

5-9 November

Athens Hilton

Athens, Greece

Session: A04

IDUG® 2007

Europe

Episode 9 – The Return of the Hierarchical Empire

XML in DB2 9 for z/OS

06 November 2007 • 09:00 a.m. – 10:00 a.m.

Platform: DB2 for z/OS



IDUG
The Worldwide DB2 User Community

GoFurther

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/>

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.)

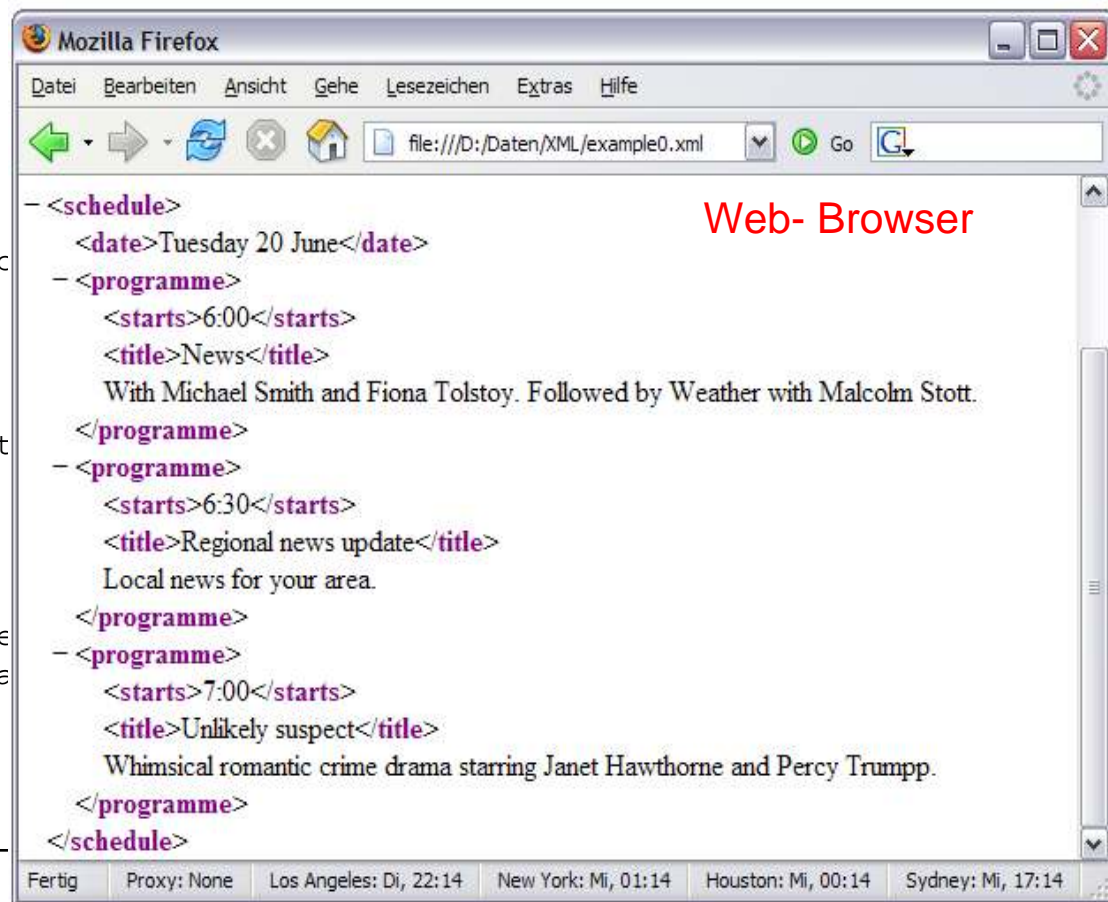
XML and the web

```

<?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 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>

```

XML Doc



XML documents and stylesheets

```
<?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>
```

XML Doc

```
@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;
  }
}
```

Stylesheet

XML documents and stylesheets

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css"
  href="css.css" />
<schedule>
  <date>Tuesday 20 June 2006</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>
```

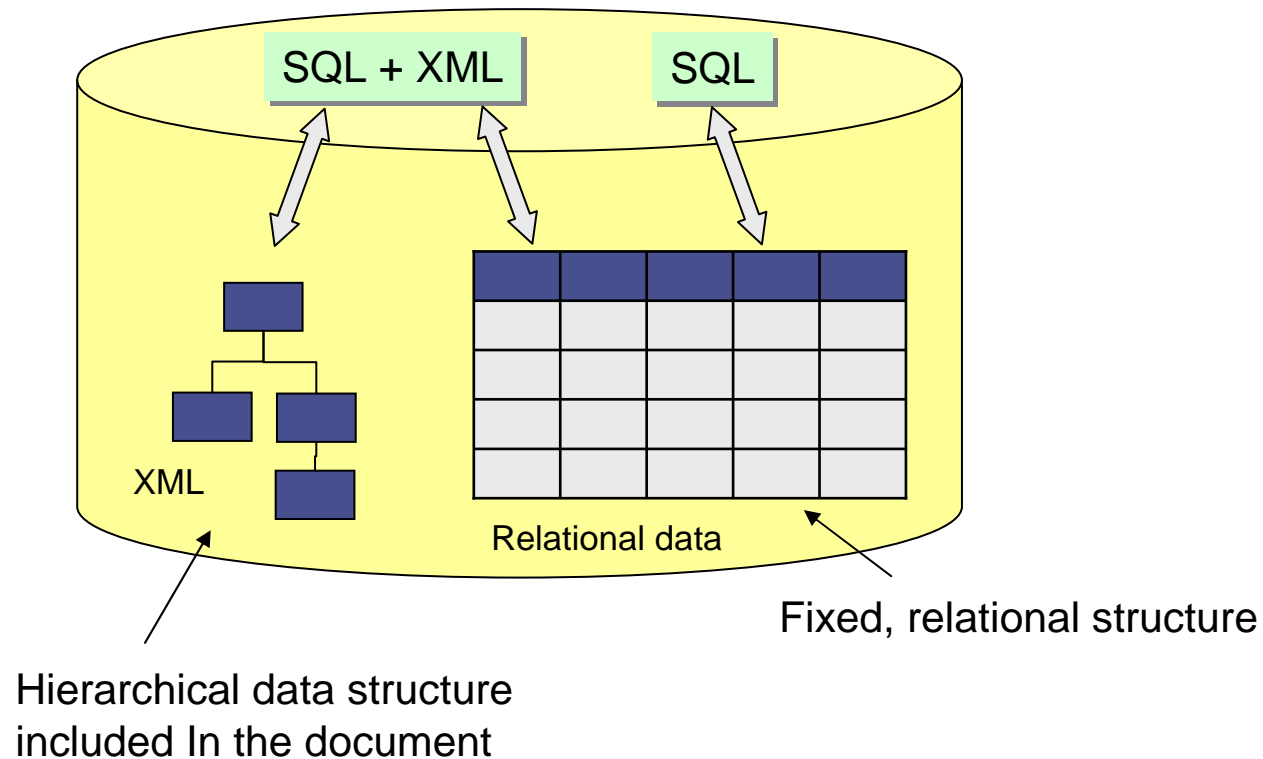
```
@media screen {
  .title {
    font-style: italic;
  }
}
```

DB2 9 and XML

3 possibilities to store XML data in DB2:

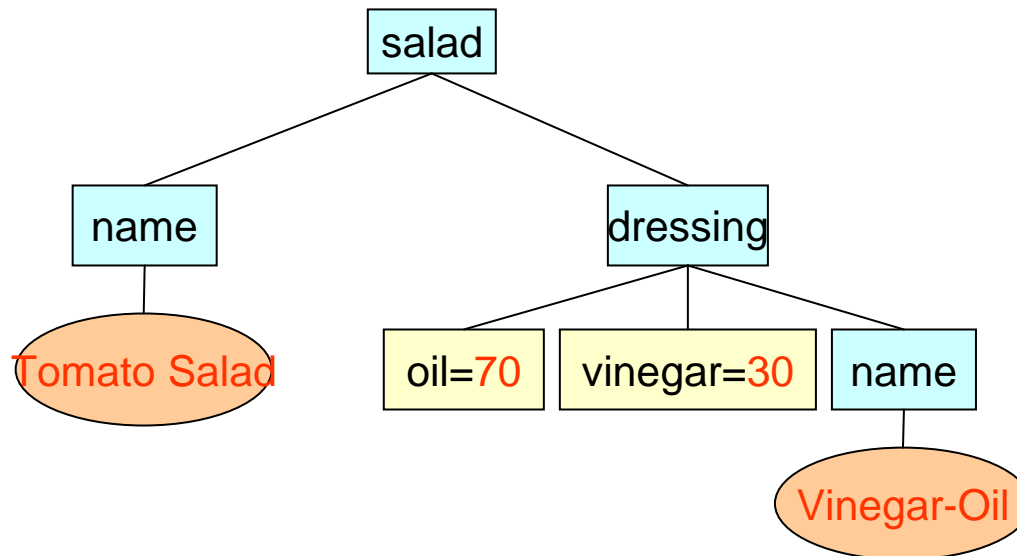
- XML as text
- XML shredding
- **Native as parstree (parsing at insert time)**

XML in DB2 architecture (z/OS):

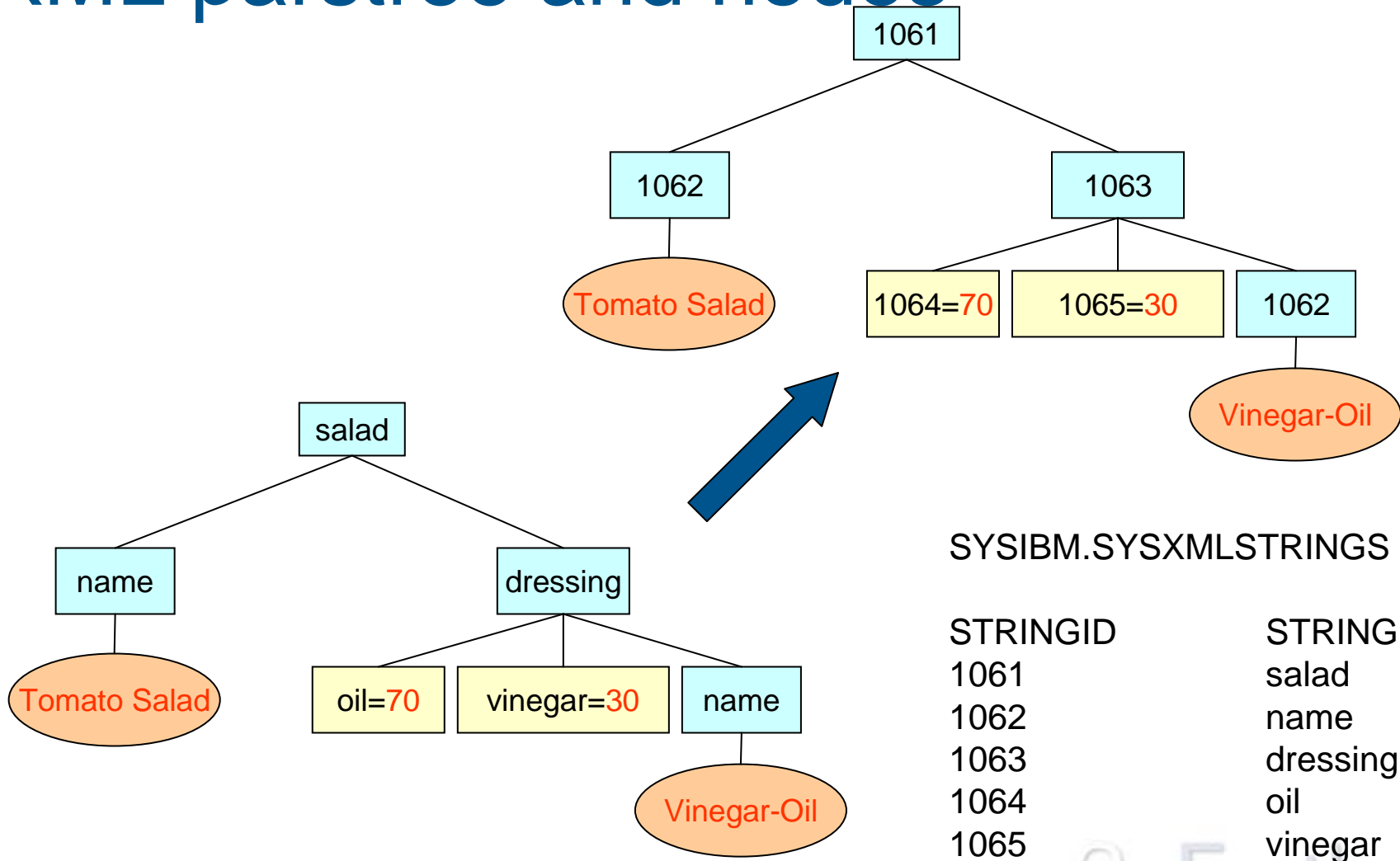


XML parstree

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



XML parstree and nodes

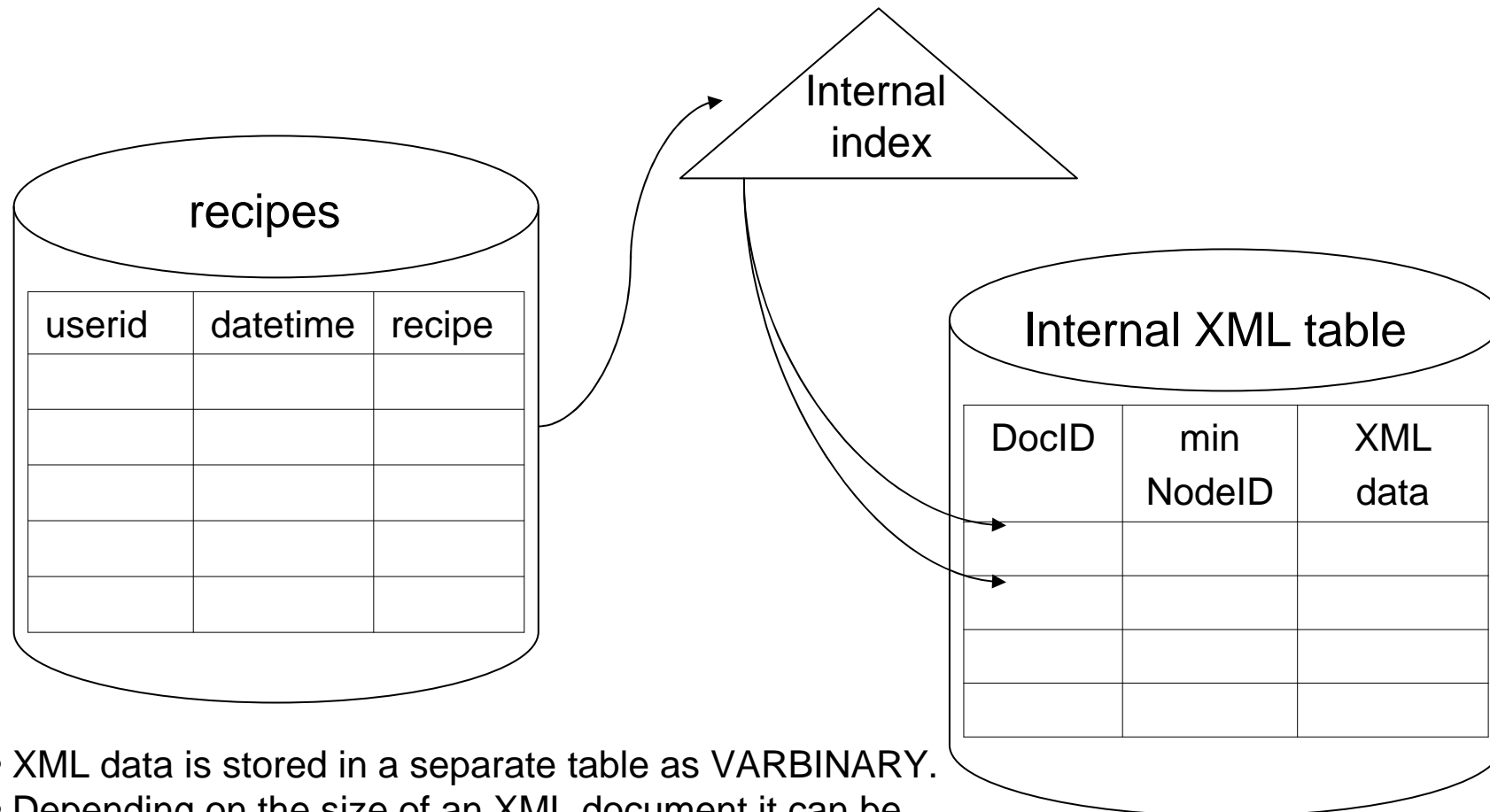


DB2 9 and XML

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 XML documents, but not to filter on elements inside the XML document

This can be done using XPath expressions imbedded in SQL

XPath

```
<salad>
```

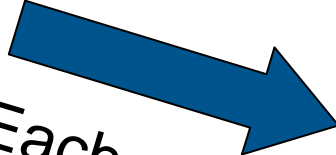
```
  <name>Tomato Salad</name>
```

```
  < dressing oil="70" vinegar="30">
```

```
    <name>Vinegar-oil</name>
```

```
  </dressing>
```

```
</salad>
```


Each node
has a path

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

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 []

Filter on XML part with XMLEXISTS

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

XMLQUERY on XML part

```
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
 - New in DB2 9: XMLText, XMLPI, XMLComment, XMLDocument

XML indexes

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

```
CREATE INDEX rec_ix1
ON recipes (recipe)
GENERATE KEY USING XMLPATTERN
'//name' ← Index on all occurrences of 'name'
AS SQL VARCHAR(96)
```

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.

Hints and tips

- Smaller documents tend to perform better
- Prefer use of fully specified Xpath expressions rather than wildcards
- 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
- To use an index the index must be equally or **less** restrictive than the predicate
- Predicate must match the index data type

Session A04

Session Episode 9 – The Return of the Hierarchical Empire

Ralf Neumann

SOFTWARE ENGINEERING

r.neumann@seg.de