AnQL: A Query Language
for Annotation Documents

Neerja Bhatnagar, Ben 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 to express semantic data annotations 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, the models presented in this paper 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 [2] and XPath [3] are complex query languages. Learning these languages might present a steep learning curve. SQL-XML [4] 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 types of 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. 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.

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 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, 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

World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:8, 2007

to the underlying RDBMS.

The rest of this paper is organized as follows – Section 2 compares and contrasts the work presented in this paper with that presented in Annotea, DBNotes, MONDRIAN, SLIMPad, and few others; Section 3 presents the data annotation models that allow users to annotate relational data at five different levels; Section 4 presents the SQL-like query language, AnQL, and its query processing operations; and conclusions and future work are discussed in Section 5.

II. RELATED WORK

This section compares and contrasts the annotation models presented in this paper with Annotea [5], DBNotes [6], MONDRIAN [7], SLIMPad [8], and with those presented in [1] and [9]. 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 of granularity – database, relation, column, tuple, and cell. DBNotes focuses on the where-provenance and propagation of annotations. 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 [9] 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 [9] utilizes LEDA graph structure and XML Document Object Model (DOM) to express annotations. The data annotation models presented in this paper also utilize 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], [9], 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 [10]. Annotations at column and tuples levels can be expressed using this scheme. However, by using this scheme, it is difficult to express annotations at the database, relation, and cell levels.

III. DATA ANNOTATION MODELS

Figures 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 text, e-mail, or electronic forms. XML is database-neutral and platform-independent. XML's database-neutrality allows the use of same data annotation models to express data annotations on base data that resides in heterogeneous databases, such as, IBM DB2 UDB [11], Oracle [12], and SQL Server [13] etc. XML's platform-independence allows users to express, share, and utilize data annotations irrespective of the operating system and platform they use. Since XML supports Unicode, the support for expressing data annotations in several different languages is already built-in [14]. Therefore, it can be used to share data seamlessly regardless of the underlying database and operating system.

```
<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

World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:8, 2007

```
<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

Each of the data annotation models presented in Figures 1-5 may be further divided into modules, namely, identification, level, annotation, annotation metadata, and cross-reference. These modules are illustrated in Fig. (6). The identification module uniquely identifies a data annotation document. It contains the hierarchy within the documentName and documentId tags. The level module represents the level (database, relation, column, tuple, or cell) that a data annotation document annotates. The level module is identified by the tag annotationAttachedTo. The annotation module contains the actual data annotation, and the annotation metadata module maintains bookkeeping information (creation time stamp and author) on the actual annotation. The annotation module is characterized by the tag applicationDomainSpecificTag, which is enclosed within the annotation tag. The annotation metadata module contains the node hierarchy within the

```
<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

```
…
<documentName>…</documentName>          ← Identification
<documentId>…</documentId>
<annotationAttachedTo>
  <database>…</database>                ← Level
</annotationAttachedTo>
<annotation>
  <applicationDomainSpecificMetatag>
    …                                   ← Annotation
  </applicationDomainSpecificMetatag>
  <annotationMetadata>
    <author>…</author>
    <recorded>…</recorded>             ← Annotation Metadata
  </annotationMetadata>
</annotation>
<referencedAnnotations>
  <documentNameList>
    <documentName>…</documentName>
       …
  </documentNameList>                    ← Cross-Reference
</referencedAnnotations>
…
```

Fig. 6 modules

annotationMetadata tag. By default, each annotation is accompanied with the annotation metadata, namely the name of the author of the annotation and the creation time stamp of the annotation. The cross-reference module (characterized by the element hierarchy within the referencedAnnotations tag) allows related annotations to cross-reference each other. Annotations are, in essence, immutable *i.e.* an annotation that semantically overrides another annotation does not cause the original annotation to be erased.

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 [15]. Annotation documents may also be stored inside a relational database. XML documents stored in their entirety inside relational databases present a few problems. The first problem is that the storage of annotation documents inside the relational database that contains the base data being annotated would require structural and schematic changes to the database. The second problem is that all

World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:8, 2007

queries that use "select *" might need to be modified. Similarly, "insert into" queries that do not mention the column names explicitly would need to be modified. Thirdly, an SQL-XML parser [4] would be needed to process queries that pertain to annotation documents. XML documents can be shredded and parsed in order to convert them into tabular format suitable for mapping on to relational data [16]. This defeats the whole concept of semi-structured data format – the data to begin with is not suitable for storage into a relational database. 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.

An alternative to using the data annotation models presented in Figures 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 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 Figures 1-5 is to declare ANNOTATION columns. An ANNOTATION column may be added for each data column. A single ANNOTATION column per relation 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 database. 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.

Example Data Annotation Documents. Assume that the manager of a real estate firm decides to transfer the listings for over a million dollars to a separate database. The manager can utilize the data model presented in Fig. (1) to annotate the existing database in order to communicate with the DBA. Fig. (7) presents this example annotation document. The DBA can review the manager's annotations, make appropriate changes, and inform the manager regarding the change via an annotation. In this case, data annotations save time and resources by eliminating the need to set up a meeting with the DBA in order to explain a simple change to him or her.

```
<annotationDocument>
  <documentName>RE1</documentName>
  <documentId>RE1</documentId>
  <annotationAttachedTo>
    <database>REAL_ESTATE</database>
  </annotationAttachedTo>
  <annotation>
    <actionRequired>
      Please create a separate database for properties
      listed at $1,000,000 or more. Separate out real
      estate agents that deal exclusively in these
      properties. Proposed database name –
      LUXURY_ESTATES. Name the relations using firm's
      standard naming conventions.
    </actionRequired>
    <annotationMetadata>
      <author>theManager</author>
      <recorded>April 2, 2004 10:37:15 PM</recorded>
    </annotationMetadata>
  </annotation>
</annotationDocument>
```

Fig. 7 example database-level annotation document

Fig. (8) presents an example data annotation document at the relation level that a paleontologist might use to annotate a relation in a paleontology database with information on topography or habitat existent during a particular period. This document annotates the LateTriassic relation. The LateTriassic relation may contain information on Coelophysis, Cynodant, and Placerias, among others. However, these relations would not contain information on the state of the earth, or the habitat, or the weather during the LateTriassic period.

```
<annotationDocument>
  <documentName>plntlogyLateTriassic1</documentName>
  <documentId>Pa11</documentId>
  <annotationAttachedTo>
    <database>PALEONTOLOGY</database>
    <relation>LATE_TRIASSIC</relation>
  </annotationAttachedTo>
  <annotation>
    <description>
      Neither flowering plants nor grass existed. The
      ground was covered with ferns and mosses. Plant life
      was very drab – just green and brown in color.
    </description>
    <annotationMetadata>
      <author>paleontologist</author>
      <recorded>Apr 17, 2004 8:02:08 PM</recorded>
    </annotationMetadata>
  </annotation>
</annotationDocument>
```

Fig. 8 example relation-level annotation document

Fig. (10) illustrates how a chef can utilize the column-level data annotation model presented in Fig. (3) to annotate the

World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:8, 2007

Quantity column in the relation CondimentList (see Fig. (9)). Fig. (11) presents an example cell-level data annotation document that a chef can use to annotate the cell that contains the data value Turmeric in the relation CondimentList.

	Id	Name	Quantity
CondimentList	1	Salt	3
	8	Turmeric	2

Fig. 9 relation CondimentList

Fig. (13) presents an example data annotation document at the tuple level. The department secretary uses the tuple-level data annotation model presented in Fig. (4) to inform the graduate coordinator that John Doe has accepted their offer of admission, and therefore, his record must be moved from relation Offered to Accepted (see Fig. (12)). For more details on these models, please refer to [17].

```
<annotationDocument>
  <documentName>cookingRecipeQuantity1</documentName>
  <documentId>cooking1</documentId>
  <annotationAttachedTo>
    <database>COOKING</database>
    <relation>RECIPE</relation>
    <column>QUANTITY</column>
  </annotationAttachedTo>
  <annotation>
    <note> All quantities in tablespoons.
     Conversion: 1 tablespoon = 5 milligrams </note>
    <annotationMetadata>
      <author>Chef Alice</author>
      <recorded>May 27, 2004 9:12:24 AM</recorded>
    </annotationMetadata>
  </annotation>
</annotationDocument>
```

Fig. 10 example column-level annotation document

IV. ANNOTATION QUERY LANGUAGE (AnQL)

AnQL, an SQL-like query language, is used to query data annotation documents based on the models presented in Figures 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 (see Fig. (14)) is employed to facilitate AnQL query processing. All annotation documents that pertain to one database are kept in a separate directory from the annotation documents that pertain to another

```
<annotationDocument>
  <documentName>cookingCondimentList</documentName>
    <documentId>cooking2</documentId>
  <annotationAttachedTo>
    <database>COOKING</database>
    <relation>CONDIMENT_LIST</relation>
    <column>CONDIMENT_NAME</column>
    <tuple>8</tuple>
  </annotationAttachedTo>
  <annotation>
   <caution>
    Be very careful with turmeric. Turmeric leaves
    yellow stains on everything incl. counter tops,
    and clothes. These stains are extremely difficult
    to remove.
   </caution>
   <annotationMetadata>
     <author>alice</author>
     <recorded>Apr 27, 2004 4:18:16 AM</recorded>
   </annotationMetadata>
  </annotation>
</annotationDocument>
```

Fig. 11 example cell-level annotation document

NEW_STUDENT_INFO

STUDENT_ID	NAME	DOB
999888777	John Doe	1/1/1984
123456789	Jane Smith	1/2/1986

ADMIT_INFO

STUDENT_ID	YEAR	DEGREE	PROGRAM
999888777	2004	B.S.	ELEC. ENGG.
123456789	2004	M.S.	COMP. SCI.

Fig. 12 admission database

```
<annotationDocument>
  <documentName>admissionsNewStudentInfo</documentName>
  <documentId>admn1</documentId>
  <annotationAttachedTo>
    <database>ADMISSIONS</database>
    <relation>NEW_STUDENT_INFO</relation>
    <tuple>999888777</tuple>
  </annotationAttachedTo>
  <annotation>
    <comment>
      John Doe (Id 999888777) accepted offer.
      Please change status.
    </comment>
    <annotationMetadata>
      <author>departmentSecretary</author>
      <recorded>May 27, 2004 4:03:08 PM</recorded>
    </annotationMetadata>
  </annotation>
</annotationDocument>
```

Fig. 13 example tuple-level annotation document

database. In other words, annotation documents that pertain to say, an Actors database, are kept in a separate directory from those that pertain to say, a Real Estate database. 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
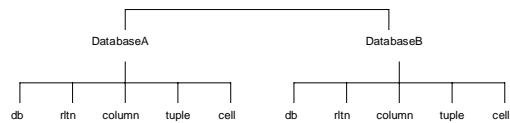

Fig. 14 naïve storage scheme

process AnQL queries. The *data annotation graph generation* (φ) 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. The nodes of a data annotation graph correspond to the elements of a data annotation document, and its edges depict the hierarchical relationship between the elements. Fig. (15) presents the cell-level data annotation graph. The data annotation graphs at the other levels (database, relation, column, and tuple) are similar, but they correspond to their data annotation models. Nodes in

World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:8, 2007

a data annotation graph may be classified as follows –

- *Root* - The *root* node of a data annotation graph is annotationDocument.

- *Element* - An *element* node contains an element defined in an annotation document. Nodes of type element include `annotationDocument`, `documentName`, `documentId`, `annotationAttachedTo`, `database`, etc.

- *ElementValue* - A node of type *elementValue* is a child node of type element, and contains the actual value of an element.

- *TextValue* - A node of type *textValue* is also a child of a node of type element, but contains the actual data annotation.
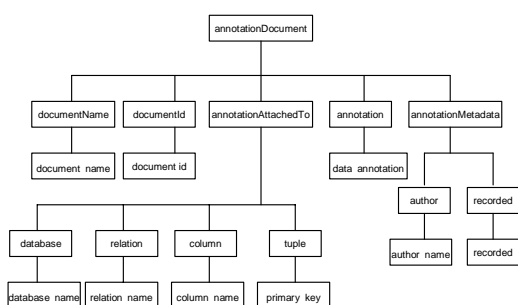

Fig. 15 cell-level data annotation graph

An example database, `RealEstate` (see Fig. (16)), is used through the rest of this paper to illustrate AnQL operations. The example `RealEstate` database maintains records of property listings, real estate agents, and which real estate agent sold which property. Figures (17) and (18) present example data annotation documents that annotate data in the RealEstate database.

Listings

PptyId	MLS	Street	City
PI1	387811	2928 Leigh	San Jose
PI2	401891	Out of Area	Out of Area

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	FI10

Sold

Id	PptyId
AG1	PI1

Fig. 16 example RealEstate 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. 17 example data annotation document - 1

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. REFeaturesPropertyIdPI2 is the result set generated by

```
<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. 18 example data annotation document – 2

$\Sigma_{documentName}$ (REFeaturesLotSizePI2). In this function, documentName is the start node, and REFeaturesLotSizePI2 is the input annotation document.

The *data annotation graph traversal* function returns the node that is directly connected to the node documentName. 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. Fig. (19) presents the result set of $\Sigma^+_{documentName}$ (REFeaturesLotSizePI2).

```
<documentName>
  REFeaturesPropertyIdPI2
</documentName>
```

Fig. 19 *transitive closure* result set

AnQL's *select* (σ) 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*

World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:8, 2007

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. Fig. (20) presents the result set generated by $\sigma_{element=annotation}$. This query signifies the selection and retrieval of the part of the annotation document presented in Fig. (18) that occurs within the start and end tags of the annotation element in the document.

AnQL's *project* (π) 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. Fig. (21) presents the result set generated

```
<result>
  <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>
</result>
```

Fig. 20 *select* result set

by issuing the query, $\pi_{\text{"sunny"}}RealEstate/Features/PptyId/PI2$, on the data annotation documents of Figures (17) and (18). This query signifies the search for the keyword "sunny" in the example data annotation documents.

```
<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>
  </annotation>
</result>
```

Fig. 21 *project* result set

AnQL's *natural join* (NJ) 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. Fig. (22) presents the result set generated by issuing the query, NJ_{author} (RealEstate/Features/LotSize/PI2), on the example annotation documents of Figures (17) and (18). This query indicates to the AnQL query engine to compare the authors within the cell-level example annotation documents, and join them if their authors are the same.

The *union* (U) 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

```
…
  <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>
  <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 12:15:14PM</recorded>
    </annotationMetadata>
  </annotation>
…
```

Fig. 22 *natural join* result set

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 can be done routinely during relatively non-busy times. Fig. (23) presents a typical *union* result set.

The result set of 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 – $\sigma_{element=annotation}$ π_{water} $nj_{Rudy\ Campos}$ (RealEstate/Features/Property_Id/PI2) is issued over an example `RealEstate` database (see Fig. 16). This query signifies the join of cell-level annotations whose author is Rudy

World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:8, 2007

Campos.

```
…
  <annotationAttachedTo>
    <database>databaseName</database>
    <relation>relationName</relation>
    <column>columnName</column>
    <tuple>primaryKey</tuple>
  </annotationAttachedTo>
  <annotation>
    <domainSpecificTag> … </domainSpecificTag>
    <annotationMetadata> … </annotationMetadata>
  </annotation>
…
```

Fig. 23 Typical *Union* Result Set

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 then 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. (24) presents the result set of the example query. [17] presents detailed discussion on AnQL's operations, along with several examples.

```
<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. 24 *SPJ* Query Result Set

V. CONCLUSION AND FUTURE WORK

This paper presents data annotation models that can annotate relational data at five different levels of granularity – database, relation, column, tuple, and cell. The motivation for these models is simplicity and ease of understanding. The models are flexible, extensible, customizable, database-neutral, and platform-independent. These models may be extended to other data models, such as hierarchical and object-oriented. This paper also presents an SQL-like query language - AnQL. AnQL is designed to exploit the abundant knowledge and skill set of SQL. The annotation system presented in this paper does not inflict any structural 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[18] or XInclude [19]. 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] D. Chamberlin, *XQuery from the Experts A Guide to the W3C XML Query Language*. Boston, MA: Addison-Wesley, 2004.

[3] "XML path language 1.0," 1999.

[4] C. M. Saracco, "Query DB2 XML data with SQL," 2006.

[5] 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.

[6] L. Chiticariu, W.-C. Tan, and G. Vijayvargiya, "DBNotes: a PostIt 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.

[7] 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.

[8] L. Delcambre, D. Maier, S. Bowers, L. Deng, M. Weaver, P. Gorman, J. Ash, M. Lavelle, and J. A. Lyman, "Bundles in captivity: An application of superimposed information," tech. rep., 2000.

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

[10] D. Bhagwat, L. Chiticariu, W.-C. Tan, and G. Vijayvargiya, "An annotation management system for relational databases," *VLDB*, 2004.

[11] IBM, "DB2 9 for Linux UNIX and Windows."

[12] Oracle, "Oracle Database."

[13] Microsoft, "Microsoft SQL Server."

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

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

[16] 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.

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

[18] "Recommendation, XML linking language 1.0," 2001.

[19] "Working draft, XML inclusions (XInclude) 1.0," 2003.

Neerja Bhatnagar received M.S. in Computer Science from California State University, Chico in 2006. She is currently pursuing PhD in Computer Science at University of California, Santa Cruz.

Dr. Ben A. Juliano is a Computer Science Associate Professor at California State University, Chico. He is the director of the Institute for Research in Intelligent Systems (IRIS) and the co-director of the Intelligent Systems Laboratory (ISL). IRIS and ISL focus on intelligent systems research that pertains to autonomous search and rescue. He teaches Theory of Computing, Data Mining, Fuzzy Logic, and Computer Architecture, among others.

Dr. Renee S. Renner is a Computer Science Associate Professor at California State University, Chico, focuses her research efforts in ISL/IRIS labs, and towards the development of a large-scale multi-agent systems tool for soft computing, complexity analysis, and intelligent systems applications. She teaches courses on Robotics, Artificial Intelligence, Expert Systems, and Databases, among others.