XBRL Repository – An Industrial approach of Management of XBRL Documents

Zhen Hua Liu , Thomas Baby, Sriram Krishnamurthy, Ying Lu, Qin Yu, Anguel Novoselsky, Vikas Arora

Oracle Corporation

500 Oracle Parkway

Redwood Shores, CA 94065, USA

{zhen.liu, thomas.baby, sriram.x.krishnamurthy)@oracle.com,

(ying.lu, qin.yu, anguel.novoselsky, vikas.arora}@oracle.com

***Abstract* - XBRL (extensible Business Reporting Language) is an XML based standard for electronic data exchange and communication of business financial documents and reports among government bodies, regulators, financial institutions and reporting entities. In this paper, we analyze the XBRL model from a perspective of data management to show why XBRL has created a new emerging vertical domain of database applications that offer opportunities and technical challenges for the database community. We outline the concept of an *XBRL repository* as a data hub to store and process collections of XBRL documents while maintaining document integrity based on XBRL semantics, providing document manageability, operational completeness and XBRL dictionary, query and business intelligence services. We then share our experiences of building the main components of the XBRL repository using the Oracle XML enabled RDBMS that leverages both XML and relational technologies as a backbone to host industrial strength XBRL applications. We also highlight technical challenges and potential solutions to each of the components. Finally we propose the potential of leveraging the XBRL data model concept for providing XML data normalization and XML having multi-hierarchy.**

I. INTRODUCTION

To realize the original motivation of XML as a common language to exchange data, the financial accounting and reporting communities created the XBRL[1,13] standard to facilitate financial data reporting among government regulators, companies and individuals. The XBRL specification [13] fully leverages XML technologies such as XML Schema[2], Xlink[3], XPath/XQuery[4]. XBRL is in use as the underlying financial reporting standard in several countries across the world and has gained adoption beyond just the community of analysts and regulators [5]. The potential market for XBRL is huge as it effectively tags financial data to facilitate communication of this data in a more consistent and self-checking manner.

An XBRL document set consists of *XBRL taxonomy documents* and *XBRL instance documents*. The XBRL taxonomy documents in turn consist of XML Schema documents and XML linkbase documents. The Schema documents describe XML schema elements, aka, **XBRL Concepts**. The Linkbases describe relationships among these concepts. Each business reporting entity submits its reports for a reporting period in an XBRL format to government regulatory bodies. These documents, which typically include XBRL taxonomy documents and XBRL instance documents, are then processed by XBRL processors, which leverage XML schema, Xlink and XQuery processors, to validate the documents. The process of XBRL validation verifies that the documents are conformant to the XBRL specification [13] and satisfy the business rules and constraints expressed in the calculation and formula linkbase documents. Various XBRL tools are used to convert XBRL instance documents to renderable XHTML reports for analysis.

The lifecycle of XBRL applications imposes different requirements on the representation of the XBRL data at different stages of processing. These representations could be, for example, a set of XBRL documents for submission and interchange, a relational representation of the structured elements within the instance documents for queryability and integration with legacy applications, an in-memory representation of the schemas and linkbases for validation, an in-memory tree representation of the instance tree for rendering. As a result, typical implementations are based on transforming the XBRL data to different representations at each stage of processing. The problem with such an approach is that it creates redundant representations of the same data with the burden on the XBRL tools and applications to maintain consistency between the representations.

As an example, XBRL document processors and tools are frequently mid-tier based, relying on a file system to store and retrieve XML documents, while also shredding the XBRL documents into relational tables for queryability, and additionally building in-memory structures for rendering reports. Shredding the data into an RDBMS causes the data to be duplicated without any system level linkage between the RDBMS and the file system. Consequently, since the RDBMS has no knowledge that the stored relational data originated from XBRL documents, neither the XBRL abstraction nor the XML abstraction of the data is preserved.

In this paper, we propose the concept of XBRL repository that provides storage of the XBRL documents in their original XML based representation, while also supporting multiple representational views over the data to support the full XBRL application lifecycle. The XBRL repository defines a set of XBRL data services that enable XBRL applications to store, process, query XBRL documents in a native, scalable and transactional manner, and to provide XBRL document

integrity, manageability with full operational completeness. The multiplicity of representational views allows XBRL applications to operate XBRL data in different abstractions including XML abstraction, relational abstraction. The XML abstraction is needed to support XBRL processors that view XBRL documents as XML documents whereas the relational data abstraction is needed for business intelligence applications that view XBRL documents as relational tables. We make use of the Oracle XML enabled RDBMS [16] to implement an XBRL repository to host industrial strength XBRL applications. The Oracle XML enabled RDBMS supports both XML and relational technologies using SQL/XML [19]. It turns out that XBRL applications can leverage both the XML technology and relational technology to their full extent because XBRL applications are neither pure classical XML applications so that XML technology by itself is enough, nor pure relational applications that relational technology by itself is enough.

In summary, the main contributions of this paper are

- We characterize the business requirements of an emerging XBRL document processing application and identify the technical challenges of managing XBRL data from the perspective of data management
- We present innovative ideas, such as, XBRL repository and scalable data service, XMLTable views, XMLTable based XMLIndex, Hierarchical XML Aggregation with recursive query extension to SQL/XML standard, automatic XBRL dimension and OLAP processing to overcome technical challenges from XBRL data management.
- We articulate the value of XBRL as a generic data model that can be generalized beyond XBRL applications. It provides a standard mechanism to do XML data normalization and support of multi-hierarchy of XML.

The rest of this paper is organized as follows. Section II starts with a motivating example to present the characteristics of XBRL document processing: the business requirements and the technical challenges from the perspective of data management. Section III presents the concept of the XBRL repository, the architecture and key components of the XBRL repository, and novel design and implementation strategies to overcome the challenges associated with XBRL processing presented in section II. Section IV discusses the merits of the XBRL data model beyond just the domain of XBRL. Section V presents experimental results to evaluate the performance characteristics of XBRL data services from the XBRL repository. Section VI discusses related work and Section VII concludes the paper.

II. XBRL Requirements & Challenges

A. Motivating Example

Consider a motivating example of a company that reports its accounting balance sheet as an XML document shown as comp-clas-bal.xml with its corresponding XML Schema shown as comp-clas-bal.xsd. The XML schema com-clas-bal.xsd is designed *classically* using an XML schema content model where all the elements are arranged in an explicit hierarchy. However, adopting the XBRL specification [13], this company report consists of an XBRL XML Schema document shown as comp-xbrl-bal.xsd, an XBRL presentation linkbase XML document shown as comp-xbrl-bal-presentation.xml and an XBRL XML instance document shown as comp-xbrl-bal.xml. The XML schema document comp-xbrl-ba.xsd merely lists all the elements (also referred as XBRL concepts using XBRL terminology) without any hierarchical relationships among the elements and the corresponding XML instance document comp-xbrl-bal.xml listing all the elements with their values in a flat representation. The XBRL linkbase document comp-xbrl-bal-presentation.xml uses the presentationArc with @xlink:from, @xlink:to and @xlink:order to specify the hierarchical relationships among elements defined in the XBRL XML schema document. Traversing the presentationArc listed in the linkbase document comp-xbrl-bal-presentation.xml, one can derive the content of comp-clas-bal.xsd and comp-clas-bal.xml. In addition to the presentation linkbase XML document, XBRL linkbase documents also include label, definition, calculation, reference, formula linkbase XML documents that specify various relationships among XBRL concepts that XBRL applications need to process and validate.

```
<Balance xmlns="comp-clas-bal.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
 <Asset>
  <Cash>14000</Cash>
  <Land>20000</Land>
 </Asset>
 <Liability>
  <AccountPayable>1000 </AccountPayable>
  <NotePayable>500 </NotePayable>
  </Liability>
 <Equity>
  <CapitalStock>14500</CapitalStock>
  <RetainedEarning>18000</RetainedEarning>
 </Equity>
</Balance>
```

Fig. 1 comp-clas-bal.xml

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="comp-clas-bal">
<element name="Balance">
 <complexType>
  <sequence>
   <element name="Asset">
    <complexType>
     <sequence>
      <element name="Cash" type="decimal"/>
      <element name="Land" type="decimal"/>
     </sequence>
    </complexType>
   </element>
   <element name="Liability">
    <complexType>
     <sequence>
      <element name="AccountPayable" type="decimal"/>
      <element name="NotePayable" type="decimal"/>
```

```
      </sequence>
    </complexType>
  </element>
  <element name="Equity">
    <complexType>
     <sequence>
      <element name="CapitalStock" type="decimal"/>
      <element name="RetainedEarning" type="decimal"/>
     </sequence>
    </complexType>
  </element>
 </sequence>
</complexType>
</element>
```

Fig.2 comp-clas-bal.xsd

B. Separation of Concept and Relationship in XBRL

A fundamental difference between the XBRL data model and XML is that entities and relationships are separated out in XBRL. The advantage of the classical approach is that the hierarchical relationships among elements defined in the XML schema and used in the corresponding instance documents are explicit and are thus simple to follow and understand. However, the problem of explicit hierarchy is that it is not flexible and extensible enough because the hierarchical relationships among elements are statically frozen instead of having the flexibility of being dynamically configurable depending on different business requirements. To avoid the issue of determining single root hierarchy for all companies, XBRL is designed without relying on the XML schema content model. Instead, it separates a schema element, (XBRL concept) from its relationships. The relationships among concepts are specified in linkbase documents so that there can be multiple hierarchies formed among concepts. The explicit hierarchy among XBRL concepts is derived during actual financial report generation. *Therefore, a primary challenge of XBRL processing is to derive explicit hierarchical relationships effectively and efficiently for reporting purposes, in real time, given large size of XML schema and linkbase documents.*

C. Large XBRL Taxonomy Documents

Classically, XML Schema documents are considered meta-data that describe XML instance data and are typically smaller than the actual XML data instance. However, the XBRL taxonomy can be very large and can have many other imported linkbases and schemas that form a large taxonomy set. The XBRL taxonomy set is large in both the size of documents and number of documents. For example, the U.S. GAAP taxonomy set [6] has close to 12000 elements defined.

Although the XBRL taxonomy is conceptually like meta-data, the meta-data is huge and more akin to a dictionary whereas each actual XBRL data instance is small in size compared to the collective size of the taxonomy documents that the instance refers to. Furthermore, many companies create their own extension schema and linkbase documents based on the standard XBRL schemas, such as U.S. GAAP[6] and IFRS [7], and there can be many versions of them for different reporting periods. Therefore, modelling and processing XBRL taxonomies as meta-data in the way similar to that of traditional meta-data modelling in a database system is not appropriate anymore. Instead taxonomies have to be treated as data with full queryability. Moreover, the idea of loading all taxonomy documents in memory in order to process XBRL instances is not a scalable solution. A scalable solution for XBRL requires an abstract dictionary layer that can provide queryability over the XBRL taxonomies by properly indexing the XBRL Taxonomy documents. *The challenge here is to index XBRL taxonomies documents so as to effectively and efficiently provide data dictionary types of services for XBRL applications.*

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xbrli="http://www.xbrl.org/2003/instance"
xmlns:link="http://www.xbrl.org/2003/linkbase"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:ci="comp-
xbrl-bal" targetNamespace="comp-xbrl-bal"> <annotation>
<appinfo> <link:linkbaseRef xlink:type="simple"
xlink:role="http://www.xbrl.org/2003/role/presentationLinkbaseRef"
xlink:arcrole="http://www.w3.org/1999/xlink/properties/linkbase"
xlink:href="comp-xbrl-bal-presentation.xml"/> </appinfo>
</annotation> <import
namespace="http://www.xbrl.org/2003/instance"
schemaLocation="xbrl-instance-2003-12-31.xsd" /> <element
id="ci_Balance" name="Balance" substitutionGroup="xbrli:item"
xbrli:periodType="instant" abstract="true"
type="xbrli:stringItemType" /> <element id="ci_Asset"
name="Asset" substitutionGroup="xbrli:item"
xbrli:periodType="instant" abstract="true"
type="xbrli:stringItemType" /> <element id="ci_Liability"
name="Liability" substitutionGroup="xbrli:item"
xbrli:periodType="instant" abstract="true"
type="xbrli:stringItemType" /> <element id="ci_Equity"
name="Equity" substitutionGroup="xbrli:item"
xbrli:periodType="instant" abstract="true"
type="xbrli:stringItemType" /> <element id="ci_Cash"
name="Cash" type="xbrli:monetaryItemType"
substitutionGroup="xbrli:item" xbrli:balance="debit"
xbrli:periodType="instant" /> <element id="ci_Land"
name="Land" type="xbrli:monetaryItemType"
substitutionGroup="xbrli:item" xbrli:balance="debit"
xbrli:periodType="instant" /><element id="ci_AccountPayable"
name="AccountPayable" type="xbrli:monetaryItemType"
substitutionGroup="xbrli:item" xbrli:balance="credit"
xbrli:periodType="instant" /> <element id="ci_NotePayable"
name="NotePayable" type="xbrli:monetaryItemType"
substitutionGroup="xbrli:item" xbrli:balance="credit"
xbrli:periodType="instant" /> <element id="ci_CapitalStock"
name="CapitalStock" type="xbrli:monetaryItemType"
substitutionGroup="xbrli:item" xbrli:balance="credit"
xbrli:periodType="instant" /> <element id="ci_RetainedEarning"
name="RetainedEarning" type="xbrli:monetaryItemType"
substitutionGroup="xbrli:item" xbrli:balance="credit"
xbrli:periodType="instant" /></schema>
```

Fig.3 comp-xbrl-bal.xsd

```
<linkbase
  xmlns="http://www.xbrl.org/2003/linkbase"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xsi:schemaLocation="http://www.xbrl.org/2003/linkbase xbrl-
linkbase-2003-12-31.xsd">
 <presentationLink xlink:type="extended"
xlink:role="http://www.xbrl.org/2003/role/link"
xlink:title="Presentation, All">
   <loc xlink:type="locator" xlink:href="comp-xbrl-
bal.xsd#ci_Balance" xlink:label="ci_Balance" />
```

```
    <loc xlink:type="locator" xlink:href="comp-xbrl-
bal.xsd#ci_Asset" xlink:label="ci_Asset" />
    <loc xlink:type="locator" xlink:href="comp-xbrl-
bal.xsd#ci_Liability" xlink:label="ci_Liability" />
    <loc xlink:type="locator" xlink:href="comp-xbrl-
bal.xsd#ci_Equity" xlink:label="ci_Equity" />
    <loc xlink:type="locator" xlink:href="comp-xbrl-
bal.xsd#ci_AssetTot" xlink:label="ci_AssetTot" />
    <loc xlink:type="locator" xlink:href="comp-xbrl-
bal.xsd#ci_LiabilityTot" xlink:label="ci_LiabilityTot" />
    <loc xlink:type="locator" xlink:href="comp-xbrl-
bal.xsd#ci_EquityTot" xlink:label="ci_EquityTot" />
    <presentationArc xlink:type="arc"
xlink:arcrole="http://www.xbrl.org/2003/arcrole/parent-child"
xlink:from="ci_Balance" xlink:to="ci_Asset" order="1"
use="optional" />
    <presentationArc xlink:type="arc"
xlink:arcrole="http://www.xbrl.org/2003/arcrole/parent-child"
xlink:from="ci_Balance" xlink:to="ci_Liability" order="2"
use="optional" />
    <presentationArc xlink:type="arc"
xlink:arcrole="http://www.xbrl.org/2003/arcrole/parent-child"
xlink:from="ci_Balance" xlink:to="ci_Equity" order="3"
use="optional" />
    <loc xlink:type="locator" xlink:href="comp-xbrl-
bal.xsd#ci_Cash" xlink:label="ci_Cash" />
    <loc xlink:type="locator" xlink:href="comp-xbrl-
bal.xsd#ci_Land" xlink:label="ci_Land" />
    <presentationArc xlink:type="arc"
xlink:arcrole="http://www.xbrl.org/2003/arcrole/parent-child"
xlink:from="ci_Asset" xlink:to="ci_Cash" order="1"
use="optional" />
    <presentationArc xlink:type="arc"
xlink:arcrole="http://www.xbrl.org/2003/arcrole/parent-child"
xlink:from="ci_Asset" xlink:to="ci_Land" order="2"
use="optional" />
     <loc xlink:type="locator" xlink:href="comp-xbrl-
bal.xsd#ci_AccountPayable" xlink:label="ci_AccountPayable" />
    <loc xlink:type="locator" xlink:href="comp-xbrl-
bal.xsd#ci_NotePayable" xlink:label="ci_NotePayable" />
    <loc xlink:type="locator" xlink:href="comp-xbrl-
bal.xsd#ci_IncomePayable" xlink:label="ci_IncomePayable" />
    <presentationArc xlink:type="arc"
xlink:arcrole="http://www.xbrl.org/2003/arcrole/parent-child"
xlink:from="ci_Liability" xlink:to="ci_AccountPayable" order="1"
use="optional" />
    <presentationArc xlink:type="arc"
xlink:arcrole="http://www.xbrl.org/2003/arcrole/parent-child"
xlink:from="ci_Liability" xlink:to="ci_NotePayable" order="2"
use="optional" />
    <loc xlink:type="locator" xlink:href="comp-xbrl-
bal.xsd#ci_CapitalStock" xlink:label="ci_CapitalStock" />
    <loc xlink:type="locator" xlink:href="comp-xbrl-
bal.xsd#ci_RetainedEarning" xlink:label="ci_RetainedEarning" />
    <presentationArc xlink:type="arc"
xlink:arcrole="http://www.xbrl.org/2003/arcrole/parent-child"
xlink:from="ci_Equity" xlink:to="ci_CapitalStock" order="1"
use="optional" />
    <presentationArc xlink:type="arc"
xlink:arcrole="http://www.xbrl.org/2003/arcrole/parent-child"
xlink:from="ci_Equity" xlink:to="ci_RetainedEarning" order="2"
use="optional" />
     </presentationLink>
</linkbase>
```

Fig.4 comp-xbrl-bal-presentation.xml

D. The need for multiple representations of XBRL data

The XML data model is a natural representation for XBRL content, serving as the perfect exchange format. An XBRL application frequently needs to operate on the XBRL content purely in its original XML representation when submitting, publishing, exchanging these reports. At the same time, XBRL also requires relational accesses because unlike content-centric or semi-structured XML documents whose structures are vague and irregular, XBRL based XML documents are well structured. Fundamentally, XBRL data reports numerical facts with its well-defined hierarchy. Its strength is its flexibility to capture data values and structure from a wide variety of reporting entities across industries, geographies, and time periods. As a result, the core value proposition of XBRL is based on extracting these structured elements to allow querying across large aggregations of XBRL documents, performing analytics and mining as described in section II.E. This fundamentally requires a relational model to slice and dice the data and roll up along different axes, as well as to integrate that with the dominant tools and applications designed for this. As a result, a fundamental part of querying XBRL content requires a relational view of the structured elements within the XBRL documents. And finally, applications need to access the XBRL representation of the data to navigate relationships based on the reconstructed trees as described in section II.A. *The challenge is that XBRL applications require not a choice of one of these representations, but all of these representations*. XBRL applications need to be able to go back and forth between the data models at different stages of processing.

```
<xbrl xmlns="http://www.xbrl.org/2003/instance"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:link="http://www.xbrl.org/2003/linkbase"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ci="comp-xbrl-bal"
    xmlns:iso4217="http://www.xbrl.org/2003/iso4217"
    xmlns:scenarios="http://www.xbrl.org/frta/scenarios">
  <link:schemaRef xlink:type="simple" xlink:href="comp-xbrl-
bal.xsd" />
  <context id="I-2008">
    <entity><identifie/></entity>
    <period> <instant>2008-12-31</instant></period>
   <scenario>
     <scenarios:ReportingScenario><scenarios:Actual
/></scenarios:ReportingScenario>
   </scenario>
  </context>
  <unit id="UM"><measure>iso4217:EUR</measure></unit>
  <ci:Land contextRef="I-2008" unitRef="UM"
decimals="INF">20000</ci:Land>
  <ci:Cash contextRef="I-2008" unitRef="UM"
decimals="INF">14000</ci:Cash>
  <ci:AccountPayable contextRef="I-2008" unitRef="UM"
decimals="INF">1000</ci:AccountPayable>
  <ci:NotePayable contextRef="I-2008" unitRef="UM"
decimals="INF">500</ci:NotePayable>
  <ci:IncomePayable contextRef="I-2008" unitRef="UM"
decimals="INF">400</ci:IncomePayable>
  <ci:CapitalStock contextRef="I-2008" unitRef="UM"
decimals="INF">14500</ci:CapitalStock>
  <ci:RetainedEarning contextRef="I-2008" unitRef="UM"
decimals="INF">18000</ci:RetainedEarning>
</xbrl>
```

Fig. 5 comp-xbrl-bal.xml

E. Analytics over XBRL Instance Documents

Just as a data warehouse is established over operational data so that business-intelligence based decisions can be made using OLAP techniques over stored warehouse data [12], the same principle can be applied to XBRL documents. In particular, the requirement to run population queries over a large collection of XBRL instance documents is a critical reason why XBRL documents need to be warehoused in a database. Furthermore, the XBRL dimension specification [14] enables a filing entity to establish hypercube and dimension relationships among XML concepts via definition linkbase documents. This opens up opportunities to execute data data-cube oriented relational queries [9] over XBRL documents. *The challenge is to be able to automatically extract and load data in XBRL linkbase and instance XML form into relational data form so that mature relational data warehouse techniques, such as data- cube analytical queriesy [10] and materialized views [15], can be fully leveraged.*

F. Life Cycle management of XBRL Documents

XBRL documents are supposed to be stored for long periods of time so that historical data auditing and analysis are feasible. Paper [8] argues that information has its own life cycle, which demands different storage and operational characteristics for data at its different life cycle period. Furthermore, there are many cross-references among XBRL documents so that document integrity among them needs to be maintained. For example of section II.A, Fig.3 comp-xbrl-bal.xsd has a reference to comp-xbrl-bal-presentation.xml via *link:linkbaseRef* so that the presentation linkbase file can not be deleted with the presence of the schema file. Similarly, comp-xbrl-bal.xml has a reference to comp-xbrl-bal.xsd and hence comp-xbrl-bal.xsd can not be deleted. Finally, XBRL documents need to track document versions [29]. XBRL application needs to manage different versions of the document, and to keep track of integrity relationships among multiple versions and to keep track of which version of an XBRL instance document is validated against with which version of taxonomy documents. *So, the challenge is to provide an infrastructure to facilitate the document integrity, versioning and life cycle management of XBRL documents.*

III. XBRL REPOSITORY

In this section we describe our XBRL approach of extending the Oracle database with XBRL semantics to serve as an XBRL repository by leveraging XML capabilities in Oracle XDB RDBMS [16]. Figure 1 shows the architecture of the XBRL repository. The repository is organized into three layers. The storage layer provides native XML based storage and indexed based on XBRL semantics. The Query Layer provides dictionary services over XBRL taxonomies as well as queryability over the instance documents. The XBRL services layer provides XBRL specific add-on services.

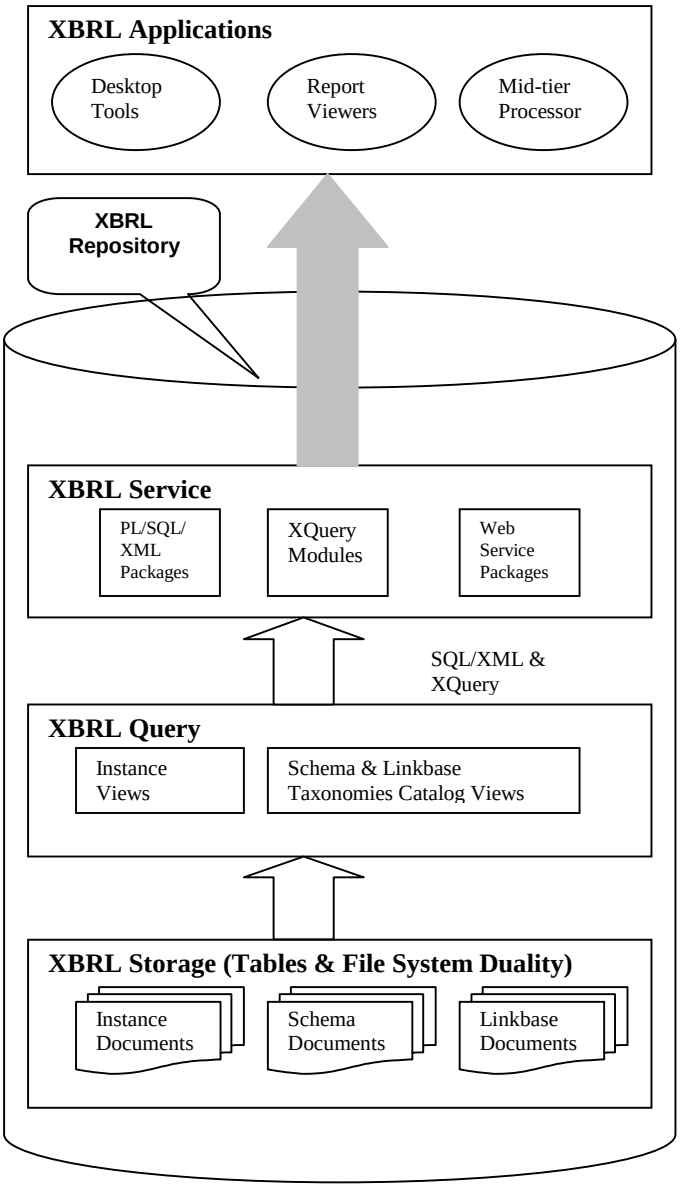

Fig.6 XBRL Repository Architecture

A. XBRL Repository - Document Storage, Index and Integrity Enforcement

Native XML Storage: The XBRL storage layer stores the XBRL content in its original representation as a set of XML schema, linkbase and instance files to preserve the content in its original submitted form. The documents are stored in primary three XMLType tables, each of which stores schemas, linkbases, and instance documents respectively. The XMLType tables in the XBRL repository leverage Oracle binary XML[16] tokenization capability to achieve compact storage since a lengthy element name is not uncommon in XBRL XML Schema. E.g. USGAAP[6] XML schema has elements whose length is 116 characters long.

Table Partition: An integral part of XBRL data management is managing the lifecycle of XBRL content that

have a distinctive temporal nature with reports being generated at regular timely intervals. To facilitate this, the XMLType tables storing the XBRL content support the concept of horizontal partitioning [11] so that XBRL applications can do range partition of their XBRL documents based on their reporting period. In this way, aged XBRL documents can be moved into partition with relatively inexpensive storage devices.

File/Folder & Table Duality: The XBRL repository provides user access in both SQL centric and file/folder centric way of uploading XBRL documents into the repository. The SQL centric way is through PL/SQL XBRL document loading APIs. The file folder centric way is through protocols, such as FTP and WebDav that Oracle XML enabled RDBMS supports [16]. One of the unique advantages of Oracle XML enabled RDBMS is that it is able to provide a *file system abstraction* over the stored XML documents so that XML documents can be accessed as if they were stored in a file system with native file system protocols. This makes the XBRL repository capable of providing both file system and relational duality of XBRL documents in contrast with mid-tier based XBRL applications that merely shred XBRL documents into relational tables and thus loose the XML abstraction. Irrespective of how the XBRL documents are loaded into the XBRL repository, all XBRL documents are inserted into the underlying XMLType tables.

Document Integrity: The references among XBRL schema, linkbases, and instance documents are tracked internally. This ensures that the XBRL document integrity is maintained. The references from XBRL instance documents to XBRL schema and linkbase documents are uni-directional, so XBRL instance documents can be deleted without restriction. However, in addition to the uni-directional references from the XBRL instance documents, XBRL schema and linkbase document references can form cycles. For example from section II.A, comp-xbrl-bal.xsd uses *link:linkbaseRef* to refer to comp-xbrl-bal-presentation.xml and comp-xbrl-bal-presentation.xml has "*locs*" element that refers to comp-xbrl-bal.xsd. The integrity is maintained by disallowing the deletion of taxonomy documents unless they are not referenced in the XBRL repository. Cross-referenced XBRL schema and linkbase taxonomy documents are deleteable as one single taxonomy unit. Document integrity check API is available for users to do post-load check of document integrity within the XBRL repository. The efficiency of document integrity check is achieved by building B+ indexes on reference relationship among documents.

B. XBRL Repository Query

The XBRL Query layer provides queryability over the XBRL content based on XBRL semantics. The XBRL data is queryable using XQuery since the data is in XML form. However, the XBRL repository needs to provide different representational views as discussed in section II.D. These views include different relational views over the instance documents, over the taxonomy documents for dictionary access, XML views to reconstruct the XBRL trees based on an XBRL presentation linkbase.

XBRL Taxonomy Dictionary Query: Because of the nature of XBRL taxonomies described in section II.C, an XBRL based dictionary layer is needed to provide queryability over the taxonomy data. For example, the view *XBRL_SCHEMA_ELEMENT* provides all the element information in the XBRL schema documents while the view *XBRL_PRESENTATION_LINKBASE* provides all the information in presentation linkbases. Both SQL access and XQuery access to these views are provided so that an XBRL application can access them as efficient dictionary lookups without having to parse and load the taxonomies in memory.

Relational views over the schema and linkbase documents are implemented using XMLTable [20] view backed up by XMLTable based XMLindexes (XTI) [17] that are created over the linkbase and schema XMLType tables. XTI is populated efficiently using SAX API and binary XML stream evaluation over the native XML storage of XBRL documents so as to scale with the size of XBRL documents.

We do not use the classical path-value based XMLIndex (PVI) [21] that is common in native XML database because the layout of the PVI is fundamentally the same as that of a vertical schema approach [22] which provides sub-optimal performance for querying a set of related properties of a node compared with that of XTI [17]. A second reason is that using PVI requires XML structural matching during run time. This is completely avoided when using XTI because structural matching in XTI is done during query compile time.

The main content of the *presentationArc* table of XTI for the comp-xbrl-presentation.xml in section II.A is shown as table 1 below. Note the table resembles the edge table shredding of XML discussed in paper [31]. However, the flexibility aspect of XBRL requires the additional *use* and *priority* columns. This enables one to delete a parent-child link by setting *use* column value to be '*restricted*' or raising *priority* column value to override prior '*restricted*' link. In this way, one can express multi-hierarchical instead of single-hierarchical relationship among XBRL concepts.

TABLE I
PRESENTATIONARC

Sid	from	To	Order	Use	Priority
1	*Balance*	*Asset*	*1*	*Optional*	*1*
1	*Balance*	*Liability*	*2*	*Optional*	*1*
1	*Balance*	*Equity*	*3*	*Optional*	*1*
1	*Asset*	*Cash*	*1*	*Optional*	*1*
1	*Asset*	*Land*	*2*	*Optional*	*1*
1	*Liability*	*Accountpayable*	*1*	*Optional*	*1*
1	*liability*	*NotePayable*	*2*	*Optional*	*1*
1	*Equity*	*CapitalStock*	*1*	*Optional*	*1*
1	*Equity*	*RetainedEarning*	*2*	*Optional*	*1*

Taxonomy Concept Hierarchical Tree Generation Query: As discussed in section II.A, hierarchical relationships among XBRL concepts are expressed in a presentation linkbase document. To provide the hierarchical view among XBRL concepts, one needs to generate the XBRL concept hierarchy from the presentation linkbase document. Query in fig. 7 shown below can be used to compute the XBRL concept

hierarchy from any starting concept that is passed in as an input parameter to the query. Shown below in Fig. 8 is the result of query in Fig. 7 with the *input_concept* value set to '*ci_Balance*'. The query runs a SQL recursive query over the *presentationARc* table created by XTI shown in table 1. Query can be modified to actually generate the comp-clas-xbrl.xsd schema.

```
SELECT HIERXMLAGG(
                    XMLQUERY('element {ename}'
                       PASSING arc. From AS "ename" ))
FROM presentationArc arc
WHERE  arc.use != 'prohibited'
AND arc.priority = (SELECT MAX(priority)
                     FROM presentationArc arc2
                    WHERE arc2.from = arc.from
                          AND arc2.to = arc.to
                         AND arc2.use != 'prohibited'
                   GROUP BY arc2.from, arc2.to
             )
CONNECT BY PRIOR arc.to = arc.from
START WITH arc.from = :input_concept /*starting concept*/
           AND arc.sid = :1
ORDER SIBLINGS BY  arc.order
```

Fig. 7 - Recursive-SQL/XML-Query-for-Concept-Tree

```
<ci_Balance>
  <ci_Asset>
     <ci_Cash/>
    <ci_Land/>
  </ci_Asset>
  <ci_Liability>
    <ci_AccountPayable/>
    <ci_NotePayable/>
  </ci_Liability>
  <ci_Equity>
   <ci_capitalStock/>
  <ci_RetainedEarnings/>
  </ci_Equity>
</ci_Balance>_
```

Fig. 8 Result-of-Recursive-SQL/XML-Query-for-Concept-Tree

Here are some important points to understand query in Fig 7:

- The SQL recursive query computes a transitive closure from a starting concept with parent-child relationship connected by the *from* and *to* columns of the *presentationArc* table. SQL recursive query is defined in the SQL standard [26] and is researched in literature [23,24]. Oracle uses the *CONNECT BY* construct to express the recursive SQL with the *ORDER SIBLINGS BY* clause to order the silbling nodes based on their *order* column values.
- Since the XBRL specification [13] allows a presentation linkbase arc to be prohibited and the arc with the highest priority for a given *from* and *to* pair defines the final relationship, therefore, Q1 needs to use MAX() scalar sub-query to compute the highest priority and to exclude prohibited rows. This illustrates the flexibility of the XBRL design. The hierarchy is controlled by raising priority value or prohibiting links. Such flexible hierarchy control enables individual companies to add new hierarchies, drop and modify existing hierarchies.
- Query in Fig. 7 uses *HIERXMLAGG()* aggregate function. Instead of merely aggregating all the element nodes generated from each row into one XML, *HIERXMLAGG()* assembles the element nodes generated from each row into a node hierarchy whose parent-child relationship is the same as that is computed from the underlying recursive CONNECT BY query. The critical difference between *HIERXMLAGG()* evaluation with *XMLAGG()* is that for each XML node it aggregates, it also gets the hierarchical level of the node. The hierarchical level is generated from the underlying SQL *CONNECT BY* query. Using the level information, *HIERXMLAGG()* knows how to attach each XML node to the correct hierarchical position.

Due to the spirit of the XBRL design to allow flexibility into the presentation and calculation hierarchy, we have identified recursive SQL and HIERXMLAGG() as the two crucial declarative constructs to be able to compute hierarchies dynamically. Furthermore, we can always support *materialized view* to cache the computed result on an as-needed basis.

Instance Facts Population Query: An XBRL instance document contain facts, each of which is the value for a concept defined in the schema. XTI is also created on XMLType storage table for XBRL documents so that facts can be accessed relationally. Table 2 shows the fact table for the *comp-xbrl-bal.xml* instance document of section II.A. The fact table storing data among all XBRL instances enables XBRL applications to construct ad hoc population queries across multiple instance documents.

TABLE II
XBRL INSTANCE FACT

InstanceId	ContextId	Concept	Fact_value
1	*I-2008*	*Cash*	*14000*
1	*I-2008*	*Land*	*22000*
1	*I-2008*	*AccountPayable*	*1000*
1	*I-2008*	*NotePayable*	*500*
1	*I-2008*	*CapitalStock*	*14500*
1	*I-2008*	*RetainedEarning*	*18000*

Instance Hierarchical Tree Report Query: As discussed in section II.A, XBRL application needs to be able to derive the XBRL instance document whose facts are arranged in the hierarchical manner defined by the presentation linkbase. Query in Fig. 9 shown below can produce the result XML shown as comp-clas-xbrl.xml. Note that query in Fig. 9 is structurally the same as one in Fig. 7 other than using the scalar sub-query to compute the concept values by joining with the xbrl_instance_fact table.

```
SELECT HIERXMLAGG(
   XMLQUERY('element {ename}'
         PASSING arc.from AS "ename")
            (SELECT FACT_VALUE
             FROM  xbrl_instance_fact fact
             WHERE  fact.Concept = arc.from
                   AND InstanceId = :1)))
FROM presentationArc arc
WHERE  arc.use != 'prohibited'
```

```
AND arc.priority = (SELECT MAX(priority)
        FROM presentationArc arc2
        WHERE arc2.from = arc.from
        AND arc2.to = arc.to
        GROUP BY arc2.from, arc2.to
        )
CONNECT BY PRIOR arc.to = arc.from
START WITH arc.from = :input_concept
ORDER SIBLINGS BY arc.order
```

Fig.9 Recursive-SQL/XML-Query-for-Instance-Tree

C. XBRL Repository Services

The XBRL Services provides useful XBRL based functions that are commonly needed as services for XBRL applications such as rendering, business-intelligence OLAP analysis.

Instance Rendering Service: Although the comp-clas-xbrl.xml shows the structure of the reported facts, the final rendering of the instance document requires an XHTML version of it with the XBRL concepts replaced by their label names for a particular language defined in the label linkbase document. So the XBRL repository also provides an XBRL instance rendering service that accepts XSLT stylesheets as input and applies the XSLT to the instance documents generated by the hierarchical tree generation service. This transformation is executed inside the XBRL repository and leverages the high performance XSLT processor [18] in the Oracle XML enabled RDBMS.

Dimension OLAP ETL & Query Service: XBRL relies on the definition linkbase document to define hypercube and dimensional relationships among XBRL concepts as specified in XBRL dimension specification [14]. This is an important foundation for business intelligence analytics over the XBRL data. Full analytical analysis of XBRL data requires defining fact tables and its associated dimensional columns that can be used as axes for *GROUP BY* rollups and cubes [10]. To enable this, the XBRL repository provides ETL type of service APIs to create basic fact table and dimension tables, and a join between them to generate the final fact table with dimensions as columns.

The XBRL instance fact table as shown in table 2 is a flat name-value pair table populated from the XTI over XBRL instance documents. However, the table layout for normal OLAP SQL operations requires that each dimension be an individual column. To illustrate this, consider measuring of concept *M* having two dimensions *D1* and *D2* with *D1* having domain members (*d1v1*, *d1v2*) and *D2* having domain members (*d2v1*, *d2v2*). The instance fact table is shown in table 3.

TABLE III
BASIC FACTS FOR PRIMARY ITEM M

Instance Id	ContextId	Concept	Fact Value
1	*d1v1-d2v1*	*M*	*v1*
1	*d1v1-d2v2*	*M*	*v2*
1	*d1v2-d2v1*	*M*	*v3*
1	*d1v2-d2v2*	*M*	*v4*

By analysing the primary item *M*'s hypercube tree, the XBRL repository ETL service create the following two dimension tables, and a join between the basic fact table and the dimension tables.

TABLE IV
DIMENSION TABLE FOR DIMENSTION D1

Instance Id	ContextId	Doman Value
1	*d1v1-d2v1*	*d1v1*
1	*d1v1-d2v2*	*d1v1*
1	*d1v2-d2v1*	*d1v2*
1	*d1v2-d2v2*	*d1v2*

TABLE V
DIMENSION TABLE FOR DIMENSTION D2

Instance Id	ContextId	Doman Value
1	*d1v1-d2v1*	*d2v1*
1	*d1v1-d2v2*	*d2v2*
1	*d1v2-d2v1*	*d2v1*
1	*d1v2-d2v2*	*d2v2*

TABLE VI
FINAL FACT TABLE FOR PRIMARY ITEM M

InstanceId	D1	D2	value
1	*d1v1*	*d2v1*	*v1*
1	*d1v1*	*d2v2*	*v2*
1	*d1v2*	*d2v1*	*v3*
1	*d1v2*	*d2v2*	*v4*

Now group by cube SQL query can run on *TM* to grouping by cubes across different dimensions as shown below.

```
SELECT instanceid,D1, D2, value
FROM  TM
GROUP BY CUBE(D1, D2)
```

Fig.10 Instance cube query

IV. MERITS OF XBRL DATA MODEL

XML as a singly rooted tree data model has its limitations. Paper [28] illustrates the need and the theory of normalization of XML. We think that XBRL in fact shows a way of fully normalizing XML data and in practice turning a singly rooted tree model into multi-hierarchy model. It is through the mechanism of linkbases that different relationships among data elements can be established. Thus, the tree form of the data can be computed as a view dynamically from the normalized data by traversing the linkbase information. Furthermore, the normalization of XML data in the XBRL approach is beyond update anomalies that avoid updating data multiple times. Rather, it is a mechanism to create multiple relationships among elements. Paper [30] in fact sees the single rooted hierarchy as the main disadvantage of XML being a generic data model and thus advocates the multi-hierarchical XML data model. XBRL now provides a standard mechanism to express such a multi-hierarchical data model using Xlink.

Although XBRL is built on a foundation of XML technologies, interestingly, the design principle behind XBRL actually aligns with that of the design of the relational data

model. The relational model separates the concept of entity from relationship, so that entities can participate and form many different relationships, instead of one fixed hierarchical relationship required by XML data model. Similarly, XBRL separates the specification of XBRL concepts from the specification of their relationships so that many different relationships, such as multiple presentation relationships, calculation relationships, definition relationships, etc., among the XBRL concepts can be expressed. With the separation of entities from relationships, data tend to be normalized. This is reflected in the design of XBRL in the sense that XBRL schema defines all the XBRL concepts with identifiers. These identifiers are then referenced in various linkbase documents to define and express relationships among these identifiers.

V. Performance Experiments

A. XBRL Taxonomies Loading and Indexing Time

We generate five XBRL schema documents (E2, E3, E4, E5, E6), each of which defines 2000, 3000, 4000, 5000, 6000 elements respectively. Then for each XBRL schema document, we generate the corresponding presentation linkbase documents (L2, L3, L4, L5, L6) that define the hierarchical relationships among the elements. The number of hierarchical links (*representationArc* elements) in each presentation linkbase document is the same as the number of elements in the corresponding schema document.

As discussed in section III.B, when XML schema and linkbase documents are loaded into the XBRL repository, XTI are created on top of the native XML storage XMLtype tables. XTI provide scalable XBRL dictionary services. The goal of this performance experiment is to demonstrate the scalability of the index loading time. Fig. 11 shows the index creation time ratio. E2/E2 is the base ratio 1 for indexing XBRL schema with 2000 elements. E3/E2, E4/E2, E5/E2 and E6/E2 is the ratio of XTI creation time for indexing XBRL schema with 3000, 4000, 5000 and 6000 elements over 2000 elements respectively. As Fig.11 shows, the creation time ratio is linear. This is expected because XTI indexing using SAX stream over XML native storage whose performance is linear to number of element definitions in the schema documents.

Fig. 12 shows the same linear performance for indexing the corresponding XBRL linkbase documents. In Fig.12, L3/L2, L4/L2, L5/L2,L6/L2 is the ratio of XTI creation time for indexing XBRL linkbase document with 3000, 4000, 5000, 6000 hierarchical links over 2000 hierarchical links respectively.

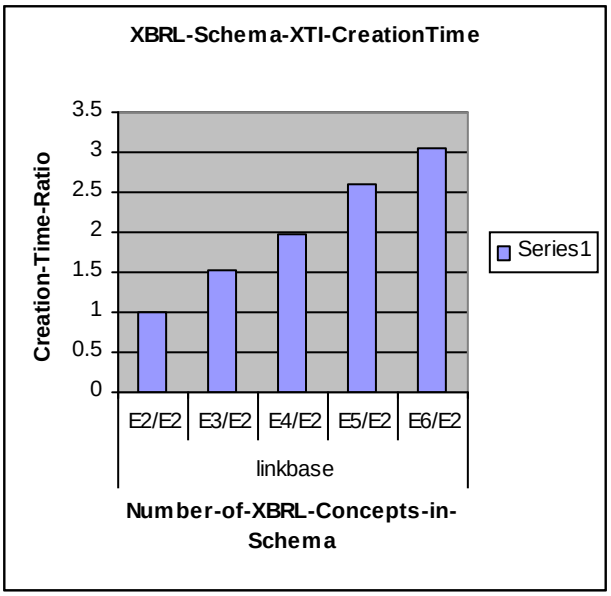

Fig. 11 XBRL Schema Indexing Time

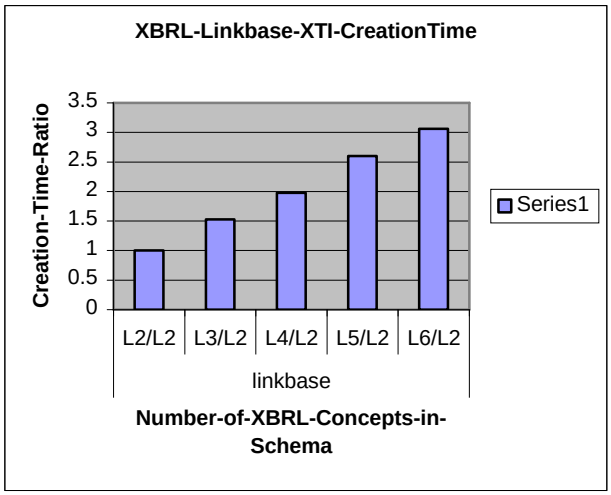

Fig. 12 XBRL Linkbase Indexing Time

Ensuring the XTI indexing time is linear to the number of elements and links justifies that the relational indexing approach over XBRL XML documents is a viable indexing strategy. Consequently, the XBRL dictionary service over XBRL taxonomies can deliver performance at the level of relational performance.

B. XBRL Concept Tree Generation Time

Using the same XBRL schema and presentation linkbase documents and XTI created earlier, we measure the hierarchical tree generation time using query Q1 discussed in section III.B. The goal of this performance experiment is to demonstrate the scalability of the XBRL hierarchical tree generation time. Fig. 13 shows the query time ratio of the recursive SQL/XML query. E2/E2 is the base ratio 1 for querying time of 2000 elements with 2000 hierarchical links. E3/E2, E4/E2, E5/E2 and E6/E2 is the ratio of query time of

3000, 4000, 5000 and 6000 elements with 3000, 4000, 5000, 6000 hierarchical links over 2000 elements with 2000 hierarchical links respectively. As Fig. 13 shows, the query time ratio is close to linear. This shows that generating hierarchical tree dynamically using SQL/XML recursive query is a viable approach and can deliver scalable tree query service.

C. *Varying Depth/Width of XBRL Concept Tree Generation Time*

We use the same XBRL schema of 6000 elements of section V.B. However, unlike presentation linkbase document used in section V.B where 6000 hierarchical links are at the same level, we create five presentation linkbase documents (D1, D2, D4, D8, D16), all of which have 6000 hierarchical links among 6000 elements. However, the depth and width of each hierarchy varies. D1 defines hierarchical depth of 1 with total of 6000 at one level; D2 defines hierarchical depth of 2 with total of 3000 nodes at each level; D4 defines hierarchical depth of 4 with total of 1500 nodes at each level; D8 defines hierarchical depth of 8 with total of 750 nodes at each level and D16 defines hierarchical depth of 16 with total of 375 nodes at each level.

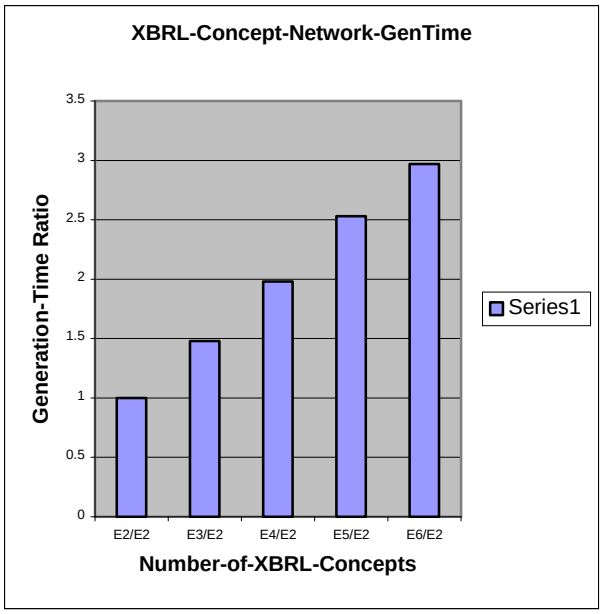

Fig. 13 XBRL Concept Tree Generation Time

Fig. 14 shows the generation time ratio. D1/D1 in Fig. 14 is the base ratio 1. D2/D1 is the query time ratio of concept tree generation time for D2 linkbase document over D1 linkbase document. D4/D1, D8/D1 and D16/D1 are the query time ratio of generation time for D4, D8 and D16 linkbase document over D1 linkbase document respectively. As Fig. 14 shows that the generation time decreases as the width of the hierarchy decreases and depth of the hierarchy increases. Common XBRL linkbase documents usually have average hundreds of elements per hierarchical level with average hierarchical depth of 8. This experiment shows that width contributes more than depth when generating XBRL concept tree. The reason is that ordering sibling nodes on the same level takes more time than running recursive queries.

VI. Related Work Comparison

To the best of our knowledge, there is no XBRL literature in the area of data management that analyzes emerging XBRL applications and identifies the challenges of XBRL document processing form the database community perspective. However, there are various mid-tier based XBRL products and tools that do XBRL validation, XBRL taxonomy editing, XBRL instance filing, etc. Most of these are based on shredding of XBRL documents into relational tables and storing them in an RDBMS. Our approach focuses on solving XBRL problems from the perspective of data management.

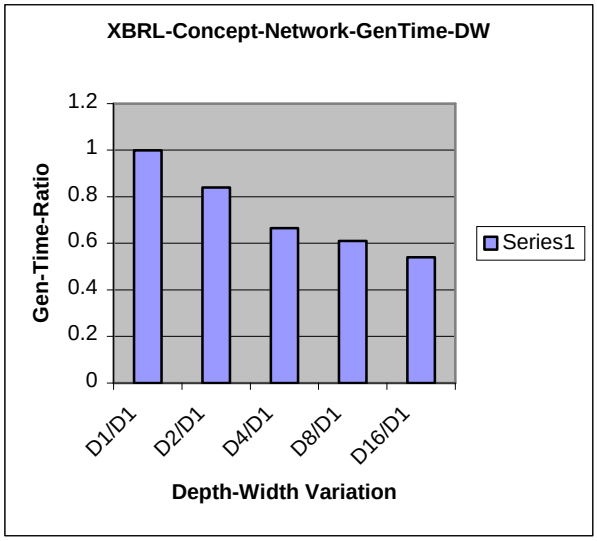

Fig. 14 XBRL Concept tree Generation by Varying depth/width of hierarchy

The first key differentiator in our approach is that we recognize that XBRL applications need both XML and relational abstraction of the data. XML abstraction is natural because all XBRL documents are XML documents. Relational abstraction is needed because fundamentally XBRL data is structured, and the relational data model proves itself as the best performing model to manage structured data. Furthermore, many mature OLAP and data warehousing technologies are based on the relational data model. Therefore, we leverage Oracle SQL/XML enabled RDBMS as the basis to support XBRL applications. We have leveraged the XMLTable concept [20] in SQL/XML as a bridge between XML to relational mode that can be efficiently supported by XMLTable based XMLIndex [17]. Furthermore, we have developed the idea of HIERXMLAGG() as the way to declaratively generate XML from a recursive SQL query. HIERXMLAGG() is not defined in the SQL/XML standard and to our best knowledge, it has not been discussed in past literature.

The second key differentiator in our approach is that XBRL applications need an XBRL repository that can manage all the XBRL documents: enforcing document level integrity constraints among them, providing document versioning, data

partitioning, and full life cycle management of them while also providing XBRL data services for all the mid-tier based XBRL applications and tools. To do this we have leveraged the Oracle XML database capabilities to load, store and manage XBRL documents without losing the file system abstraction. Users can load and access XBRL documents using native file system based protocols. Furthermore, XBRL data services can be delivered via web service protocols, all of which are in addition to the classical SQL and XQuery interface from RDBMS.

An alternative potential approach of solving the XBRL document processing problem is to leverage XML only database. However, since XBRL data are in well-structured XML form, from a performance perspective, it makes sense to do a relational decomposition of the XBRL data so that queries do not need to do structural matching during run time. Furthermore, instead of forcing users to write user defined functions for procedurally processing XBRL documents, XQuery needs to be enhanced with declarative group by cube type construct and declarative recursive query concept, just as SQL has been gone through, to effectively process XBRL documents.

VII. CONCLUSIONS

In this paper, we have identified challenges of XBRL document processing from the perspective of the database community, and have described the idea and design of the XBRL repository as a data hub to host XBRL applications. There are several key points that we can conclude from our approach. XBRL data requires multi-representation view of the data. The XBRL data model proposes a practical way of doing XML normalization and multi-hierarchy for structured XML data. Both XML and relational technologies are needed for processing XBRL documents and this is best done by leveraging SQL/XML based RDBMS. In particular, the XMLTable construct and HIERXMLAGG() serve as a bridge between XML and relational abstractions of XBRL documents.

REFERENCES

[1] XBRL: http://www.xbrl.org/Home/.
[2] XML Schema: http://www.w3.org/XML/Schema
[3] Xlink: http://www.w3.org/TR/xlink.
[4] XQuery/XPath: http://www.w3.org/TR/xquery/
[5] http://www.agacgfm.org/research/downloads/CPAGNo16.pdf
[6] U.S. Generally Accepted Accounting Principle (GAAP): http://www.xbrl.org/Taxonomy/us/2008/gaap/XBRL-US-GAAP-Taxonomy-1.0-Summary-2008.htm
[7] International Financial Reporting Standards (IFRS): http://www.iasb.org/XBRL/IFRS+Taxonomy/Latest+Taxonomy/Latest+taxonomy.htm
[8] P.P. Tallon, R. Scannell: Information Life Cycle Management. Communications of the ACM, Nov 2007, 65-69
[9] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, H. Pirahesh: Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub Totals. Data Min. Knowl. Discov. 1(1): 29-53 (1997)
[10] J. Gray, A. Bosworth, A. Layman, H. Pirahesh: Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total. ICDE 1996: 152-159
[11] S. Ceri, M. Negri, G. Pelagatti: Horizontal Data Partitioning in Database Design. SIGMOD 1982, 128-136
[12] S. Chaudhuri , U. Dayal, An Overview of Data Warehousing and OLAP Technology, ACM SIGMOD Record, **26**:1, 1997
[13] Extensible Business Reporting Languages: (XBRL) 2.1: http://www.xbrl.org/Specification/XBRL-RECOMMENDATION-2003-12-31+Corrected-Errata-2008-07-02.htm
[14] XBRL Dimesion 1.0: http://www.xbrl.org/Specification/XDT-REC-2006-09-18.htm
[15] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom, View Maintenance in a Warehousing Environment, ACM SIGMOD, 316—327, 1995.
[16] R. Murthy, Z. H. Liu, M. Krishnaprasad, S. Chandrasekar, A. Tran, E. Sedlar, D. Florescu, S. Kotsovolos, N. Agarwal, V. Arora, V. Krishnamurthy: Towards an enterprise XML architecture. SIGMOD Conference 2005: 953-957
[17] Z. H. Liu, M. Krishnaprasad, H.J. Chang, V. Arora: XMLTable Index An Efficient Way of Indexing and Querying XML Property Data. ICDE 2007: 1194-1203
[18] A. Novoselsky: The Oracle XSLT Virtual Machine. http://www.idealliance.org/proceedings/xtech05/papers/04-02-01/
[19] I.O. for Standardization (ISO). Information Technology-Database Language SQL-Part 14: XML-Related Specificaitons (SQL/XML)
[20] F Zemke, M. Rys, K. Kulkarni, J. Michels, B. Reinwald, F. Oczan, Z. H. Liu, I. Davis, K. Hare, "XMLTable" , ISO/IEC JTC1/SC32 WG3:SIA-051 ANSI NCITS H2 2004-039 http://www.wiscorp.com/H2-2004-039-xmltable.pdf
[21] M. Yoshikawa, T. Amagasa, T. Shimura, S. Uemura: Xrel: A Path-Based Approach to Storage and Retrieval of XML documents Using Relational Databases.
[22] R. Agrawal, A. Somani, Y. Xu: Storage and Querying of E-Commerce Data. VLDB 2001: 149-158
[23] K. Koymen, Q. Cai. SQL*: a recursive SQL. Inf. Syst., 18(2):121-128,1993
[24] C. Ordonez: Optimizing recursive queries in SQL. SIGMOD 2005: 834-839
[25] M. Stonebraker, J.M. Hellerstein: "What Goes Around Comes Around." http://mitpress.mit.edu/books/chapters/0262693143chapm1.pdf
[26] ISO-ANSI. Database Language SQL-Part2: SQL/Foundation. ANSI, ISO 9075-2 edition, 1999.
[27] C. J. Date: An Introduction to Database Systems, Volume I, 5th Edition. Addison-Wesley 1990
[28] M. Arenas: Normalization Theory for XML. SIGMOD Record, Vol. 35, No. 4, December 2006
[29] XBRL Versioning: http://www.xbrl.org/Specification/Versioning/XVS-Part1-PWD-2008-06-03.rtf
[30] H. V. Jagadish, Laks V. S. Lakshmanan, Monica Scannapieco, Divesh Srivastava, Nuwee Wiwatwattana: Colorful XML: One Hierarchy Isn't Enough. SIGMOD Conference 2004: 251-262
[31] Daniela Florescu, Donald Kossmann: Storing and Querying XML Data using an RDMBS. IEEE Data Eng. Bull. 22(3): 27-34 (1999)