4

Towards Business Intelligence over Unified Structured and Unstructured Data Using XML

Zhen Hua Liu and Vishu Krishnamurthy
Oracle University,
USA

1. Introduction

Traditional data warehousing has been very successful in helping business enterprises to make intelligent decisions through declarative analysis of large amount of structured data stored in a relational database. However, not all enterprise data naturally fit into a relational model. Within an enterprise, there are huge amount of unstructured data, such as document content, emails, spreadsheets, that do not have a fixed schema, or have a very sparse or loose schema that cannot be effectively modeled using relational model. Yet, like relational data, unstructured data record many useful facts that are equally essential and important to be analyzed by businesses to make intelligent decisions. In this chapter, we propose an XML-enabled RDBMS that uses XML as the underlying logical data model to uniformly represent both well-structured relational data, semi-structured and unstructured data in building an enterprise data warehouse that is able to store and analyze any data regardless of existence of schema or not. We show how XQuery used in SQL/XML as a declarative language to do data query, analysis and transformation over both structured data and unstructured content in the data warehouse. We present the rationale for using XML as the logical data model for unified data warehouse query, XML extended inverted text index to integrate structured data query and context aware full text search for unstructured content so as to support efficient data analysis over large volume of structured and unstructured data. We argue that the technical approach of using XML to unify both structured and unstructured data in a warehouse has the potential to push business intelligence over all enterprise data to a new era.

2. Concept of a data warehouse

Inductive reasoning refers to human arriving at a conclusion based on their observations. The inductive reasoning is a bottom up process where a general conclusion is reached from many instances observed and analyzed. (Myers,1986) Data Warehouse and decision support capabilities in modern database management system (DBMS) reflect the human inductive reasoning process. Data Warehouse (DWH) and decision support system (DSS), typically based on an RDBMS, involve extraction of operational data from business activities, transformation of the operational data, and loading of the results conforming to a fixed relational data model into a DWH store. Sophisticated data transformation, analysis, and mining can then be applied to a DWH to derive useful conclusions that assist businesses in

making intelligent decisions. Such evidence based decision-making process achieved through DWH is generally accepted as standard business intelligence practices in Enterprises.

To achieve the goal of business intelligence, the design of DWH in DBMS must address the following requirements that are different from operational data store in Online Transaction Processing (OLTP) environment: Data Heterogeneity, Data Extraction and Batch Loading, Large Data Volumes, and Declarative Ad-Hoc Query Performance.

Data Heterogeneity: since DWH store loads data from different operational store, therefore, it is likely that data may not be as homogeneous as operational store is. That is, data may not have well-defined common schema or may not have schema at all. The general trend is that unstructured data content and semi-structured data are more common than well-structured data to process and to query.

Data Extraction and Batch Load: building decision support system involves extraction of data from various operational stores and bulk loading of them into a central DWH store. This is known as ETL (ExTract Load) process. Data transformation is applied during ETL process to convert data from different operational stores into canonical form. To handle large data volume, tables can be partitioned and managed by several data server instances in a clustered environment. Query can use table partition criteria as selection qualification to work on different partitions of data. Data in DWH is typically partitioned based on certain criteria, such as timestamp based range partition criteria or hash partition based on record key or hybrid combination of the two. Such partition scheme facilitates life cycle management of data and enables query parallelism.

Large Data Volume: Given the fast growing of memory core, it is reasonable to assume that operational data are able to all fit in memory such that in memory database processing becomes very attractive to overcome the gap between disk I/O speed and CPU speed. However, the amount of data for DWH store shall never be assumed to fit in main memory. Therefore, DWH design must take into consideration of selecting data layout to be disk I/O friendly. For example, design favoring small number of sequential large I/O requests generally delivers better performance than that of large number of random small I/O requests. Consequently, DWH design usually lays out data in a way to be optimized for large number of read requests instead of laying out data to be optimized for a large number of random frequent data modification requests.

Declarative Ad-hoc Query Performance: declarative query is an attractive property for DBMS so that users can declaratively specify what they want to ask instead of procedurally programming the system on how to obtain the answer. Declarative query language processing with superior performance is critical for the success of DBMS. For operational store, the supported operations over data are usually pre-determined, therefore, the data query and modification requests have deterministic patterns. Operational store query is typically point query using id lookup that selects small amount of data using simple query criteria. However for DWH environment, the query requests are ad-hoc and exploring in nature. The query pattern is less predictive than that of operational store. DWH Query typically involves processing large amount of data to get summarized report to facilitate decision-making process or to mine data to derive insightful conclusion based on statistical analysis. So DWH query can be long running compared with short running point query in operational store.

3. Data warehouse in SQL/RDBMS – its success and limitations

Based on a strong foundation in relational model and algebra (Codd, 1970), relational database management system (RDBMS) is a great success. The practical realization of relational model using Entity/Relationship (E/R) design (Chen,1975) greatly facilitates users to model real world objects into entities with relationships so that they can be managed by RDBMS and queried or modified declaratively using SQL. RDBMS has been very successful in supporting On Line Transaction Processing (OLTP) workloads. Following the success of OLTP, businesses have successfully built DWH using RDBMS to make intelligent business decisions. The strength of DWH using SQL and RDBMS is described in section 3.1.

3.1 Strength of current DWH practices with SQL and RDBMS

3.1.1 Well-defined relational model for structured data

E/R design models structured data objects, that is, objects having well-defined schema, very well. Objects are decomposed into entities (tables) with primitive attributes (columns). Hierarchical tree shaped objects are modeled by one-to-one and one-to-many relationships among entities via primary and foreign key relationships among tables. Graphical objects are modeled by many-to-many relationships among entities via intermediate key mapping tables. Many-to-many relationship enables construction of objects with different hierarchies. Given that data is frequently updated in OLTP workload, high degree of data normalization provides good update performance because same data is not duplicatively stored so that data update occurs only in one place.

DWH schema design in RDBMS follows the same E/R design principle. The common practice is to use star schema where there is a fact table with a set of dimension tables. An example is shown in Tables 1,2,3,4,5 below. Fact table records business transactions. A fact record is a WWWW tuple (Who purchased What,Where and When). In the example below, Table 1 is a Fact table, while Tables 2-1,2,3,4,5 are dimension tables containing Customer, Item, Store and Date dimensions respectively. There is a primary key on id column in each dimension table. The id column of a dimension table is referred as *dimension id*. The dimension id is also stored in the fact table. There is a foreign-key and primary-key relationship between each column of fact table that stores the dimension id and its corresponding referencing dimension table. In addition, for each dimension table, there are *dimensional columns* that specify composite values for that dimension. DWH SQL query involves joins among fact table and multiple dimension tables shown as Query 1. The WHERE clause of DWH SQL query may use combinations of multiple predicates on dimensional values and thus is ad-hoc in nature.

CustomerId	ItemId	StoreId	DateId	QuantitySold	Description
1454	1456	123	13579	2	<sale_comment> This is sold via special summer sale promotion program at the store. The promotion program is conducted along with the independence celebration event in the city. </sale_comment>

Table 1. Transaction fact table

CustomerId	CustomerName
1454	John Smith

Table 2. Customers dimension table

ItemId	ItemName	ItemPrice	ItemCategoryCode
1456	T.V	$250.32	Electronic

Table 3. Items dmension table

DateId	Date	Month	Year
13579	4	July	2004

Table 4. Dates dimension table

StoreId	StoreName	StoreZipCode
123	Eletronic-Supply	45789

Table 5. Stores dimension table

```
SELECT StoreId, count(*)
FROM TransactionFact f, Customers c, Items i, Dates d, Stores s
WHERE f.CustomerId = c.CustomerId AND f.ItemId = i.ItemId AND f.StoreId = s.StoreId AND
f.DateId = d.DateId AND i.ItemName = 'TV' AND s.StoreZipCode >= 45000 and
s.StoreZipCode <= 46000 AND i.ItemPrice < 300 AND d.Year between 2003 and 2005
GROUP BY StoreId
```

Query 1. SQL on Star Schema

3.1.2 Declarative SQL language with high performing SQL engines

SQL is powerful enough to express analytical queries declaratively. Beyond basic conventional aggregation functions, such as sum() and avg(), contemporary RDBMS supports data mining capabilities as built-in functions so that statistical analysis of data can be done declaratively (Milenova et al., 2005). Through database extensibility work from Object Relational DBMS (Stonebraker et al.,1998), user defined aggregation functions and table functions are supported by contemporary RDBMS so that customized analytical logic over virtual table row sources can be integrated into the SQL engine.

DWH has promoted fruitful research results and industrial practices in the past three decades resulting in RDBMSs with well-engineered optimizers and executors to support DSS type of SQL query workload. Advanced algebraic query transformation based on mathematical property of relational set algebra (Seshadri et al., 1996), sophisticated statistics gathering and dynamic sampling techniques, parallel query execution infrastructure (DeWitt & Gray 1992) are well-developed compile time and execution time techniques to speed up SQL query for DWH workloads. Furthermore, the recent trend of hardware acceleration to speed up query execution is a new direction in greatly improving SQL query performance.

3.1.3 I/O friendly bitmap join index structures

The logical query plan of Query 1 on star-schema consists of three phases. The first phase is to probe each Dimension table to find a set of dimension ids given the dimension value.

Then use resulting dimension ids to find a set of row ids of the rows in Fact table that have foreign key value matching the dimension ids using primary key and foreign key relationship between the Fact table and each Dimension table. The *row id* (RID) in RDBMS represents a physical locator to a row, which is typically composed of file id, page number and slot number of a page where the row resides on a page. The second phase is to perform set intersection of the set of RIDs of the Fact table to get a common set of RIDs that satisfy all the AND qualifications in the where clause. The third phase is to use RIDs from second phase to probe the Fact table to do further processing, such as Group By and Aggregation. This phase may involve probing Dimension tables again using dimension ids from the Fact table row to obtain dimension column values if the query selects dimension column value.

Given that a Dimension table is typically small enough to fit in memory, table scans can be used to find a set of dimension ids given a dimension value. However, finding the set of RIDs of the Fact table having these dimension ids via table scan of Fact table yields poor performance due to the large size of the Fact table. The first attempt to improve the performance is to leverage B+ tree index on foreign key columns of Fact table containing dimension ids so that RIDs of the Fact table can be found via B+ tree index access. However, B+ tree index results in many random I/O requests. Excessive seeks followed by small reads after each seek is not I/O friendly.

To resolve B+ tree index performance problem, Bitmap Join Index (O'Neil & Graefe, 1995) can be created between the foreign key column of the Fact table containing the dimension ids and its referencing dimension id column of the Dimension table. In bitmap join index, each row is given a *row ordinal position* (ROP). The bitmap join index maps a dimension value of the Dimension table to a bitmap representing ROPs of the fact table that having that dimension value. Bitmap can be compressed so that small amount of I/O is needed to load all the relevant bitmaps into memory. During second phase, set intersection using bitmaps of ROPs can be executed much more efficiently than that of using RIDs. At the end of second phase, ROPs are converted into RIDs for further processing.

3.1.4 I/O Friendly columnar storage

Columnar storage (Stonebraker et al., 2005) for DWH is an I/O friendly approach to store data in columnar fashion in order to avoid unnecessary disk I/O to read data that is not needed by the query. Furthermore, when all values of a column are stored together, it is more amenable to data compression that reduces amount of I/O and facilitates query processing strategies that directly operate on compressed columnar data (Abadi et al., 2008). We will explore this further in section 5.

3.2 Limitations of current DWH practices with SQL and RDBMS

3.2.1 Limited capability of handling unstructured data

Unlike structured data that has a static schema that fit into the relational model, we define *unstructured data* to represent loosely structured data, arbitrarily structured data, or data with high degree of schema variability. Such data do not fit into the structured relational model. For data having highly varying schema, fitting it into relational model requires constant schema evolution which is not a scalable and maintainable solution. It is more natural to model such data as semi-structured data or as a string of attribute-value pairs.

Without a schema, such semi-structured data can only be stored in LOB (CLOB or BLOB) columns in an RDBMS. Data without conforming to a rigid schema is increasingly becoming common in enterprises. Like structured data, there is valuable information embedded inside unstructured data that can help business enterprise to make better decision. Users are looking to store and query unstructured data without defining a schema first. This exposes a limitation of relational model that requires data to have schema before they are storable and queryable.

3.2.2 Limited queriability of unstructured data

While unstructured data could be stored as LOBs in the RDBMS, there is effectively no means to query the data inside these lobs. Popular RDBMSs have been extended to support Full-text functionality that can be exercised through SQL. Therefore, text in these lobs could be queried using Full-text search. For example, in the transaction Fact table, there is a description column that records the comments of each purchase transaction. Users can query transaction comments using text CONTAINS() function that does keyword search as shown in Query 2. Users can create an inverted text index on description column of the fact table to speed up CONTAINS() function. Organization of inverted text index follows the traditional Information Retrieval (IR) technique (Salton & McGill, 1983) where a posting list is created for each keyword. Given a keyword, the posting list identifies a set of DOCIDs (Zobel & Moffat, 2006). The DOCID is the same as that of ROP, which is a simple integer based sequence number in contrast with RID which is concatenated raw bytes from different components. It identifies the row of the Fact table that has the keyword in the description LOB column.

```
SELECT StoreId, count(*)
FROM TransactionFact f, Customers c, Items i, Dates d, Stores s
WHERE f.CustomerId = c.CustomerId AND f.ItemId = i.ItemId AND f.StoreId = s.StoreId AND
f.DateId = d.DateId AND i.ItemName = 'TV' AND s.StoreZipCode >= 45000 and
s.StoreZipCode <= 46000 AND i.ItemPrice < 300 AND d.Year between 2003 and 2005 AND
CONTAINS(description, 'promotion')
GROUP BY StoreId
```

Query 2. SQL on Star-Schema with Text Search

However, traditional IR (Salton & McGill, 1983) keyword based full text search is restricted to text content search only, it doesn't address querying structured data with text search, nor does it address the capability of providing context aware text search (Yates & Navarro 1996). There is no support for efficiently querying structure and content together. We will explore and resolve this limitation in section 4 and section 5 using XML data model and XML extended text index.

3.2.3 Lack of query optimizer and engine that is able to optimize both structured and unstructured data query

Consider how Query 2, which consists of both structured data query and unstructured content search via CONTAINS() function, is processed in RDBMS, Contemporary processing strategy optimizes and executes the query in two parts. One part is the structured query processing as described in for Query 1 to get a set of RIDs of the fact table.

The other part is unstructured text content search query processing that gets a set of DOCIDs of the fact table rows whose description column containing the search keyword '*promotion*'. Given a keyword, inverted text index returns a set of DOCIDs containing the keyword. The set of DOCIDs is then mapped to set of RIDs. Then these two sets of RIDs are intersected to get common RIDs of fact table. Note the inverted text index stores DOCIDs instead of RIDs in its posting list because DOCIDs are more amenable for delta compression to achieve very compact posting list size and thus reducing I/O traffic (Zobel & Moffat,2006). This two-part processing strategy is rooted in different indexing and query processing strategies in relational and IR approaches. However, the two-part processing strategy does not deliver ideal performance for queries spanning both structured data and unstructured data. We shall present a better strategy for handling such queries in section 5.

4. Using XML data model in a data warehouse for both structured and unstructured data

4.1 Why XML data model in DBMS ?

Paper (Stonebraker & Hellerstein 2005) reveals the evolution of data models in database community starting with the hierarchical model, then network, relational, object, object-relational, leading up to the XML data model. The criticism on XML data model is that it appears to date back to the hierarchical model that is well supported in IMS systems. Indeed XML is a tree-based model forming single hierarchy and thus is not adequate to handle multiple hierarchies. For example, for data with many-to-many relationship, such as the classical example of students taking courses, relational model offers the most flexible way of presenting such relationship without duplicating data. In a relational model, entity is identified by an id, the ids are recorded in a relationship mapping table to capture many-to-many relationships. With many-to-many relationships, two different hierarchies: the hierarchy of student taking multiple courses and the hierarchy of a course being taken by multiple students can be modeled without data duplication. For OLTP workloads where data is read and updated frequently, relational model with application of high degree of normalization rules is the best model. Therefore, there is no reason to suggest XML model/XQuery as a replacement of the relational model/SQL.

However, the rationale for a DBMS is that it can store and retrieve all kind of data, not just one kind of data. In RDBMS, structure of the data is well-defined so that data and its structure can be separated out cleanly. Structure of data is defined using the relational schema (table definitions) and managed as meta-data by RDBMS. Unstructured content is one extreme of data where there is no meaningful structure to describe the collection of data. Semi-structured data is data whose structure is not easily separable from data because the structure is dynamic and evolving. Semi-structured data is represented by adding tags to mark and annotate the data. The tag denotes inline structures of the data so that they can be referred via navigation of tags. Each tag has a name and tags can form a hierarchy. Querying semi-structured data involve querying tag names, tag hierarchy and data together.

XML data model defines a hierarchical tree composed of nodes having both tag and content. Structure of data is recorded as XML tags annotating the data. Both structure and data can be stored and queried together using XQuery, a declarative language. XML can be used as a

canonical data model for representing all: unstructured data, semi-structured data and structured data, in a DWH. The following is a list of key points illustrating how XQuery/XML model differentiates from SQL/Relational model and keyword text search/IR model.

Schema Flexibility: "Data first, schema later or never" approach is the key strength of XML data model that differentiates itself from relational model. Data can be stored and queried without defining schema for the data first. Data annotated with tags become self-describing and self-contained. Data with high schema variability, data with rapidly evolving schema can be modeled using XML model. Schema evolution, which is a common issue for RDBMS administrators and developers, is not an issue for XML model. Although XML can be schema-based, XML schema is primary used for the purpose of data validation instead of being used as relational schema that defines table structures to store XML data. One XML document collection can hold XML document instances without any XML schemas or different XML schemas. This differentiates an XML model from a relational model where one table can hold only entities conforming to one particular relational schema.

Structure Query Capability: In the XML data model, structures are not specified apriori as meta-data. Instead, structures are embedded into the data, therefore, they can be searched and queried as if they were data. Capabilities of wildcard tag name search and descendant tag name search in XQuery essentially support the notation of searching structures without knowing the exact names of the structure or the exact hierarchy of the structure. This differentiates from SQL where exact names of the relational tables that hold the data and exact join column keys to join tables must be specified in the query. Using XQuery on XML, users have the flexibility of writing specific query with precise structural names and hierarchy or generic query without precise structural names and hierarchy. The latter capability gives user the flexibility of querying structures and data together while structure of data is dynamic and evolving.

Full context aware text search Capability: With XQuery full text capability, XML data model is an ideal data model to manage document contents. Classical keyword text search for unstructured content from the IR community lacks declarative language to search content with mark-up tag annotation. However, after document content is modeled as XML document, XQuery full text search can be leveraged to provide hierarchical context-aware full text search with context defined by XML tags. Classical keyword text search capability is logically a degeneration of XQuery full text search over one XML text node capturing the entire unstructured content. Therefore, XQuery/XML model completely subsumes the keyword text search/IR model.

Declarative data transformation and construction: XQuery is an expressive language for not only queries that enable "finding needles in the haystack", but also transformation constructs that enable transformation of existing data, construction of new data in a declarative way. This differentiates from classical keyword text search where there is no declarative language construct to process the searched keywords. Keyword highlighting functionality is not part of the full text language. This also differentiates from SQL where there is no declarative language facilitating hierarchical result construction. Therefore, in the relational model that flattens everything, hierarchical object construction has to be done outside SQL in a procedure-oriented host programming language code. With XML data

model and XQuery, hierarchical object construction can be defined declaratively as hierarchical views over the relational model.

In summary, Figure 1 shows the positions and value of XML. Currently unstructured data is stored in LOB column without the need of schema. So they are easy to store, however, there is no strong declarative language, such as SQL, to query them. This falls into NO-SQL approach. Structured data are stored in relational model and is easy to query them using SQL. However, it is relatively hard to store them without coming out schema first. XML sits in the middle to bridge these two worlds and provides value as a flexible data model with declarative XQuery language.

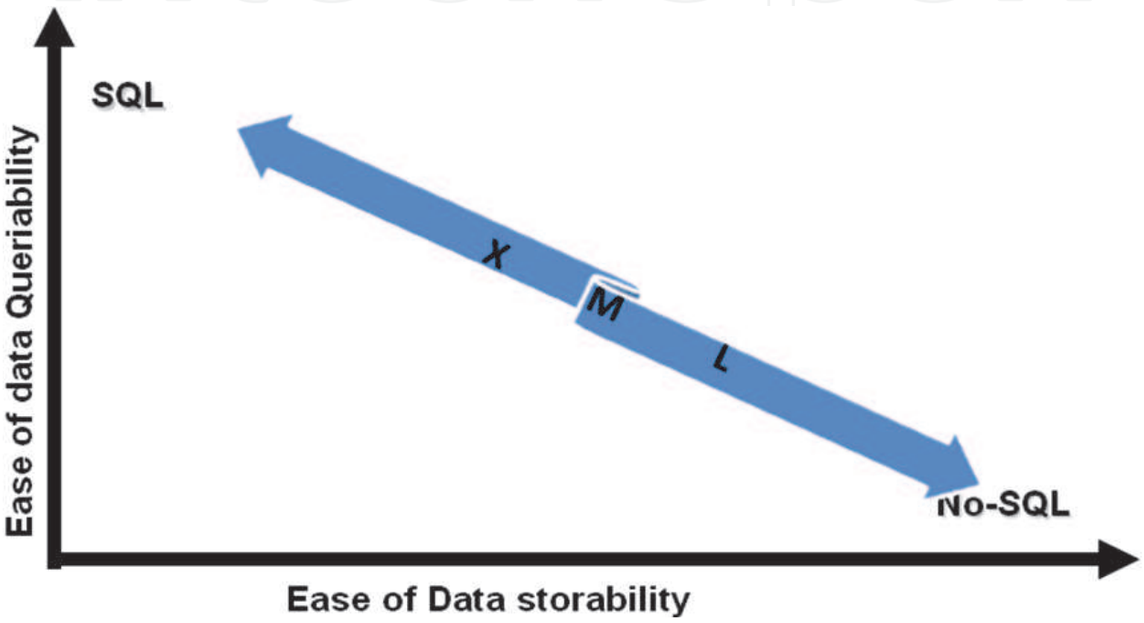


Fig. 1. XML bridges SQL and No-SQL approaches

4.2 Why XML for DWH?

Besides representing semi-structured data and unstructured content, there are advantages to leveraging XML data model to handle DWH processing. We present the key points here.

4.2.1 ETL flexibility

Building DWH requires Extracting, Transforming, and Loading (ETL) of data from multiple data sources. XML is increasingly being used as an exchange format to facilitate data transfer. In the past decade, different industries have defined XML schemas, such as XBRL schema for business reporting, HL7 schema for health industry, FPXML schema for financial industry to capture and exchange domain specific XML data. When an XML schema is rigid, it is possible to shred the XML data into relational tables. However, common industrial XML schema is designed to capture all possible data representations, therefore, industrial XML schema is highly variant with usage of many XML constructs, such as XML nodes of any data type, mixed content data, choice models, etc., so that mapping XML schema into relational schema is either infeasible or will result in the creation of many tables having

many null columns that are sparsely populated with data. From the perspective of XML schema, this is reasonable because XML schema is supposed to be used for XML data validation instead of being used to decompose XML into relational data. This indeed highlights the crux of the challenges faced with data integration and extraction. That is, coming up with a small, rigid relational like schema that covers every piece of data variant is not feasible. To overcome this problem, common ETL process for RDBMS DWH store defines a minimal relational schema that captures the most commonly used data. The rest of the data that do not fit the relational schema are either stored in LOB columns or as files in the file system. This results in limited query capability over the non-relational data.

Using an RDBMS that supports XML persistence overcomes the above deficiencies because using XQuery the structure and data in the XML can be queried together. Furthermore, well-structured data in the XML can be extracted into relational tables by building XML indexes on top of XML storage (Liu et al., 2007). **This approach of persisting data as XML first and extracting well-structured data within XML as XMLTable based XMLIndex is the practical means to fulfill the spirit of "data first schema later".**

4.2.2 Integrated structured, semi-structured, unstructured content search query

Query 2 is a SQL query involving both text search and regular relational search. Such a query enables users to integrate well-structured data search based on classical RDBMS technology and unstructured content search query from classical IR technology together. Paper (Yates & Navarro 1996) illustrates the need of integrating contents and structure search tightly in text retrieval. As XQuery full text capability subsumes the classical IR text search capability, Query 3 shows how SQL/XML that provides XQuery capability in SQL is able to do integrated data query regardless of the availability of a pre-defined structure for the data. Description column of the Fact table is of XMLType that stores XML data. The *'. contains text "promotion" ftand "independence"'* is XQuery full text syntax to specify searching the keyword '*promotion*' and '*independence*' anywhere in all the descendent text nodes from the input context node. XMLEXISTS() is a SQL/XML operator that takes as input an XMLType column, which in this case is the Description column of the Fact Table. This predicate executes the XQuery full text functions to see if the evaluation of the text node returns any nodes. If it does, XMLEXISTS() returns the boolean TRUE value.

```
SELECT f.StoreId, count(*)
FROM TransactionFact f, Customers c, Items i, Dates d, Stores s
WHERE f.CustomerId = c.CustomerId AND f.ItemId = i.ItemId AND f.StoreId = s.StoreId AND
f.DateId = d.DateId AND i.ItemName = 'TV' AND s.StoreZipCode >= 45000 and
s.StoreZipCode <= 46000 AND i.ItemPrice < 300 AND d.Year between 2003 and 2005 AND
XMLEXISTS('. contains text "promotion" ftand "independence"' PASSING description)
GROUP BY f.StoreId
```

Query 3. SQL/XML Query on Hybrid XML and Relational Star-Schema

4.2.3 I/O friendly XML object retrieval and object transformation

In a DWH, data is loaded once and read many times. In a typical relational database, to retrieve the original entity data in full, tables are joined over primary keys and foreign keys which results in generation of random I/O requests compared to the option of storing data aggregated and contiguous on disk which results in typically one I/O to bring the whole

object entity into memory. Therefore, for a DWH, it is desirable to store data in a de-normalized way compared to an OLTP database where data is normalized to avoid update anomalies. XML persistency essentially extends the idea of de-normalization further by materializing hierarchy without joint keys and thus provides the fastest whole object retrieval from the perspective of disk friendly I/O. *From I/O perspective, excessive seeks with small read per seek is worse in performance compared with few seeks with large read per seek.*

Consider the third phase of processing of Query 1 where the set of RIDs from the Fact table is computed. In addition to fetching the rows from Fact table based on RID, Dimensional table lookup may still be needed to bring in relevant dimensional values using dimension ids stored in the Fact table to re-construct the original transaction object. On the other hand, XML data may be stored in a compressed binary format without shredding the data into relational tables resulting in the advantage of efficient retrieval of the whole object when needed. This form of XML persistency is essentially a generalization of data de-normalization concept in DWH design. Furthermore, XQuery provides constructs to transform XML into different hierarchical shapes and forms for different presentations of the data.

Having discussed the value of XML data model for DWH, we now show the XML and XQuery functionality support in RDBMS leveraging the SQL/XML standard.

4.3 SQL/XML in RDBMS for XML based DWH

SQL/XML is an ISO SQL standard that defines XML as a datatype conforming to the XQuery data model with three new built-in operators: XMLQuery(), XMLExists(), and XMLCast(); and one new table construct - XMLTable to facilitate XQuery invocation in SQL and to query and modify XML datatype. Furthermore, it has introduced new built-in SQL/XML publishing operators: XMLElement(), XMLForest(), XMLComment(), and XMLPI(); and a new built-in aggregation operator - XMLAgg() that facilitate constructing XML from structured relational data. Therefore, with the SQL/XML standard, XML data can be managed by RDBMS along with structured data without the need of a specialized XML database. It is more elegant to enable management of all data in one system, and query them using SQL with domain object extension (such as SQL/XML), than to manage different data in different systems, and to correlate them in the application tier or mid-tier. Indeed, SQL/XML approach reflects the success of object relational approach that provides a type, function, index extensibility framework for managing any data in Object Relational DBMS - ORDBMS (Stonebraker et al., 1998). Contemporary RDBMS supports SQL/XML standard leveraging ORDBMS principle (Krishnaprasad et al., 2005). There are three approaches to supporting SQL/XML in a post-relational DBMS.

4.3.1 Approach 1: Relational and XML hybrid

In this approach, the relational table is extended with XMLType column instead of LOB column to store content that do not have rigid schema to be shredded into relational columns. This approach is conservative and represents the first generational adoption of XML in RDBMS. Query 3 shows this approach. It has the advantage of specifying both SQL and XQuery declaratively in one query. However, like processing of Query 2, processing

Query 3 still requires the two-part processing strategies, that is, join of relational indexing for relational part of the query and XML indexing for XML part of the query. Furthermore, the conceptual issue with this hybrid approach is that in many cases, structured data and unstructured data are embedded within each other, i.e. there could be islands of structured data embedded in top-level unstructured content. Therefore, it is natural to store the whole top-level data as one XML object and only project out the islands of structured data as index. This leads to Approach2 discussed below.

4.3.2 Approach 2: XML persistence with XML table based XMLIndex

Approach1 reveals the underlying problem that structures inside XML can be too diverse to be captured by relational tables. In this approach, the XML is stored as is, while an XML Index is created to index the islands of structured data for relational processing. XQuery can be used to declaratively specify the extraction of the structured data from the XML in the XML Index creation.

For example, consider that the purchase transaction captured in table 1,2,3,4,5 is stored as one XML document, shown in XML-Document 1 below, in a table containing all XML documents.

```
<Transaction>
<Customer id = 1454>
        <CustomerName>John Smith</CustomerName>
</Customer>
<Store storied =123>
        <StoreName>Electronic-Supply</StoreName>
        <StoreZipCode>45789</StoreZipCode>
</Store>
 <Item ItemId=1456>
        <ItemName>T.V</ItemName>
         <ItemPrice>250.32</ItemPrice>
         <QuantitySold>2</QuantitySold>
 </Item>
<poDate dateId=13579>2004-07-20</poDate>
 <sale_comment>
This is sold via special summer sale promotion program at the store. The promotion program is conducted along with the independence celebration event in the city.
</sale_comment>
</Transaction>
```

XML-Document 1. Transaction Fact

We create a transactions table having an XMLType column that stores each transaction fact as an XML document in a row. Then we create XMLTable based XMLIndex to extract structured data out into a relational table: TransactionFact as shown in SQL-DDL 1. The XML Index uses XMLTable construct with XPath/XQuery. The equivalent of Query 3 is now formulated as Query 4 using the XMLTable construct. In Query 4, the original fact table in Query 3 becomes a virtual table computed by XMLTable() construct over XMLType column.

```
Create table Transactions(xmldata XMLType);

Create Index tabIdx on Transactions(xmldata) as
(transactionfact XMLTable('/Transaction'
                    column
                       CustomerId varchar(20) PATH 'Customer/@id',
                       ItemId varchar(20)  PATH 'Item/@ItemId',
                       StoreId varchar(20)  PATH 'Store/@storeid',
                       DateId  integer PATH 'poDate/@dateId',
                       QuantitySold integer PATH 'Item/QuantitySold')
);
```

SQL-DDL 1. XMLTable based XMLIndex Creation

```
SELECT f.StoreId, count(*)
FROM Transactions,
XMLTable('/Transaction' PASSING Transactions.xmldata
        Column
          CustomerId varchar(20) 'PATH Customer/@id',
          ItemId varchar(20) PATH 'Item/@ItemId',
          StoreId varchar(20) PATH 'Store/@storeid',
           DateId  integer PATH 'poDate/@dateId',
           QuantitySold integer PATH 'Item/QuantitySold') Fact f,
Customers c, Items i, Dates d, Stores s
WHERE f.CustomerId = c.CustomerId AND f.ItemId = i.ItemId AND f.StoreId = s.StoreId AND
f.DateId = d.DateId AND i.ItemName = 'TV' AND s.StoreZipCode >= 45000 and
s.StoreZipCode <= 46000 AND i.ItemPrice < 300 AND d.Year between 2003 and 2005 AND
XMLEXISTS('. contains text "promotion" ftand "independence"' PASSING description)
GROUP BY f.StoreId
```

Query 4. SQL/XML Query on XML Persistency with XMLTable

Compared to the hybrid approach, this XML persistency approach with XMLTable based XMLIndex has the following advantages for DWH environment:

- There is no need to force all data into a common integration schema; all data can be captured without any data loss.
- XMLTable based XML index can be used to index the islands of structured data. This results in a flexible mechanism because index can be dropped and re-created without affecting the base storage. Users have the flexibility to decide what to index and how to index without the need for changing the base XML persistency. This genuinely fulfills the goal of 'data first, schema later' approach. Such an approach is superior to that of a relational approach because data can be stored without first defining the schema. Yet, it does not lose the advantage of relational processing because the projection of the embedded structured data as an index allows for queries over this data to be processed using relational access methods. (Liu et al., 2006). XMLTable based XMLIndex conceptually represents partial relational shredding approach of XML. Furthermore, it gives users the flexibility of not decomposing XML relationally even if XML is well-structured.

- For a DWH, where data is not frequently modified, maintenance of the XMLTable based XMLIndex is acceptable. For cases where the index definition is complex enough requiring more time to be processed, XMLIndex can be maintained like materialized views that can be refreshed asynchronously.
- Finally, all existing business intelligence tools on current relational model based DWH are completely useable on XML via the XMLTable constructs supported by XMLTable based XMLIndex.

4.3.3 Approach 3: XML persistence with XML-extended Inverted Index (XEIIX)

This approach uses the same XML persistence as that of Approach 2. Approach 2 above still conceptually requires users to split structured query from full text query, and therefore forces users to write the query to explicitly join the two parts of the query together. In Approach 3, using XQuery on XML, it is feasible to express the structures and unstructured data query intermixed naturally and natively as shown in Query 5.

```
SELECT XMLCAST(XMLQUERY('$doc/Transaction/StoreId') AS INTEGER) as StoreId, count(*)
FROM Transactions
WHERE XMLExists('$doc/Transaction[Item/ItemName = "TV" and fn:year-from-dateTime(poDate) >= 2003 and fn:year-from-dateTime(poDate) <= 2005 and xs:integer(StoreZipCode) >=45000 and xs:integer(StoreZipCode) <= 46000 and saleComment contains text "promotion" ftand "independence" ]'
            PASSING description AS "doc")
GROUP BY StoreId
```

Query 5. SQL/XML Query on XML Persistency with Full XQuery

The significance of Query 5 is that there is no explicit relational join query specified in SQL FROM clause. From the perspective of user, it is more natural to express structured and unstructured data query using Query 5 instead of Query 4. This illustrates the advantages of XML as a hierarchical data model and XQuery as a user friendly language to access hierarchical structures using XPath syntax instead of specifying explicit joins of relational tables. Internally RDBMS can either process XPath relationally by decomposing XPath traversal as joins of relational tables, if XML is indexed relationally, or by processing XPath natively if XML is persisted in aggregated binary form. Although XQuery and SQL/XML processing can be decomposed into storage/index independent query logical rewrite transformation followed by storage/index dependent physical rewrite transformation (Liu et.al, 2008), from the user's perspective, the query is written independent of physical storage/index methods. To process Query 5 efficiently, however, we need to extend inverted text index to support XML; such an XML extended inverted index (XEIIX) is discussed in section 5.

5. Efficient processing of DWH query on structured and unstructured data

In this section, we discuss the extended inverted (XEIIX) index layout to efficiently process both structured and unstructured data search. XEIIX is an extension of the classical inverted text index to index structured data, XML hierarchies and keywords in unstructured content all together.

5.1 Comparison of bitmap join index, columnar storage and IR inverted text index

Before we go into XML Extended Inverted Index (XEIIX), let's look at the following three common query processing strategies in RDBMS and IR systems to motivate the rationale for developing XEIIX.

Bitmap Join Index: Recall that in the bitmap join index approach presented in section 3.1.3, to efficiently process star-join in DWH, each row in the Fact table is mapped to a ROP (Row Ordinal Position). Each bitmap index maps a set of dimension values of a given dimension to ROPs. SQL *and, or, not* boolean predicates for different dimension values are processed using set-based multi-dimensional joins. The set-based multi-dimensional join is done efficiently by computing intersection, union and difference among a set of bitmaps. Bitmap join-index representation is very compact and thus requires less amount of disk I/O.

Columnar Storage: Columnar storage (Stonebraker et al., 2005) for DWH is a similar approach that uses invisible join strategy to process star-schema join query (Abadi et al., 2008). All values of a column of the base table are stored together in a compressed form to reduce the amount of disk I/Os to read them into memory. Each column value is implicitly identified by its corresponding ROP of the row containing that column value. Fast scan of columnar data using small amount of disk I/O results in a set of bitmaps representing ROPs for rows that match the given dimension ids. The multi-dimension join is then done efficiently using bitmap intersections.

IR Inverted text index: Each document stored in the base table row is given a document id (DOCID). The DOCID is identical to the ROP, It is a unique sequential integer identifying each document using the position of the row that contains the document. Inverted text index maps a keyword to a posting-list which consists of a set of sorted DOCIDs that identify documents that contain the keyword (Zobel & Moffat, 2006). The posting-list is delta-compressed leveraging the sequential integer property of the DOCID. This results in small amount of disk I/Os. Keyword search query using '*and, or, not*' predicate is done via pre-sorted merge join among posting-lists (Zobel & Moffat, 2006).

All three presented techniques have the following common query processing and indexing properties:

- They all handle boolean predicates of *and, or, not* on a set of dimension values extracted from a collection of objects. In RDBMS, each table can be considered as a collection and each row of the table can be considered as an object. The dimension values are column values extracted from each row. In an IR system, each document collection can be considered as a collection, and each document can be considered as an object. The dimension values are keywords extracted from each document.
- They identify each collection object using a unique sequential integer, henceforth referred to as DOCID. There exists an I/O friendly layout of a mapping structure so that looking up the mapping between a dimension value to a set of DOCIDs having the dimension value is very efficient. In RDBMS, the mapping structure is either bitmap join index or columnar layout of relational table. The mapping structure in IR system is an inverted text index. Although the mapping structure can be logically modeled as a relational table, storing each mapping as a row in the relational table causes poor performance due to the on-disk layout of the table rows. This is one of the key reasons

why an IR text index layout using a relational table could be an order of magnitude slower than a customized text search engine implementation (Brewer 2005). In fact, contemporary RDBMS products that support inverted text index do not store keyword and DOCID mapping as individual rows in a relational table. Instead, the disk layout of posting list of the inverted text index is highly compressed and stored in LOB structures. Therefore, it requires small amount of disk I/O to fetch the entire posting list into memory.

- They all process boolean *and, or, not* predicates on dimension values using set intersection, union and difference computation among sets of DOCIDs or bitmaps identified by ROPs. This set join processing essentially converts the polynomial-bound multiple binary-joins into a linear bound multi-way join. The inverted text index's posting list for a given keyword stores DOCIDs in a sorted order. The bitmap positions stored in bitmap join index for a given dimension value stores ROPs in sorted order. Therefore, DOCIDs are pre-sorted in all of these approaches and the linear multi-way join is an effectively pre-sorted merge join (Sort-merge join has linear join property excluding the sorting time.). Paper (Zhang et al., 2001) shows that Multi-Predicate Merge Join strategy commonly employed by full text search engine with hardware cache utilization are the two key reasons that conventional relational engines with conventional join methods do not yield comparable performance to a full text search engine. Furthermore, hardware based vector instructions can be leveraged to further accelerate multi-way join process. For example, bitmap join, can be speedup by using hard-ware vector processing instructions.

It has also been shown that inverted index is more space efficient and delivers better query performance than that of bitmap index when the attribute indexed has a high cardinality (Bjørklund et al., 2009). This leads to the conclusion that applying IR inverted index to DWH is a fruitful direction to take.

5.2 XML-extended inverted index (XEIIX)

Having examined the above three cases, we propose to process the structured and unstructured DWH query 5 in a disk I/O friendly manner based on Multi-Predicate Pre-Sorted Merge join (MPPSMJ), a technique, employed by customized inverted index text engines.

We extend the classical inverted text index to form the XEIIX. The XML element and attribute tags are indexed as regular keywords. All XML text content is indexed by their keywords. There is already keyword position stored in posting list in classical inverted text index so that phrase search is done by comparing keyword position information during MPPSMJ process (Zobel & Moffat, 2006). The XEIIX contains XML tag hierarchical position information so that keyword and XPath containment is processed during MPPSMJ process. Parent-child relationship between XML tags can also be tracked in the index so as to speed up hierarchical relationship check. The key idea behind XEIIX is that it captures both XML structures (tags and their hierarchical relationships) and content data together in one index. With such an integrated index, the search of structure and data together can be processed efficiently, thereby realizing the full potential of XML and XQuery. From a RDBMS perspective, being able to query without differentiating structures and data is a conceptual milestone. From an IR perspective, being able to do content search within structures so that text search becomes context aware is also a conceptual milestone.

But there is still one thing missing in that classical full text search is not capable of performing range predicates using common scalar datatypes, such as, number, date, time, etc, that are widely used in RDBMS. However, XQuery supports the capability of querying data embedded in XML using datatype aware range predicate. For example, to process XQuery expressions of Query 5, such as '*fn:year-from-dateTime(poDate)* >= *2003 and fn:year-from-dateTime(poDate)* <= *2005 and xs:integer(StoreZipCode)* >=*45000 and xs:integer(StoreZipCode)* <= *46000*', we need to further enhance the XEIIX to index range type data embedded in XML. Even for XQuery expression '*Item/ItemName = "TV"*', *ItemName* range comparison is performed using character string datatype which is semantically different from full text search because full text search is subject to stemming, thesaurus, diacritics, etc. options supported by a text search engine.

We, therefore, propose to enhance XEIIX to also index well-defined relational data in the XML document. The XMLTable construct used in XMLTable based XMLIndex presented in section 4.3.2 can be used as a conceptual framework to allow users to declaratively locate the data to be indexed as range-typed data. This is shown in SQL-DDL2 as an illustrative syntax to create XEIIX covering relational scalar datatypes data. However, unlike XMLTable based XMLIndex that physically implements the index as relational tables, the index layout is exactly the same as that of mapping a keyword to a posting list of sorted DOCIDs. The range-data index structure maps a range typed data value to a set of sorted DOCIDs having that value. The MPPSMJ processing can join posting lists for both ranged-typed data and text keywords. This essentially accomplishes the integration of structured data query and unstructured content search at the index level. This integrated index approach performs better than the conventional approach of evaluating structured predicates using relational indexes and retrieving ROWIDS and evaluating text predicates using Text indexes and retrieving DOCIDs and then finally joining DOCIDs and ROWIDs. The XEIIX structure and the MPPSMJ process essentially flattens all joins uniformly using DOCIDs. In fact, both the bitmap index join approach and invisible join in columnar storage for pure relational queries have demonstrated the performance advantage of doing join using ROPs instead of ROWIDs due to small sequential I/O traffic and linear-bound MPPSMJ join. From this perspective, XEIIX leverages the benefits of both the bitmap join-index in RDBMS and inverted text inedex in IR. This is an efficient design to fulfill the goal of XQuery with full text capability that tightly integrates both structured data query and unstructured content search together that otherwise would have to been done separately (one in RDBMS and one in IR) and joined in the end.

```
Create Index  xml-text-index on Transactions(xmldata) as
(xmlfull text,
Range-data: XMLTable('/Transaction'
                    column
                       ItemName  varchar(20) PATH 'Item/ItemName',
                       StoreZipCode integer PATH  'Store/StoreZipCode',
                       Year  integer  PATH  'fn:year-from-dateTime(poDate/@dateId)'
                       )
);
```

SQL-DDL 2. XML-extended Inverted Index

5.3 Declarative & efficient object construction and transformation

Because the relational model normalizes entity objects into a set of tables, in an RDBMS, in order to re-create the original entity object, one has to issue a SQL query that joins the tables and selects all the columns out into the mid-tier where a host programming language is used to construct the original object entity. In a DWH star-join query, the retrieval of the original transactional object entails pulling columns of both the fact transaction table and all the dimensional tables. This is illustrated in the SQL example in Query 6. If all fields of an entity object needs to be retrieved in the end, one of the challenges for an RDBMS using columnar storage is to develop efficient materialization strategies to delay the row construction (Abadi et al., 2007).

```
SELECT c.CustomerId, c.CustomerName, d.Date, d.Month, d.Year, s.StoreZipCode, f.Description
FROM TransactionFact f, Customers c, Items i, Dates d, Stores s
WHERE f.CustomerId = c.CustomerId AND f.ItemId = i.ItemId AND f.StoreId = s.StoreId AND
f.DateId = d.DateId AND i.ItemName = 'TV' AND s.StoreZipCode >= 45000 and
s.StoreZipCode <= 46000 AND i.ItemPrice < 300 AND d.Year between 2003 and 2005 AND
CONTAINS(description, 'promotion')
```

Query 6. SQL All Fields Selection

To contrast, now consider the SQL/XML queries, supported by an XRDBMS, illustrated below in Query 7 and Query 8. Query 7 shows the usage of XQuery in the select list to selectively project out fields from the original stored XML to construct new XML objects. Query 8 illustrates the capability in XQuery to transform the original XML object into a new object by deleting the *sale_comment* node.

In comparison to Query 6, Query 7 and 8 illustrate the following advantages of an XRDBMS and XQuery over RDBMS and SQL:

- The XML support in XRDBMS frees the user from figuring out exactly what tables to join which depends on how the transaction object has been normalized into relational tables. XML preserves the abstraction of the entity object, and enables the user to issue a query based on this abstraction using XQuery, which is then efficiently processed by the XRDBMS using XEIIX l and MPPSMJ process.
- XRDBMS allows users to construct new object or transform objects declaratively using XQuery instead of programmatically constructing or transforming the object in the mid-tier by issuing a relational SQL query in a plain RDBMS, The I/O efficiency in an XRDBMS is realized by storing XML in a compressed binary form that can be loaded into memory using a smaller amount of I/O. Then all XQuery evaluation in the select list is performed on the in-memory XML object. This is in contrast to the columnar storage of an RDBMS where various pieces of the column values need to be fetched from disk and assembled in the end. In this regard, it may be argued that it is a better option for the RDBMS to use columnar store as an index rather than as a persistence mechanism, thereby taking advantage of the columnar layout of data for efficient evaluation of predicates while avoiding the overhead of piecing the columns together by going to the row storage for retrieving the entire row. **Therefore, we think columnar index instead of columnar storage is the ideal way to bridge row store and column store.** In this way, not only row filtering that leverages the disk I/O friendly columnar layout of columns and invisible bitmap joins can be done efficiently, but also full row retrieval can be done efficiently without unnecessary assembly from columnar storage as well!

```
SELECT XMLQUERY(
'<TVTransaction>
(
<Customer @id = {$doc/Transaction/Customer/@id}>
    <CName>{$doc/Customer/CustomerName/text()}</CName>
</Customer>
,
<transactionDate>{$doc/Transaction/poDate/text()}</transactionDate>
<zip>{$doc/Transaction/Store/StoreZipCode/text()}</zip>
,
$doc/<sale_comment>
)
</TVTransaction>'
PASSING description AS "doc")
FROM Transactions
WHERE XMLExists('$doc/Transaction[Item/ItemName = "TV" and fn:year-from-dateTime(poDate) >= 2003 and fn:year-from-dateTime(poDate) <= 2005 and xs:integer(StoreZipCode) >=45000 and xs:integer(StoreZipCode) <= 46000 and saleComment contains text "promotion" ftand "independence" ]'  PASSING description AS "doc")
```

Query 7. SQL/XML Query with XML Construction

```
SELECT XMLQUERY(
'copy $cpy := $doc modify
      delete nodes $cpy/sale_comment
  return $cpy'
PASSING description AS "doc")
FROM Transactions
WHERE XMLExists('$doc/Transaction[Item/ItemName = "TV" and fn:year-from-dateTime(poDate) >= 2003 and fn:year-from-dateTime(poDate) <= 2005 and xs:integer(StoreZipCode) >=45000 and xs:integer(StoreZipCode) <= 46000 and saleComment contains text "promotion" ftand "independence" ]'  PASSING description AS "doc")
```

Query 8. SQL/XML Query with XML Transformation

6. Challenges and future directions

In this section, we discuss the challenges and future work for Data Warehousing and business intelligence.

6.1 Domain specific object type handling

So far, we have discussed the rationale for extending DWH technology to cover both structured and unstructured data,with integrated search over both data. However, we focused on only data of scalar and text data types. There are domain specific data, such as spatial, image, etc. that can be embedded inside XML documents. There is valuable information stored in these domain specific objects that need to be semantically queried and analyzed together with scalar and text data to assist making intelligent business decisions. ORDBMS (Stonebraker et al., 1998) framework enables the addition of user defined types to

model domain specific objects, user defined functions on user defined types to manipulate such objects, and user defined index (domain index) with user defined operators to speed up queries over such objects. The end result is that ORDBMS appears to understand domain specific object types as if they were native built-in types with built-in functions, operators and indices. For example, an ORDBMS may be extended to support management of Multi-media data, images, and handle queries over such data with matching and resemblance operators that are efficiently evaluated using domain indexes created over such data (Candan & Sapino 2010). Modern ORDBMS also manage spatial objects that are built on the domain index and extensibility framework (Kanth et al., 1999, Kanth et al., 2002). The one point to highlight is that the domain index probes in a typical ORDBMS return ROWIDs that are then used to map and merge with the rest of the SQL query evaluation and execution.

Built on the same principle of ORDBMS and Object Relational SQL, XRDBMS and XQuery have built-in extensibility to support user defined type and user defined functions. This enables queries over structured, unstructured and domain specific object instances all together using XQuery and SQL/XML. However, the key difference is that in an XRDBMS, the XEIIX returns a set of DOCIDs instead of ROWIDs. Whereas the domain index described return a set of ROWIDs. One way to handle this is to convert DOCIDs into ROWIDs and then join them together. However, ROWID joins are slower than DOCIDs and also incur the cost of conversions between DOCIDs and ROWIDs. Pre-sorted merge join techniques on sorted DOCID is much faster than general purpose ROWID joins. Future work is needed to investigate the best way of joining domain specific index results with inverted text index results, that is, whether MPPSMJ can be extended to cover domain indices. An even more fundamental question is whether the inverted style index that we described handling both text content and structured data can be extended to cover domain specific indexing as well.

6.2 XML in text mining, enterprise document search and information extraction

Enterprise search crawls document data from various data sources, builds inverted text index to facilitate keyword search based on classical I/R techniques (Salton & McGill, 1983). Structured data within the documents, such as document authors, types, publishing dates etc, are extracted out to provide facet navigational search. Text mining does statistical analysis of document content to derive concepts and topics contained in the document collection. Entity and relationship extraction from documents provide foundation for text mining and pattern discovery. All of the derivative data from these unstructured content analysis and discovery process need to be captured and persisted so that they could be queried and analyzed. These derivative data can be retained in context of the document they were discovered in via XML. Future work is needed to investigate how XML can be effectively used as a data model to facilitate text mining and analytical work.

6.3 Real-time DWH support

So far we have seen that DWH assumes read-only batch update model where decisions are made based on relatively stale data. The challenge is to support near real-time DWH given that all index and data layout in DWH do not favor in-place updates of the data. We think that the timestamp based consistent read query semantics and in-memory indexing with batch index merges is the direction worth exploring.

First, data is not in place updated. Instead updating data implies a new version of the data is inserted and old version of the data is marked for deletion. Queries run with an implicit timestamp so that the query results are consistent, although it may not always correspond to the latest timestamp. With timestamp based data version technique, the database would essentially allow time traversal query where history could be queried. Consistent read is well-practiced in RDBMS (Bamford 1992), so is time traversal query capability (Gawlick 2004).

Second, index for new version of data is built in memory and periodically merged with on-disk index structures. Such incremental index maintenance is discussed in inverted text index implementations (Zobel & Moffat 2006). The challenge is how such technique can be applied to other domain specific indexing structures. We believe that for DWH environment, data may never fit in memory, but index in compressed form can fit in memory resulting in fast query and search.

7. Generic star schema and star query

Classical DWH practices develop schema first and then load data later. In this chapter, we have shown we can do data store first via XML persistency and exploit schema later via different XML indexing methods. The XML Index can be a relational projection of XML to facilitate relational modeling over XML - this is the idea of the XMLTable based XMLIndex discussed in section 4.3.2. The XML Index can be keyword and tag extractions to facilitate context aware text search over XML - this is the idea of XML extended inverted text index discussed in section 5.2. The XML Index can also project out domain specific object instances embedded in XML to facilitate domain object specific query as discussed in section 6.1. Therefore, generalizing all this, we think the future DWH practice is going to be a more ***generic star-schema*** where the fact table is a collection of XML documents, each of which is identified by a DOCID. Different dimensional tables represent different dimension values extracted from the base XML document, and thus a dimensional table serves as the role of a ***dimension index*** into the XML document collection. The dimension value can be as simple as well-typed relational data or can be as complex as domain specific data. Text keywords and XML tags are default dimensional index. The dimensional index maps a dimensional value to a set of pre-sorted DOCIDs that satisfy a relationship with the dimensional values. That relationship is essentially an operator that can be evaluated via the dimensional index. The ***generic star-query*** is on the single fact table with where clauses of a set of boolean predicates, each of which specifies a domain index specific operator searching on some dimension values. The generic star-query is first processed via probing different dimensional indexes, followed by computing the set of common DOCIDs using pre-sorted merge joins among sets of pre-sorted DOCIDs obtained from dimension index lookup. The DOCIDs are then used to retrieve base XML document upon which further data extraction, transformation and aggregation operations in the query select list are performed. All of these phases of processing can be executed in parallel, and may exploit specific hardware accelerations when feasible. New dimensional index can be added as new dimensions are discovered for the underlying data. Dimensional index can be dropped when such dimension search is not needed. Thus this generic star-schema/query model fully embraces the concept of data first, schema later.

8. Conclusion

In this chapter, we have shown that XML is flexible enough to handle both structured and unstructured data. Declarative XQuery language with SQL/XML can be used to effectively build and query data warehouses comprising of all enterprise data. Both structured data and unstructured content can be managed by one XRDBMS – an XML enabled RDBMS with XQuery and SQL/XML. This obviates the need to migrate relational data into a pure XML database; instead an XML view over relational data can be defined. Unstructured and semi-structured data can be stored natively as XML, without relational shredding, in the XRDBMS. All XML data can be uniformly queried via XQuery using SQL/XML. XML extended inverted text index can be used to efficiently support XQuery processing. SQL/XML bridges the structured and unstructured world: relational data can be viewed as XML via SQL/XML view, and XML data can be cast as relational data via XMLTable construct. This provides for a very flexible system that enables all relational tools and application to access XML Data while new XML tools and applications can access both XML and relational data.

Various industries are defining XML Schemas for data exchange, transformation and reporting. Domain specific object instances can be embedded in XML. Such XML data can be persisted as native XML in XRDBMS and then queried using XQuery, or relationally using SQL/XML via the XMLTable construct. The management of XML in an extended Relational Database Management system is benefitted by the leverage of secular DBMS technologies, such as data partitioning, parallel query execution, clustered server operating environments, etc. all of which are generally available in a contemporary RDBMS.

To efficiently support DWH query over any data, the design has to realize the significant performance gap between disk I/O and CPU speed. Therefore, I/O friendly data and index layout and pre-sorted multi-way merge join processing remain to be the two key strategies to deliver superior query performance over large volume of data. This really leads to the confluence of inverted text index and its query processing strategies from SIGIR community and columnar oriented data/index layout and its query processing strategies from DBMS SIGMOD community. The integration of the two shall deliver high performance of query that spans structured data and context aware full text search together.

XML and XQuery efforts have led us to explore a new post-relational world where business intelligence over structured, semi-structured and unstructured data is becoming feasible. This post-relational world requires us to embrace the concept of data first, schema later model and to provide declarative query that integrates structure and content search, transformation together. This genuine spirit from post-relational world shall empower business to access and make decisions over any type of data in a unified way.

9. References

Abadi, D.J.; Madden, S; Hachem, N : Column-stores vs. row-stores: how different are they really? SIGMOD Conference 2008: 967-980

Abadi, D.J; Myers, D.S; , DeWitt, D.J.; Madden, S: Materialization Strategies in a Column-Oriented DBMS. ICDE 2007: 466-475

Bamford, R: Using Multiversioning to Improve Performance Without Loss of Consistency. SIGMOD Conference 1992: 164

Bjørklund, T.A; Grimsmo, N; Gehrke, J; Torbjørnsen, Ø: Inverted indexes vs. bitmap indexes in decision support systems. CIKM 2009: 1509-1512

Brewer E.: *Combining Systems and Databases: A Search Engine Retrospective In readings in Database Systems*. The MIT Press, 4th edition, 2005.

Candan, K. S; Sapino, M.L: Data Management for Multimedia Retrieval, Cambridge University Press, ISBN-10: 0521887399, ISBN-13: 978-0521887397 May 31, 2010

Chen P.: *The Enity-Relationship Model: Toward a Unified View of Data*. VLDB 1975: 173

Codd, E. *A Relational Model of Data for Large Shared Data Banks*. Commun. ACM 13(6): 377-387 (1970)

DeWitt, D.J; Gray, J: Parallel Database Systems: *The Future of High Performance Database Systems.* Commun. ACM 35(6): 85-98 (1992)

Gawlick, D: Querying the Past, the Present, and the Future. ICDE 2004: 867

Kanth, K.V.R; Ravada, S; Abugov, D: Quadtree and R-tree indexes in oracle spatial: a comparison using GIS data. SIGMOD Conference 2002: 546-557

Kanth, K.V.R; Ravada, S; Sharma, J; Banerjee, J: Indexing Medium-dimensionality Data in Oracle. SIGMOD Conference 1999: 521-522

Krishnaprasad, M; Liu, Z.H; Manikutty, A; Warner, J.W; Arora, V: *Towards an Industrial Strength SQL/XML Infrastructure.* ICDE 2005: 991-1000

Liu, Z.H; Chandrasekar, S; Baby, T; Chang, H.J: Towards a physical XML independent XQuery/SQL/XML engine. PVLDB 1(2): 1356-1367 (2008)

Liu, Z. H; Krishnaprasad, M; Chang, H.J.; Arora, V: *XMLTable Index An Efficient Way of Indexing and Querying XML Property Data*. ICDE 2007: 1194-1203

Milenova, B.L; Yarmus, J; Campos, Marcos: *SVM In Oracle Database 10g: Removing the Barriers to Widespread Adoption of Support Vector Machines.* VLDB 2005; 1152-1163

Myers, D: (1986) *Psychology*, Worth Publishers, Inc, ISBN: 0-87901-311-7, New York, New York 10003

O'Neil, P.; Graefe, G. *Multi-Table Joins Through Bitmapped Join Indices*. SIGMOD Record, Vol. 24, No.3, Sep 1995

Salton, G; McGill, M: (1983) *Introduction To Modern Information Retrieval*, McGraw-Hill, Inc. ISBN: 0-07-054484-0, U.S.A

SQL/XML: I.O. for Standardization (ISO). Information Technology-Database Language SQL-Part 14: XML-Related Specifications

Seshadri, P; Hellerstein, J.M; Pirahesh, H; Leung, T.Y, C; , Ramakrishnan, R; Srivastava, D; Stuckey, P; Sudarshan, S: *Cost-Based Optimization for Magic: Algebra and Implementation*. SIGMOD Conference 1996: 435-446

Stonebraker, M.; Abadi, D; Batkin, A.; Chen, X.; Cherniack, M; Ferreira, M.; Lau, E.; Lin, A.; Madden, S; O'Neil, E.; O'Neil, P.; Rasin, A.; Tran, N.; Zdonik, S. *C-Store: A Column-oriented DBMS*. VLDB 2005: 553-564

Stonebraker, M; , Brown, P; Moore, D. *Object-Relational DBMSs*, Second Edition Morgan Kaufmann 1998

Stonebraker, M.; Hellerstein, J. *What Goes Around Comes Around*. In readings in Database Systems. The MIT Press, 4th edition, 2005.

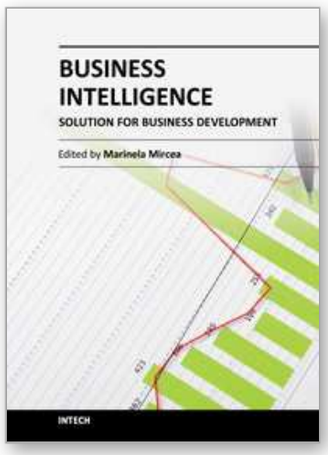
XQuery. http://www.w3.org/TR/xquery/

XQuery and XPath Full Text. http://www.w3.org/TR/xpath-full-text-10/

Yates, R.; Navarro, G. *Integrating Contents and Structure in Text Retrieval*. SIGMOD Record, Vol.25, No.1, Mar 1996

Zhang, C.; Naughton, J; DwWitt, D; Luo, Qiong; Lohman, G. *On Supporting Containment Queries in Relational Database Management Systems*. SIGMOD Conference 2001: 425-436

Zobel, J; Moffat, A. *Inverted files for text search engines*. ACM Comput. Surv. 38(2): (2006)



Business Intelligence - Solution for Business Development

Edited by Dr. Marinela Mircea

ISBN 978-953-51-0019-5
Hard cover, 108 pages
Publisher InTech
Published online 01, February, 2012
Published in print edition February, 2012

The work addresses to specialists in informatics, with preoccupations in development of Business Intelligence systems, and also to beneficiaries of such systems, constituting an important scientific contribution. Experts in the field contribute with new ideas and concepts regarding the development of Business Intelligence applications and their adoption in organizations. This book presents both an overview of Business Intelligence and an in-depth analysis of current applications and future directions for this technology. The book covers a large area, including methods, concepts, and case studies related to: constructing an enterprise business intelligence maturity model, developing an agile architecture framework that leverages the strengths of business intelligence, decision management and service orientation, adding semantics to Business Intelligence, towards business intelligence over unified structured and unstructured data using XML, density-based clustering and anomaly detection, data mining based on neural networks.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen