Chapter 1
Introduction

Abstract When working with those XML data, there are three different functions that need to be performed: adding information to the repository, searching and retrieving information from the repository, and updating information from the repository. A good XML database must handle those functions well. In this chapter, we will introduce solutions for XML database, including flat files, relational database, object relational database, and other storage management system.

Keywords Relational database • Object relational database

1.1 XML Data Model

An XML document always starts with a prolog markup. The minimal prolog contains a declaration that identifies the document as an XML document. XML identifies data using tags, which are identifiers enclosed in angle brackets. Collectively, the tags are known as "markup." The most commonly used markup in XML data is element. Element identifies the content it surrounds. For example, Fig. 1.1 shows a simple example XML document. This document starts with a prolog markup that identifies the document as an XML document that conforms to version 1.0 of the XML specification and uses the 8-bit Unicode character encoding scheme (Line 1). The root element (Line 2–14) of the document follows the declaration, which is named as bib element. Generally, each XML document has a single root element. Next, there is an element book (Line 3–13) which describes the information (including author, title, and chapter) of a book. In Line 9, the element text contains both a subelement keyword and character data *XML stands for* . . . .

J. Lu, *An Introduction to XML Query Processing and Keyword Search*,
DOI 10.1007/978-3-642-34555-5_1,

```
1.  <?xml version = "1.0" encoding = "UTF–8"?>
2.  <bib>
3.   <book>
4.    <author>Suciu</author>
5.    <author>Chen</author>
6.    <title> Advanced Database System </title>
7.    <chapter><title>XML</title>
8.     <section><title>XML specification</title>
9.      <text><keyword>markup</keyword> XML stands for...
10.     </text>
11.    </section>
12.   </chapter>
13.  </book>
14.  </bib>
```

Fig. 1.1 Example XML document

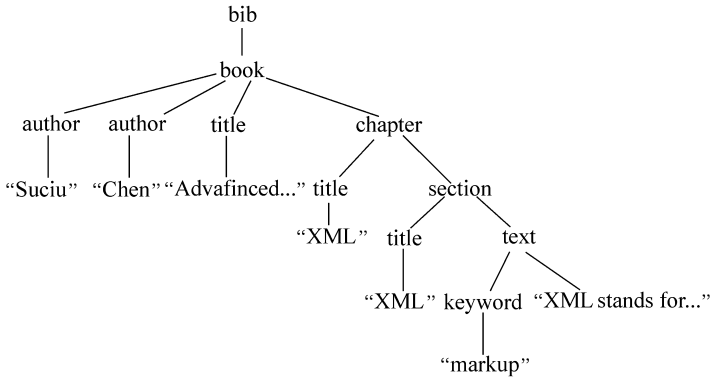

Fig. 1.2 Example XML tree model

Although XML documents can have rather complex internal structures, they can generally be modeled as trees,[1] where tree nodes represent document elements, attributes, and character data and edges represent the element–subelement (or parent–child) relationship. We call such a tree representation of an XML document as an XML tree. Figure 1.2 shows a tree that models the XML document in Fig. 1.1.

XML has grown from a markup language for special purpose documents to a standard for the interchange of heterogenous data over the Web, a common language for distributed computation, and a universal data format to provide users with different views of data. All of these increase the volume of data encoded in XML, consequently increasing the need for database management support for XML documents. An essential concern is how to store and query potentially huge amounts of XML data efficiently [AJP+02, AQM+97, JAC+02, LLHC05, MW99, ZND+01].

[1]For the purpose of this book, when we model XML document as trees, we consider IDREF attributes as not reference links but subelements.

1.2 Emergence of XML Database

XML has penetrated virtually all areas of Internet-related application programming and become the frequently used data exchange framework in the application areas [Abi97, CFI+00, DFS99]. When working with those XML data, there are (loosely speaking) three different functions that need to be performed: adding information to the repository, searching and retrieving information from the repository, and updating information from the repository. A good XML database must handle those functions well. Many solutions for XML database have been proposed, including flat files, relational database [FTS00, Mal99, SSK+01, STZ+99, TVB+02, ZND+01], object relational database [ML02, SYU99], and other storage management system, such as Natix [FM01], TIMBER [JJ06, JLS+04, PJ05, YJR03], and Lore [MAG97]. We briefly discuss these solutions as follows.

1.2.1 Flat File Storage

The simplest type of storage is flat file storage, that is, the main entity is a complete document; internal structure does not play a role. These models may be implemented either on the top of real file systems, such as the file systems available on UNIX, or inside databases where documents are stored as binary large objects (BLOBs). The operation store can be supported very efficiently at low cost, while other operations, such as search, which require access to the internal structure of documents may become prohibitively expensive. Flat file storage is not most appropriate when search is frequent, and the level of granularity required by this storage is the entire document, not the element or character data within the document.

1.2.2 Relational and Object Relational Storage

XML data can be stored in existing relational database. They can benefit from already existing relation database features such as indexing, transaction, and query optimizers. However, due to XML data that is a semistructured data, converting this data model into relation data is necessary. There are mainly two converting methods: generic [FK99] and schema-driven [STZ+99]. Generic method does not make use of schemas but instead defines a generic target schema that captures any XML document.

Schema-driven depends on a given XML schema and defines a set of rules for mapping it to a relational schema. Since the inherent significant difference between rational data model and nested structures of semistructured data, both converting methods need a lot of expensive join operations for query processing.

Mo and Ling [ML02] proposed to use object relational database to store and query XML data. Their method is based on ORA-SS (Object-Relationship-Attribute model for Semistructured Data) data model [DWLL01], which not only reflects the nested structure of semistructured data but also distinguishes between object classes and relationship types and between attributes of objects classes and attributes of relationship types. Compared to the strategies that convert XML to relational database, their methods reduce the redundancy in storage and the costly join operations.

1.2.3 Native Storage of XML Data

Native XML engines are systems that are specially designed for managing XML data [MLLA03]. Compared to the relational database storage of XML data, native XML database does not need the expensive operations to convert XML data to fit in the relational table. The storage and query processing techniques adopted by native XML database are usually more efficient than that based on flat file and relational and object relational storage. In the following, we introduce three native XML storage approaches.

The first approach is to model XML documents using the Document Object Model (DOM) [Abi97]. Internally, each node in a DOM tree has four pointers and two sibling pointers. The filiation pointers include the first child, the last child, the parent, and the root pointers. The sibling pointers point to the previous and the next sibling nodes. The nodes in a DOM tree are serialized into disk pages according to depth-first order (filiation clustering) or breadth-first order (sibling clustering). Lore [MAG97, MW99] and XBase [LWY+02] are two instances of such a storage approach.

The second approach is TIMBER project [JA02], at the University of Michigan, aiming to develop a genuine native XML database engine, designed from scratch. It uses TAX, a bulk algebra for manipulating sets of trees. For the implementation of its Storage Manager module, it uses Shore, a back-end storage system capable for disk storage management, indexing support, buffering, and concurrency control. With TIMBER, it is possible to create indexes on the document's attribute contents or on the element contents. The indexes on attributes are allowed for both text and numeric content. In addition, another kind of index support is the tag index, that, given the name of an element, it returns all the elements of the same name.

Finally, Natix [FM01] is proposed by Kanne and Moerkotte at the University of Mannheim, Germany. It is an efficient and native repository designed from scratch tailored to the requirement of storing and processing XML data. There are three features in Natix system: (1) subtrees of the original XML document are stored together in a single (physical) record; (2) the inner structure of subtrees is retained; and (3) to satisfy special application requirements, the clustering requirements of subtrees are specifiable through a split matrix. Unlike other XML DBMS which

provide fully developed functionalities to manage data, Natix is only a repository. It is built from scratch and has no query language, no much work done on indexing and query processing, and no use of DTDs or XML schema.

1.3 XML Query Language and Processing

To retrieve such tree-structured data, a few XML query languages have been proposed in the literature. Examples are Lorel [AQM+97], XML-QL [DFF98], XML-GL [CCD+99], Quilt [CRF00], XPath [BBC04], and XQuery [BCF03]. Of all the existing XML query languages, XQuery is being standardized as the major XML query language. XQuery is derived from the Quilt query language, which in turn borrowed features from several other languages such as XPath. The main building block of XQuery consists of path expressions, which addresses part of XML documents for retrieval, both by value search and structure search in their elements. For example, the following path expression */bib/book[author= 'Suciu']/title* asks for the title of the book written by "Suciu." In Fig. 1.1, this query returns the title *Advanced Database System*.

1.4 XML Keyword Search

The extreme success of web search engines makes keyword Search the most popular search model for ordinary users. As XML is becoming a standard in data representation, it is desirable to support keyword search in XML database. It is a user-friendly way to query XML databases since it allows users to pose queries without the knowledge of complex query languages and the database schema.

Most previous efforts in this area focus on keyword proximity search in XML based on either tree data model or graph (or digraph) data model. Tree data model for XML is generally simple and efficient for keyword proximity search. However, it cannot capture connections such as ID references in XML databases. In contrast, techniques based on graph (or digraph) can capture those connections, but the algorithms based on the graph model are very expensive in many cases. In this book, we will show interconnected object trees model for keyword search to achieve the efficiency of tree model and meanwhile to capture the connections such as ID references in XML by fully exploiting the property and schema information of XML databases. In particular, we will propose ICA (Interested Common Ancestor) semantics to find all predefined interested objects that contain all query keywords. We will also introduce novel IRA (Interested Related Ancestors) semantics to capture the conceptual connections between interested objects and include more objects that only contain some query keywords. Then a novel ranking metric, RelevanceRank, is studied to dynamically assign higher ranks to objects that are

more relevant to given keyword query according to the conceptual connections in IRAs. We will design and analyze efficient algorithms for keyword search based on our efficient, which outperforms most existing systems in terms of result quality.

1.5 Book Outline

The content of this book can be divided into five parts.

Part I is an introduction, which contains Chap. 1. Chapter 1 gives a brief introduction of XML, including the emergence of XML database, XML data model, and searching and querying XML data.

Part II discusses query processing, which focuses on tree pattern queries. Part II contains Chaps. 2, 3, 4, and 5. In Chap. 2, in order to facilitate query process over XML data that conforms to an ordered tree-structure data model efficiently, a number of labeling schemes for XML data have been proposed.

The emergence of the Web has increased interests in XML data. Without a structural summary and efficient index, query processing can be quite inefficient due to an exhaustive traversal on XML data. To overcome the inefficiency, several path indexes have been proposed in Chap. 3.

Answering twig queries efficiently is important in XML tree pattern processing. In order to perform efficient processing, Chap. 4 introduces two kinds of join algorithms, both of which play significant roles. Also, solutions about how to speed up query processing and how to reduce the intermediate results to save spaces are present in Chap. 4.

Previous algorithms focus on XML tree pattern queries with only P-C and A-D relationships. Little work has been done on extended XML tree pattern queries which contain wildcards, negation function, and order restriction, all of which are frequently used in XML query languages. Chapter 5 will show a set of holistic algorithms to efficiently process the extended XML tree patterns.

Part III discusses XML keyword search, which contains Chaps. 6, 7, and 8. Chapter 6 presents a survey on the existing XML keyword search semantics algorithms and ranking strategy. In XML keyword search, user queries usually contain irrelevant or mismatched terms, typos, etc., which may easily lead to empty or meaningless results. Chapter 7 introduces the problem of content-aware XML keyword query refinement and offers a novel content-aware XML keyword query refinement framework. Chapter 8 is an introduction to LCRA and LotusX, which provides a concise and graphical interface where users can explicitly specify their search concerns.

Finally, Part IV contains Chap. 9, which summarizes this book and presents several future works.

References

[Abi97] Abiteboul, S.: Querying semi-structured data. In: Proceedings of Database Theory, 6th International Conference, Delphi, Greece, pp. 1–18 (1997)

[AJP+02] Al-khalifa, S., Jagadish, H.V., Patel, J.M., Wu, Y., Koudas, N., Srivastava, D.: Structural joins: a primitive for efficient XML query pattern matching. In: Proceedings of the 20th International Conference on Data Engineering, San Jose, pp. 141–152 (2002)

[AQM+97] Abiteboul, S., Quass, D., Mchugh, J., Widom, J., Wiener, J.L.: The Lorel query language for semistructured data. Int. J. Digit. Libr. **1**(1), 68–88 (1997)

[BBC04] Berglund, A., Boag, S., Chamberlin, D.: XML path language (XPath) 2.0, W3C Working Draft 23 July 2004

[BCF03] Boag, S., Chamberlin, D., Fernandez, M.F.: XQuery 1.0: an XML query language. W3C Working Draft 22 Aug 2003

[CCD+99] Ceri, S., Comai, S., Damiani, E., Fraternali, P. Paraboschi, S., Tanca, L.: XML-GL: a graphical language for querying and restructuring XML documents. In: Proceedings of the 8th International World Wide Web Conference, Toronto, May 1999

[CFI+00] Carey, M.J., Florescu, D., Ives, Z.G., Lu, Y., Shanmugasundaram, J., Shekita, E.J., Subramanian, S.N.: XPERANTO: publishing object-relational data as XML. In: Proceedings of the 3rd International Workshop on the Web and Databases, Dallas, TX, USA (Informal proceedings), pp. 105–110 (2000)

[CRF00] Chamberlin, D.D., Robie, J., Florescu, D.: Quilt: an XML query language for heterogeneous data sources. In: Proceedings of the Third International Workshop on the Web and Databases, Dallas, Texas, USA, pp. 53–62 (2000)

[DFF98] Deutsch, A., Fernandez, M.F., Florescu, D.: A query language for XML, World Wide Web Consortium (1998)

[DFS99] Deutsch, A., Fernandez, M.F., Suciu, D.: Storing semistructured data with STORED. In: Proceedings ACM SIGMOD International Conference on Management of Data, Philadelphia, pp. 431–442 (1999)

[DWLL01] Dobbie, G., Wu, X., Ling, T.W., Lee, M.: ORA-SS: object-relationship-attribute model for semistructured data, Technical Report TR 21/00 National University of Singapore (2001)

[FK99] Florescu, D., Kossmann, D.: Storing and querying XML data using an RDMBS. IEEE Data Eng. Bull. **22**(3), 27–34 (1999)

[FM01] Fiebig, T., Moerkotte, G.: Algebraic XML construction in NATIX. In: Proceedings of WISE, Kyoto, Japan, pp. 212–221 (2001)

[FTS00] Fernandez, M.F., Tan, W.C., Suciu, D.: SilkRoute: trading between relations and XML. Comput. Netw. **33**(1–6), 723–745 (2000)

[JA02] Jagadish, H.V., Al-khalifa, S.: TIMBER: a native XML database, Technical report, University of Michigan (2002)

[JAC+02] Jagadish, H.V., Al-khalifa, S., Chapman, A., Lakshmanan, L.V.S., Nierman, A., Paparizos, S., Patel, J.M., Srivastava, D., Wiwatwattana, N., Wu, Y., Yu, C.: TIMBER: a native XML database. VLDB J. **11**(4), 274–291 (2002)

[JJ06] Jayapandian, M., Jagadish, H.V.: Automating the design and construction of query forms. In: Proceedings of the 22nd International Conference on Data Engineering, Atlanta (2006)

[JLS+04] Jagadish, H.V., Lakshmanan, L.V.S., Scannapieco, M., Srivastava, D., Wiwatwattana, N.: Colorful XML: one hierarchy isn't enough. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, pp. 71–82 (2004)

[LLHC05] Li, H., Lee, M.L., Hsu, W., Cong, G.: An estimation system for XPath expressions. In: Proceedings of the 22nd International Conference on Data Engineering, Tokyo, Japan, pp. 54–65 (2005)

[LWY+02] Lu, H., Wang, G., Yu, G., Bao, Y., Lv, J., Yu, Y.: XBase: making your gigabyte disk files queriable. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Madison, pp. 630–630 (2002)

[MAG97] Mchugh, J., Abiteboul, S., Goldman, R., Quass, D., Widom, J.: Lore: a database management system for semistructured data. SIGMOD Rec. **26**(3), 54–66 (1997)

[Mal99] Malaika, S.: Using XML in relational database applications. In: Proceedings of the 15th International Conference on Data Engineering, Sydney, pp. 167–167 (1999)

[ML02] Mo, Y., Ling, T.W.: Storing and maintaining semistructured data efficiently in an object-relational database. In: Proceedings of the 3rd International Conference on Web Information Systems Engineering, Singapore, pp. 247–256 (2002)

[MLLA03] Meng, X., Luo, D., Lee, M., An, J.: Orientstore: a schema based native XML storage system. In: Proceedings of 29th International Conference on Very Large Data Base, Berlin, pp. 1057–1060 (2003)

[MW99] Mchugh, J., Widom, J.: Query optimization for XML. In: Proceeding of the 25th International Conference on Very Large Data Bases, Edinburgh, pp. 315–326 (1999)

[PJ05] Pararizos, S., Jagadish, H.V.: Pattern tree algebras: sets or sequences. In: Proceedings of 31th International Conference on Very Large Data Bases, Trondheim, Norway, pp. 349–360 (2005)

[SSK+01] Shanmugasundaram, J., Shekita, E.J., Kiernan, J., Krishnamurthy, R., Viglas, S., Naughton, J.F., Tatarinov, I.: A general techniques for querying XML documents using a relational database system. SIGMOD Rec. **30**(3), 20–26 (2001)

[STZ+99] Shanmugasundaram, J., Tufte, K., Zhang, C., He, G., Dewitt, D.J., Naughton, J.F.: Relational database for querying XML documents: limitation and opportunities. In: Proceedings of 25th International Conference on Very Large Data Bases, Edinburgh, pp. 302–314 (1999)

[SYU99] Shimura, T., Yoshikawa, M., Uemura, S.: Storage and retrieval of XML documents using object-relational databases. In: Database and Expert Systems Applications, 10th International Conference, Florence, pp. 206–217 (1999)

[TVB+02] Tatarinov, I., Viglas, S., Beyer, K.S., Shanmugasundaram, J., Shekita, E.J., Zhang. C.: Storing and querying ordered XML using a relational database system. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Madison, pp. 204–215 (2002)

[YJR03] Yu, C., Jagadish, H.V., Radev, D.R.: Querying XML using structures and keywords in TIMBER. In: Proceeding of SIGIR, Toronto, pp. 463–463 (2003)

[ZND+01] Zhang, C., Naughton, J.F., Dewitt, D.J., Luo, Q., Lohman, G.M.: On supporting containment queries in relational database management systems. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Santa Barbara, pp. 425–436 (2001)

Chapter 2
XML Labeling Scheme

Abstract With the rapid development of the Internet, XML has become the widely popular standard of representing and exchanging data. Documents conforming to the XML standard can be viewed as trees. Elements in XML data can be labeled according to the structure of the document to facilitate query processing. To facilitate query process over XML data that conforms to an ordered tree-structured data model efficiently, this chapter shows a number of labeling schemes for XML data. We classify labeling schemes into two types: static labeling schemes and dynamic labeling scheme.

Keywords Static labeling scheme • Dynamic labeling scheme

2.1 Introducing XML Labeling Scheme

With the rapid development of the Internet, XML has become the widely popular standard of representing and exchanging data. Documents obeying the XML standard can be viewed as trees. Element in XML data can be labeled according to the structure of the document to facilitate query processing. Query language like XPath uses path expressions to traverse XML data. The traditional and most beneficial technique for increasing query performance is the creation of effective indexing. A well-constructed index will allow a query to bypass the need of scanning the entire document for results. Normally, a labeling scheme assigns identifiers to elements such that the hierarchical orders of the elements can be reestablished based on their identifiers. Since hierarchical orders are used extensively in processing XML queries, the reduction of the computing workload for the hierarchy reestablishment is desirable.

To facilitate query process over XML data that conforms to an ordered tree-structured data model efficiently, a number of labeling schemes for XML data have been proposed. We classify labeling scheme into two types: static labeling schemes and dynamic labeling scheme.

J. Lu, *An Introduction to XML Query Processing and Keyword Search*,
DOI 10.1007/978-3-642-34555-5_2,

When XML data are static, the labeling schemes, such as containment scheme (or called region encoding labeling scheme) [BKS02] and prefix scheme (or called Dewey ID labeling scheme) [CKM02, OOP+04, TVB+02], can determine the ancestor–descendant (A-D), parent–child (P-C), etc., relationships efficiently in XML query processing. Some variants have appeared for different purposes. For example, extended Dewey labeling scheme [LLCC05] is developed from Dewey ID labeling scheme [TVB+02]; the unique feature of this scheme is that from the label of an element alone, one can derive the name of all elements in the path from the root to this element. Twig pattern matching also can benefit from it, because TJFast only needs to access labels of leaf nodes to answer queries and significantly reducing I/O cost.

When XML data become dynamic, to efficiently update the labels of labeling scheme, a lot of dynamic XML labeling schemes have been designed for needs, such as region-based dynamic labeling scheme, prefix-based dynamic labeling scheme, and prime labeling scheme. However, most of the techniques have high update cost; they cannot completely avoid relabeling in XML updates. Therefore, we will introduce a compact dynamic binary string (CDBS) encoding [LLH08] and a compact dynamic quaternary string (CDQS) encoding [LLH08] which can be applied broadly to different labeling schemes to efficiently process order-sensitive updates.

2.2 Region Encoding Scheme

Elements in XML data can be labeled according to the structure of the document to facilitate query processing. The region encoding scheme uses textual positions of start and end tags. It can determine the ancestor–descendant (A-D), parent–child (P-C), etc., relationships efficiently in XML query processing if XML data are static.

In the region encoding scheme (or called containment scheme), every node is assigned three values: *start*, *end*, and *level*. *start* and *end* can be generated by counting node numbers from beginning of the document until the *start* and the *end* of the current node. *level* is the depth of the node in the document.

For any two nodes u and v:

1. **[ancestor–descendant]**: If u is an ancestor of v if and only if $u.start < v.start$ and $u.end < v.end$. In other words, the interval of node v is contained in the interval of node u.
2. **[parent–child]**: If u is the parent of v if and only if $u.start < v.start$ and $u.end < v.end$, $u.level = v.level$-1.
3. **[sibling]**: Node u is a sibling of node v if and only if the parent of node u is also a parent of node v.
4. **[preceding–following]**: Node u is a preceding (following) node of node v if and only if $u.start <(>) v.start$.