Developing an Indexing Scheme for XML Document Collections using the Oracle8*i* Extensibility Framework

Seema Sundara Ying Hu Timothy Chorma Jagannathan Srinivasan

{Seema.Sundara, Ying.Hu, Timothy.Chorma, Jagannathan.Srinivasan}@oracle.com

Oracle Corporation
One Oracle Drive, Nashua, NH 03062, USA

Abstract

Despite the success of the Oracle8*i* Extensibility Framework to index data from diverse domains (including text, images, spatial objects, chemical compounds, molecular structures, and genomic sequences), developing an indexing scheme is perceived as a difficult task, to be embarked upon only by experts, that too, for building support for complex domains. The goal of this demonstration is to show that: 1) the task of building and integrating an indexing scheme with the Oracle8*i* Extensibility Framework is quite simple and 2) the applicability of the framework is not limited to complex domains. We chose to develop an indexing scheme for XML document collections, since XML is becoming widely popular. Using the Oracle8*i* Extensibility Framework we will demonstrate 1) the ability to define domain operators with user-defined cost and selectivity functions, 2) the ability to define domain-specific indexing schemes, 3) the ability to specify user-defined index cost and statistics collection functions, 4) the ability to optimize queries involving domain operators via user-defined optimizer functions and 5) the ability to execute queries via domain indexes.

1. Introduction

For emerging new applications that deal with complex data, the major DBMS vendors like Oracle and IBM have introduced extensibility mechanisms that allow users to extend the DBMS functionality. These mechanisms include the ability to create user-defined types, user-defined functions or operators, as well as specialized indexing schemes. The Oracle8*i* Extensibility Framework includes:

- *Extensible Type System:* It enables the creation of domain specific object types with associated attributes and methods that define their behavior.
- *Extensible Indexing Framework:* It allows users to register new indexing schemes with the DBMS [7]. The user provides the code for defining the index structure, maintaining the index, and for searching the index data during query processing. The index structure can be stored in Oracle tables, and can exploit all the performance and scalability features of the Oracle8*i* server.
- *Extensible Optimizer Framework:* It allows users to provide cost and selectivity functions for user-defined predicates as well as the cost and statistics collection functions for domain indexes [5].

The task of developing the indexing and optimizer related extensions are generally perceived to be very complex. However, in this demonstration, we intend to show how a new indexing scheme, along with the optimizer related support, can be easily developed. Also, the domain of interest need not be complex like chem-informatics, but could be something simple like an XML document collection.

Currently, two approaches are generally adopted for storing, indexing, and querying XML data:

1) XML documents are parsed and the individual scalar values are stored as columns in a table. The table can be queried using relational predicates, which can make use of B^+-tree or bitmap indexes.

2) XML documents are stored as an opaque CLOB or a BLOB column in a table. The table is queried using the *Contains* operator which provides full-text search capability. Text indexes can be built, for example, using the Oracle interMedia Text product, which parses XML

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the 27th VLDB Conference, Roma, Italy, 2001

documents and builds a traditional inverted index structure for fast access [1].

Our approach is similar to 2) in that it stores the XML document in the server as an opaque CLOB column in a table. However, the table is queried using *ExistsNode* and *ExistsNodeWithValue* operators. The *ExistsNode* operator takes in a column value, and an xpath argument (a path to a node in an XML document [2]), and returns TRUE or FALSE depending on whether the specified xpath was found in the XML document. The *ExistsNodeWithValue* operator takes in a column value, an xpath, and a value, and returns TRUE or FALSE depending on whether the specified value exists at a leaf node that can be reached via the xpath specified. A specialized indexing scheme is developed for XML collections as described below.

2. Overview of the Indexing Scheme

When a domain index is created on a column with XML data, the documents stored in that column are parsed, and information about the structure and data of the XML documents is stored in two index-organized tables - the `eix_data` table, and the `eix_dataguide` table.

The sequence of slash-separated labels that are traversed to reach a node of an XML parse tree will be called the *path* of that node. The `eix_data` table stores the path to every leaf node in every XML document in the indexed column, along with the leaf's value, and the document's row identifier. This is similar to the edge table approach of storing XML data [3], with a few exceptions. Instead of storing each edge of the XML tree, only full paths to leafs are stored, making every target object a value. The key compression feature of index-organized tables [6] is used to reduce storage requirements by compressing common column prefixes in the path column.

The `eix_dataguide` table models a DataGuide [4], by storing the set of all paths (both leaf and branch) appearing in all indexed XML documents, with duplicates removed. Thus, the DataGuide provides a structural summary of the XML document collection. The DataGuide stores selectivity estimates for each path, which are used to assist in query cost estimation. The DataGuide can also be used to re-write queries involving an inexact path (e.g. '…/author') but this functionality was not implemented for this demonstration.

3. Demonstration

For the demonstration, we use a data set from the DBLP Bibliography consisting of about 200,000 citations (each an XML document) with a total size of 104MB. The demonstration will show the following aspects:

1) *Developing an Indexing Scheme in Oracle8i*: This will walk the user through the various steps involved. Namely, registering domain operators, indexing scheme, and optimizer functions.

2) *Query Optimization Using Extensible Optimizer Framework*: For a query involving one of the user-defined operators, the plan chosen by the optimizer will be examined before and after user-defined statistics have been associated with the index. Before the association, the optimizer will always choose the domain index to evaluate the query. After the association, and after analyzing the domain index, a different execution plan might be chosen based upon the costs estimated by the extensible optimizer routines. During statistics collection, a selectivity is stored along with each path in the `eix_dataguide` table, which the extensible optimizer routines use to derive approximate costs of index scans and functional evaluations. The Oracle8*i* EXPLAIN PLAN command will be used to show the query execution plan chosen.

3) *Query Execution Using Extensible Indexing Framework*: Query executions involving an *ExistsNode* operator will be compared before and after creating a domain index on a table containing XML documents. In the absence of the domain index, this query will use a functional implementation of *ExistsNode*, which parses documents from the table, searching for the specified path. In the presence of the domain index, an index-based scan will be used to evaluate the operator. Consider the following query:

```
SELECT citation FROM Bibliography WHERE
  ExistsNode(citation,'/book/author')=TRUE
```

The underlying index-based scan implicitly generates the following query on the `eix_data` table:

```
SELECT DISTINCT rowid FROM eix_data
  WHERE path = '/book/author'
```

The row identifiers returned by the query on the `eix_data` table identify the set of documents satisfying the query on the Bibliography table.

References

[1] S. Banerjee, V. Krishnamurthy, M. Krishnaprasad, R. Murthy: Oracle8*i* - The XML Enabled Data Management System. ICDE 2000: 561-568.

[2] J. Clark, S. DeRose, Xml path language (xpath) version 1.0 w3c recommendation. In http://www.w3.org/TR/xpath.html, 1999

[3] D. Florescu, D. Kossmann, A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database, Rapport de Recherche No. 3680 INRIA, France, 1999.

[4] R. Goldman, J. Widom: DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. VLDB 1997: 436-445

[5] Oracle Corporation, Oracle8*i* Data Cartridge Developer's Guide, Release 8.1.5, Part No. A68002-01, 1999.

[6] J. Srinivasan, S. Das, C. Freiwald, E. I. Chong, M. Jagannath, A. Yalamanchi, R. Krishnan, A.-T. Tran, S. DeFazio, J. Banerjee: Oracle8*i* Index-Organized Table and Its Application to New Domains. VLDB 2000: 285-296

[7] J. Srinivasan, R. Murthy, S. Sundara, N. Agarwal, S. DeFazio, Extensible Indexing: A Framework for Integrating Domain-Specific Indexing into Oracle8*i*, ICDE 2000: 91-100, 2000.