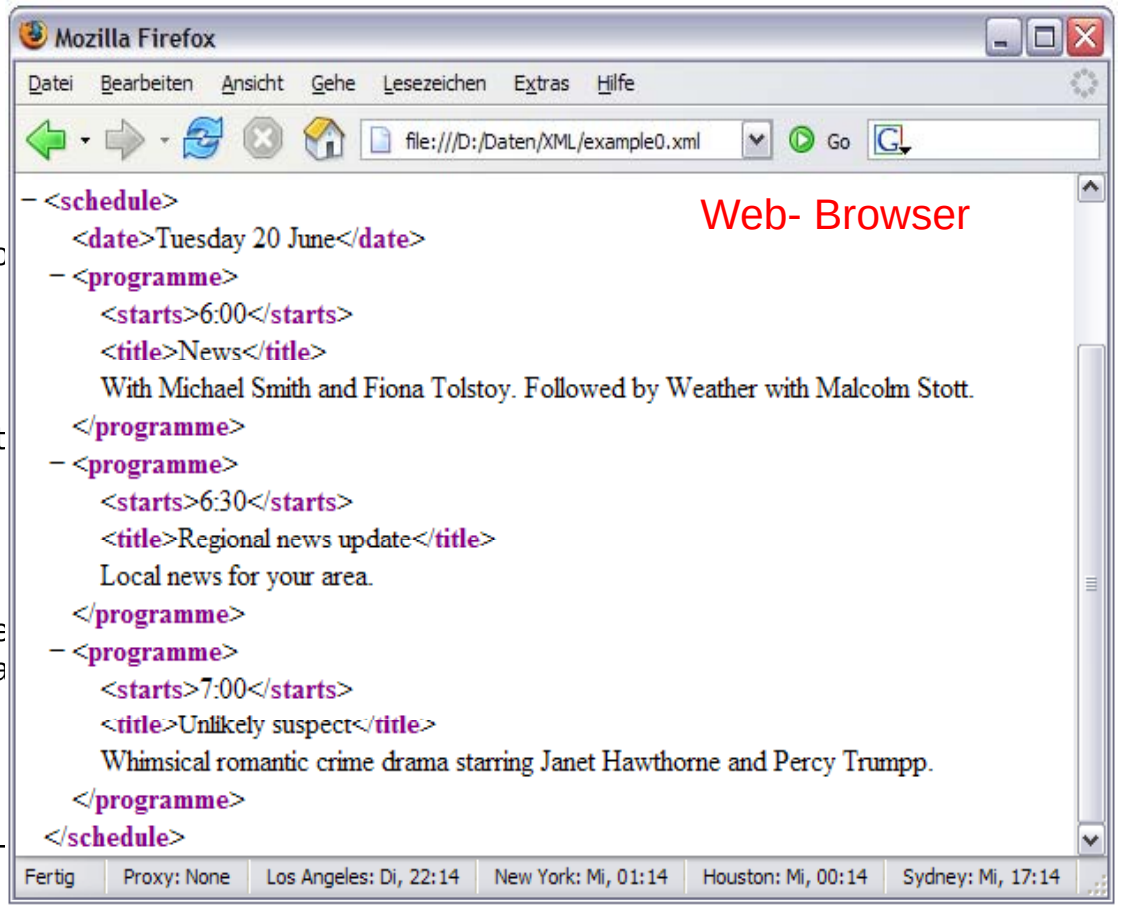
(DTD can be used to define the legal building blocks of an XML document.)

XML and the web



```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css"
    href="css.css"?>
<schedule>
  <date>Tuesday 20 June</date>
  <programme>
    <starts>6:00</starts>
    <title>News</title>
    With Michael Smith and Fiona
    Tolstoy.
    Followed by Weather with Malcc
    Stott.
  </programme>
  <programme>
    <starts>6:30</starts>
    <title>Regional news update</t
    Local news for your area.
  </programme>
  <programme>
    <starts>7:00</starts>
    <title>Unlikely suspect</title
    Whimsical romantic crime drama
    starring Janet
    Hawthorne and Percy Trumpp.
  </programme>
</schedule>
```

XML Doc



Web- Browser

XML documents and stylesheets

XML Doc

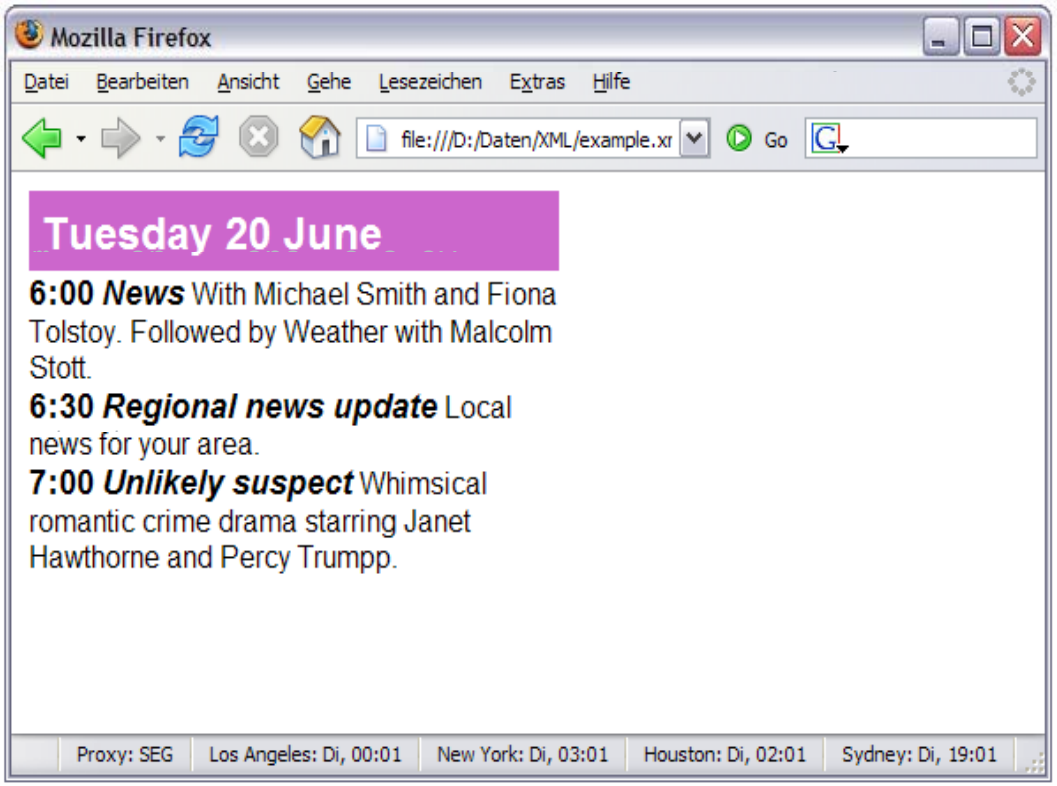
```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css"
    href="css.css"?>
<schedule>
  <date>Tuesday 20 June</date>
  <programme>
    <starts>6:00</starts>
    <title>News</title>
    With Michael Smith and Fiona
    Tolstoy.
    Followed by Weather with Malcolm
    Stott.
  </programme>
  <programme>
    <starts>6:30</starts>
    <title>Regional news update</title>
    Local news for your area.
  </programme>
  <programme>
    <starts>7:00</starts>
    <title>Unlikely suspect</title>
    Whimsical romantic crime drama
    starring Janet
    Hawthorne and Percy Trumpp.
  </programme>
</schedule>
```

Stylesheet

```
@media screen {
  schedule {
    display: block;
    margin: 10px;
    width: 300px;
  }
  date {
    display: block;
    padding: 0.3em;
    font: bold x-large sans-serif;
    color: white;
    background-color: #C6C;
  }
  programme {
    display: block;
    font: normal medium sans-
serif;
  }
  programme > * {
    font-weight: bold;
    font-size: large;
  }
  title {
    font-style: italic;
  }
}
```

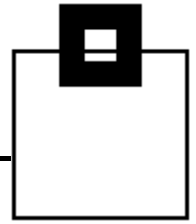
XML documents and stylesheets

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet
    href="css.css
<schedule>
  <date>Tuesday
  <programme>
    <starts>6:00
    <title>News</
    With Michael
    Followed by
    Stott.
  </programme>
  <programme>
    <starts>6:30
    <title>Regio
    Local news f
  </programme>
  <programme>
    <starts>7:00
    <title>Unlik
    Whimsical ro
    starring Jane
    Hawthorne an
  </programme>
</schedule>
```




```
@media screen {
                                    ylesheet

                                    ans-serif;

                                    6C;

                                    sans-

  title {
    font-style: italic;
  }
}
```

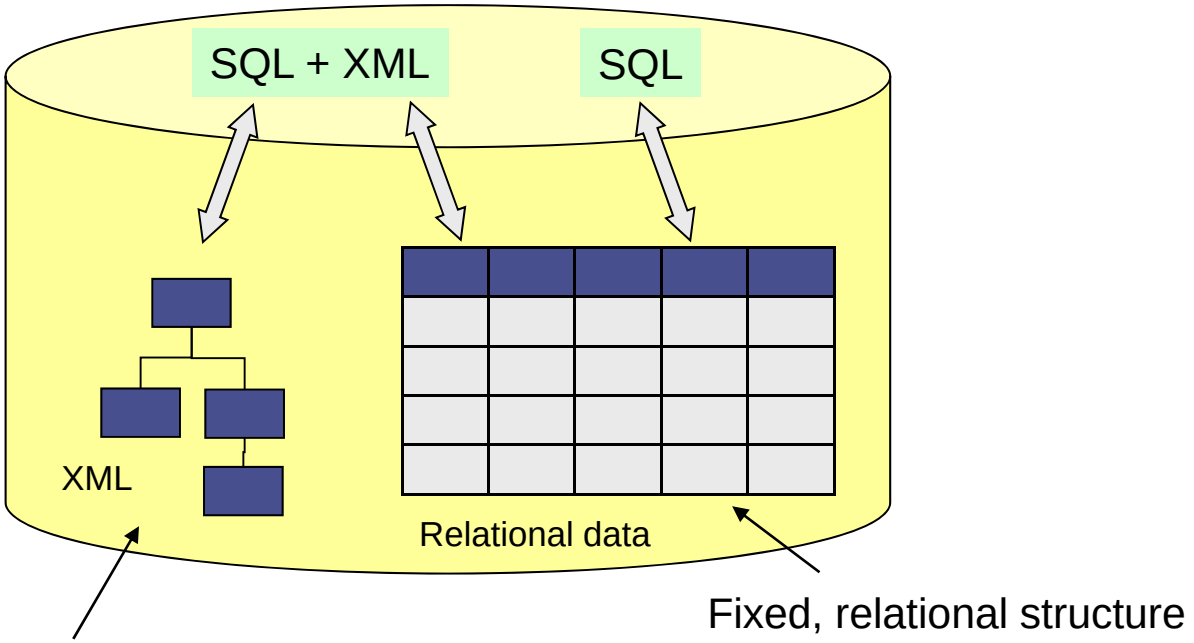
DB2 9 and XML



3 possibilities to store XML data in DB2:

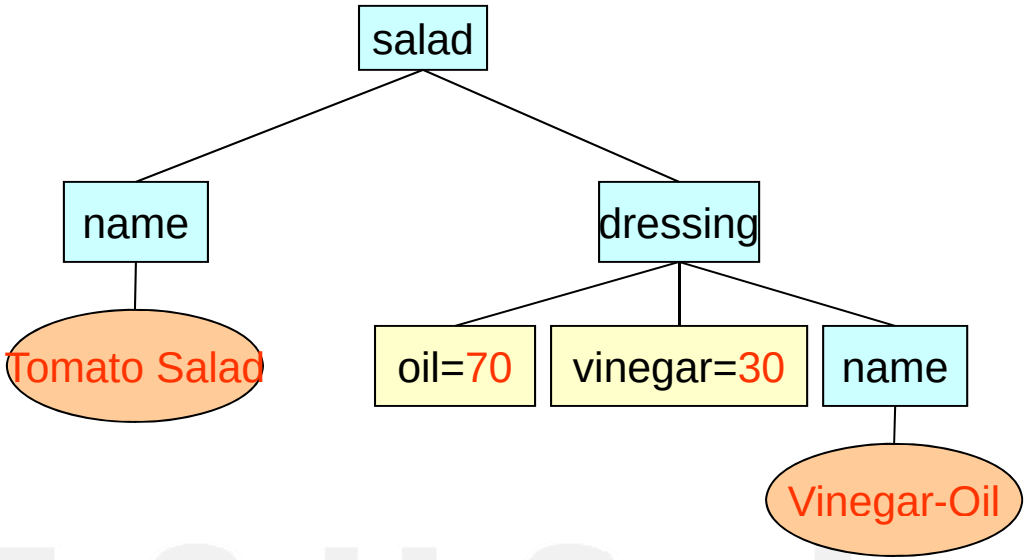
- XML as text
- XML shredding
- Native as parstree (parsing at insert time)

XML in DB2 architecture (z/OS):

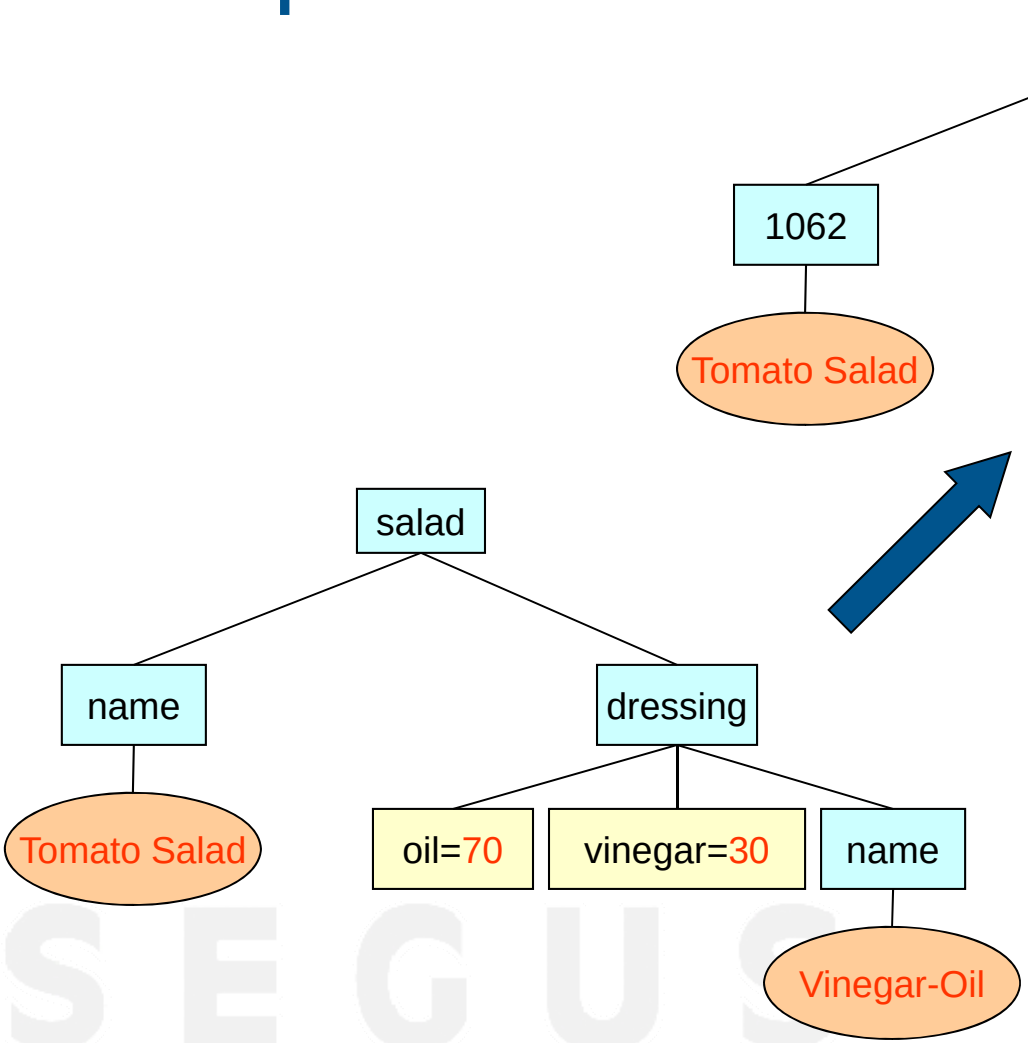


XML parstree

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



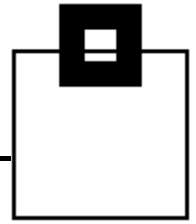
XML parstree and nodes



SYSIBM.SYSXMLSTRINGS

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

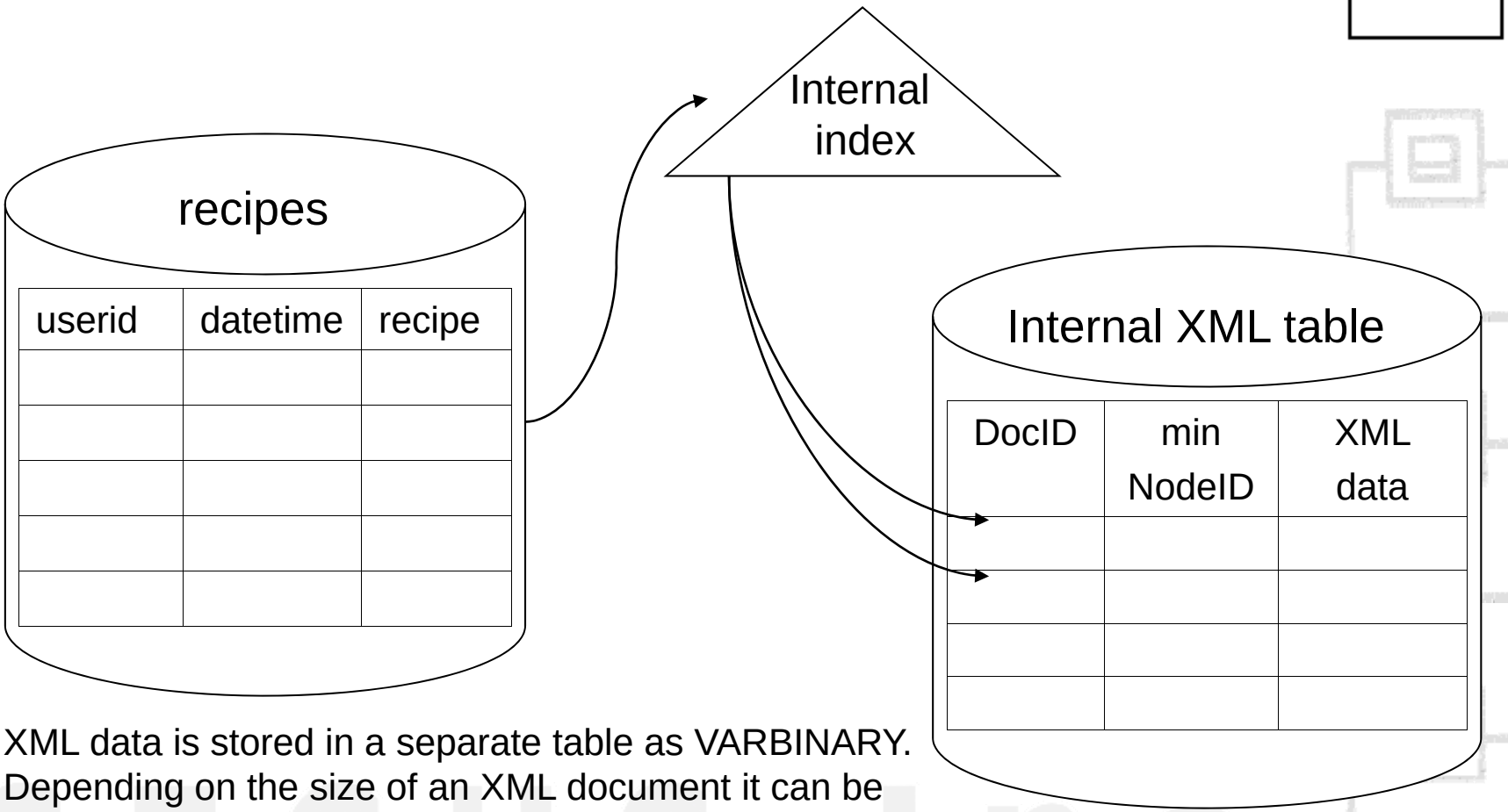
DB2 9 and XML



Column Type XML:

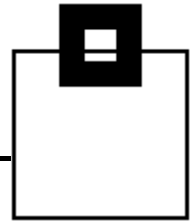
```
CREATE TABLE recipes (
    userid              VARCHAR(30),
    datetime            TIMESTAMP,
    recipe              XML
);
```

XML Storage in DB2



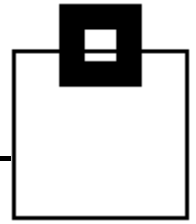
- XML data is stored in a separate table as VARBINARY.
- Depending on the size of an XML document it can be split up into more than one row.
- The XML column in the original table contains a link to the XML table.
- An internal, implicit DocID index is created to access the XML data.

XML Storage in DB2



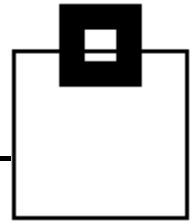
- XML table and base table are located in separate tablespaces
- RUNSTATS on base table does not collect statistics for XML tablespace nor for XML indexes
- Separate RUNSTATS necessary for XML tablespace

XML objects in the DB2 catalog



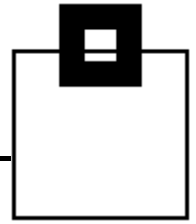
- SYSIBM.SYSXMLRELS
  - Contains one row for each XML table that is created for an XML column.
- SYSIBM.SYSXMLSTRINGS
  - Each row contains a single string and its unique ID that are used to condense XML data. The string can be an element name, attribute name, name space prefix, or a namespace URI.

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)
    - For an XML index, this is the XML pattern that is used to generate the key-target value.

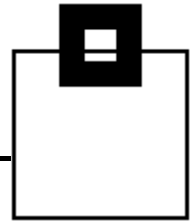
How to get it in - XMLPARSE



```
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>
   ') )
```

* DSN_XMLVALIDATE can be used to define an XML schema

Accessing XML data with SQL

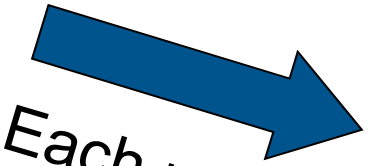


```
SELECT userid, recipe
   FROM recipe
   WHERE userid = 'Fred'
```

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

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

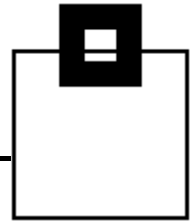


/
/salad
/salad/name
/salad/dressing
/salad/dressing/@oil
/salad/dressing/@vinegar
/salad/dressing/name

XPTH allows

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

Path expressions and wildcards

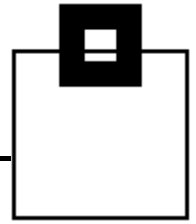


@	specifies an attribute
text()	specifies a node under an element
*	matches any tag name
//	is the “descendent-or-self“ wildcard
.	specifies the current context
..	specifies the parent context

Predicates are enclosed in square brackets [ ]

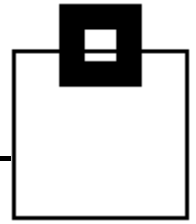
[*n*] is a shortcut to [fn:posotion ()*n*]

Filter on XML using XMLEXISTS



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

XMLQUERY on XML

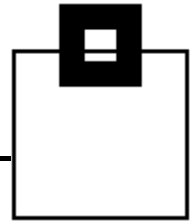


```
SELECT XMLQUERY ('$c/salad/greens'
            PASSING recipe as "c")
FROM recipes
WHERE XMLEXISTS('$c/salad[name="Tomato Salad"]'
            PASSING recipe as "c");
```

Result:

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

XMLSERIALIZE and text()



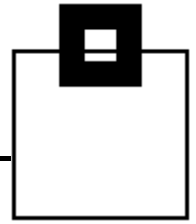
```
SELECT XMLSERIALIZE ( XMLQUERY ('$c/salad/greens/text()'
        PASSING recipe as "c") AS CLOB (10k) )
FROM recipes
WHERE XMLEXISTS('$c/salad[name="Tomato Salad"]'
        PASSING recipe as "c");
```

Result:

Tomatoes

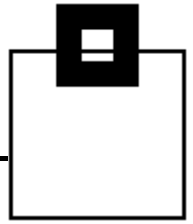
Onions

Parameter Markers



```
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"
);
```

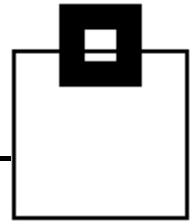
Filtering in XMLQUERY vs. XMLEXISTS



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.

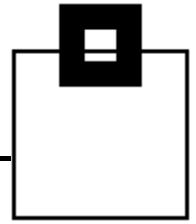
DB2 9 and XML



XML in DB2 (z/OS):

- SQL/XML functions with XPath
  - XMLEXISTS, XMLQUERY
  - XMLPARSE, XMLSERIALIZE
  - XMLTABLE
- Update
  - Not yet implemented in XPath
  - 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
  - New in DB2 9: XMLText, XMLPI, XMLComment, XMLDocument

XML indexes



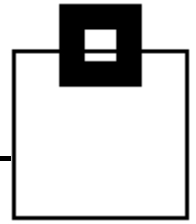
```
CREATE INDEX rec_ix1
 ON recipes (recipe)
 GENERATE KEY USING XMLPATTERN
 '/salad/name'
 AS SQL VARCHAR(96)
```

Exact path

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

Index on all occurrences of ‘name‘

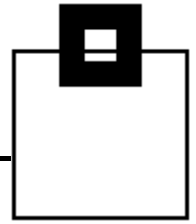
XML indexes



Some specialties for XML Indexes:

- The number of keys for each document (each base row) depends on the document and XMLPattern.
- 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.
- For a string (VARCHAR(n)) index, if a key value is longer than the limit, INSERT or CREATE INDEX will fail.

XML Acccess 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 don't match

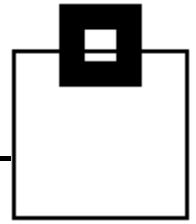
→ DocScan will apply (ACCESS TYPE = R-scan

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 Acccess Path



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

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

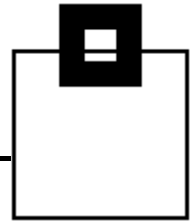
For AND predicates:

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:

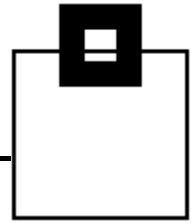
1. Union DOCID lists from all indexes
2. Produce a unique DOCID list
3. Continue with single access plan process

XML - What's new



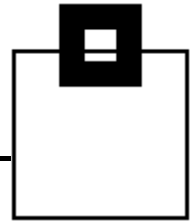
- New XPATH functions
  - PK55585
  - PK55831
- XMLCAST() and XMLTABLE()
  - PK51571
  - PK51572
  - PK51573
- XML Load performance improvements
  - PK47594
  - PK58766

XMLPATH functions



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

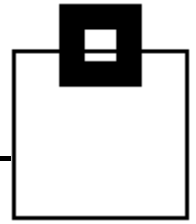
XMLCAST



- 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

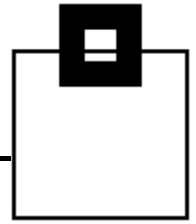


- XMLTABLE() returns a “repeating group” in an XML document as in a “table”

→ This allows using wildcards and column names accessing XML data

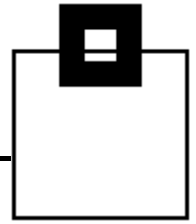
, but it requires materialization of the data BEFORE the predicate can be applied and may lead to a performance gotcha

XML LOAD improvements



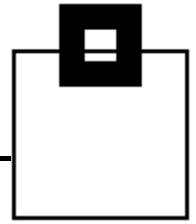
- XML parser is only called once when loading XML data

Hints and tips

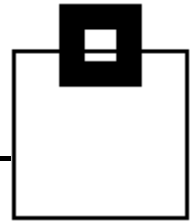


- Smaller documents tend to perform better, but may incur more overhead
- Prefer use of fully specified Xpath expressions rather than wildcards
  - Espacially // or * patterns
- XMLQUERY in a SELECT does not filter documents or rows
- XMLQUERY in a SELECT does not use indexes
- If XMLEXISTS returns false no filtering occures
  - 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



Ulf Heinrich

SEGUS, Inc

u.heinrich@segus.com