Managing E-Commerce Catalogs in a DBMS with Native XML Support

Lipyeow Lim
IBM T.J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532
lipyeow@us.ibm.com

Min Wang
IBM T.J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532
min@us.ibm.com

Abstract

Electronic commerce is emerging as a major application area for database systems. A large number of e-commerce stores provide electronic product catalogs that allow customers to search products of interest and store owners to manage various product information.

Due to the constant schema evolution and the sparsity of e-commerce data, most commercial e-commerce systems use the so-called vertical schema for data storage. However, query processing for data stored using vertical schema is extremely inefficient because current RDBMSs, especially its cost-based query optimizer, are specifically designed to deal with traditional horizontal schemas.

In this paper, we show that e-catalog management can be naturally supported in IBM's System RX, the first DBMS that truly supports both XML and relational data in their native forms. By leveraging on System RX's hybrid nature, we present a novel solution for storing, managing, and querying e-catalog data. In addition to traditional queries, we show that our solution supports semantic queries as well. Our solution does not require a separate query optimization layer, because query optimization is handled within the hybrid DBMS engine itself.

1 Introduction

Electronic commerce is emerging as a major application area for database systems. A large number of e-commerce sites provide electronic product catalogs that allow buyers, sellers, and brokers to search products of interest.

Imagine we are running a marketplace for a big retail chain such as Sears. The e-catalog of this marketplace may contain tens of thousands of products. Each product has its own set of attributes. For example, a "T-shirt" product in woman's shirt category may be associated with the attribute set $\{size, style, color, price\}$. Another product "TV set" in the electronics category may have a quite different attribute set $\{brand, view\_type, signal\_type, screen\_size, price\}$.

A natural technique for storage of the product information in a relational database system is the *horizontal schema*. Each product is represented as a row in a table, and the columns are the union of the attribute sets across all products (see Figure 1(a). However, this natural approach is not practical due to the following reasons [3]:

- The total number of attributes across all the products can be huge, but current commercial DBMSs do not permit a large number of columns in a table (e.g., both DB2 and Oracle permit only up to 1012 columns in a table).
- Even if a DBMS were to allow huge number of columns in a table, we would have a lot of nulls in most of the fields (e.g., any T-shirt product has a null value in $view\_type$ column). The large number of null values creates a big storage overhead and increases the size of indexes on these columns.
- Due to the large number of columns and null values, query performance would be very poor since the data records are very wide and only a few columns are used in a query.
- In an electronic marketplace, products traded and sold vary from day to day. We would need frequent altering of the table to accommodate new products. Maintenance and processing of altered tables can be quite expensive in RDBMSs.

An alternative is to use *binary schema* [4, 8]. We create one table for each attribute. Each table contains two columns, one is an attribute column and the other is an OID column that ties different fields of a tuple across tables (see Figure 1(c)). While there are no null values when using a binary schema, the number of join operations is usually large even when processing simple queries. Hence query performance is a big issue for the binary schema solution. An enhancement is to define a table per product set. This becomes unwieldy since the requirement is for dynamically adding and deleting new products.

Many commercial e-commerce systems use the so-called

OID	A_1	A_2	A_3	A_4	A_5
1	v_1	v_2	v_3		v_4
2	v_5	v_6	v_7		v_8
3	v_9	v_{10}			v_{11}
4			v_{12}	v_{13}	v_{14}
5			v_{15}		v_{16}
6				v_{17}	v_{18}

(a) Horizontal representation.

OID	*Attr*	*Value*
1	A_1	v_1
1	A_2	v_2
1	A_3	v_3
1	A_5	v_4
2	A_1	v_5
2	A_2	v_6
2	A_3	v_7
2	A_5	v_8
3	A_1	v_9
3	A_2	v_{10}
3	A_5	v_{11}
4	A_3	v_{12}
4	A_4	v_{13}
4	A_5	v_{14}
5	A_3	v_{15}
5	A_5	v_{16}
6	A_4	v_{17}
6	A_5	v_{18}

(b) Vertical representation.

OID	A_1
1	v_1
2	v_5
3	v_9

OID	A_2
1	v_2
2	v_6
3	v_{10}

OID	A_3
1	v_3
2	v_7
4	v_{12}
5	v_{15}

OID	A_4
4	v_{13}
6	v_{17}

OID	A_5
1	v_4
2	v_8
3	v_{11}
4	v_{14}
5	v_{16}
6	v_{18}

(c) Binary representation

Figure 1. An example of how the same data can be represented using the horizontal, vertical and binary schemas.

vertical schema design for their e-catalogs. Like the horizontal method, only one table is used. However, this table only has three main attributes: OID, $Attr$ (for attribute name), and $Value$ (for attribute value). [1] Each product is represented as a number of tuples in this table, one for each attribute-value combination and multiple attributes of a product are tied together using the same OID across multiple tuples in the table (see Figure 1(b)).

The vertical schema has the following advantages when compared to the others:

- *High flexibility:* The schema can handle any number of products and attributes.
- *Ease of schema evolution:* When products are added or deleted, alter or create table operations are not needed. We only need to add/delete tuples that correspond to the attributes of the added/deleted products.
- *Low storage overhead:* In contrast to horizontal schema, the vertical table does not contain null values.

Due to the above reasons, many commercial e-commerce systems (e.g., IBM Websphere Commerce Server, Ariba Marketplace, I2 Technology) utilize the vertical schema design for their e-catalogs.

Unfortunately, the vertical schema's flexibility introduces performance challenges for one important activity on e-catalogs, namely, *parametric search queries*. A parametric search query is a lookup of the e-catalog using specific constraints. Since a product usually contains many attributes, a search query in general is likely to impose several constraints that could be combined in a logical expression using AND/OR or even more sophisticated operators. Writing SQL queries against vertical schema could be cumbersome and error-prone. To solve this problem, Agrawal et al. proposed a method of creating a logical horizontal view on top of a vertical schema [3]. They also presented a set of query rewrite algorithms to convert relational algebra operators against the horizontal view to that against the vertical table. However, the performance of the converted query remains an issue. They show that queries against vertical schema performs no better than against binary schemas in most cases. The main reason for the poor performance is that there is a mismatch between the data model of a vertical schema and the storage model of traditional RDBMSs. The query optimization and planning techniques in traditional RDBMSs are designed for the horizontal schema data model where each table contains well-defined and meaningful attributes.

To achieve good query performance in an RDBMS when using vertical schema, a solution called *SAL*, Search Assistant Layer, is proposed in [10]. As a middle-ware solution, even though SAL can speed up parametric search on vertical schema significantly, it is not tightly integrated with the RDBMS engine and thus cannot fully leverage on the capability of the query optimizer to achieve the best possible performance.

Our Contributions. In this paper we propose a novel solution to store, query and manage e-catalog data using the hybrid relational-XML paradigm. In particular, a hybrid relational-XML DBMS called System RX [9] has already been prototyped by IBM and we describe our work in the context of System RX. We show that the hybrid relational-XML approach addresses many of the open issues in managing e-catalog data using the relational approach. Moreover, we show that the hybrid relational-XML approach exceeds the relational approach especially in its ability to support interesting semantic queries. In summary, the contributions of this paper are:

[1] In reality, the $Value$ column needs to be extended to accommodate values of different data types.

1. We proposed using a hybrid relational-XML DBMS to manage e-catalog data. Our hybrid relational-XML approach addresses (a) the schema evolution problem associated with using a pure relational approach, and (b) the query performance problems in using the vertical schema in a RDBMS.
2. We design a hybrid relational-XML database schema for storing e-catalog data and show how various important queries and data manipulation operations can be accomplished using either SQL/XML [5, 6] or XQuery [2].
3. We show that semantic queries, which are difficult to process in a pure relational system, can be easily expressed and processed in a hybrid relational-XML DBMS.

The rest of the paper is organized as follows. We introduce IBM's System RX, the first hybrid relational-XML DBMS in the next section. In Section 3 we describe our proposed approach to storing e-catalog data. In Section 4 we show how various product search operations can be accomplished using SQL/XML or XQuery. In Section 5 we outline how to maintain and manipulate the e-catalog data in a hybrid relational-XML DBMS as the e-catalog data changes. In Section 6, we show how a hybrid relational-XML DBMS can easily combine meta-data such as category hierarchy with the e-catalog data to support semantic queries. We summarize and draw conclusions in Section 7.

2 IBM's System RX: The First Hybrid Relational-XML DBMS

Most business data are stored in relational database management systems (RDBMSs) and accessed using SQL. With XML [1] becoming the standard for data retrieval and exchange, new functionality to support XML data is being expected from traditional RDBMSs.

When we store XML data in a traditional DBMS, the relational approach is not adequate for processing them: Either we decompose and store the XML data in relational format, in which case we get a major performance hit because we have to convert them to and from relational format whenever we store or retrieve them, or we have to store them as binary large objects, in which case we cannot do any processing with them. Ideally, a DBMS should be able to store XML data in its native form and support the search and manipulation of XML data as first-class citizens along with existing relational data types [7]. Towards this goal, IBM is building a hybrid database system, System RX, that handles both relational data and XML data in their native form [9].

System RX is a hybrid relational-XML DBMS that unifies new native XML storage, indexing and query processing technologies with existing relational storage, indexing and query processing [9]. The system thereby provides a natural path to XML for SQL applications, as well as scalable, transactional support for XQuery applications. It extends an existing RDBMS with the following components: (1) a native XML storage that stores an XML document as an instance of the XQuery Data Model (QDM), i.e., as a structured, typed, binary tree, (2) new index types for XML data including structural indexes, value indexes, and full-text indexes, (3) a hybrid query compiler that can process XQuery and SQL, and (4) an enhanced query runtime that supports XQuery and SQL/XML operators.

In System RX, a new data type, XML, is supported as a basic data type. Users can create a table with one or more XML type columns. A collection of XML documents can therefore be defined as a column in a table. For example, a user can create a table `ecatalog` with the following statement:

```
CREATE TABLE ecatalog(productID integer,
                      categoryID varchar(20),
                      info XML);
```

For this paper, we will use the `varchar` type for the category identifier in the interest of readability. The reader should understand that in a real e-commerce system, the category identifier is usually declared to be an integer for efficient processing.

To insert an XML document into a table, it must be parsed, placed into the native XML storage, and then indexed. We use the SQL/XML function, `XMLParse`, for this purpose:

```
insert into ecatalog values(1, "Women's shirt",
XMLParse('<?xml version='1.0'>
<productinfo id="1" category="women's shirt">
   <name>T-shirt</name>
   <size>XL</size>
   <color>Blue</color>
   <style>Round-neck</style>
   <listprice>20.00</listprice>
   <material>Cotton</material>
   <description>
      I love NY printed on the front
   </description>
   <supplier ID='19'>
      <name>Bob's Garment factory<name>
   <supplier>
</productinfo>'));
```

Users can query relational columns and XML column together by issuing SQL/XML query. For example, The following query returns product ids for all T-shirts that are red in color:

```
SELECT productID
FROM ecatalog AS C
WHERE XMLExists('$t/productInfo[ name =
      ''T-shirt'' and color =''Red'']'
      PASSING BY REF C.info AS "t")
```

Note that `XMLExists` is an SQL/XML boolean function that evaluates an XPath expression on an XML value.

If XPath returns a nonempty sequence of nodes, then `XMLExists` is true, otherwise, it is false. Since query optimization is handled within the hybrid DBMS engine, no further query optimization middle-ware is required for processing queries on XML data.

3 Storing E-Catalog Data in a Hybrid Relational-XML System

In this section, we describe how e-catalog data can be stored in a hybrid relational-XML DBMS. The main question we address is how to design the physical database schema for e-catalog data: what tables and columns are needed ?

The database schema needs to support the following operations:

- Searching for products that satisfy certain constraints on some attributes, eg., the user is shopping for a gift that is red in color and less than $20.
- Search for products within a given category, eg., find all products in the electronics category that is in the price range $100 – $200.
- Adding a new category of products, eg. a bookstore decides to offer audio compact discs (CDs) for sale as well.
- Removing a category.
- Adding a new product (possibly with new attributes).
- Removing a product.
- Removing all products associated with a given supplier.

Pure relational schemas are limited in their ability to handle the variability in the information associated with each product. We propose a schema that leverages on native XML support by localizing the highly variable product information into an XML column. In this case, product information and supplier information are stored in XML format. Our proposed schema creates one single table for the entire e-catalog,

```
CREATE TABLE ecatalog(
              productID integer,
              categoryID varchar(20),
              info XML);
```

The `ecatalog` table is illustrated in Figure 2. Each row in the `ecatalog` table corresponds to a single product. Each product has an unique product identifier and is associated with a category (denoted by the category identifier[2])

An alternative to the proposed schema is a schema that creates one table for each category, for example,

[2]Recall that we use strings for category identifiers in the interest of readability. Real systems use numeric category identifiers for efficient processing.

Figure 2. The proposed `ecatalog` table. The `info` column is of XML type and contain pointers to XML data stored natively as trees.

```
CREATE TABLE womensshirt(
              productID integer,
              info XML);
CREATE TABLE electronics(
              productID integer,
              info XML);
```

Although this alternative schema is also able to support all the important e-commerce operations listed above, its main drawback is that a new table needs to be created whenever a new category is added.

In addition to the table schema we proposed, XML schemas can be associated with each category of products. For example, the fragment of an XML schema for the products in women's shirt category might look like,

```
<xsd:complexType name = "productInfo">
  <xsd:attribute name = "id"
                 type = "xsd:integer" />
  <xsd:attribute name = "category"
                 type = "xsd:string" />
  <xsd:sequence>
     <xsd:element name = "style"
                  type = "xsd:string"
                  minOccurs = "0" />
     <xsd:element name = "listprice"
                  type = "xsd:decimal"
                  minOccurs = "1"
                  maxOccurs = "1" />
  </xsd:sequence>
</xsd:complexType>
```

The use of XML schemas is completely optional. The system designer may choose not to use any XML schemas at all if she/he does not wish to impose any restrictions on the type of XML data that can be stored in the system. System RX allows the user to leverage the full flexibility of XML by not mandating XML schemas be used. However, when XML schemas are used, it is mainly for two reasons: (1) it allows the hybrid relational-XML DBMS to verify the integrity of the XML data (for example type checking), and (2) it serves as a structural guide for writing queries XQuery. System RX provides a mechanism for validating XML data against a given XML schema. The XML schemas first need to be registered and stored in the XML schema repository. The SQL/XML command `XMLValidate` can then be invoked to validate an XML document against a given schema.

When using XML schemas, the system designer needs to decide the granularity of the XML schema. Usually, having one XML schema per category is the appropriate granularity, because we expect less variability in the XML structure for products within the same category. However, as long as the mapping between which XML schema describes which XML document is maintained, any number of XML schemas can be used.

4 Product Search on E-Catalog

Product search is arguably the most important operation on an e-catalog. In this section we show how product search queries can be formulated in both SQL/XML and XQuery.

4.1 Product Search within a Given Category

One of the most common type of queries in an e-commerce system is to find all products within a given category that satisfy certain constraints on some attributes. Since product information are stored as XML, these constraints on product attributes correspond to XPath expressions. Writing such queries is then equivalent to specifying constraints on the XPath expressions. For example, a shopper might want to find all women's shirt that are red in color, that are made from the material Lycra, and whose list price is under $20. We can formulate this query in SQL/XML as,

```
SELECT productID
FROM ecatalog AS C
WHERE XMLExists('$t/productInfo[color =''Red''
      and material=''Lycra''
      and listprice < 20]'
      PASSING BY REF C.info AS "t")
      AND categoryID="women's shirt";
```

In this query, the constraints on color corresponds to the path expression `/productInfo/color`, the attribute material to `/productInfo/material`, and the constraints on list price to `/productInfo/listprice`. The search is restricted to the women's shirt category by the putting the condition on the `categoryID` column in the `WHERE` clause.

Equivalently, we can also express the same query using XQuery as

```
for $p in xmlcolumn('ecatalog.info')
    /productInfo
where $p/@category = "Women's shirt"
      and $p/color = "red"
      and $p/material= "Lycra"
      and $p/listprice < 20
return
  <product>
    <id> {$p/@id} </id>
    <name> {$p/name} </name>
  </product>;
```

The `xmlcolumn` function takes as input the name of an XML column. The result of a query formulated as an XQuery statement is returned in XML format.

In general, if the desired output format is relational, the query should be formulated in SQL/XML. If the desired output format is XML, the XQuery language should be used.

4.2 Product Search across Multiple Categories

Another common type of queries is to find products across multiple or all categories given certain constraints. For example, a shopper is shopping for a birthday present. She/he has a budget under $20 and requires that the present be red in color. The following SQL/XML query may be used,

```
SELECT productID
FROM ecatalog AS C
WHERE XMLExists('$t/productInfo[color =''Red''
      and listprice < 20]'
      PASSING BY REF C.info AS "t");
```

Since the search is over all categories, there is no constraint on the `categoryID` column.

The same query expressed in XQuery is:

```
for $p in xmlcolumn(
      'ecatalog.info')/productInfo
where $p/color = ''red''
      and $p/listprice < 20
return
  <product>
    <id> {$p/@id} </id>
    <name> {$p/name} </name>
  </product>;
```

4.3 Product Search using Keywords

Keyword search is yet another important type of query for e-commerce systems. For example, when searching for a book, we often do not remember the title exactly, but we do remember a few keywords that appear in the title. Suppose we want to find all books that have the keyword "algorithms" in the title, we can write the following query,

```
SELECT productID
FROM ecatalog AS C
WHERE XMLExists('$t/productInfo[fn:contains(
      booktitle,''algorithms'')]'
      PASSING BY REF C.info AS ''t'')
      AND categoryID=''Books'';
```

The XQuery function `fn:contains` returns true if `booktitle` contains "algorithms" as a substring. While relational DBMSs do not support keyword search efficiently, full-text indexes can be used in hybrid relational-XML DBMS (such as System RX) to process keyword search very efficiently.

4.4 Optimizing Search with Indexes

For path expressions that are frequently queried, System RX allows indexes to be created on these path expressions. For example, in the queries mentioned above, the list price of a product is queried frequently. We can therefore create an index on list price in order to speedup the range predicates on list price:

```
CREATE INDEX listpriceindex ON ecatalog
USING //listprice AS DOUBLE;
```

If query workloads are available and there is a high probability that these workloads are representative of future queries, frequently queries path expressions can be extracted from these workloads and indexes can be created on these path expressions.

5 Maintenance of E-Catalog Data

Besides search, an e-catalog system often needs to support operations like adding a new product category, deleting an outdated product category, adding new products, and deleting old products.

Adding a new product category is a relatively simple operation. When a product belonging to the new category is inserted into the `ecatalog` table, the new category is automatically added. Removing a category requires all rows (i.e. all products) associated with the category to be deleted. For example, if the women's shirt category is to be deleted, the following SQL statement can be used.

```
DELETE FROM ecatalog
WHERE categoryID = "women's shirt";
```

In our schema design, we assume that each product corresponds to a row in the `ecatalog` table. Adding and removing a product is therefore equivalent to removing the row associated with the product.

We consider a more interesting delete scenario next. Suppose the e-store no longer buys products from a particular supplier with ID 19. All the products that are associated with this supplier needs to be removed. However, some products may be supplied by more than one supplier. Hence, we need to delete products with only one supplier whose supplier ID is 19. For products that have more than one supplier and of which supplier 19 is a supplier, the XML info for these products needs to be updated. To accomplish the deletion, we can use the following SQL/XML statement,

```
DELETE FROM ecatalog
WHERE XMLExists('$t/productInfo[
      fn:count(./supplier) = 1
      and ./supplier/@ID=19]'
      PASSING BY REF ecatalog.info AS "t");
```

The XPath expression `$t/productInfo[fn:count(./supplier) = 1 and ./supplier/@ID = 19]` returns a productInfo node in the XML document that has exactly one supplier child node and the supplier identifier is 19.

To update the products supplied by supplier 19 and that also have more than one supplier, we use the SQL/XML `XMLUpdate` function,

```
UPDATE ecatalog
SET info = XMLUpdate(info,
    '/productInfo/supplier[@ID=19]',
    XMLParse(''))
WHERE XMLExists('$t/productInfo[
      fn:count(./supplier) > 1
      and ./supplier/@ID=19]'
      PASSING BY REF ecatalog.info AS "t");
```

The `XMLUpdate` function takes as its first argument an XML value, as its second argument, the XPath expression to be evaluated on the XML value, and as its third argument the XML value that will replace the node specified by the second argument. The resultant XML value is returned.

XML Schema Evolution. Product offering on an e-commerce website change over time. It is almost certain that the structure of product information will also need to change. For example, after the MP3 audio format had become popular, compact disc players started to have a new attribute "MP3 compatible". This addition (and removal) of attributes is one reason why the relational model is inadequate to handle e-catalog data. The traditional Entity-Relationship model will require the table schemas to change. Vertical schemas addresses this schema evolution problem but at the cost of performance.

In our proposed solution using a hybrid relational-XML DBMS, changes in the structure of the product information is no longer a problem, because we leveraged on the flexibility of XML to store the product information. In the case where no XML schemas are used, no further processing is required other than adding the new information into the XML column (for example, adding the XML element `<feature>MP3-compatible</feature>` to the product information for compact disc players). If XML schemas are used, the XML schema needs to incorporate the addition of such new attributes. It is possible that XML data in the `ecatalog` may need to be re-validated as a result of the XML schema change. Managing XML schema evolution is still an open problem and is part of our future work.

6 Supporting Semantic Queries

6.1 Queries using the Category Hierarchy

E-commerce stores often have a hierarchy of categories and sub-categories to help users (i.e. shoppers) navigate their diverse array of product offerings. An example of such

Figure 3. A subset of the category hierarchy from the Yahoo website. The sub-category relationship is denoted by indentation.

a hierarchy is shown in Figure 3. Users can therefore also search for products that fall in any sub-category. For example, one user might want to browse through all products in the `BookMagazines->SportsRecreation` category, another user might want to only browse through products in the `BookMagazines->SportsRecreation ->Outdoor->Hiking` category.

Such queries are hard to process in a relational system. First, the relational data model is inherently not suitable for storing the hierarchical data. Hence the hierarchical tree-structured data will need to be shredded into an edge-list representation that can be stored in a RDBMS. Second, expressing queries that requires tree-traversal in SQL is awkward. Recursive procedures can be used, but they require many self-joins on the edge-list data.

A hybrid relational-XML DBMS, on the other hand, is especially suited for supporting such hierarchical data and the associated queries. We can create a `category` table with one XML column named "`info`". The category hierarchy can be stored as XML data in the `category.info` column. Each subcategory name becomes an XML element optionally with an `ID` attribute that stores the associated category identifier. To simplify the design of the entire e-catalog system, the category identifier column of the `ecatalog` table should only contain the identifiers of subcategories that are leaves in the `category.info` hierarchy.

To find all products in the `BookMagazines ->SportsRecreation` category, the following SQL/XML query can be used,

```
SELECT productID
FROM ecatalog AS E, category AS C
     XMLTable('$t/Shopping/BookMagazines
     /SportsRecreation//*[fn:empty(*)]'
     PASSING C.info as "t",
     COLUMNS "categoryID" VARCHAR(20)
     PATH 'fn:string(@ID)') AS X
WHERE E.categoryID = X.categoryID;
```

The path expression `/Shopping/BookMagazines /SportsRecreation//*[fn:empty(*)]` finds all leaf nodes in the subtree at `/Shopping/BookMagazines /SportsRecreation`. The `XMLTable` function constructs a virtual table aliased "X" consisting of one column named "categoryID" whose values are the "`ID`" attribute of the leaf nodes. Conceptually, the query contracts a list of category IDs that belong to `BookMagazines ->SportsRecreation` and returns all products whose category ID belongs to the constructed list.

To find all products in the `BookMagazines ->SportsRecreation->Outdoor->Hiking` category, we first determine if the path `//BookMagazines /SportsRecreation/Outdoor/Hiking` leads to a leaf node. If it is a leaf node, the query corresponds to a simple retrieval using the associated category ID. If it is not a leaf node, the query reduces to the `XMLTable` query in the previous example.

6.2 Queries using Semantic Relationships

A more interesting type of semantic queries is when the user specify a descriptive keyword and wants to search for all products associated with the given keyword. For example, a shopper is shopping for a present for a friend who likes "orienteering"[3]. Ideally, the e-commerce system should be able to return all products associated with the activity "orienteering". The problem is that the keyword "orienteering" is not a category name. In the absence of further information, the best that we could do is to find all products whose description or booktitle element contains the word "orienteering".

Suppose the e-commerce system has access to some ontological data that describes the semantic relationship between certain keywords and certain product categories. In our example, suppose there is an `ontology` table with an XML column named `info` that contains the following XML fragment

```
<activity>
   <keyword>hiking</keyword>
   <keyword>orienteering</keyword>
   <keyword>walking</keyword>
   <related>
      <category>BooksMagazines</category>
      <keyword>Maps</keyword>
      <keyword>Hiking</keyword>
```

[3]Orienteering is an outdoor activity where a person uses a map and compass to navigate through some terrain.

```
   </related>
   <related>
      <category>Telecommunications<category>
      <keyword>GPS</keyword>
   </related>
   <related>
      <category>Shoes<category>
      <keyword>Hiking Boots</keyword>
   </related>
   <related>
      <category>CampingHiking<category>
      <keyword>Walking stick</keyword>
   </related>
</activity>
```

Using these semantic relationship, we will be able to associate the activity "orienteering" with several product categories. Moreover, for each associated category, additional keywords are supplied so that relevant products can be retrieved from that category. Note that such meta-data can also be (semi-)automatically obtained by performing market-basket data-mining on data-warehouses and historical sales information.

To find all products semantically related to the activity "orienteering", we can issue the following XQuery,

```
for $r in xmlcolumn('ontology.info')
   /activity[keyword="Orienteering"]/related
for $k in $r/keyword
for $p in xmlcolumn('ecatalog.info')
   /productInfo[@category=$r/category
       and fn:contains(description,$k)]
return
<product>
   {$p/@id}
</product>;
```

The first `for`-loop iterates through each related category associated with the "orienteering" activity, the second `for`-loop iterates through each additional keyword supplied under each related category, and the last `for`-loop iterates through each product checking for matches to each pair of category and keyword. Note that if the category from the ontology is not a leaf category in the `category` hierarchy, we can always use another `for`-clause to generate all the leaf category IDs using the `category` hierarchy. In summary, this query will return products associated with hiking books, maps, global positioning (GPS) devices, hiking boots, and walking sticks.

7 Conclusions

In this paper, we have introduced a novel solution to store, manage and query e-commerce catalog data using a hybrid relational-XML DBMS. We have designed an appropriate database schema for the e-catalog that localizes product information into a native XML-type column. Our solution leverages on the flexibility of XML to address the high variability in the structure of e-catalog data and thus is able to avoid the relational schema evolution problem faced by relational solutions. Moreover, our solution does not suffer from the query optimization problems faced by the vertical schema approach, because there is no mismatch in the structure of the data and the storage model of a hybrid relational-XML DBMS. Query optimization is handled within the hybrid DBMS engine and hence no separate optimization middle-ware is required. We have shown that the e-catalog data can be easily queried and maintained in a hybrid relational-XML DBMS. Besides conventional queries, our solution is also capable of processing more interesting queries, such as semantic queries, that are not well handled in relational systems.

References

[1] Extensible markup language (XML) 1.0 (third edition). `http://www.w3.org/TR/REC-xml`.

[2] XQuery 1.0: An XML query language. `http://www.w3.org/TR/xquery/`.

[3] Rakesh Agrawal, Amit Somani, and Yirong Xu. Storage and querying of e-commerce data. In *VLDB*. Morgan Kaufmann, 2001.

[4] George P. Copeland and Setrag Khoshafian. A decomposition storage model. In Shamkant B. Navathe, editor, *SIGMOD*, pages 268–279. ACM Press, 1985.

[5] Andrew Eisenberg and Jim Melton. SQL/XML is making good progress. *SIGMOD Record*, 31(2):101–108, 2002.

[6] Andrew Eisenberg and Jim Melton. Advancements in SQL/XML. *SIGMOD Record*, 33(3):79–86, 2004.

[7] J. E. Funderburk, S. Malaika, and B. Reinwald. XML programming with SQL/XML and XQuery. *IBM Systems Journal*, 41(4), 2002.

[8] Setrag Khoshafian, George P. Copeland, Thomas Jagodis, Haran Boral, and Patrick Valduriez. A query processing strategy for the decomposed storage model. In *ICDE*, pages 636–643. IEEE Computer Society, 1987.

[9] Fatma Ozcan, Roberta Cochrane, Hamid Pirahesh, Jim Kleewein, Kevin Beyer, Vanja Josifovski, and Chun Zhang. System RX: One part relational, one part XML. In *SIGMOD*, 2005.

[10] Min Wang, Yuan chi Chang, and Sriram Padmanabhan. Supporting efficient parametric search of e-commerce data: A loosely-coupled solution. In *EDBT*, volume 2287, pages 409–426. Springer, 2002.