University of Rostock

Department of Computer Science

Full Text Search in XML Documents

Cornelia Laudien
born 13.04.1977 in Rostock
Tutors: Andreas Heuer, Albert Maier, Denny Priebe

Abstract

The goal of this paper is to show how XML structure information can be used for full text search in XML documents. Existing products for full text search are investigated regarding their support of XML. The main aspect of this investigation is how the search scope of queries is specified and narrowed by taking advantage of the XML format. Considering the results of this investigation, a suggestion how to realize XML support for IBM's text search engine GTR is developed.

Contents

1 Introduction

Full text search functionality has become an integrated part of modern database management systems. With the fast growing XML technology and the rapidly increasing number of XML documents, the importance of providing extended retrieval functionality for these structured documents is rising every day.
This means, it is a new feature of full text search functionality to use XML structure information in order to narrow the search scope. The search scope, for example, can be narrowed to specific sections of a document. In order to do so, text retrieval products should be able to separate structure and content of documents. They have to take advantage of the additional information about structure provided by documents of XML format.
The main part of this paper will investigate how market leading companies like IBM and Oracle have realized this new feature.
At the beginning of this paper a brief overview of XML technology is given. Major components are explained with an example, which will also be part of further chapters.
Then criteria are introduced, which are used to analyze and compare the functionality of existing products that already support full text search in XML documents. This investigation is performed particularly with regard to requirements of XML documents and will not analyze general full text search functionality.
Furthermore, one of IBM's text search engines will be presented. It is called GTR, and its functionality will be investigated regarding its suitability for XML support. The intention is to use GTR's existing internal functionality and extend it for XML support. Alternative search engines and the possibility of combining ORDBMS with independent document management systems providing sophisticated retrieval techniques (see [Nit00]) are not subject of this paper.
At the end the results of the comparison of existing products and the possibilities of XML technology will be used to give a vision of future products with improved text retrieval functionality for XML documents.

2 XML

This chapter focuses on XML technology that is already used or will be used in the near future. The first part of this chapter gives an overview of XML 1.0. Although the major components of XML are mentioned, the intention of this chapter is not to be an XML tutorial. An XML document example will be presented to which further chapter will refer to. The second part of this chapter introduces a language called XPath for addressing parts of an XML document. In contrast to other languages, e.g. XQL (XML Query Language), XPath is a standard maintained by the World Wide Web Consortium that uses intuitive directory notation.
The last part of this chapter concentrates on XML Schema, a language to describe a schema for a class of XML documents. Unlike RDF (Resource Description Framework) schemas, which provide information about the interpretation of the statements given in an RDF data model, XML Schema gives specific constraints on the structure of an XML document.

2.1 XML Overview

XML, the Extensible Markup Language, is an independent standard which is maintained by the World Wide Web Consortium's XML Working Group and endorsed by software industry market leaders. It is an extensible markup language for structured document definition and exchange. XML is a meta language with a simple, intuitive syntax and a formal and verifiable design. This makes it extremly flexible for every conceivable application.
The major components of XML will be described in the following.

Physical and logical structure: XML documents are comprised of reusable storage units called **entities**, which can contain any type of named data, e.g. character or binary. Entities can be parsed and unparsed, latter are not read by XML parser. There are two main entity categories general and parameter entities. General entities are used within the document content itself. Parameter entities are parsed entities that appear only in DTDs (see below).
Besides the physical structure represented by entities, XML has a logical structure that allows to subdivide documents into sections and subsections, so called **elements** and subelements. Each element has a type, identified by a name, and may have a set of **attributes**. Attributes are in contrast to subelements atomic qualifier of elements. The boundaries of an element are delimited by tags, e.g. <paper>content</paper> or in case of an empty element <empty_paper/>. Users have the ability to define new tags to suit their specific requirements.

Well-formedness and validity: **DTDs**, document type declarations, represent document schemas which specify the valid tags and the valid structure for document classes. That means, an XML document is considered to be valid, if it has an associated DTD and if the document complies with the DTD's constraints. But an XML document can be delivered without a DTD, it doesn't need to be valid as long as it is well-formed. The well-formedness of an XML document requires that the document contains at least one element, but no element can partially overlap any other element, that means elements must be nested properly. Moreover, there has to be exactly one root element which cannot be contained in other elements.
Non-validating processors process the document entity and any internal DTD subset for well-formedness. Validating processors must process the entire DTD and all external parsed entities referenced in the document and must report violations of all validity constrains of the XML Specification.

The following *paper* example presents an XML document with its DTD.

```
<?xml version="1.0" ?>
<!DOCTYPE paper [
<!-- BEGIN DOCUMENT TYPE DECLARATION -->
<!ELEMENT paper      (meta,body)>
<!ATTLIST paper      language CDATA #IMPLIED>
<!ELEMENT meta       (title,author+,year)>
<!ELEMENT title      (#PCDATA)>
<!ELEMENT author     (#PCDATA)>
<!ELEMENT year       (#PCDATA)>
<!ELEMENT body       (chapter+,references?)>
<!ELEMENT chapter    (paragraph*)>
<!ATTLIST chapter    title CDATA #IMPLIED>
<!ELEMENT references (#PCDATA)>
<!ELEMENT paragraph  (#PCDATA)>
<!-- END DOCUMENT TYPE DECLARATION -->
]>
<!-- BEGIN DOCUMENT -->
<paper language="en">
  <meta>
    <title>Full Text Search in XML Documents</title>
    <author>Cornelia Laudien</author>
    <year>2000</year>
  </meta>
  <body>
    <chapter title="Introduction">introduction</chapter>
    <chapter title="XML">
      <paragraph>XML Overview</paragraph>
```

```
      <paragraph>XPath</paragraph>
      <paragraph>XMLSchema</paragraph>
    </chapter>
    <chapter title="Existing Products">
      <paragraph>
       IBM/DB2 XML Extender and Text Extender
      </paragraph>
      <paragraph>Oracle8i InterMedia Text</paragraph>
    </chapter>
    <chapter title="GTR">
      <paragraph>Search Types</paragraph>
      <paragraph>Indexing</paragraph>
    </chapter>
    <references>http://www.w3.org</references>
  </body>
</paper>
<!-- END DOCUMENT -->
```

This example shows an instance of the *paper* document class.
The valid structure of the document class *paper* is specified in the first part of the example as internal DTD subset. It defines that each paper element consists of exactly one element called *meta* and exactly one called *body*. The meta element, however, consists of exactly one element called *title* and one called *year*, but it allows one or more authors. This is represented by "+" after *author*. "?" in case of *references* signifies that the reference section can appear once within the document but not necessarily. "*" next to *paragraph* means, that a paragraph section can occur several times, but it doesn't have to. Besides that, no default value is set for the *language* and *title* attributes recognizable by the IMPLIED keyword. Furthermore, the paper element is refered to as the root element, because it is not contained in other elements.
The instance itself consists of meta data and a body. The meta data encompass the title of the paper, its author and the year it was written. The body contains some chapter and a reference part. Moreover, some of the chapter consist of several paragraphs. This document is valid, because it complies with its DTD's constraints.
Text within this kind of tags "$<! - -text - - >$" represents a comment. Empty elements and elements containing alternative subelements do not occur within this example. However, they should be mentioned at this point, because they are of interest for further investigation. Latter could look like this:

```
<!ELEMENT author (#PCDATA | ( first_name, last_name))>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT last_name (#PCDATA)>
```

In this example *author* is an element of alternatives, because it can contain PCDATA or the two subelements *first_name* and *last_name*. That means, documents containing the author's name simply as a string are valid as well as documents that distinguish between first and last name. Further information, for example regarding different content types, can be found in [W3C98].

2.2 XPath

The XML Path Language (XPath) is a standard maintained by the World Wide Web Consortium. Besides that, XPath is a powerful language for referencing information within XML documents, and therefore it is of significant interest for XML supporting retrieval systems. In order to address parts of an XML document, XPath models the document as a tree of element **nodes**, attribute nodes and text nodes. Moreover, it uses a **directory notation** to perform queries. Querying XML documents by XPath means selecting nodes to determine which elements within a document satisfy a given set of criteria.

Location Path: The most important construct of XPath is the location path. It selects a set of nodes relative to the **context node** which represents the current position within the document tree. The following list describes the abbreviated location path syntax according to the example of chapter 2.1. This list reflects the most interesting features without claiming completeness. A comprehensive description can be found in [W3C99].

1. **/** Selects the document root.
2. **/paper/body/chapter** Selects all *chapter* element children of the *body* element child of *paper*. That means, all four chapters are selected.
3. **/paper/body/chapter[1]/paragraph** Selects all *paragraph* element children of the first *chapter* of the *body* element child of *paper*. The result of this query against the example document of chapter 2.1 would be empty, because the first chapter does not contain any paragraph elements.
4. **/paper/body/chapter[@title="XML"]** Selects all *chapter* of the *body* element child of *paper* that have a *title* attribute with value "XML". In this case, the second chapter will be returned as result.
5. **chapter//paragraph** Selects all *paragraph* element decendants of the *chapter* element children of the context node.

6. **paragraph** Selects the *paragraph* element children of the context node. If the context node represents a chapter element, then all paragraphs will be returned otherwise the result will be empty, because only chapter elements contain paragraph elements.

7. **.** Selects the context node.

8. **../@*** Selects all attributes of the context node's parent. If the context node is a paragraph, then the title of the chapter would be selected.

2.3 XMLSchema

XML Schema is a definition language that can be used to formally describe a schema. The purpose of a schema is to define a class of XML documents, and the term "instance document" is used to describe an XML document that conforms to a particular schema. Beside that, XML Schema is a working draft of the World Wide Web Consortium.
Unlike DTDs, the mechanism supplied by XML 1.0 for declaring constraints on the use of markup, XML Schema uses XML syntax to describe a schema. This is convenient since it does not require to learn a completely new syntax just to describe the grammar. In addition, XML Schema offers a number of other significant advantages over DTDs. Some of them will be described in the following using the *paper* document class of chapter 2.1 as an example.

```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<xsd:element name="paper" type="PaperType"/>

<xsd:complexType name="PaperType">
<xsd:element name="meta" type="MetaType"/>
<xsd:element name="body" type="BodyType"/>
<xsd:attribute name="language" type="xsd:language"/>
</xsd:complexType>

<xsd:complexType name="MetaType">
<xsd:element name="title" type="xsd:string"/>
<xsd:element name="author" type="xsd:string"
     minOccurs="1" maxOccurs="3"/>
<xsd:element name="year" type="xsd:positiveInteger"/>
</xsd:complexType>
```

```
<xsd:complexType name="BodyType">
<xsd:element name="chapter" minOccurs="1"
     maxOccurs="unbounded"/>
     <xsd:complexType>
     <xsd:element name="paragraph" type="xsd:string"
          minOccurs="0" maxOccurs="unbounded"/>
     <xsd:attribute name="title" type="xsd:string"/>
     </xsd:complexType>
<xsd:element name="references" type="xsd:string"
     minOccurs="0" maxOccurs="1"/>
</xsd:complexType>

</xsd:schema>
```

Types: All elements and attributes of the *paper* doctype of chapter 2.1, e.g. paper and title, can be found within this schema. However, unlike a DTD, XML Schema allows to specify a data type for elements and attributes. XML Schema provides **simple and complex types**. Simple types built-in to XML Schema shown in the example are string, language and positiveInteger. Attributes can only be of simple type. ENTITY, NMTOKEN, ID and IDREF are also provided by XML Schema, but they can only be used in attributes to ensure compatibility with XML 1.0 DTDs. Elements that contain subelements or carry attributes are said to be of complex types, whereas elements that contain numbers, strings, dates .. ,but do not contain any subelements are said to be of simple types. The paper, meta, chapter and body elements are of complex type, because they contain subelements and attributes. Furthermore, simple and complex types can be named or unnamed. The paper element is of a named type called PaperType, but the type of the chapter element is an unnamed respectively **anonymous type**. Anonymous types are convenient especially if many types have to be defined that are referenced only once and contain very few constraints. A type defined as anonymous type saves the overhead of having to be named and explicitly referenced. New types can be derived from existing types by restriction or extension. For example, there could be a new type derived by restriction of the positiveInteger type called YearType, that allows values equal to 1990 or bigger.

```
<xsd:simpleType name="YearType"
     base="xsd:positiveInteger">
<xsd:minInclusive value="1990"/>
</xsd:simpleType>
```

Occurrences of elements: The definitions of complex types in the *paper* schema all declare sequences of elements that must appear in the instance document. The occurrences of individual elements declared in the so-called content models of these types may be optional, as indicated by a 0 value for the attribute **minOccurs** or otherwise constrained depending upon the values of minOccurs and **maxOccurs**. For example, the references element is optional. Furthermore, the number of authors could be limited to 3, which is not possible to realize with DTDs.

Namespace: A schema can be viewed as a collection of type definitions and element declarations whose names belong to a particular namespace. Each of the elements in the *paper* schema has a prefix xsd: which is associated with the XML Schema namespace through the declaration, xmlns:xsd="http://www.w3.org/1999/XMLSchema", that appears in the schema element. The prefix xsd: is used by convention to denote the XML Schema namespace. The same prefix, and hence the same association, also appears on the same of built-in simple types, e.g. xsd:string. The purpose of the association is to identify the elements and simple types as belonging to the vocabulary of the XML Schema language rather than the vocabulary of the schema author.

Within this XML chapter, major components of XML have been introduced and it has been shown that XML is an intuitive and flexible language. In the next chapter, the investigation of how market-leading companies provide full text search within XML documents will be of interest.

3 Existing Products for Text Search and Their Support of XML Documents

Today's database technology vendors provide different products for full text search with different support of XML documents. The object-relational DBMSs of IBM and Informix have already been analyzed in [Por99] regarding unstructured and structured document support. Since the publication of this master thesis new releases have been delivered, and the new functionalities especially concerning XML support are of interest. However, the requirements of XML document's on retrieval systems discussed in [Por99] will be the base to evaluate the new functionalities of IBM's, Oracle's, Informix and Microsoft's object-relational DBMSs. For this reason, the criteria will be listed again briefly at this point. The criteria do not demand the complete functionality of an XML query language like XQL. To understand the following part, knowledge about the logical structure of XML documents is required; a brief description gives the chapter 2.1 XML Overview.
There are **criteria for update operations and search criteria**. An XML supporting text retrieval system:

- Should be able to insert and update XML documents with and without a DTD.
- Should be able to extract XML data, that means, it should have the ability to extract sections as well as entire documents.
- Has to update the text search index after insert, update and delete operations.
- Has to be able to separate structure and content, in order to take advantage of structured documents.
- Should have the ability to search certain sections and attributes.
- Should support queries that search the distance not only among terms but also among sections.
- Should support queries whether an element contains a certain subelement or not. This can be important for elements containing alternative subelements.
- Should support data-typing. It is an important criterion for comparing attribute values or querying element content that are not of type string.
- Should not leave out structure elements when creating the index, not even structure elements that occur in stopword lists[1].

[1] A stopword list contains words that will be disregarded when creating an index.

The current SQL Multimedia Standard for Full-Text from 1999 requires certain search methods for the user-defined type FullText, but they do not apply to structured documents like XML documents. For this reason, the conformance to the standard is not a criterion for evaluating the new functionalities of the following products.

3.1 IBM/ DB2 XML Extender and Text Extender

The XML Extender and the Text Extender support XML in database applications. DB2's XML Extender Version 7 provides the ability to store and access XML documents. Furthermore, XML documents can be generated from existing relational data or decomposed respectively shred into relational data. Therefore, the XML Extender supports two storage and access methods, XML column and XML collection.

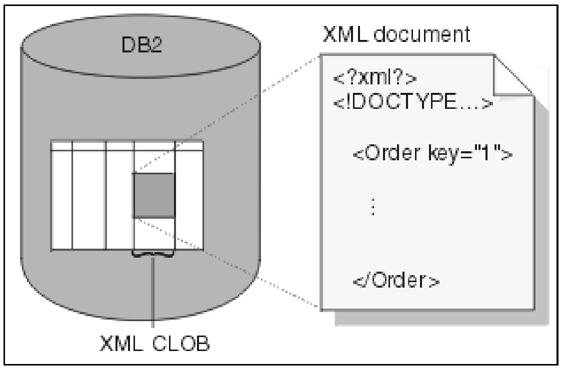

Figure 1: Storing an XML document in a DB2 table column

XML column enables the user to store intact XML documents as shown in figure 1. The document is inserted into a column that is enabled for XML and can be updated, retrieved and searched. Element and attribute data can additionally be mapped to DB2 tables, so-called side tables, which in turn can be indexed for fast structural search. The mapping of the XML document structure to DB2 side tables is defined with the XML-based document access definition (DAD) language supported by the XML Extender. There is one DAD for each XML column. The XML Extender provides XMLVarchar, XMLCLOB and XMLFILE as user-defined types for use with XML column. Using the XML column method, the user can perform structural text search with the Text Extender. However, XMLFILE is not supported by the Text Extender.
Structured document support has been added to DB2's Text Extender Version 6. In order to limit the scope of a search to a particular section of documents,

the user must create a document model. This document model contains the markup tags that identify the chosen sections and descriptive section names, which will be used in queries against that section. Current information can be found in [IBM99].

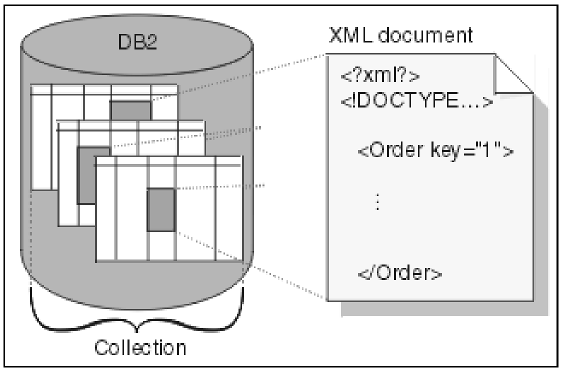

Figure 2: Storing a decomposed XML document in DB2 tables

The XML collection method gives the user the ability to either decompose incoming XML documents, or compose outgoing XML documents. Figure 2 shows a decomposed XML document in DB2 tables. Elements and attributes are mapped to relational schema according to the XML collection's DAD.

For further information about the XML column or XML collection method see [IBM00].

Criteria for Update Operations

Insert: On the one hand, the XML Extender provides a DTD repository to store DTDs. When the user inserts an XML document either with the XML column method or with the XML collection method, he can specify a DTDID in the DAD file to relate a certain DTD to it. The DTDID is a path specifying the location of the DTD on the local system. That means, validation is only possible against an external, locally residing DTD. When the DAD file is processed, the XML document will be validated according to this DTD. On the other hand, XML documents can be stored without a DTD. In this case, documents will not be processed for well-formedness.

Update: It is possible to update the entire XML document by replacing the XML column data, but it is also possible to update values of specified elements or attributes using XPath like expressions. Updating element content is restricted. Only text content of the element itself can be changed. It is not possible to add or delete subelements, that means, it is not possible to update the structure of a document.

The following example shows the update of the *year* element's content using the Update UDF (User Defined Function):

```
UPDATE paper_table
set xml_doc = Update(xml_doc,'/paper/meta/year','2000')
WHERE paper_id = 1
```

xml_doc is a column enabled for XML containing entire XML documents. In general, the user can choose between two methods, casting functions or the Update UDF, for updating the data. Both methods do not relate to the DTD repository, therefore updated documents will not be processed for well-formedness and validity.
When updating a column that is enabled for XML, the XML Extender automatically updates the side tables to reflect the changes.
Updating decomposed documents results from executing the usual SQL Update statement.

Update text index after updating XML documents: Updating the search index is not a database managed event. It occurs either according to the Text Extenders update frequency settings or immediatly after executing the update index command.

Extract XML data: On the one hand, the user has the possibility to retrieve an entire document by using casting functions. The following example retrieves XMLVARCHAR and stores it in memory as a VARCHAR data type:

```
SELECT db2xml.varchar(xml_doc) FROM paper_table
```

On the other hand, the user can retrieve element content or attribute values by using user-defined extract functions.

```
SELECT extractVarchar(xml_doc,'/paper/meta/author')
FROM paper_table
```

In this example, the extract UDF retrieves the element *author* from the column *xml_doc* as a VARCHAR data type. There are about ten different data types supported by extract functions.
The path specified as parameter of the extract UDF should be a path down to an element containing PCDATA. If a path like '/paper/meta' for the example document of chapter 2.1 is specified, then an empty result is returned, because the meta section itself does not contain PCDATA. The result is not a content summary of the subelements title, author and year. Querying a path that does not specify a leaf of the document tree is only useful, in case of this UDF, when the specified section is of mixed content type (see [W3C98]).
Empty elements return empty results.

Search Criteria

Separation of structure and content: Besides the ability of separating sentences and paragraphs, which has already been available in earlier versions, the Text Extender can now search a particular section of a document. Sections are delimited by start and end tags. The Text Extender provides two kinds of sections. There are plain-text sections that have no type and there are sections with a declared type that are called attribute sections. Attribute sections are simple sections delimited by start and end tags and should not be mistaken for element's attributes. The supported attribute section types are DATE, TIME, FLOAT, and INTEGER. Nested attribute sections are not allowed. Attribute sections are supported in Text Extender version 7.
However, the power of section search is limited:

- The dual index cannot be used to take advantage of document structure.
- The ngram index doesn't support the XML format.
- The user has to know the structure of the document in order to specify sections that will be searched. It is not possible to query whether a document contains a certain section or not.

With the XML Extender, elements and attributes can be distinguished. Furthermore, they can be mapped to object relational schema according to the DAD file of XML collection. That means, search against the structure of the document can be mapped to schema queries. Content search represents queries against the content of the table's particular column. To specify sections in the DAD file, XPath compliant expressions are used.
Moreover, the XML Extender creates a default view in case of XMLcolumn, that joins the application table and the side tables. The user can use this view to search column data and query side tables.
Although XML Extender and Text Extender are able to separate sections, they are not able to distinguish, for example, between the first and the last chapter of the *paper* example. It is not possible to narrow the search to this point.

Section search: There are two possibilities for section search with the Extenders. On the one hand, the user can decompose the XML document with the XML collection method and enable the required columns for Text Extender's search functionality. For example, assume that the *paper* example document is decomposed whereat paragraph sections are stored in a separate column *para* in the *para_table*. The column is enabled for Text Extender's text search with the handle column named *para_handle*[2].

[2]The handles in the para_handle column indicate where the paragraph text index is located.

Then a query could look like this:

```
SELECT para FROM para_table
WHERE DB2TX.contains(para_handle,
     'PRECISE FORM OF "Indexing"')=1
```

As a result of this query, all paragraphs of *para_table* that contain the precise form of the string "Indexing" will be returned.
On the other hand, the user performs the XML column method, enables the column for Text Extender and specifies the required sections in the document model file before creating the index. In case of empty elements, empty results will be returned.
The model name must be the same as the tag name of the root element.

A document model file could look like this:

```
[MODELS]
modelname = paper

[paper]
paper = paper
paper/anystring = paper/body/chapter/paragraph
```

First the model's name *paper* is defined. Then sections are related to this model. The right-hand side specifies the path through the document tree down to the particular element. Elements are seperated by "/". However, the section name identifier on the left-hand side that is used in queries could be any name. Moreover, the possibilities of specifying the search scope is limited to point 2 of the XPath expressions of chapter 2.2.
Using the XML column method, the above example query would look like this, when the paper document is stored in a column of the *paper_table* with the handle column named *paperhandle*.

```
SELECT paper_id FROM paper_table
WHERE DB2TX.contains(paperhandle,
     'MODEL paper SECTION (paper/anystring)
      PRECISE FORM OF "Indexing"')=1
```

As a result of this query, all paper identifier of *paper_table* whose paper contain the precise form of "Indexing" within the anystring sections (paragraph sections) will be returned. In this case, it is necessary to use the MODEL and SECTION keyword in order to limit the scope of a search to particular sections. Besides, the Text Extender allows to mask a section name within a

query using wildcared characters.
Furthermore, the user can use equality comparison and value ranges to search documents containing attribute sections. For example, the *year* section of the *paper* example is an attribute section of type INTEGER. Then the search term:

```
'MODEL paper SECTION (paper/meta/year) = 1999'
```

will search for documents published in 1999.
By the way, the user does not have to use the XML Extender to save XML documents in DB2 in order to perform section search by Text Extender. It is also possible just to enable the entire XML document as text file for Text Extender.
Moreover, the XML Extender can be used for searching XML documents with direct queries on side tables or searching from a joined view, for example the default view as mentioned above, or with extract UDFs.

```
SELECT paper_id FROM paper_table
WHERE db2xml.extractVarchar(xml_doc,
       '/paper/meta/author')
       like 'Laud%'
```

This query using an extract UDF will return all paper identifier of *paper_table* whose paper contain author elements with a value that starts with 'Laud'.

Search attributes: Attribute values can easily be queried in three different ways:

- When an attribute has been mapped to a separate column by XML collection method using SQL
- When an attribute has been mapped to a side table by XML column method using SQL
- When an attribute resides as part of an intact document using XML Extender's extract UDFs

The following shows an example for extract UDFs:

```
SELECT paper_id FROM paper_table
WHERE db2xml.extractVarchar(xml_doc,
       '/paper/body/chapter/@title')
       like '%XML%'
```

This query will return all paper identifier of *paper_table* whose paper contain chapter title similar to 'XML'.
Attributes cannot be specified in the Text Extender's document model file. That means, attributes and its values cannot be querried by Text Extender's functionality.

Proximity search: Text Extender's proximity search can be combined with section search. That means, search terms can be found in the same sentence or paragraph within a specified section. For example, the search term:

```
'MODEL paper SECTION (paper/body/chapter) "search"
IN SAME SENTENCE AS "documents"'
```

searches documents where "search" and "documents" appear in the same sentence and the sentence is part of a chapter section.
However, the distance among elements, e.g. the distance to the root element, cannot be investigated.

Inclusion: With the Text Extender it is not possible to query whether an element contains a certain subelement or not.

Data-typing: When elements or attributes are mapped to relational schema by the XML collection method, a different type besides string can be related to it. This type can be any basic datatype provided by DB2. In this case, identity and range search can be performed according to the datatype.
Text Extender's search terms can only be strings except if attribute sections appear within the document. Then range search can be performed as described in section Section search.

Indexing: All tags of sections enabled for section search are not indexed and cannot be searched.
Tags that are not defined in the document model file are indexed according to the index type. Stopwords[3] are disregarded even if they appear within tags.

Generally speaking, the XML Extender is a powerful DB2 extension for inserting, updating and extracting XML data. The Text Extender provides sophisticated full text search functionality that can be used to search XML documents. IBM's solution of providing the possibility of combining both extenders enables the user to perform extensive full text search in XML documents.

[3]The user can define words that will not be indexed so-called stopwords, e.g. and, or, then, if. Defining stopwords keeps the index small.

3.2 Oracle/ *Inter*Media Text

Oracle provides several components, utilities and interfaces to take advantage of XML technology. Oracle's key product that supports full text search in XML documents is *Inter*Media Text.
*Inter*Media Text version 8.1.5 can be used to perform searches on XML documents stored in Oracle8*i* by indexing the XML as plain text or as document sections. High full text functionality is provided, such as exact match search, search with boolean expressions, word stem, fuzzy and free text search, phonetic and proximity search, search for synonyms and section search. In the following the main focus will be on section search.
The user can follow three basic strategies when storing and searching XML documents in Oracle8*i*:

- The first is to store an **XML document as a single, intact object** with its tags in a CLOB or BLOB. In order to store an XML document as an intact object Oracle's SQL*Loader tool can be used.
- The second strategy is to store the **XML document as data** and distribute it untagged across object-relational tables. This functionality is provided by the XML SQL Utility (XSU). XSU provides the means to map XML documents to the underlying object-relational storage. It allows to extract data from an XML document, then insert the data into a table, update a table, or delete corresponding data from a table. The corresponding command is 'OracleXML putXML'. It also allows to retrieve the object-relational data as an XML document. This means, the user can generate an XML document given an SQL query or a JDBC resultset object. The corresponding command is 'OracleXML getXML'.
- The last strategy is **combining XML documents and data using views**. The user just creates a view that, for example, combines an XML document stored in a column and its meta data stored in different columns.

Criteria for Update Operations

Insert: When an XML document is inserted, it will not be processed for validity or well-formedness. It is the user's responsibility whether to include Oracle's XML Parser in his application or not.
If the document gets mapped to the object-relational schema and its structure is not compatible with the structure of the database schema, the user must transform the data into the correct format before writing it to the database. An element name has to be equal to its corresponding relation's attribute name or vice versa. In order to do so, XSL (Extensible Stylesheet Language) stylesheets or other programming approaches are recommended.

Update: It is possible to update the entire XML document by replacing the existing one using the SQL*Loader Replace statement.
It is also possible to update values of elements, but only when the specified element sections have been mapped to a separate column. Then the SQL Update statement or XSU (OracleXML putXML) can be used to update the element's content.
Updated documents will not be processed for well-formedness or validity.

Update text index after updating XML documents: Oracle provides two ways of updating the text index. On the one hand, the index is updated immediatly and automatically whenever there is an insert, delete, or update to the base table. This is known as background DML[4] processing. On the other hand, the user can update the index in batch mode by executing the alter index command. When synchronizing the index in batch mode, Oracle processes pending updates and inserts stored in the DML queue.

Extract XML data: On the one hand, the user has the possibility to retrieve an entire document. However, it is not possible to extract parts of an intact XML document.
On the other hand, the user can retrieve element content by executing the SQL Select statement if the XML document was decomposed. This does not apply to attribute values.

Search Criteria

Separation of structure and content: Oracle gives the user the ability to separate sections. These sections can be zone sections, field sections or special sections. Latter specify sentence and paragraph sections. Zone sections and field sections are sections delimited by start and end tags. Zone sections can be nested within one another, can overlap and can occur more than once in a document in contrast to field sections that cannot nest or overlap and are non-repeating. To be able to index element content as zone or field sections and to search these sections, the user has to know the structure of the document.
Although *Inter*Media Text is able to separate sections, it is not able to distinguish, for example, between the first and the last chapter of the *paper* example. It is not possible to narrow the search to this point.
If the user followed the second strategy, storing the XML document as data across object-relational tables, search against the structure of the document can be mapped to schema queries. Content search represents queries against the content of the table's particular column.

[4]Data Manipulation Language

Section search: In order to search sections within an intact XML document the user has to create a section group of type XML_SECTION_GROUP. Afterwards field sections and zone sections can be added or removed from this section group. For example:

```
begin
ctx_ddl.create_section_group('xmlgroup','XML_SECTION_GROUP');
ctx_ddl.add_zone_section('xmlgroup','Chapter','chapter');
ctx_ddl.add_field_section('xmlgroup','Title','title',TRUE);
end;
```

In this example an XML section group called *xmlgroup* is created and a zone section called *Chapter* and a field section called *Title* are added to it. The third parameter of the add_xxx_section statement is the tag that marks the section.
Besides, the user can decide whether the content of a field section is visible or not. For example, assume that the title element of the *paper* example is a field section defined with visible flag set to false. Then the search request for:

```
SELECT paper FROM paper_table
WHERE CONTAINS(xml_doc,
        'Full Text Search in XML Documents')>0
```

without specifying the field section title will not find the example document. Only the query:

```
SELECT paper FROM paper_table
WHERE CONTAINS(xml_doc,
        'Full Text Search in XML Documents
         WITHIN Title')>0
```

will be successful, because it explicitly specifies the title field.
If the user followed the second strategy of storing XML documents decomposed, he can create an index for the required column and query it by searching its index without using an xml_section_group.

Search attributes: Attributes cannot be searched by *Inter*Media Text's section search. Furthermore, XSU does not provide the functionality to map XML attributes to columns. For this reason, it is not possible to search for attribute values.

Proximity search: Oracle's proximity search is basically performed by using the WITHIN operator. However, this operator cannot nest. For example, it is not possible to query:

```
'(search and documents) WITHIN SENTENCE
         WITHIN Chapter'
```

which should find documents where "search" and "documents" appear in the same sentence and the sentence is part of a chapter section. Another way to perform proximity search is to use a combination of the WITHIN operator and the NEAR operator. NEAR returns a score based on the proximity of two or more search terms. For example:

```
SELECT paper FROM paper_table
WHERE CONTAINS(xml_doc,'near((search, documents),10)
        WITHIN Chapter')>0
```

This will return documents where the maximum distance between "search" and "documents" is 10, and both terms appear in the same chapter section. The NEAR score defines the distance between the search terms. The highest score is 100.
The distance among elements, e.g. the distance to the root element, cannot be investigated.

Inclusion: With Oracle's *Inter*Media Text it is not possible to query whether an element contains a certain subelement or not.

Data-typing: When elements are mapped to relational schema by the XML SQL Utility, a different type besides string can be related to it. This type can be any basic datatype provided by Oracle. In this case, identity and range search can be performed according to the datatype. Oracle's *Inter*Media Text's search term can only handle strings.

Indexing: Tags are not indexed and cannot be searched when an XML section group is specified as parameter for indexing.

Generally, Oracle's *Inter*Media Text is a powerful database extension that enables the user to perform sufficient full text search in XML documents.

3.3 Microsoft SQL Server 2000 Text Search

The new Microsoft SQL Server 2000 is XML-enabled. Among other things, it allows queries to be sent directly to SQL Server via URL with the results returned as XML formatted documents. The SQL Server 2000 provides full text search concepts. Although the supported features are entry-level, they still serve many full text searching purposes for users. Exact match search, search terms with boolean expressions, ranking search, word stem and free text search are provided. Moreover, restricted use of wildcared and proximity search are part of the features. However, all these features do not apply to documents of XML format.

3.4 Informix/ Excalibur Text Search DataBlade Module

XML will be supported by the newest version of the Informix Web DataBlade module. An early developer's release will be made available later this year. Hierarchical XML Data Storage will give the ability to import, export, store and query XML structures in their native format.
Informix provides high full text search functionality, which covers exact match search, search with boolean expressions, ranking, word stem and free text search, wildcared and proximity search and search for synonyms. This is described in detail in [Por99]. The Excalibur Text Search DataBlade Module does not yet provide section search in XML documents.

Having looked at different products regarding their support of full text search functionality for the XML format, the next section will compare the results of the above investigation.

3.5 Comparison

In this chapter the previously analyzed products will be compared. However, Microsoft and Informix do not yet provide products that are enabled for full text search in XML documents. For this reason, they are not regarded in the following comparison.
At first the advantages and disadvantages of IBM's and Oracle's products regarding the criteria for update operations will be listed. Afterwards the products will be compared according to the search criteria. Final remarks will complete this chapter.

Criteria for Update Operations

Insert: IBM's and Oracle's products provide the functionality to store an XML document decomposed as relational data as well as an entire, intact XML document.
It is the advantage of IBM's XML Extender that it provides additional functionality like:

- Providing XML specific types
- Validation against DTD when storing XML documents
- Ability of decomposing attributes
- Generating additional side tables when storing intact XML documents.

However, in both cases the problem of mapping the XML document structure to relational schema results for decomposing. The user is in charge of providing structure and mapping information. In case of the XML Extender the user has to write a DAD file. In case of XSU the user has to change the document structure according to the relational schema. In both cases, the user is not able to decompose XML documents of unknown structure.

Update: IBM's and Oracle's products provide the functionality to update entire XML documents and update content of decomposed elements.
Providing an additional Update UDF is the advantage of IBM's XML Extender, because it allows to update element's text content within an intact XML document, although the document structure itself cannot be changed. Furthermore, attribute values can be updated by the XML Extender.
However, in both cases updated documents will not be processed for well-formedness and validity.

Update text index after updating XML documents: Text Extender and *Inter*Media Text provide functionality for updating the text index immediatly after an XML document has been changed, and later as a batch update. The advantage of Oracle's *Inter*Media Text is the fact that the search index update is a database managed event.

Extract XML data: IBM's and Oracle's products provide the functionality to retrieve entire documents and content of decomposed elements. It is the advantage of IBM's XML Extender that it provides additional extract UDFs. These extract UDFs can extract elements content as well as attribute values of intact XML documents and convert the result to a selected data type. The disadvantage is the fact that in contrast to user's expectation the result does not include the content of existing subelements.

Figure 3 summarizes the results of this paragraph.

	IBM	Oracle
Insert	+	o
Update XML docs	+	o
Update text index	o	+
Extract XML data	o	o

Figure 3: Comparison - criteria for update operations

The characters have the following meaning:
- : The functionality is not supported.
o : The functionality is supported.
+ : The functionality is supported and additional, helpful features are provided.

Search Criteria

Separation of structure and content: IBM's and Oracle's products provide the functionality to recognize sections, that means, they are able to separate structure information and content. But in case of an element that occurs multiple times within a document, they are not able to narrow the search to the first, second or last section of this element.

Section search: IBM's and Oracle's products provide high full text search functionality in combination with section search; it does not matter whether a section is part of an intact document or a result of a decomposition.
Both provide similar concepts; *Inter*Media Text's zone section is similar to Text Extender's plain-text sections and field sections are similar to attribute sections. The advantage of Oracle's field section is the fact that the user can decide whether it is visible or not. It is the advantage of IBM's attribute section that it allows range search.
However, both Text Extender and *Inter*Media Text ask the user for specifying in advance which sections he wants to search. In case of Text Extender, the user has to specify a document model. In case of *Inter*Media Text, the user has to create an XML section group with its components.

It is an advantage of IBM's XML Extender that it can also be used to perform simple search queries without specifying the required sections in advance.

Search attributes: IBM's Text Extender and Oracle's *Inter*Media Text and XML SQL Utility do not support attributes. The advantage of IBM's XML Extender is the fact that it makes attribute search possible.

Proximity search: IBM's and Oracle's products provide the functionality to combine proximity search and section search. However, the distance among elements cannot be investigated.

Inclusion: IBM's and Oracle's products do not provide the functionality to allow queries whether an element contains a certain subelement or not.

Data-typing: IBM's and Oracle's products provide the functionality to relate data types provided by their DBMS to element sections that will be decomposed. It is an advantage of IBM's XML Extender that it provides additional extract UDFs that convert their result to a selected data type. Providing internal data type mapping for attribute sections is an advantage of the Text Extender.

Indexing: Oracle's *Inter*Media Text does not index any XML structure information. IBM's Text Extender, in contrast, does not index information specified in the document model file. Both solutions are not satisfying when queries against structure information are demanded.

Figure 4 summarizes the results of this paragraph.

	IBM	Oracle
Separation of structure and content	o	o
Section search	+	+
Search attributes	+	-
Proximity search	o	o
Inclusion	-	-
Data-typing	+	o
Indexing	-	-

Figure 4: Comparison - search criteria

The characters have the following meaning:
- : The functionality is not supported.
o : The functionality is supported.
+ : The functionality is supported and additional, helpful features are provided.

Concluding it is to say that IBM and Oracle provide products that make full text search in XML documents possible. They are able to separate structure and content in order to take advantage of the information about structure. They recognize sections, so that it is possible to narrow the search scope within documents. This results in improved information retrieval.
However, this can be more improved, for example, through supporting all functionality provided by XPath or automatic structure analysis and indexing.
Of course, IBM's advantage is the XML Extender. It provides XML specific types and functions that allow extensive information retrieval in combination with the Text Extender.

4 GTR

GTR is an IBM text search engine. GTR stands for Global Text Retrieval. It will replace the existing IBM Text Search Engine (TSE), which is currently used within the DB2 Text Extender. Until now, GTR does not support XML. This chapter will introduce GTR's concepts and provide suggestions how to support XML by GTR. The intention is to use GTR's existing internal search functionality and extend it for XML support.

4.1 Concepts

In contrast to many other text retrieval products, GTR is not based on dictionaries. Instead of performing keyword search, it focuses on the aspect of consecutive character pattern. No stopword list will be created. Every information of a document is kept, when it is indexed by GTR.
GTR consists of a set of functions for high speed text search. GTR's text search functions can easily be integrated into an application program. Moreover, GTR provides thesaurus functions to improve search conditions, see [IBM97] for further information.
In the following GTR's search type and indexing concepts will be introduced.

4.1.1 Search Types

Two different search types are provided by GTR, full text search and item search.

Full Text Search Full text search is a function to search for documents using search terms contained in the documents, e.g. search for "XML" within the abstract "This article is about XML .." . The following search functionality is supported for full text search:

- *Exact match:* GTR searches for the terms exactly as typed.
  '"database"#C' finds documents containing the term "database". A case sensitve search is performed.
- *Fuzzy search:* GTR searches for words that are spelled in a similar way to the search term. Occurences of misspelled words could be found.
  '"database"%75' $\Longrightarrow$ The string "database" is searched for with 75 percent matching level.
- *English word stemming:* GTR searches also for variations of the search terms, such as the plural of a noun, or a different tense of a verb.
  '"communicate"%STEM' $\Longrightarrow$ "communication", "communicating" are also searched based on English inflection rule.

- *Wildcard search for alphabetical/ alphanumerical words:* Masking characters represent optional characters within the search term. Therefore, search is more flexible and the number of matching documents increases.
  ”data*” $\Longrightarrow$ Documents containing ”database”, ”data”, ”datatransfer” can be found.
- *Field search:* Only specific parts of documents will be searched.
  ’”database”@F10’ $\Longrightarrow$ Only field 10 is searched for ”database”.
- *Boolean operations:* Search terms can be combined with other terms using the boolean operators ”*”(AND), ”+”(OR), and ”!” or ”-”(NOT).
  ’(”database” + ”storage”) * ”IBM”’ $\Longrightarrow$ Documents that contain either ”database” or ”storage” and ”IBM” are searched.
- *Adjacent operation (proximity search):* GTR searches for documents that contain all the specified search terms within the same segment. A segment could be a sentence or a paragraph or others. For example search for documents that contain ”database” and ”IBM” within one sentence.
- *Ranking search:* GTR returns an absolute value that indicates how well the document met the search criteria relative to other found documents. The following three factors are selectable to determine the score of documents: frequency of search terms in the document, frequency of search terms in the whole set of documents, weight parameter specified explicitly by the user program. An example for a weight specified by user program:
  ’”more important term”$200 + ”less important term”$100’ $\Longrightarrow$ ”more important term” is interpreted twice as important as ”less important term”.

Item Search While full text search scans natural language document text itself, item search is a function to search for documents using formatted attribute data, e.g. author name, creation date or category code. The formatted attribute data can be of character or numeric type. Besides, item without and item with multiple values for one document are allowed, that means, if an author name item exists, multiple authors could be assigned to one document.
The following search functionality is provided for item search:

- Character item search
  - *Exact match:* GTR searches for documents which character item value is identical to the specified value. For example, a search by an author named ”John” will find documents whose author item value is ”John” but not ”John Doe”.
  - *Wildcard:* GTR searches for documents whose character item value differs from the specified value only in the masked characters. The search term ”Jo* Doe” can find documents that have an author item value like ”Jo Doe” or ”John Doe”.

- Numerical item search
  - *Numerical item matching:* GTR searches for documents whose numerical item value is equal to the specified value. For example, searching for all documents published in 1999 means, the pubyear item value of a document has to be equal to 1999.
  - *Range search:* GTR searches for documents whose numerical item value is smaller or bigger than the specified value. Searching for documents published after 1995 and before 1999 means: 1995 < pubyear item value < 1999.

The difference between the character item search and the full text field search is that the character item search returns documents whose character item value *is equal* to the given value instead of returning documents that *contain* the search term in a certain field. Figure 5 shows an example containing text and attributes.
Ranking is not supported in the case of item search.
Text search condition and item search condition cannot be specified together, but it is possible to combine text and item search. The easiest way to realize it is to specify the result of item search as a scope of text search. From the documents satisfying the search condition, only the documents contained in the specified scope are selected.

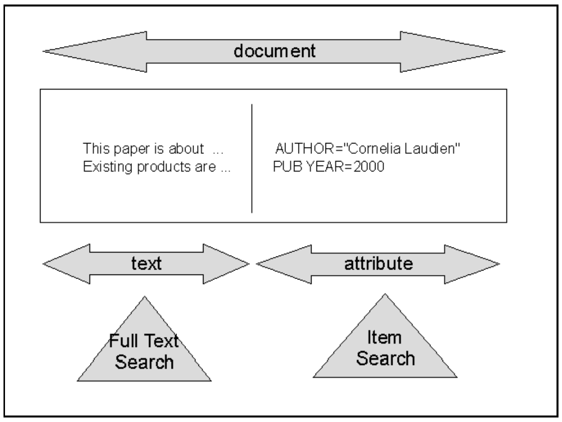

Figure 5: Text Search versus Item Search

4.1.2 Indexing

GTR's function set allows to create, update and delete indexes. Therefore, two different index files are provided by GTR, the **main index** file and a **supplemental index** file. The index data for newly added documents is automatically written into the supplemental index file. Both indexes can be merged. Search, of course, is done to main and supplemental index. The advantage of the supplemental index file is the fact that search systems can achieve real-time frequent document registration. To keep track of every document the application program has to provide a unique **document number** for each document. This number is at least zero and must be assigned in ascending order. The document number will be passed to the create index function for indexing and will be returned as the result of a search request.

4.2 GTR's Suitability for XML Support

With the ever-increasing importance of XML, retrieval needs to support the structure of documents efficiently. GTR provides interesting functionality that can be used to support XML. This chapter is a collection of ideas, it is not a specification and does not claim completeness.
In the following three sections GTR's field search, adjacent operation and item search will be described more detailed. The given information is of interest for further investigation regarding GTR's aptitude for XML support.

4.2.1 GTR's Field Search

As mentioned above GTR provides the ability to search only specific parts of a document by field search. The specific part to be searched is identified by a field number and the user program has to specify where the field is located and what field number is assigned to it. This information will be used for indexing. There can be more than one field assigned to one field number. However, the fields assigned to the same field number are not allowed to be nested or to overlap. The field number can reach from 1 to 65535. When a field search is requested the field number needs to be assigned to the search terms.
On the one hand, field search could be used to **search element's content**. In order to recognize a certain element, a field number would be assigned to its section. If an element occurs several times then all its sections will be assigned to the same field number. Within a specified field all other full text search functionality, e.g. fuzzy search or wildcard, is provided.
For example to search the author, chapter and paragraph sections of the example in section 2.1 the following would be done: The author element could be assigned to field number 1, all chapter sections could be assigned to field number 2 and all paragraph sections could be assigned to field number 3. It would not be allowed to assign all chapter and paragraph sections to the same field number, because they nest.
On the other hand, field search could be used for **searching the structure** of an XML document. Questions whether documents contain certain elements or attributes could be answered. Therefore, each tag itself would be considered as a field that starts with ”<” and ends with ”>”. The restriction does not apply in this case, because tags never nest or overlap in well-formed XML documents. All

tags could be assigned to the same field number, there is no restriction mentioned regarding the number of sections assigned to one field number. Otherwise each tag gets its own field number. To limit the cost only start tags would be considered, because they contain all necessary information.

4.2.2 GTR's Adjacent Operation

With the adjacent operation the user can search for documents that contain search terms within the same segment. In order to enable such a way of search, the information identifying the locational limitation needs to be built into the index when those documents are indexed. There can be different kinds of text segments. Each kind of segment relates to a different separation-rule. A separation-rule number is assigned to each separation-rule. GTR allows 33 different separation rules. The separation-rule number 0 is already assigned to the sentence separation-rule. Within each rule the end of a segment needs to be specified. In this case: ".", "!" and "?" followed by a blank character mark the end of a sentence. When an adjacence search is requested the separation-rule number needs to be assigned to the search terms.

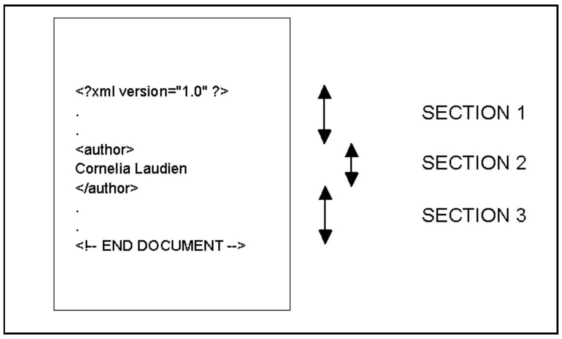

Figure 6: Adjacent Operation

A user program can give a meaning to separation-rules from number 1 to 32. To have the ability of section search a separation-rule could be created for each element that will be searched. Therefore, each tag of the element will be a separation mark. The resulting separation of the *paper* example regarding the author element is shown in figure 6. Within each rule only the end of a segment can be specified, that means, start and end tags are treated in the same way. This raises the problem that the user cannot focus on the required sections (section 2 in case of the example in figure 6). The problem could be solved by managing the segment separation pointer in a particular way. In case of empty element tags, the separation rule could be defined in a way that there won't be any segment separation pointers for this kind of tags.

However, compared to GTR's field search, adjacent operation does not really meet the requirements of exactly recognizing sections.

4.2.3 Item Search

Item search could be used in the case when XML documents are mapped to relational schema. Columns containing attribute values or element's content could be indexed and searched by item search. Moreover, GTR's function set allows it to combine full text search and item search. A search request could look like this:

```
SELECT paper_id
FROM paper_table
WHERE CONTAINS(paperhandle,' "XML" and "year=2000"')=1
```

instead of :

```
SELECT paper_id
FROM paper_table
WHERE CONTAINS(paperhandle,' "XML"')=1 and year=2000
```

This query searches for papers that contain the string "XML" and were published in the year 2000. The year element seperately stored as attribute is considered to be an item. In the first case, the string would be searched by full text search and the year attribute would be searched by item search. The specified paperhandle would contain the index information for the text and for the item index. In the second case, the year would be searched by the DBMS itself.
Even though item search is faster than full text search, DBMS are optimized for the kind of queries that can be answered by item search. For this reason, using DBMS functionality should be prefered in this case.

4.2.4 Other Considerations

Structure information: The fact that the user has to know the document structure and has to specify the sections he wants to search in advance has been criticized in earlier chapters. For this reason, it is suggested, that the document model should be created automatically containing all possible search paths. These paths should be of the intuitive XPath notation. Moreover, this document model file can then be viewed by the user in order to get structure information.

Furthermore, XML gives users the possibility to define tags that suit their specific requirements. Usually a given element name relates to the element's content. Assume documents that contain an element called database, and its content provides information about DB2, Informix or Oracle. Moreover, tags are not indexed. Then a search request looking for database information within these documents will not be successful unless the term database itself occurs again within the section. Because of that, it would be reasonable to keep structure information as part of the index. In this case, the document model file itself can be indexed.

Highlighting: GTR provides a function for highlighting the found character string. As a result of a search, the location of the found character string is returned, by the unit of character. GTR supports a function to convert the location into the location by the unit of byte. Through this function, the user program can find the location of hits as the format of offset bytes from the beginning of the document, and can easily highlight the searched character string when browsing the document.
In the case that structure information is indexed, and the searched character string is an element name, then the entire content of the element should be highlighted since browsers do not display structure information.

This chapter shows that GTR is able to separate structure and content in order to take advantage of structured documents. GTR's field search provides the ability to search certain sections and attributes. Furthermore, the idea of automatically generating and indexing the model file allows queries whether an element contains certain subelements or not. Performing this kind of queries is not possible in current products and has been critized in chapter 3.
Because this chapter is a collection of ideas, further, more detailed investigations are necessary in order to develop a complete specification of GTR's support for the XML format.

5 Outlook

Important steps have been made in order to extend the existing full text search functionality to use the advantages of the XML format. However, XML technology provides more features that can be used by text retrieval systems. Some of them have already been mentioned in earlier chapters. One of them is the XPath standard, which should be used more extensively to specify search scopes. Furthermore, XML Schema provides significant advantages over DTDs as described in chapter 2.3, and it is already used in many applications. For this reason, functionality in order to validate not only according to DTDs, but also according to XML Schemas should be provided.
Moreover, XML specific query languages should be considered. The SQL-Contains-extension with its search argument becomes more complex the more structure and other search information it contains. For example, the XML information server of the Software AG called Tamino uses XQL as query language besides SQL.
Finally the SQL Multimedia Standard for Full-Text should be reviewed and updated regarding full text search in structured documents.

A Remarks about New Versions

It should be noticed that new versions of products have been delivered during the time of writing this paper. One of them is a new version of Oracle's *Inter*Media Text, version 8.1.6. This is mentioned because functionality that was critizised in this paper has been improved, and suggestions that were made have been realized. Although the practical investigation of the new features could not be done as part of this paper, they should at least be listed.

A.1 Changes from Oracle *Inter*Media Text version 8.1.5 to version 8.1.6

Search attributes: Attribute search is now possible in the new version. In order to search attributes, attribute sections can be added to XML section groups. The attribute itself is specified in tag@attribute form.

Section search: An additional section group type called AUTO_SECTION_GROUP has been added to the new version. When it is specified, it automatically creates a zone section for each start-tag/end-tag pair in an XML document. Attribute sections are also created automatically if applicable.

Proximity search: It is now possible to create nested WITHIN queries.

List of Figures

References

[Bou00] Bourret, R.: XML Database Products, March 2000, http://www.informatik.tu-darmstadt.de/DVS1/staff/bourret/xml/XMLDatabaseProds.htm

[IBM97] IBM Corporation: *GTR V3 Specification, 1997*

[IBM99] IBM Corporation: *DB2 Universal Database Text Extender: Administration and Programming, June 1999.*

[IBM00] IBM Corporation: *XML Extender Administration and Programming, 2000.*

[Inf97] Informix Software Inc.: *Excalibur Text Search DataBlade Module, July 1997.*

[Jun99] Jung, F.,Software AG: XML Backgrounder Technology and Applications, December 1999, http://www.softwareag.com

[Mic98] Microsoft: *Textual Searches on Database Data Using Microsoft SQL Server 7.0, 1998.*

[Nit00] Nitzsche,R.: Kopplung von Volltext- und Datenbanksystemen. Master Thesis, University of Rostock, Germany, 2000.

[Opp99] Oppel,K.,Software AG: *Tamino, Der Information Server für Electronic Business, October 1999.*

[Ora99] Oracle Corporation: *Oracle8i interMedia Text Reference, 1999.*

[Ora00] Oracle Corporation: *Oracle XML SQL Utility (XSU), April 2000.*

[Por99] Porst, B.: Untersuchungen zu Datentyperweiterungen für XML-Dokumente und ihre Anfragemethoden am Beispiel von DB2 und Informix. Master Thesis, University of Rostock, Germany, 1999.

[W3C98] W3C XML Working Group: Extensible Markup Language (XML) 1.0, February 1998, http://www.w3.org/TR/REC-xml

[W3C99] W3C XML Working Group: XML Path Language (XPath) 1.0, November 1999, http://www.w3.org/TR/xpath

[W3C000] W3C XML Working Group: XML Schema Part 0: Primer, April 2000, http://www.w3.org/TR/xmlschema-0

[W3C001] W3C XML Working Group: XML Schema Part 1: Structure, April 2000, http://www.w3.org/TR/xmlschema-1

[W3C002] W3C XML Working Group: XML Schema Part 2: Datatypes, April 2000, http://www.w3.org/TR/xmlschema-2