bm.com

e-business

DB2 UDB Extenders for iSeries
XML and Text Extenders

BP02

ITSO iSeries Technical Forum

Hernando Bedoya

International Technical Support Organization

Notes

The DB2 UDB Extenders for iSeries is shipped as a separate license program 5722-DE1. This means that the Extenders LPP is not a part of the DB2 UDB for iSeries runtime.

The current release contains two extenders:

DB2 UDB XML Extender

DB2 UDB Text Extender

In fact, you can think of both extenders as a middleware that resides on top of the DB2 UDB for iSeries database and enhances its functionality by providing a range of user defined types (UDTs), user defined functions (UDFs), and stored procedures.

Disclaimer

Putting DB2's Power to Support XML

XML represents a fundamental change in computing ... away from proprietary file and data formats to a world of open interchange

- XML = portable data

DB2 provides stability, scalability and security

Your mission-critical business data is currently stored in DB2

- Where do you store your XML documents?
- How can you convert your business data into XML documents?
- How can you turn the XML data into database data?

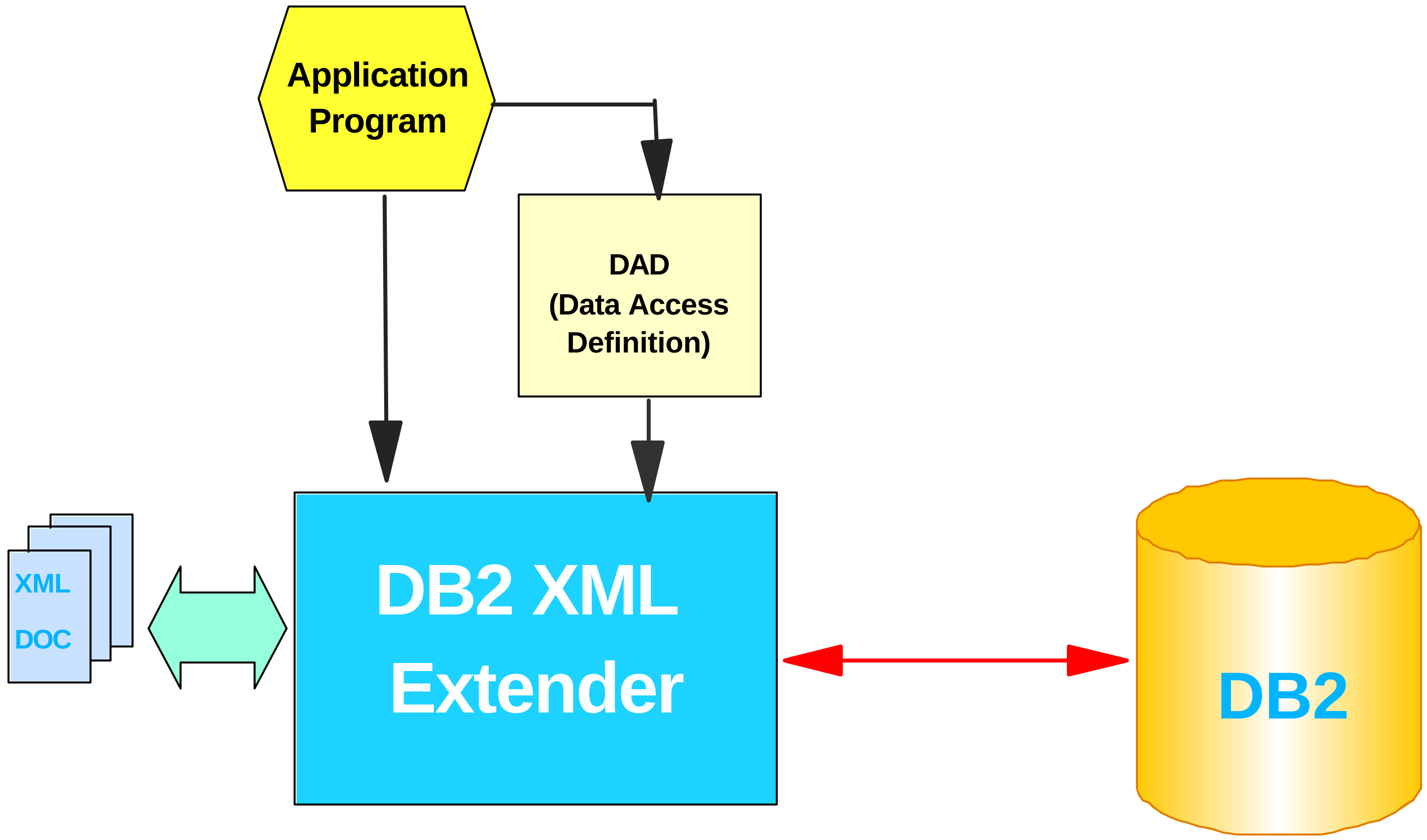
Answer: DB2 UDB for iSeries Extenders

Notes

XML is universal not only by its range of applications but also by its ease of use: Its text-based nature makes it easy to create tools, and it is also an open, license-free, cross-platform standard, which means anyone can create,develop, and use tools for XML.

Although XML solves many problems by providing a standard format for data interchange, some challenges remain. In the real world, application design has always had to address issues such as sharing data between applications, replication, transformation, exporting and saving of data. These kinds of issues can be addressed only by a database management system. By incorporating the XML information and meta-information directly in the database, you can more efficiently obtain the XML results that your other applications need. With the content of your structured XML documents in a DB2 database, you can combine structured XML information with traditional relational data. With the help of the XML Extender, the XML documents can be stored entirely in the database columns or you can setup mapping so that XML documents can be decomposed in to existing columns or generated from data in existing columns.

XML Extender Overview



DB2 XML Extender provides

- new data types that let you store XML documents in DB2 databases
- new functions that assist you in working with these structured documents

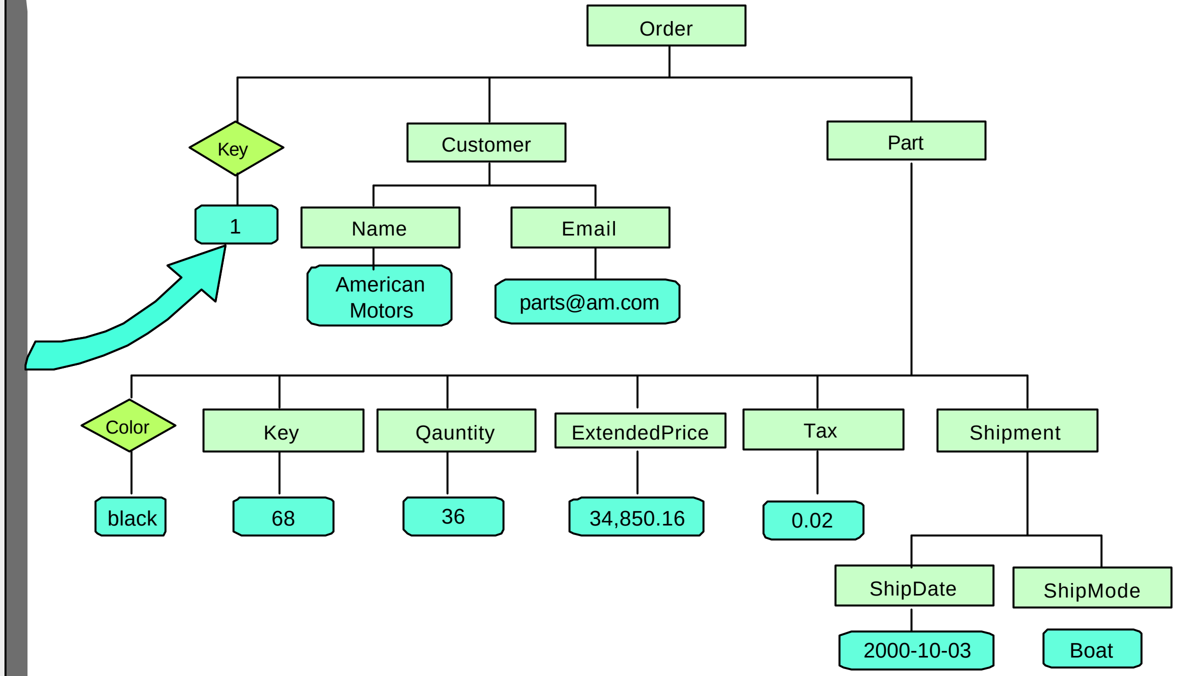
Notes

There are several components that make up the XML Extender architecture. DB2 is used to store and retrieve XML data and also generates helper side tables that can greatly improve the performance of the XML retrieval process. Additionally there is the extender itself which mediates between DB2 and the application requester.

The extender is functional and flexible depending on whether you have relational data that need to be transformed into XML or XML data to store into DB2 tables. The extender contains a rich set of user defined types (UDTs), user defined functions (UDFs) and stored procedures to manage XML data in DB2. XML documents can be stored in DB2 databases as character data or stored as external files but still managed by DB2. Retrieval UDFs allow you to retrieve either the entire XML document or individual elements or attributes.

XML document logical structure

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM
"/dxx/samples/dtd/getstart.dtd">
<Order key="1">
  <Customer>
    <Name>American Motors</Name>
    <Email>parts@am.com</Email>
  </Customer>
  <Part color="black ">
    <key>68</key>
    <Quantity>36</Quantity>
<ExtendedPrice>34850.16</ExtendedPrice>
    <Tax>6.000000e-2</Tax>
    <Shipment>
      <ShipDate>2000-10-03</ShipDate>
      <ShipMode>BOAT  </ShipMode>
    </Shipment>
   ....
  </Part>
</Order>
```



- XML document has a tree-like hierarchical structure
- XML uses start and end tags as containers
- XML document must have only one root element

Notes

Before we cover the details of the DB2 XML Extender implementation, we need to explain several basic concepts used throughout this presentation.

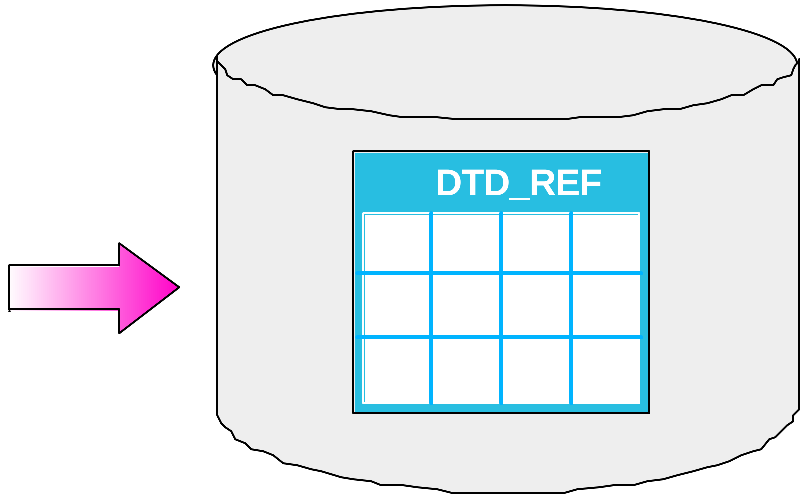
As you can see from this foil, an XML document has a tree-like structure, with the root element (<Order>) at the top of the tree. All the elements that are inside the root element are also contained within each other. The document must contain one and only one root element. We say that an element is the parent of the elements it contains. The elements that are inside an element are called its children. Similarly, the elements that have the same parent element are called siblings.

In our example, <Order> is parent of all other elements, <Name> is a child of <Customer>, and <Customer> and <Part> are siblings. Going down the element tree, each child element must be fully contained with its parent element. Sibling elements may not overlap.

Document Type Definition Repository

- DTD reference table created when database is enabled for XML
- Each row of the DTD reference table represents a DTD
- Users can insert their own DTDs
- The DTDs used to validate XML documents

```
<?xml encoding='IBM037'?>
<!ELEMENT Order (Customer, Part+)>
<!ATTLIST Order key CDATA #REQUIRED>
<!ELEMENT Customer (Name, Email)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Email (#PCDATA)>
<!ELEMENT Part (key,Quantity,ExtendedPrice,Tax,
Shipment+)>
<!ELEMENT key (#PCDATA)>
<!ELEMENT Quantity (#PCDATA)>
<!ELEMENT ExtendedPrice (#PCDATA)>
<!ELEMENT Tax (#PCDATA)>
<!ATTLIST Part color CDATA #REQUIRED>
<!ELEMENT Shipment (ShipDate, ShipMode)>
<!ELEMENT ShipDate (#PCDATA)>
<!ELEMENT ShipMode (#PCDATA)>
```



Notes

The DTD specifies the structure of an XML document, thereby allowing XML parsers to understand and interpret the document's contents. The DTD contains the list of tags which are allowed within the XML document and their types and attributes. More specifically, the DTD defines how elements relate to one another within the document's tree structure, and specifies which attributes may be used with which elements. Therefore, it also constrains the element types that can be included in the document and determines its conformance: An XML document which conforms to its DTD is said to be valid.

DB2 XML Extender provides an XML DTD repository. A DTD reference table called DTD_REF is created at the time when the database is enabled. Enabling the database creates the following objects:

• The db2xml schema (library).

• The user-defined types (UDT) XMLVARCHAR, XMLCLOB, XMLFILE, as well as all required user-defined functions (UDF) and stored procedures

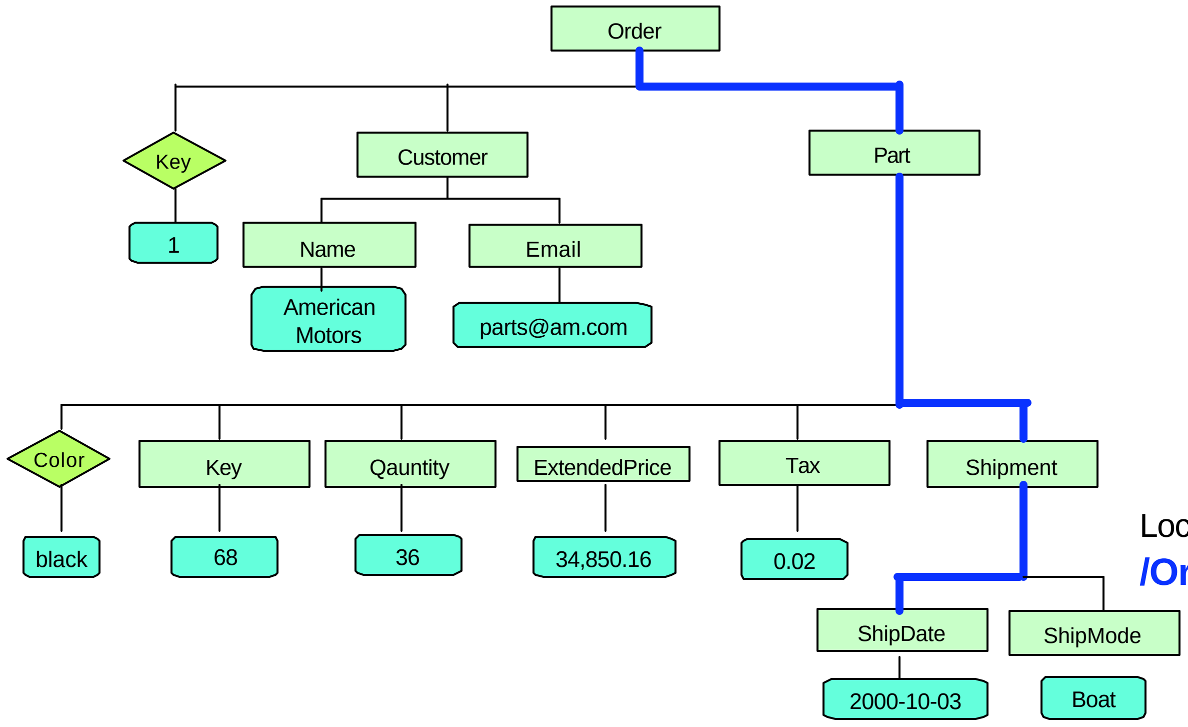
• The necessary control tables required by DB2 XML Extender such as XML_USAGE and DTD_REF

Each row in the DTD_REF table contains a DTD with additional metadata information about it. You can insert your own DTDs into this table. The DTDs in this table are used to validate the XML documents.

Location path

A sequence of XML tags that identify an XML element or attribute

- used by XML Extender to map an XML element or attribute to a DB2 column
- used by Text Extender for structural text search



Location path for the ShipDate element:
/Order/Part/Shipment/ShipDate

Notes

A location path is a sequence of XML tags separated by a forward slash (/) that identifies an XML element or attribute. Location paths are used in the following situations within DB2 XML Extender and DB2 Text Extender:

• They are given as input to extracting UDFs to identify elements and attributes to be extracted.

• They are used to specify the mapping file between an XML element or attribute and a DB2 column when defining the indexing scheme in the DAD for XML Columns

• They are used by the Text Extender for structural-text search.

The following is the location path syntax supported by DB2 XML Extender:

• /: Represents the XML root element.

• /tag1:Represents the element tag1under root.

• /tag1/tag2/..../tagn : Represents an element with the name tagn as the child of the descending chain from root, tag1, tag2, through tagn-1.

• //tagn: Represents any element with the name tagn, where double slashes(//) denote zero or more arbitrary tags.

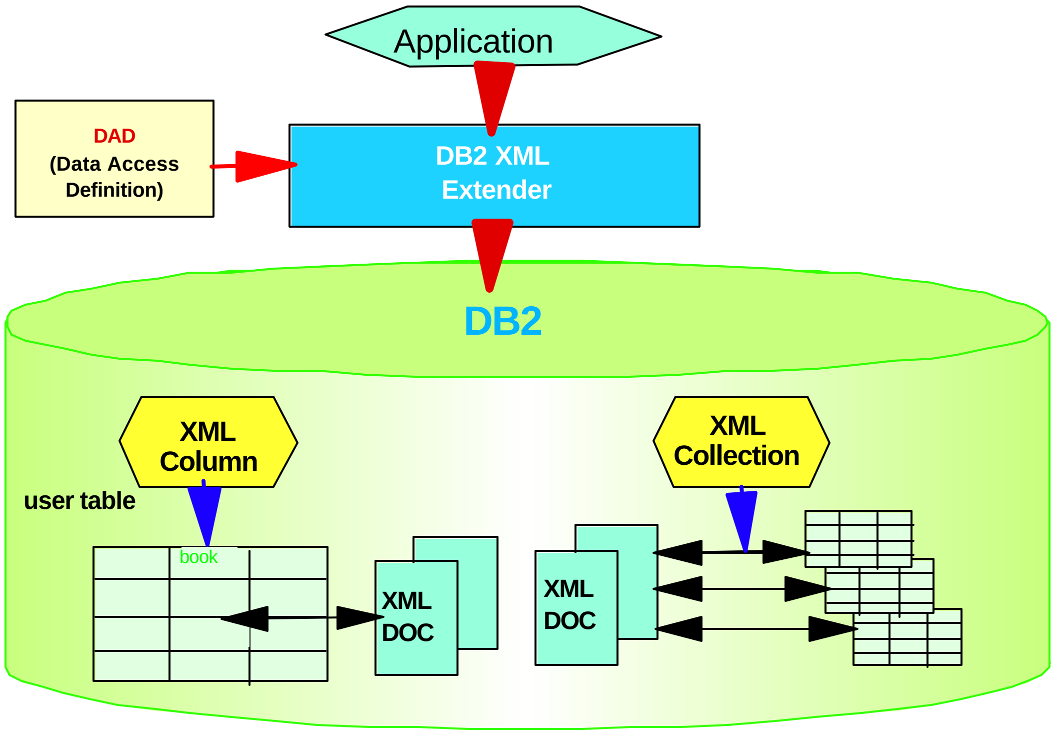
• /tag1//tagn: Represents any element with the name tagn, a child of an element with the name tag1under root, where double slashes (//) denote zero or more arbitrary tags.

• /tag1/tag2/@attr1: Represents the attribute attr1of an element with the name tag2, which is a child of element tag1under root.

• /tag1/tag2[@attr1="5"]: Represents an element with the name tag2 whose attribute attr1 has the value 5. tag2is a child of element with the name tag1under root.

• /tag1/tag2[@attr1="5"]/.../tagn: Represents an element with the name tagn, which is a child of the descending chain from root, tag1, tag2, through tagn-1, where the attribute attr1 of tag2 has the value 5.

Two Access and Storage Methods



XML column

- store and retrieve entire XML documents as DB2 column data
- XML data represented by XML column

XML Collection

- decompose XML document into a collection of relational tables
- compose XML documents from a collection of relational tables

Notes

The DB2 XML Extender provides you with the ability to use DB2 to store, manage, query and update XML data. There are two basic techniques used, the XML column method, and the XML collection method.

Using the **XML column method**, you can use DB2 tables store XML documents in columns that have been enabled for XML, or you can store them as external files. The XML data can then be retrieved, updated, and searched. Furthermore, you can extract XML element or attribute values into secondary tables called side tables which, when indexed, provide fast XML element and attribute search capabilities.

Columns that have been enabled for XML are known as XML columns, and can be implemented as one of the three user-defined types provided with the XML Extender:

- XMLVARCHAR
- XMLCLOB
- XMLFILE

The **XML collection method** allows you to compose XML documents from existing DB2 data, or decompose XML documents into DB2 data, that is, store untagged element or attribute values in DB2 tables. This method is useful for Business-to-Business (B2B) or Electronic Data Interchange (EDI) applications, particularly if the contents of XML documents are frequently updated.

XML Columns Scenario

Scenarios suitable for XML Columns:

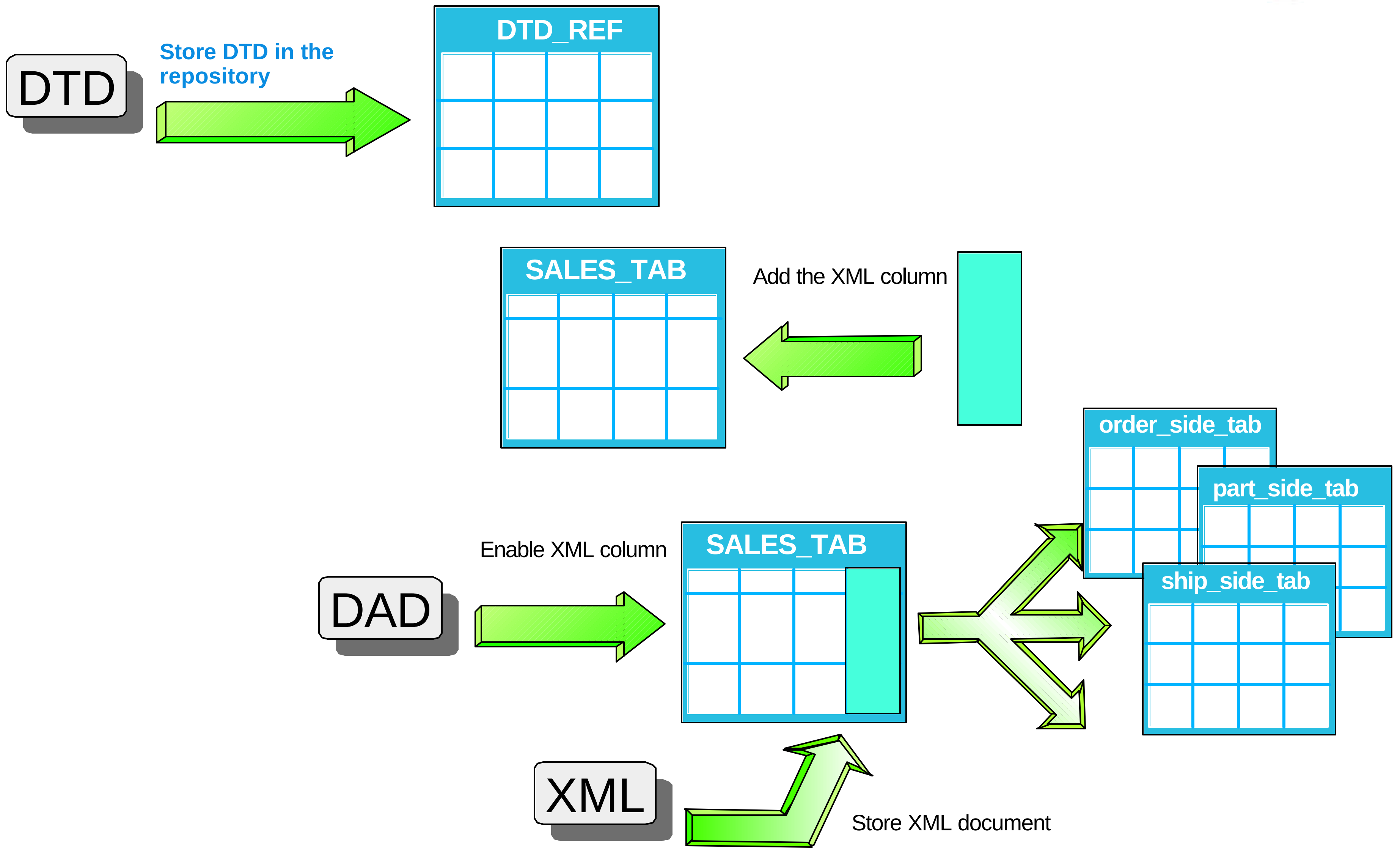
- XML documents already exist or come from some external source
  - you want to store them in DB2 for integrity or for archive and auditing purpose
  - you prefer to store documents in native XML format
- XML documents are read mostly
  - performance of update is not critical
  - range search is needed based on the values of XML elements or attributes
- The documents have elements with large text block and you want to use Text Extender for structural text search

Notes

As mentioned earlier, using the XML Columns method allows you to store the entire XML document, as it is, in a column. We recommend choosing the XML Columns method if one or more of the following criteria are met:

- The XML documents already exist — for example, you want to archive documents such as newspaper articles, orders, and so on.
- The XML documents are read-often and update-rarely.
- The performance of the update is not critical.
- You want to store the intact XML documents.
- You want to keep the XML documents externally from DB2 in a local file system.
- You know what elements or attributes will be frequently searched. To perform efficient searches on these documents, you can decide to create indexes in side tables on the elements or attributes that you need to access more often.

XML column setup



Notes

This foil presents the steps required to store an XML document in a XML column. We assume that the database is already XML-enabled.

1. If you plan to validate the XML documents, you should store the appropriate DTD document in the DTD repository.

2. The next step is to decide in which table you will store the XML documents. You can create a new table with an XML Column or just alter an existing table to add an XML Column. DB2 XML Extender provides you with three new user-defined types located in the db2xml schema (library) to store your XML documents as column data:

• XMLVARCHAR: You can store an XML document in the database, with a maximum size of 32 KB.

• XMLCLOB: The XML document is also stored in the database, but its maximum size is 2 GB.

• XMLFILE: This UDT allows you to keep the document on the local file.

3. Create the Document Access Definition file. The DAD file, itself an XML document, specifies how the XML documents that you store in the database are to be handled.

In the case of XML Columns, the DAD file is only needed if you want to validate your XML documents before storing them, or if you want to index elements or attributes in side tables. The side tables are additional tables created by DB2 XML Extender to improve performance when searching elements or attributes in an XML Column.

4. If you created a DAD file for an XML Column, it is necessary that you tell DB2 XML Extender to which XML Column and in which table this file relates. When you enable an XML Column, DB2 XML Extender does the following:

• It parses the DAD file.

• It creates the side tables with the desired columns corresponding to the elements and attributes in the XML document.

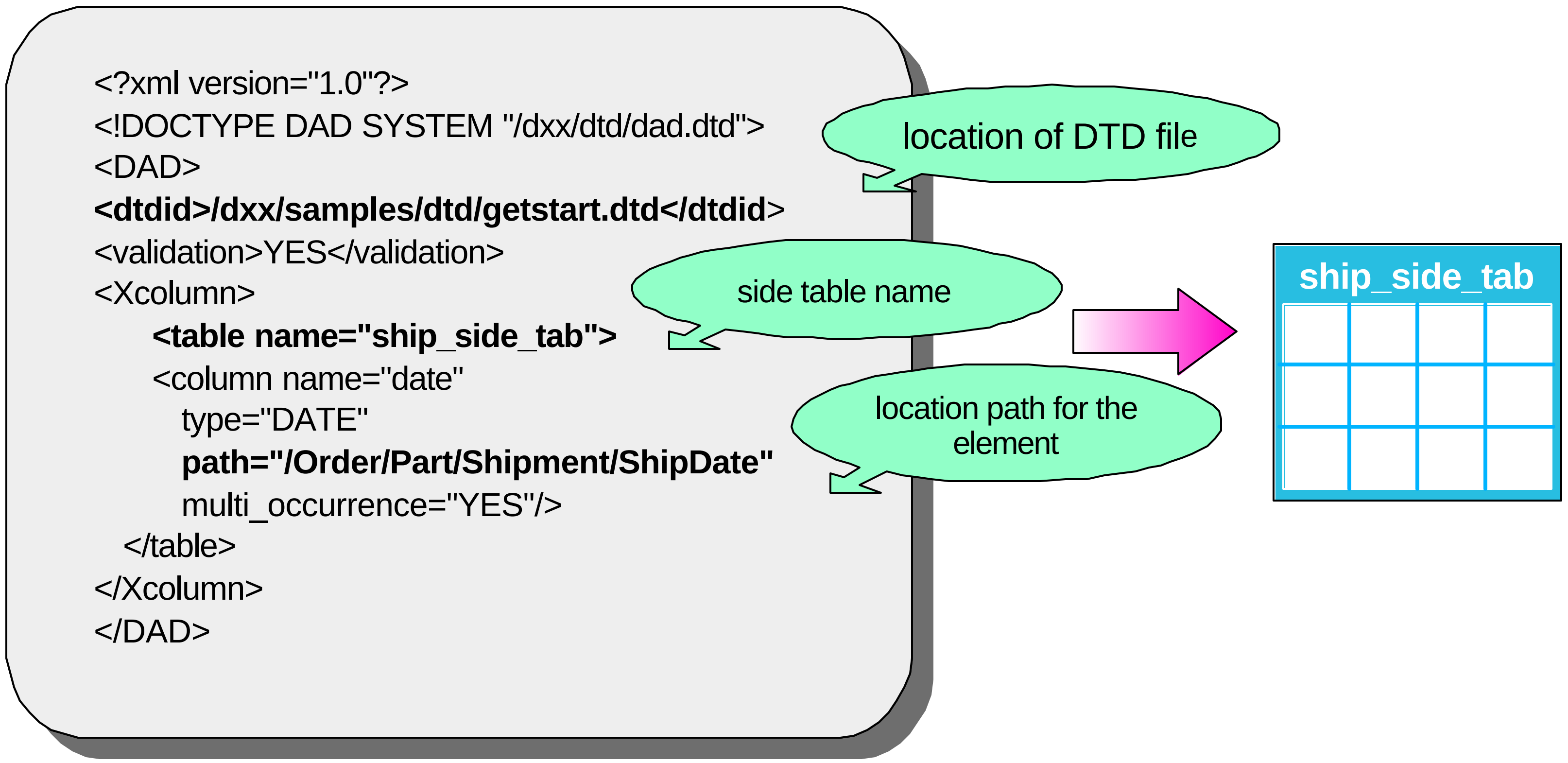
• It creates the triggers on the user table or XML table containing the XML Column to synchronize with side tables.

• It adds a new entry in the db2xml.xml_usage table created during the XML enablement of the database. This new entry keeps the relation between the user table, the XML Column in this table, the DTD ID and the DAD file. This DAD file is stored as a CLOB in the XML_USAGE table.

5. Insert an XML document into the XML Column. At this point, the side tables will be also updated by the triggers created in step 4.

Document Access Definition (DAD)

- XML document itself
  - defines the location of key files such as DTD
  - defines the mapping between XML document and relational tables
- Used for both XML Column and XML Collection



```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "/dxx/dtd/dad.dtd">
<DAD>
<dtdid>/dxx/samples/dtd/getstart.dtd</dtdid>
<validation>YES</validation>
<Xcolumn>
    <table name="ship_side_tab">
    <column name="date"
      type="DATE"
      path="/Order/Part/Shipment/ShipDate"
      multi_occurrence="YES"/>
  </table>
</Xcolumn>
</DAD>
```

Notes

Before creating the DAD file, you should:

• Decide which elements or attributes you often want to search in your XML documents

• For each element or attribute that you want to index in a side table, define:

- The location path to represent it: Use the XPath data model to map XML structure (the element and attribute) to the relational tables (the columns). In our example, we map <ShipDate> element to a date column located in the ship_side_tab table. The location path for the element is **/Order/Part/Shipment/ShipDate.**

- The data type your element or attribute should be converted to. In your XML documents, all your elements content and attributes value are considered character data. But in your side tables, you can use any DB2 data types. In the example, we map the ShipDate into a DATE data type.

- Consider whether they have multiple occurrences or not. This must match the declaration in the DTD validating your XML documents. The multi_occurrence attribute for the <ShipDate> element is set to YES, which means that there can be several shipments for one part.

• For each multiple occurring element or attribute, you need to create a new side table if you want to extract their value for indexing.

• Decide whether you want the validation to occur or not. This decision can be based on the following considerations:

- The validation has a small performance impact

- You may not want to validate XML documents that you know are valid

- The validation by DB2 XML Extender can only occur at the time the XML documents are stored into the XML table and not afterwards

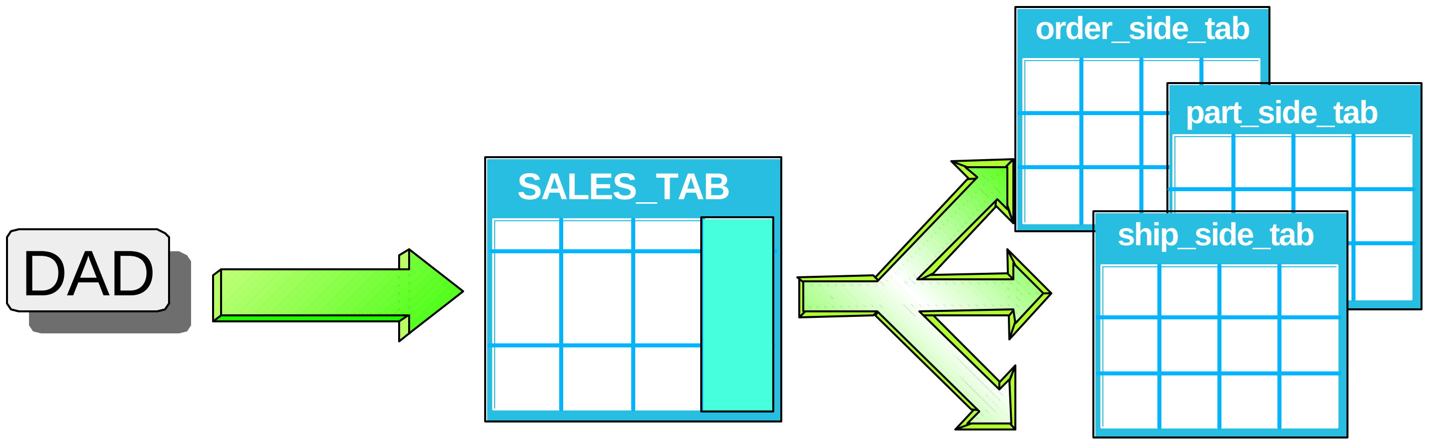
In the example, the validation is set to YES. Note that you need to use capital letters for the content of the <validation> element.

Side tables

Additional tables created by the XML Extender

- based on the DAD specification
- improve performance when searching elements or attributes in an XML column

Creating indexes on side tables further enhances the performance



Notes

As mentioned earlier, side tables are additional tables created by the XML Extender to improve performance when searching elements or attributes in an XML column. The side tables are created at the time when the XML column is enabled. The definition of the side tables is specified in the DAD associated with a given XML column. Note that side tables are not mandatory.

To further enhance the performance for the search requests, you may create indexes over the side tables.

XML Column Storage and Access Means

User Distinct Types (UDTs) provided by the XML Extender:

- XMLFile - external file name
- XMLVarchar - for internal short document
- XMLCLOB - for internal long document

User Defined Functions (UDFs) provided by the XML Extender:

- Storage
  - XMLVarcharFromFile(), XMLClobFromFile(), XMLFileFromVarchar(), XMLFileFromClob
- Retrieval
  - default cast functions varchar(XMLVarChar), clob(XMLClob) varchar(XMLFile) or Content(XMLobj, XMLFile)
- Update
  - default cast functions or storage UDFs
- Extract the content of an element or attribute
  - convert XML data to SQL data types
  - extractVarchar(XMLCol, LocationPath), extractCLOB(XMLCol, LocationPath), ...

Notes

The XML Extender user-defined types (UDTs) are data types that are used for XML columns and XML collections. All the UDTs have the schema (library) name DB2XML. The XML Extender creates UDTs for storing and retrieving XML documents.

The XML Extender provides functions for storing, retrieving, searching, and updating XML documents, and for extracting XML elements or attributes. Use XML user-defined functions (UDFs) for XML columns, but not for XML collections. There are four types of the XML Extender functions:

- storage functions

Use storage functions to insert XML documents into a DB2 database. You can use the default casting functions of a UDT directly in INSERT or SELECT statements. Additionally, the XML Extender provides UDFs to take XML documents from sources other than the UDT base data type and convert them to the specified UDT. For instance, XMLVarcharFromFile() reads an XML document from a server file and returns the document as an XMLVARCHAR type.

- retrieval functions

The XML Extender provides an overloaded function Content(), which is used for retrieval. This overloaded function refers to a set of retrieval functions that have the same name, but behave differently based on where the data is being retrieved. You can also use the default casting functions to convert an XML UDT to the base data type.

- update function

The Update() function modifies the element content or attribute value and returns a copy of an XML document with an updated value that is specified by the location path. The Update() function allows the application programmer to specify the element or attribute that is to be updated.

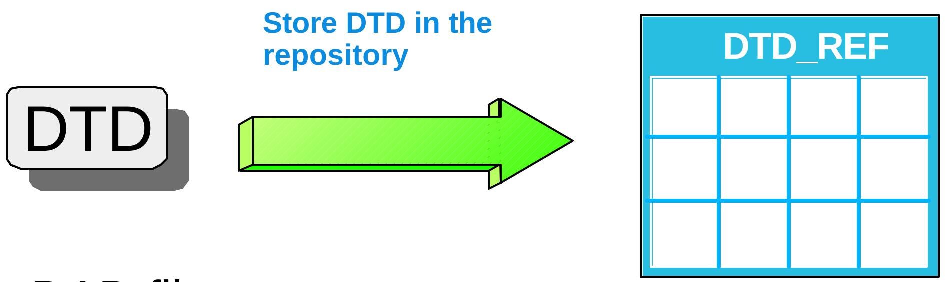
- extracting functions

Extracting functions extract and convert the element content or attribute value from an XML document to the data type that is specified by the function name. The XML Extender provides a set of extracting functions for various SQL data types.

XML column example (1/3)

Storing the DTD in the DTD repository

```
INSERT into db2xml.dtd_ref values('/dxx /samples /dtd /getstart.dtd',
db2xml.XMLClobFromFile('/dxx /samples /dtd /getstart.dtd'),0,'user1',
'user1','user1')
```



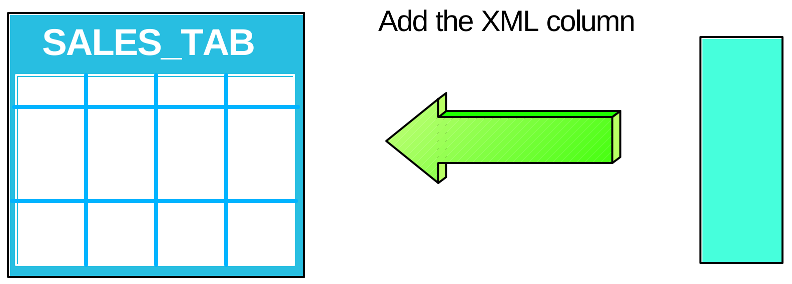
Preparing the DAD file

```
...
<Xcolumn>
<table name="ship_side_tab">
<column name="shipdate"
type="DATE"
path="/Order/Part/Shipment/ShipDate"
multi_occurrence="NO"/>
</table>
</Xcolumn>
...
```

XML Column example (2/3)

Adding the XML column

```
ALTER TABLE SALES_TAB ADD COLUMN ORDER DB2XML.XMLVARCHAR
```



Enabling the XML column (QShell)

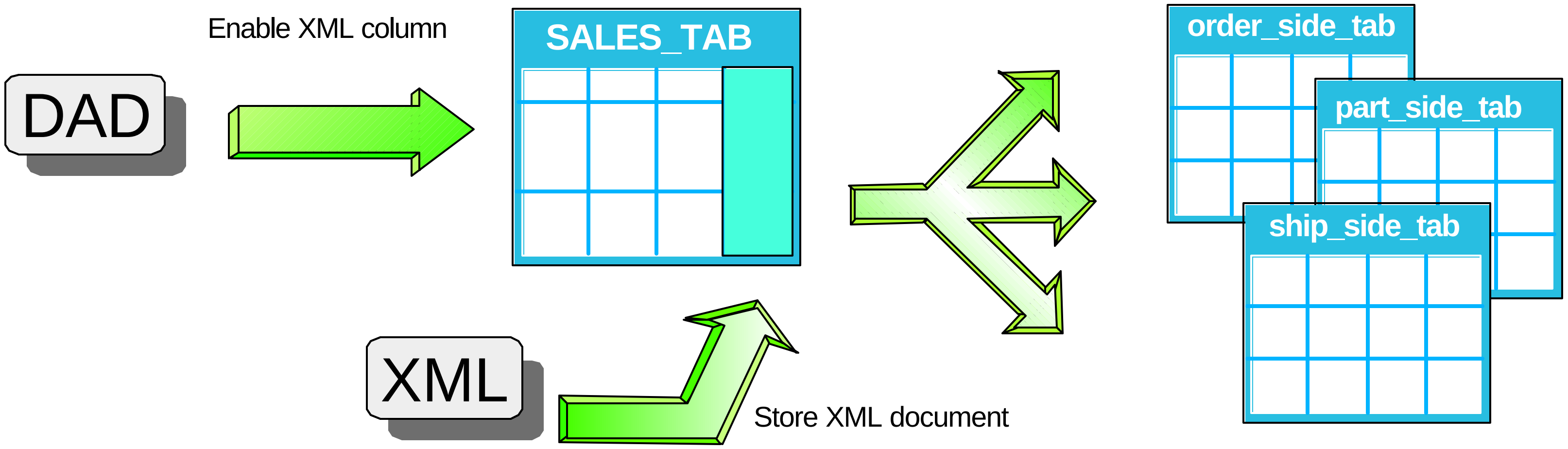
```
dxxadm enable_column PWD2 sales_tab order
/dxxsamples/dad/getstart_xcolumn.dad -v sales_order_view -r invoice_num
```

Note: The column ORDER located in the SALES_DB.SALES_TAB is enabled.
The side table SHIP_SIDE_TABLE is created based on the DAD specification. The default view SALES_ORDER_VIEW is created. The primary key for the side table is INVOICE_NUM.

XML column example (3/3)

Storing the XML document

```
INSERT INTO SALES_TAB (INVOICE_NUM,SALES_PERSON,ORDER)VALUES('123456',
'Joe Doe',db2xml.XMLVarcharFromFile('/dxx /samples/xml/getstart.xml'))
```



Verifying the content of side tables

```
SELECT * FROM SHIP_SIDE_TAB
```

Searching the XML document

```
SELECT DISTINCT SALES_PERSON FROM SALES_TAB S,SHIP_SIDE_TAB P
WHERE SHIPDATE > DATE('2000-01-01') AND
S.INVOICE_NUM=P.INVOICE_NUM
```

XML Collections Scenario

Scenarios suitable for XML Collections

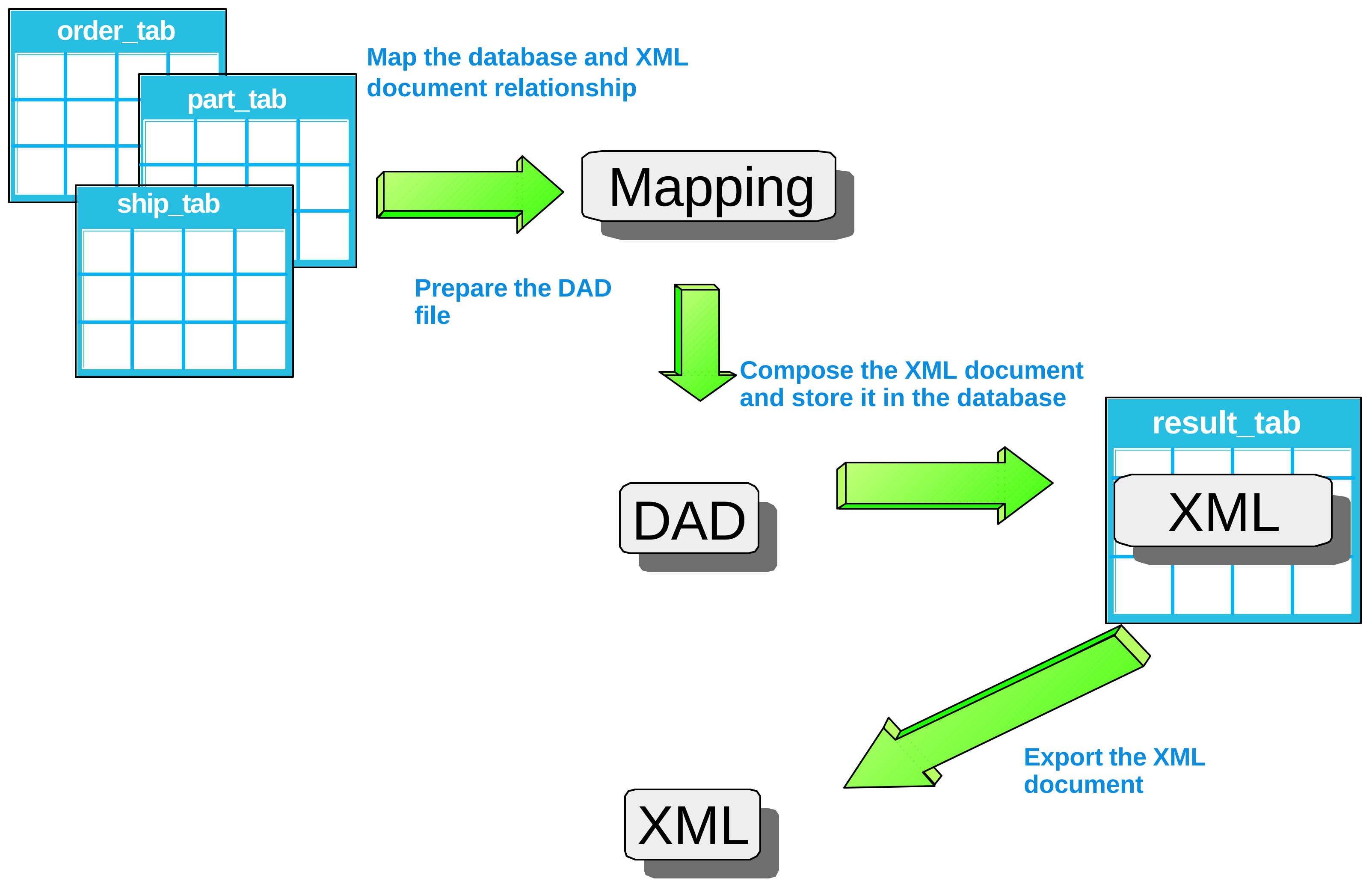
- Your have data in your existing relational tables, and you want to compose XML documents using your existing data based on DTD
- You want to create different view of your relational data using different mapping scheme
- The XML documents come from other source and you want to store pure data
- A small part of your XML documents need to be updated often, and update performance is critical
- You like to store the data of entire incoming XML documents but often only want to retrieve a subset of them;
- Your XML documents are large in size, which exceed 2 GB and you must decompose them

Notes

Using the XML Collections method allows you to map XML document structures to DB2 tables, so that you can compose XML documents from existing DB2 data or decompose XML documents into DB2 tables.

This foil lists scenarios suitable for the XML Collection method.

XML collection: composing a document


order_tab
part_tab
ship_tab
Map the database and XML document relationship
Mapping
Prepare the DAD file
Compose the XML document and store it in the database
DAD
result_tab
XML
Export the XML document
XML

Notes

This foil illustrates the steps required to compose a XML document from the underlying database tables.

You can compose XML documents using stored procedures. To use these stored procedures, you must create a DAD file, which specifies the mapping between the XML document and the DB2 table structure.

1.Map the structure of the XML document to the relational tables that contain the contents of the element and attribute values (see next foil).

2. Create the appropriate DAD file. For XML collections, the DAD file maps the structure of the XML document to the
DB2 tables from which you either compose the document, or to where you decompose the document.

For example, if you have an element called <ShipDate> in your XML document, you might need to map <ShipDate> to a column called Ship_Date. You define the relationship between the XML data and the relational data in the DAD.

3. To compose the XML document you call the dxxGenXML() stored procedure. The stored procedure constructs XML documents using data that is stored in the XML collection tables that are specified by the <Xcollection> in the DAD file and inserts each XML document as a row into the result table. The appropriate DAD is passed to the stored procedure as one of the input parameters.

4. Now you can retrieve the XML document for the result table and store in the IFS or send it over the net to your business partner or customer.

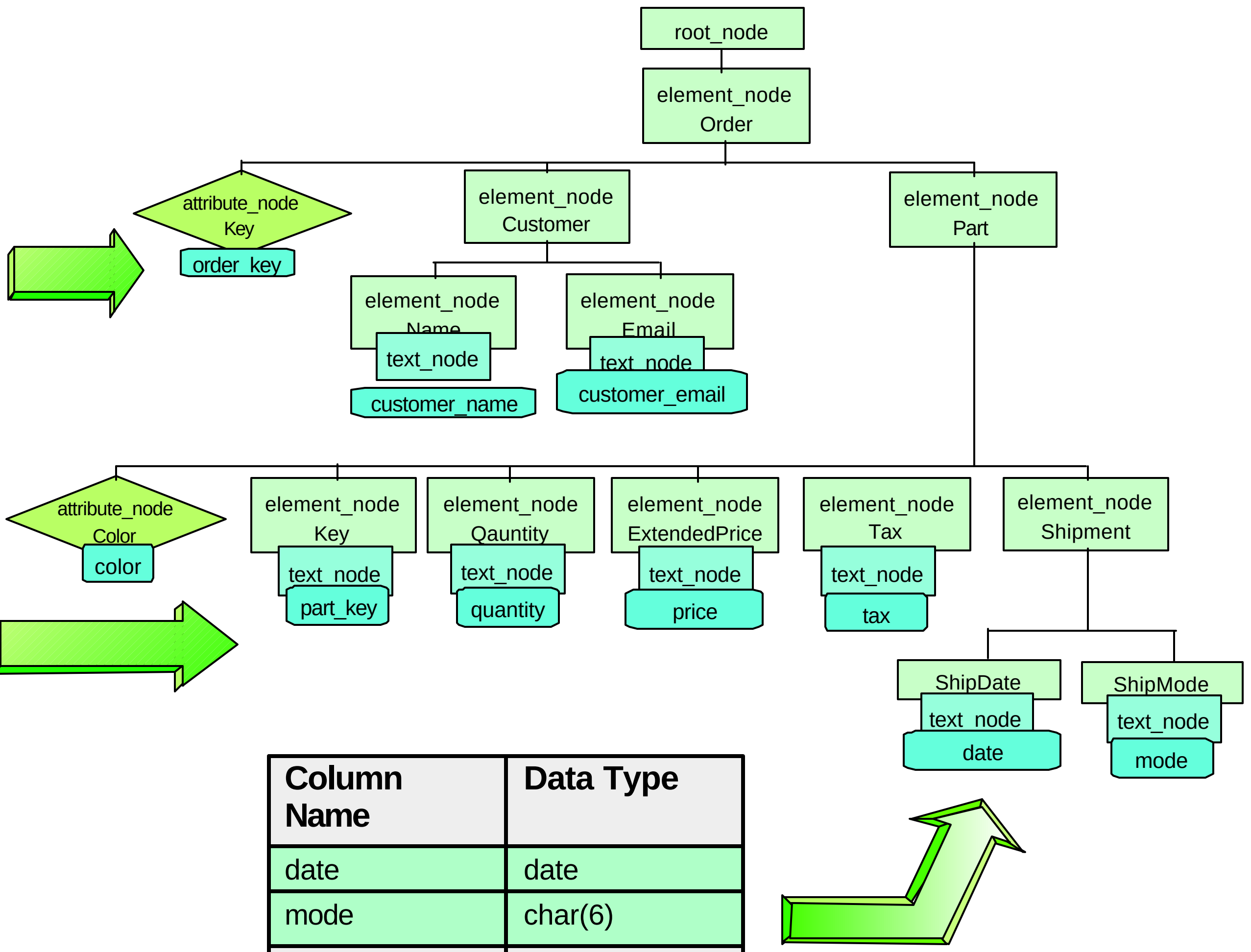
Relational tables to XML mapping

- A column is mapped to an element_node or an attribute_node

Column Name	Data Type
order_key	integer
customer	varchar(16)
customer_name	varchar(16)
customer_email	varchar(16)

Column Name	Data Type
part_key	integer
color	char(6)
quantity	integer
price	decimal(10,2)
tax	decimal(5,2)
order_key	integer

Column Name	Data Type
date	date
mode	char(6)
comment	varchar(128)
part_key	integer



Notes

If you are using an XML collection, you must select a mapping scheme that defines how XML data is represented in a relational database. Because XML collections must match a hierarchical structure that is used in XML documents with a relational structure, you should understand how the two structures compare. This foil shows how the hierarchical structure can be mapped to relational table columns. A shaded column from the database table is directly mapped to a particular element in the composed XML document. For instance, the CUSTOMER_NAME columns will be mapped to the <Name> element. Note that in this hierarchy, <Name> is a child of <Customer>.

Mapping methods

SQL mapping

- uses SQL statement element to specify the SQL query to retrieve DB2 data
- can be used only for composing XML documents

RDB_node mapping

- uses an XML Extender-unique element called RDB_node
- used to specify tables, columns, conditions, and order for XML data
- can be used for both composing and decomposing

RDB_node example:

```
<RDB_node>
 <table name="order_tab"/>
 <table name="part_tab"/>
 <table name="ship_tab"/>
 <condition>order_tab.order_key=part_tab.order_key
  AND part_tab.part_key=ship_tab.part_key </condition>
 </RDB_node>
```

Notes

The mapping scheme is specified in the <Xcollection> element in the DAD file. The XML Extender provides two types of mapping schemes: SQL mapping and Relational Database (RDB_node) mapping. Both methods use the XPath model to define the hierarchy of the XML document.

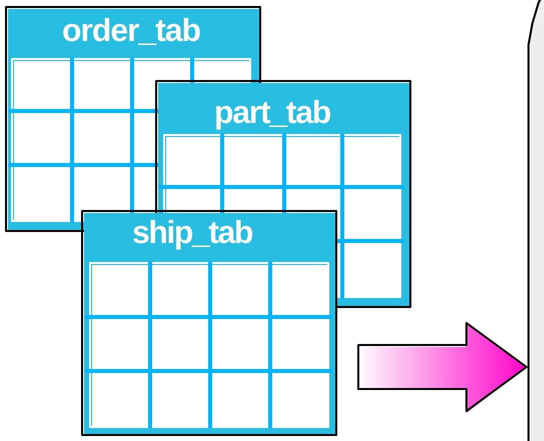
SQL mapping

Allows direct mapping from relational data to XML documents through a single SQL statement and the XPath data model. SQL mapping is used for composition; it is not used for decomposition. SQL mapping is defined with the SQL_stmt element in the DAD file. The content of the SQL_stmt is a valid SQL statement. The SQL_stmt maps the columns in the SELECT clause to XML elements or attributes that are used in the XML document. When defined for composing XML documents, the column names in the SQL statement's SELECT clause are used to define the value of an attribute_node or a content of text_node. The FROM clause defines the tables containing the data; the WHERE clause specifies the join and search condition. The SQL mapping gives DB2 users the power to map the data using SQL. When using SQL mapping, you must be able to join all tables in one SELECT statement to form a query. If one SQL statement is not sufficient, consider using RDB_node mapping. To tie all tables together, the primary key and foreign key relationship is recommended among these tables.

RDB_node mapping

Defines the location of the content of an XML element or the value of an XML attribute so that the XML Extender can determine where to store or retrieve the XML data. This method uses the XML Extender-provided RDB_node, which contains one or more node definitions for tables, optional columns, and optional conditions. The tables and columns are used to define how the XML data is to be stored in the database. The condition specifies the criteria for selecting XML data or the way to join the XML collection tables.

DAD with SQL mapping



SELECT statement for SQL mapping

```
....
<DAD>
<validation>NO</validation>
<Xcollection>
<SQL_stmt>select o.order_key, customer_name, customer_email, p.part_key,
color, quantity, price, tax, ship_id, date, mode from order_tab o, part_tab p,
(select rrn(ship_tab) as ship_id, date, mode, part_key from ship_tab) s where
o.order_key = 1 and p.price > 20000 and p.order_key = o.order_key and
s.part_key = p.part_key ORDER BY order_key, part_key, ship_id</SQL_stmt>
<prolog>?xml version="1.0"?</prolog>
<doctype>!DOCTYPE Order SYSTEM "/dxx/samples/dtd/getstart.dtd"</doctype>
<root_node>
<element_node name="Order">
 <attribute_node name="key">
  <column name="order_key"/>
 </attribute_node>
  .....
</element_node>
</root_node>
</Xcollection>
</DAD>
```

DB2 column mapped to an element_node or an attribute node

Notes

Follow these guidelines to create the SQL statement for mapping the relational data to the XML document:

- Columns are specified in top-down order, by the hierarchy of the XML document structure.
- The columns for an entity are grouped together, and each group has an object ID column: order_key, part_key.
- The object ID column is the first column in each group. For example, order_key precedes the columns related to the order element and part_key precedes columns for the part element.
- The SHIP_TAB table does not have a single key column, and therefore, the rrn() DB2 built-in function is used to generate the ship_id column.
- The object ID columns are then listed in top-down order in an ORDER BY statements. The columns in ORDER BY should not be qualified by any schema and table name and should match the column names in the SELECT clause.

Note also that the columns in ORDER BY clause should not be qualified and they should match the column names in the SELECT clause. This iSeries-specific restriction for the ORDER BY clause will be lifted in a future release of DB2 UDB for iSeries.

XML collection example

Steps to compose XML from DB2 data with SQL mapping

1. Prepare DAD

Specify SQL for SQL mapping

```
<SQL_stmt>select o.order_key, customer_name, customer_email, p.part_key,
color, quantity, price, tax, ship_id, date, mode
from order_tab o, part_tab p, (select
rrn(ship_tab) as ship_id, date, mode, part_key from ship_tab) s
where o.order_key = 1 and p.price > 20000
and p.order_key = o.order_key
and s.part_key =p.part_key
ORDER BY order_key, part_key, ship_id</SQL_stmt>
...
```

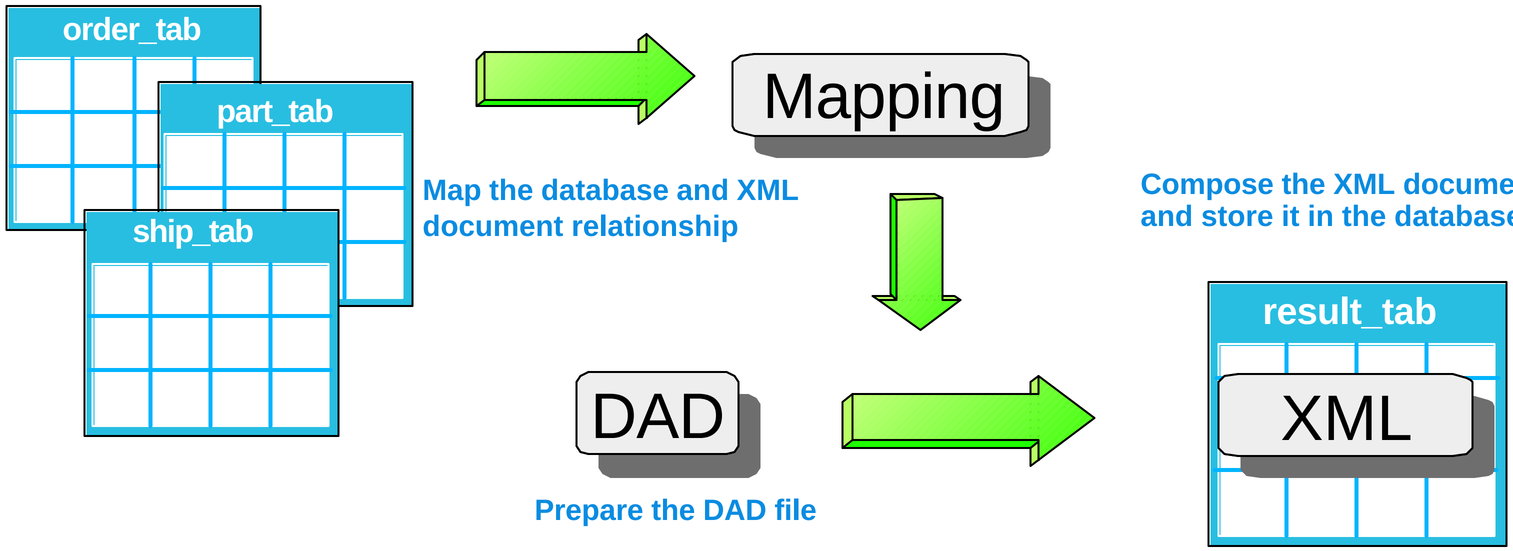
Define <element_node> tag for each element in XML document

```
<element_node name="ShipMode">
<text_node>
<column name="mode"/>
</text_node>
</element_node>
...
```

XML collection example cont.

2. Compose the XML document using the dxxGenXML stored procedure

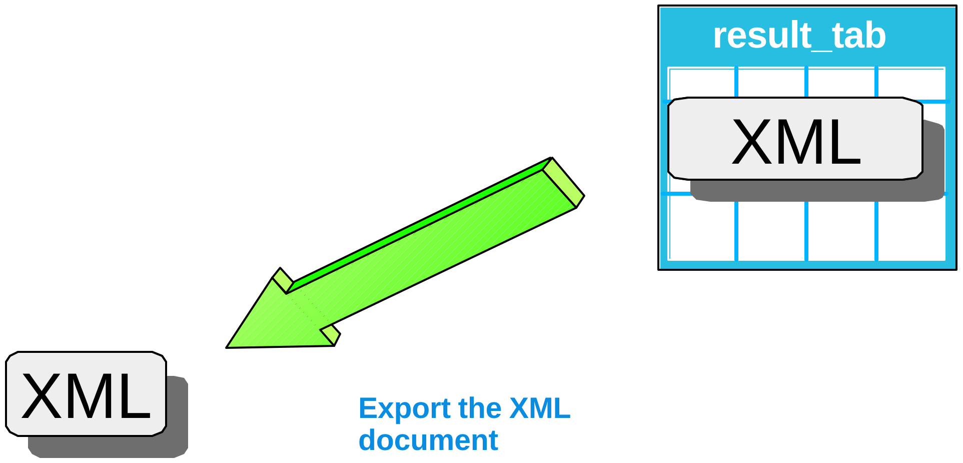
```
dxxGenXML(
CLOB(100K) DAD,                    /*CLOB containing the DAD file */
char(UDB_SIZE ) resultTabName,     /*result table name, contains one column */
integer overrideType               /*flag to indicate the type of the override */
varchar(1024)override,             /*overrides the condition in the DAD file */
integer maxRows,                   /*maximum number of rows in the result table */
integer numRows,                   /*number of generated rows */
long returnCode,                   /*return code */
varchar(1024)returnMsg)            /*message text returned in case of error */
. . .
```


order_tab
part_tab
ship_tab
Mapping
Map the database and XML document relationship
Compose the XML document and store it in the database
result_tab
XML
DAD
Prepare the DAD file

XML Collection example (3/3)

3. Export the XML document

```
SELECT db2xml.Content(db2xml.xmlvarchar(doc),
'/dxx /samples /cmd /getstart.xml')FROM RESULT_TAB
```



DB2 UDB for iSeries Text Extenders

Putting DB2's power to Support Text Mining

Looking for a fast, versatile, and intelligent full text and mining tool?

Answer: DB2 UDB Text Extender

It's fast

- can search through thousands of documents at high speed.

It's versatile

- can access text documents including XML documents (e.g. DB2 XML Extenders) and documents in a variety of languages

It's intelligent

- utilizes DB2 Universal Database's built-in support for user-defined types and user-defined functions and also exploits DB2 UDB's support for large objects

Notes

DB2 Text Extender adds the power of full-text retrieval to SQL queries by making use of features available in DB2 UDB for iSeries that let you store text documents in databases.

DB2 Text Extender offers DB2 UDB for iSeires users and application programmers a fast, versatile, and intelligent method of searching through such text documents. DB2 Text Extender's strength lies in its ability to search through many thousands of large text documents, finding not only what you directly ask for, but also word variations and synonyms.

Text Extender Overview

Powerful linguistic text searching technology

- Search for documents that contain a specific word, such as "Internet," or phrase, such as "large object"
- Make a "fuzzy" search to find words that are spelled in a similar way to the search term
- Search for documents that contain synonyms of a word or phrase.
  - search for documents that contain the word "book" and also find documents that contain "article," "volume," "manual", and other synonyms
- Search for documents that contain words in any sequence, or words in the same sentence or paragraph
  - search for the word "compress," in the same paragraph as "encryption"
- Perform wild card searches using word and character masking
- Search for documents by variations of a word, such as its plural form or the word in a different tense
  - For example, search for documents that contain the word "drive" and also find documents that contain "driving," "drove," and "driven."

Indexing

Text Extender indexes are created to enable intelligent text search capabilities

- are non-relational index structures stored in IFS stream files
- store list of significant words with a list of documents that contain them

Indexing process is asynchronous

- Search is available during indexing

Index update can be triggered

- Immediately
- Periodically

IFS file support

- Files stored in the IFS can also be indexed

Notes

There are three types of indexes that allow you for a fast text search using DB2 Text Extender. The three types of indexes available with DB2 Text Extender are linguistic, precise, and NGram.

The type of index you choose for a text column determines how you can search and what you can find. You can create more than one index (of different types) on the same text column if you want to benefit from the advantages provided by more than one index type.

Generally, a text index contains a list of the significant terms contained in your documents, and a reference to where the documents are located. All the insignificant terms such as “a” and “the”, referred to as “STOP-WORDS”,are ignored by the indexing process.

The Text Extender indexes are stored in IFS and their structure and features are completely different from the regular database indexes.

The indexing is a two-step process:

1. Record all the documents that need to be indexed in a log file (or more precisely, a log table). This step is automatically executed for you by the insert, update, and delete triggers created by DB2 Text Extender for each
text column enabled for text search.

2. Periodically, process all the documents recorded in the log and index their significant terms to keep the content of the index synchronized with the content of the database. The period between two such processes is
determined by a configuration setting.

You are not restricted to searching only in text documents stored in DB2 UDB for iSeries, you can also search in text documents stored in files.

Structured document support

Structured document support introduces the concept of sections within a document.

A section is a part of the document that is identified by an HTML-like tag.

The sections and the path to the section are recognized during indexing and stored in the index

A model definition file is used to identify the tags that are processed as sections.

Search on sections, list of sections and nested sections

Notes

Section support allows you to index and search specific sections in a structured document, for example, in the title, author, or description. The documents can be in XML or HTML format or flat-file documents with HTML-like tags.

The sections are divided in two categories:

• The plain-text sections: These have no type.

• The attribute sections: These have a declared type. Using these declared types, it is possible to search for documents whose given attribute section is within a specified range.

Structured document support

Supported formats

- ASCII with tagged sections
- HTML
- XML (with or without DTD)
  - nested sections

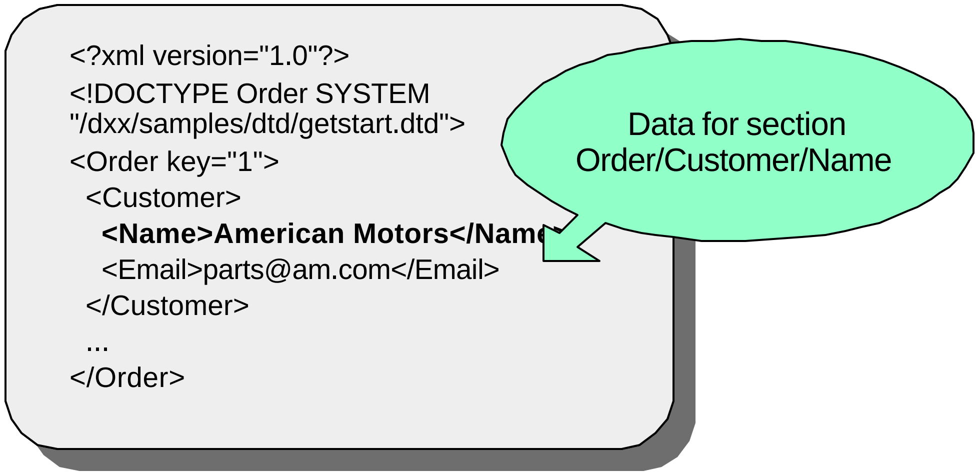
Model definition file

```
[MODELS ]
modelname=Order
;left side =section name identifier
;right side =section name tag
[Order ]
Order =/Order
Order/Customer/Name =/Order/Customer/Name
Order/Customer/Email =/Order/Customer/Email
Order/Part/@color =/Order/Part/@color
```

Notes

You define the markup tags and their corresponding section names in a document model. All the document models for the server instance are listed in the document models file. The default document models file is DESMODEL.INI, which is created automatically with the server instance. Each document model in the document models file is made up of two components: its name and its description. When describing an XML document, the document model must have the same name and case as the root element. For each element or attribute you want to define and use as a section, its complete hierarchy must be included in the model description.

XML document search example



```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM
"/dxx/samples/dtd/getstart.dtd">
<Order key="1">
  <Customer>
    <Name>American Motors</Name>
    <Email>parts@am.com</Email>
  </Customer>
  ...
</Order>
```

Enable the XML column for the Text Extender search

```
CALL PGM(QDB2TX/DB2TX) PARM('enable text column order xvarchar function
db2xml.varchartovarchar handle varcharhandle ccsid 850 language us_english
format xml indextype precise indexproperty sections_enabled')
```

Search using the section

```
select xvarchar from order where db2tx.contains(varcharhandle,
'model Order section(Order/Customer/Name)"Motors"')=1
```

Additional Information

- **DB2 UDB for iSeries home page** - **http://www.iseries.ibm.com/db2**
- **DB2 UDB Extenders Site: http://www.ibm.com/software/data/db2/extenders/**
- **Newsgroups**
  - USENET: comp.sys.ibm.as400.misc, comp.databases.ibm-db2
  - iSeries Network (NEWS/400 Magazine) SQL & DB2 Forum - http://www.iseriesnetwork.com/Forums/main.cfm?CFApp=59
- **Education Resources - Classroom & Online**
  - http://www.iseries.ibm.com/db2/db2educ_m.htm
  - http://www.iseries.ibm.com/developer/education/ibo/index.html
- **DB2 UDB for iSeries Publications**
  - Online Manuals: http://www.iseries.ibm.com/db2/books.htm
  - Porting Help: http://www.iseries.ibm.com/developer/db2/porting.html
  - DB2 UDB for iSeries Redbooks (http://ibm.com/redbooks)
    - Stored Procedures & Triggers on DB2 UDB for iSeries (SG24-6503)
    - DB2 UDB for AS/400 Object Relational Support (SG24-5409)
    - SQL Query Engine Redpiece http://publib-b.boulder.ibm.com/Redbooks.nsf/RedpieceAbstracts/sg2456598.html
    - Integrating XML with DB2 XML Extender and DB2 Text Extender (SG24-6130)

Upcoming Residencies

IS-3101 - SQL Query Enhancements on DB2 UDB for iSeries - Phase 2

This Rochester residency begins 21 Apr 2003, ends 30 May 2003 (6 weeks), and requires 3 residents.

IS-3102 - V5R1 and V5R2 DB2 UDB for iSeries Functionality Update

This Rochester residency begins 7 July 2003, ends 15 Aug 2003 (6 weeks), and requires 4 residents.

Trademarks and Disclaimers