International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:9, 2007

Data Annotation Models and Annotation Query Language

Neerja Bhatnagar and Benjoe A. Juliano and Renee S. Renner

Abstract**—** This paper presents data annotation models at five levels of granularity (database, relation, column, tuple, and cell) of relational data to address the problem of unsuitability of most relational databases to express *annotations*. These models do not require any structural and schematic changes to the underlying database. These models are also flexible, extensible, customizable, database-neutral, and platform-independent. This paper also presents an SQL-like query language, named Annotation Query Language (AnQL), to query annotation documents. AnQL is simple to understand and exploits the already-existent wide knowledge and skill set of SQL.

Keywords - annotation query language, data annotations, data annotation models, semantic data annotations.

I. Introduction

Most relational databases (RDBMSs) utilize their metadata schema to store statistical information for constraint checking and query optimization. The metadata schema provided by most RDBMSs is unsuitable for expressing *data annotations*. Data annotations are semantically rich metadata applicable to a particular application domain that help further clarify *features of interest*. A feature of interest is a data item that a user wants to annotate [1]. Types of data annotations include comments, descriptions, definitions, notes, error messages, among several others.

This paper addresses the problem of unsuitability of most RDBMSs' metadata schema by defining data annotation models that allow the annotation of relational data at five levels of granularity - database, relation, column, tuple, and cell. These data annotation models are structured using Extensible Markup Language (XML). The most important feature of these models is that they do not require any structural or schematic changes to the underlying RDBMS.

It is common knowledge that database administrators (DBAs) are resistant to structural and schematic changes to already deployed databases. Thus, these models stand a greater chance of being adopted. In addition, these models are easy to understand, flexible, customizable, extensible, database-neutral, and platform-independent. These models also allow users to cross-reference related annotations. The ability to customize these models stems from the flexibility given to users to define annotations to serve their individual requirements. Specifically, users can name the `applicationDomainSpecificTag` according to their requirements.

This paper also presents an SQL-like query language, Annotation Query Language (AnQL), to query annotation documents based on these models. AnQL is designed to take advantage of the already-abundant knowledge and skill set of SQL. XQuery and XPath are complex query languages. Learning these languages might present a steep learning curve. SQL-XQuery engines force RDBMSs to deal with semi-structured data format. Simplicity and ease of understanding are the main motivation factors for the design of the data annotation models and the query language presented in this paper.

Data annotations reduce communication and data exchange hassles and provide almost all database users (scientists, customer service providers, banks, corporations) with a more collaborative environment. A scientist, who wants to share the discovery he or she made while investigating an image can utilize annotations to annotate the image. He or she can also seamlessly share these findings with other researchers. Due to its numeric nature, it is often difficult to interpret the semantics of scientific data, simply by looking at it. As an example, it is difficult to interpret whether the data in the `TEMPERATURE` column is expressed in Metric or English units. Annotations can help scientists to annotate the `TEMPERATURE` column with the appropriate unit. If the column contains temperature in both Metric and English units, the cell-level data annotation model can be used to annotate each cell individually with its unit. An alternative is to change the data type of the `TEMPERATURE` column, from `DECIMAL` to `VARCHAR`,

Manuscript received October 31, 2006.

Neerja Bhatnagar (email: neerja@cs.ucsc.edu) received her Master's degree in Computer Science from California State University, Chico. She is currently pursuing PhD at University of California, Santa Cruz.

Dr. Benjoe A. Juliano (email: juliano@csuchico.edu) and Dr. Renee S. Renner (email: renner@csuchico.edu) are Associate Professors in the department of Computer Science at California State University, Chico. Both lead research projects at the Institute for Research in Intelligent Systems (IRIS) and the Intelligent Systems Laboratory (ISL).

to accommodate the unit. This might not be desirable since the scientists will lose the ability to manipulate the data mathematically. Moreover, this also requires a change to the underlying RDBMS. In some cases, annotations might also help reduce costs, and save time and effort. As an example, a customer can dispute a charge on his or her bank statement using annotations. The customer need not restrict himself or herself to the customer service hours. A customer service provider can also use annotations to update the customer.

II. Related Work

This section compares and contrasts the annotation models presented in this paper with Annotea [2], DB-Notes [3], MONDRIAN [4], SLIMPad [5], and with those presented in [1] and [6]. Annotea allows users to annotate documents identified by a URI with or without the knowledge of the authors. Similar to the models presented in this paper, DBNotes and MONDRIAN annotate relational data. The models presented in this paper allow relational data to be annotated at five different levels - database, relation, column, tuple, and cell. DBNotes focuses on the where-provenance and annotation propagation. MONDRIAN focuses on biological databases, and allows users to annotate both single values and the association between multiple values. SLIMPad annotates data that resides in a variety of applications, such as databases, spreadsheets, and documents, among several others. SLIMPad addresses annotations in the domain of physicians providing treatment to patients. The system presented in [6] annotates audio-visual documents. The system presented in [1] annotates neuroanatomical images at various levels of granularity.

Annotea and SLIMPad utilize RDF to define annotations. DBNotes and MONDRIAN utilize relational data to express annotations. The annotation system in [6] utilizes LEDA graph structure and XML Document Object Model (DOM) to express annotations. The data annotation models presented in this system also utilizes XML to structure data annotations. The system presented in [1] uses text-based annotations. All annotations, by default, based on the data annotation models presented in this paper, are accompanied with metadata information, such as the author's name and the creation time stamp. This behavior is common with annotations in Annotea. Both Annotea and the data annotation models presented in this paper allow annotations to cross-reference related annotations.

Annotations in Annotea reside in generic RDF databases, accessible through HTTP servers. Annotations presented in [1], [6], and in this paper reside outside the underlying database whose data they annotate. Data annotation documents based on the data annotation models presented in this paper reside on the file system of a computer system. SLIMPad modifies the base layer that it annotates in order to store annotations. DBNotes and MONDRIAN store annotations within the RDBMS. Columns may be added to relations in order to annotate relational data. Annotations at column and tuples levels can be expressed using this scheme. However, it is difficult to express annotations at the database, relation, and cell levels.

DBNotes extends SQL to pSQL, which specifies three propagation schemes for annotation propagation. pSQL can be used to query both data and annotations. MONDRIAN introduces color algebra that can also be used to query both data and annotations. In contrast, AnQL can only be used to query annotations. SQL must be used to retrieve the data that the annotation documents annotate. The query language presented in [1], similar to AnQL, utilizes data annotation graphs to query annotations. However, the semantics of nodes and edges in the data annotation graphs utilized by [1] and AnQL are different. AnQL queries annotations directly. The system presented in [1] maps nodes and edges of an annotation graph to relations stored in the underlying database, translates annotation queries into equivalent SQL queries.

III. Data Annotation Models

Fig. 1-5 present data annotation models that allow users to annotate relational data at five different levels of granularity - database, relation, column, tuple, and cell. XML is used to structure these data annotation models. XML was chosen because it provides several advantages over other data formats, such as simple text, e-mail, or electronic forms. XML is database-neutral and platform-independent. Therefore, it can be used to share annotations seamlessly regardless of the underlying database and operating system that contains the relational data to be annotated. Since XML supports Unicode, the support for expressing data annotations in several different languages is already built-in. [7].

Each of the data annotation models may be divided into identification, level, annotation, annotation metadata, and cross-reference modules. The identification module uniquely identifies a data annotation document via the `documentName` and `documentId` tags. The level module (via the element hierarchy enclosed within the tag `annotationAttachedTo`) represents the level of the relational data (database, relation, column, tuple, or cell) that the data annotation document annotates. The annotation module contains the actual data annotation. The annotation metadata module (via

```
<annotationDocument>
  <documentName>uniqueName</documentName>
  <documentId>uniqueId</documentId>
  <annotationAttachedTo>
    <database>databaseName</database>
  </annotationAttachedTo>
  <!--annotation and annotationMetadata may occur
      multiple times in a single document -->
  <annotation>
    <applicationDomainSpecificMetatag>
      dataAnnotation
    </applicationDomainSpecificMetatag>
    <annotationMetadata>
      <author>someone</author>
      <recorded>someDateAndTime</recorded>
    </annotationMetadata>
  </annotation>
  <referencedAnnotations>
    <documentNameList>
      <documentName>uniqueName</documentName>
      <documentName>uniqueName</documentName>
      …
     </documentNameList>
  </referencedAnnotations>
</annotationDocument>
```

Fig. 1. Database-Level Data Annotation Model

```
<annotationDocument>
  <documentName>uniqueName</documentName>
  <documentId>uniqueId</documentId>
  <annotationAttachedTo>
    <database>databaseName</database>
    <relation>relationName</relation>
  </annotationAttachedTo>
  <!--annotation and annotationMetadata may occur
      multiple times in a single document -->
  <annotation>
    <applicationDomainSpecificMetatag>
      dataAnnotation
    </applicationDomainSpecificMetatag>
    <annotationMetadata>
      <author>someone</author>
      <recorded>someDateAndTime</recorded>
    </annotationMetadata>
  </annotation>
  <referencedAnnotations>
    <documentNameList>
      <documentName>uniqueName</documentName>
      …
    </documentNameList>
  </referencedAnnotations>
</annotationDocument>
```

Fig. 2. Relation-Level Data Annotation Model

the `annotationMetadata` tag) maintains bookkeeping information (creation time stamp and author name) on the actual annotation. The cross-reference module allows annotations to cross-reference related annotations. Annotations are in essence immutable i.e. an annotation that overrides another annotation does not cause the original annotation to be erased.

An alternative to using the data annotation models presented in Fig. 1-5 is to keep notes in text documents. However, this approach presents a few problems. First, this alternative does not provide a uniform, consistent mechanism to express annotations. The reason is that each user would use his or her own personal format. Second, sharing text documents across platforms is difficult. Third, keeping annotations in text documents

```
<annotationDocument>
  <documentName>uniqueName</documentName>
  <documentId>uniqueId</documentId>
  <annotationAttachedTo>
    <database>databaseName</database>
    <relation>relationName</relation>
    <column>columnName</column>
  </annotationAttachedTo>
  <!--annotation and annotationMetadata may occur
      multiple times in a single document -->
  <annotation>
    <applicationDomainSpecificMetatag>
      dataAnnotation
    </applicationDomainSpecificMetatag>
    <annotationMetadata>
      <author>someone</author>
      <recorded>someDateAndTime</recorded>
    </annotationMetadata>
  </annotation>
  <referencedAnnotations>
    <documentNameList>
      <documentName>uniqueName</documentName>
           …
    </documentNameList>
  </referencedAnnotations>
</annotationDocument>
```

Fig. 3. Column-Level Data Annotation Model

```
<annotationDocument>
  <documentName>uniqueName</documentName>
  <documentId>uniqueId</documentId>
  <annotationAttachedTo>
    <database>databaseName</database>
    <relation>relationName</relation>
    <!--for composite primary keys, list values separated
        by commas-->
    <tuple>primary key</tuple>
  </annotationAttachedTo>
  <!--annotation and annotationMetadata may occur multiple
      times in a single document -->
  <annotation>
    <applicationDomainSpecificMetatag>
      dataAnnotation
    </applicationDomainSpecificMetatag>
    <annotationMetadata>
      <author>someone</author>
      <recorded>someDateAndTime</recorded>
    </annotationMetadata>
  </annotation>
  <referencedAnnotations>
    <documentNameList>
      <documentName>uniqueName</documentName>
      …
    </documentNameList>
  </referencedAnnotations>
</annotationDocument>
```

Fig. 4. Tuple-Level Data Annotation Model

```
<annotationDocument>
  <documentName>uniqueName</documentName>
  <documentId>uniqueId</documentId>
  <annotationAttachedTo>
    <database>databaseName</database>
    <relation>relationName</relation>
    <column>columnName</column>
    <!--for composite primary keys, list
        values separated by commas-->
    <tuple>primary key</tuple>
  </annotationAttachedTo>
  <!--annotation and annotationMetadata may occur
      multiple times in a single document -->
  <annotation>
    <applicationDomainSpecificMetatag>
      dataAnnotation
    </applicationDomainSpecificMetatag>
    <annotationMetadata>
      <author>someone</author>
      <recorded>someDateAndTime</recorded>
    </annotationMetadata>
  </annotation>
  <referencedAnnotations>
    <documentNameList>
      <documentName>uniqueName</documentName>
          …
    </documentNameList>
  </referencedAnnotations>
</annotationDocument>
```

Fig. 5. Cell-Level Data Annotation Model

does not automatically maintain metadata information on annotations. The data annotation models presented in this paper not only address all of these problems, but also present useful features, such as, cross-referencing and annotation metadata.

Another alternative to using the data annotation models presented in Fig. 1-5 is to declare `ANNOTATION` columns. An `ANNOTATION` column may be added for each data column. A single `ANNOTATION` column can express tuple-level annotations. However, this technique requires the addition of columns to an already deployed database. This might be problematic because it is common knowledge that DBAs are resistant to making any changes to already deployed databases. Secondly, this technique is only effective for annotating data at the column and tuple levels. The third problem with this technique is that all queries with "`select *`" must be modified to exclude `ANNOTATION` columns, particularly, if the users want to view data only. Similarly, all data insertion queries that use "`insert into relation values()`" without specifying the columns into which data has to be inserted must also be modified. All applications that retrieve data from the database, and manipulate and process this data must be modified accordingly to accommodate the changes in the underlying RDBMS. Such changes to the underlying database, and applications that work in association with the database, can prove to be particularly difficult for production systems that are typically heavily used. Another major disadvantage of this technique is that it might not be possible to cross-reference related annotations since the annotations stored in database columns are no longer uniquely distinguishable.

Although, it might be feasible to utilize `ANNOTATION` columns to annotate relational data at the column and tuple level, this technique cannot be used effectively to annotate at the database, relation, and cell levels. This is because it is hard to decide which relation should host the column that annotates the entire database. Similarly a column must be added to each relation to store annotations at the relation and cell levels.

Data annotation documents based on the models presented above reside on the file system provided by a computer system. When the number of data annotation documents becomes prohibitively large, a smart indexing scheme would become necessary to quickly locate and retrieve annotations. An alternative is to store annotation documents on disks that are being designed specifically to store and efficiently retrieve semi-structured data [8]. Annotation documents may also be stored inside an RDBMS. XML documents stored in their entirety inside RDBMSs present the same problems as those presented above. XML documents can be shredded and parsed in order to convert them into tabular format suitable for mapping on to relational data [9]. This defeats the whole concept of semi-structured data format - the data to begin with is not suitable for storage into an RDBMS. Moreover, putting shredded XML documents back together is expensive since it requires the computation of several joins. Annotation documents can also be stored in XML native databases. However, retrieving these annotations from XML native databases requires the knowledge of XQuery. XQuery is a complex query language, and might present users with a steep learning curve.

IV. Annotation Query Language (AnQL)

AnQL, an SQL-like query language, is used to query data annotation documents based on the models presented in Fig. 1-5. AnQL is designed to take advantage of the existent abundant SQL knowledge and skill set. AnQL query operations include *select*, *project*, *natural join*, and *union*. A naive, yet clever, storage scheme is employed to facilitate AnQL query processing. All annotation documents pertaining to one database are kept in a separate directory. In other words, annotation documents that pertain to say, an `Actors` database are kept in separate directories. Within a directory, annotations pertaining to each level are kept in separate subdirectories. In other words, annotations at cell-level and database-level reside in separate subdirectories. An alternative is to use a tagged file format in which all annotations are kept in a single file. Tags uniquely identify the start and end of each annotation. Storing annotations in a tagged file requires an additional access to the index file for each query. Moreover, the index file must be updated for each addition and deletion. Using a tagged file would also make the annotations and their querying system dependent.

AnQL query engine utilizes *data annotation graph generation* and *data annotation graph traversal* functions to process AnQL queries. The *data annotation graph generation* ( ) function generates a *data annotation graph* corresponding to a well-formed and validated data annotation document provided as input. A data annotation graph is a special graph structured especially for AnQL query processing and is modeled in spirit of the XQuery data model [10]. Its nodes correspond to the elements of a data annotation document, and edges depict the hierarchical relationship between the elements. The *data annotation graph traversal* (Σ) function traverses a data annotation graph in depth-first manner. The function accepts as input a start node contained within a well-formed and validated data annotation document, and returns a set of nodes directly connected to the start node. The *transitive closure* (Σ^+) of the *data annotation graph traversal* accepts the same input as the *data annotation graph traversal* function, but outputs the set of all nodes that are connected directly or indirectly to the start node.

AnQL's *select* ( ) operation accepts as input a well-formed and validated data annotation document and a boolean predicate of the form `element=value` or `elementValue=value`. It returns the node hierarchy that satisfies the boolean predicate of the input document. In order to process a *select* operation, the query engine generates a data annotation graph, using the *data annotation graph generation* function, corresponding to the input data annotation document. The query engine then traverses the data annotation graph, using the *data annotation graph traversal* function, in order to find a node that satisfies the boolean predicate. It returns as the result set the node hierarchy that satisfies the boolean predicate.

AnQL's *project* ( ) operation accepts as input a non-boolean constraint (a maximum of three keywords), and a *project criterion* (the level - database, relation, column, tuple, or cell). It returns the node hierarchy that satisfies the non-boolean constraint (keywords joined by *and* logic) and the *project criterion*. The processing of *project* operation is similar to that of a *select* operation.

AnQL's *natural join* ($\bowtie$) operation joins data annotation documents at a specific level based on a *natural join criterion* (author or creation time stamp). If the *natural join criterion* is not specified, the operation simply appends all the data annotation documents at the specified level. The result set contains the hierarchy within the `annotation` tag. AnQL's definition of *natural join* also includes the definition of an *intersection* operation. AnQL's query engine compares the value of the corresponding nodes (either author or creation time stamp) in data annotation graphs corresponding to all documents at the specified level. If the values match, the query engine includes the `annotation` in the result set.

The *union* ($\cup$) operation generates a consolidated report that groups annotations at a specified level i.e. all data annotations pertaining to a particular cell. The query engine browses through the relevant directory (determined by the *union criterion*), and groups all data annotations at one level into one large document to provide the users with a comprehensive, consolidated view of annotations at that level. The processing of *union* operation is inefficient since no indexing mechanism has been employed. Therefore, computation of a *union* operation is done routinely during relatively non-busy times. The result set for all operations, by default, returns the element hierarchy within `annotationMetadata` when the hierarchy within the `annotation` tag is returned.

Select, *project* and *natural join* operations may be combined to form *Select-Project-Join (SPJ)* queries, and *select*, *project*, and *union* operations may be combined to form *Select-Project-Union (SPU)* queries.

A *Select-Project-Natural Join* Query Example. The example *SPJ* query - `element=annotation` "`water`" $\bowtie_{RudyCampos}$ `(RealEstate/Features/Property_Id/PI2)` is issued over an example `RealEstate` database (presented below). Fig. 6 and 7 present example data annotation documents that annotate data in the `RealEstate` database. This query signifies the join of cell-level annotations whose author is Rudy Campos.

Listings

PptyId	MLS	Street	CITY	ZIP
PI1	387811	2928 Leigh	San Jose	95127
PI2	401891	Out of Area	Out of Area	95148

Features

PptyId	LotSize	Bed	Bath	Age	Style
PI1	-	4	2	47	Detached
PI2	0.62	-	-	-	Land

Details

FtrId	FtrName
FI1	Fireplace
FI10	L-Shaped Pool

Agents

Id	Name
AG1	Rudy Campos
AG2	Carla Gallegos

Contains

PptyId	FtrID
PI1	FI1
PI2	FI2

Sold

Id	PptyId
AG1	PI1

AnQL's query engine first processes the *natural join* operation. Using the *data annotation graph generation* function, it generates data annotation graphs corresponding to the input documents. Next, using the *data annotation graph traversal* function, it traverses to the author node and compares them. If the nodes match, the query engine returns the node hierarchy within the `annotation` tag. Next, the query engine processes the *project* clause by searching for the keyword "water", in nodes of type `elementValue` and `textValue`, within the intermediate result set returned by *natural join*. The query engine, next, processes the *select* clause by traversing through the nodes of type `element` within the intermediate result set generated by the processing of *natural join* and *project* clauses. Fig. 8 presents the result set of the example query. [11] presents detailed discussion of AnQL's operations, along with several examples.

V. Conclusions and Future Work

This paper presented data annotation models that can annotate relational data at five different levels - database,

```
<annotationDocument>
  <documentName>REFeaturesPropertyIdPI2</documentName>
  <documentId>RE11</documentId>
  <annotationAttachedTo>
    <database>REAL_ESTATE</database>
    <relation>FEATURES</relation>
    <column>PROPERTY_ID</column>
    <tuple>PI2</tuple>
  </annotationAttachedTo>
  <annotation><comment>
    Build your dream home. Sunny and private. Water
    and electricity at site. Plans and permits
    approved and ready for a 2683+ sq. ft home. Septic
    and geo approved. </comment>
    <annotationMetadata>
      <author>Rudy Campos</author>
      <recorded>Apr 1, 2004 10:15:25 AM</recorded>
    </annotationMetadata>
  </annotation>
</annotationDocument>
```

Fig. 6. Example Data Annotation Document (REFeaturesPropertyIdPI2)

```
<result>
  <annotationAttachedTo>
    <database>REAL_ESTATE</database>
    <relation>FEATURES</relation>
    <column>PROPERTY_ID</column>
    <tuple>PI2</tuple>
  </annotationAttachedTo>
  <annotation>
    <comment>
      Build your dream home. Sunny and private.
      Water and electricity at site. Plans
      and permits approved and ready for a 2683+
      sq. ft home. Septic and geo-approved.
   </comment>
    <annotationMetadata>
      <author>Rudy Campos</author>
      <recorded>May 28, 2004 12:15:14 PM</recorded>
    </annotationMetadata>
  </annotation>
</result>
```

Fig. 8. Result Set of Example SPJ Query

```
<annotationDocument>
  <documentName>REFeaturesLotSizePI2</documentName>
  <documentId>RE9</documentId>
  <annotationAttachedTo>
    <database>REAL_ESTATE</database>
    <relation>FEATURES</relation>
    <column>LOT_SIZE</column>
    <tuple>PI2</tuple>
  </annotationAttachedTo>
  <annotation>
    <description>
        Lot size is in acres.
    </description>
    <annotationMetadata>
      <author>Rudy Campos</author>
      <recorded>May 28, 2004 12:15:14 PM</recorded>
    </annotationMetadata>
  </annotation>
</annotationDocument>
```

Fig. 7. Example Data Annotation Document (REFeaturesLotSizePI2)

relation, column, tuple, and cell, along with an SQL-like query language - AnQL. The motivation for the models is the simplicity and ease of understanding. The models are flexible, extensible, customizable, database-neutral, and platform-independent. These models may be extended for other data models, such as hierarchical and object-oriented. AnQL is designed to exploit the abundant knowledge and skill set of SQL. The annotation system presented in this paper does not inflict any strucutral or schematic changes to the underlying database. AnQL's *project* operation may be extended to more keywords and *or* and *not* logic. Cross-referencing can be enhanced with XLink or XInclude. The *union* operation can benefit from a smart indexing scheme. A system, as a virtualization over the underlying RDBMS, can be developed to incorporate these models and AnQL.

REFERENCES

[1] M. Gertz, K.-U. Sattler, F. Gorin, M. Hogarth, and J. Stone, "Annotating scientific images: A concept-based approach," in *SSDBM '02: Proceedings of the 14th International Conference on Scientific and Statistical Database Management*, (Washington, DC, USA), pp. 59–68, IEEE Computer Society, 2002.

[2] J. Kahan and M.-R. Koivunen, "Annotea: an open rdf infrastructure for shared web annotations," in *WWW '01: Proceedings of the 10th international conference on World Wide Web*, (New York, NY, USA), pp. 623–632, ACM Press, 2001.

[3] L. Chiticariu, W.-C. Tan, and G. Vijayvargiya, "DBNotes: a post-it system for relational databases based on provenance," in *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 942–944, ACM Press, 2005.

[4] F. Geerts, A. Kementsietsidis, and D. Milano, "Mondrian: Annotating and querying databases through colors and blocks," in *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, (Washington, DC, USA), p. 82, IEEE Computer Society, 2006.

[5] D. Lois, M. David, B. Shawn, D. Longxing, W. Mathew, G. Paul, A. Joan, L. Mary, and L. J. A., "Bundles in captivity: An application of superimposed information," tech. rep., 2000.

[6] E. Egyed-Szigmond, Y. Pri, A. Mille, and J. Pinon, "A graph-based audiovisual document annotation and browsing system," in *RIAO (CAIR)*, April 2000.

[7] K. B.Sall, *XML Family of Specifications A Practical Guide*. Boston MA: Addison Wesley, 2002.

[8] M. Bhadkamkar, V. Hristidis, and R. Rangaswami, "Efficient native XML storage," tech. rep., Florida International University, April 2005.

[9] A. H. Al-Azzawe, "IBM video online for e-business - DB2 inbound XML data fragments," *http://www-106.ibm.com/developerworks/db2/library*, June 2004.

[10] D. Chamberlin, *XQuery from the Experts A Guide to the W3C XML Query Language*. Boston, MA: Addison-Wesley, 2004.

[11] N. Bhatnagar, "Data annotation models and annotation query language," Master's thesis, California State University, Chico, May 2006.