Linkage Query Writer

Oktie Hassanzadeh, Reynold Xin,
Renée J. Miller
University of Toronto
{oktie,rxin,miller}@cs.toronto.edu

Anastasios Kementsietsidis, Lipyeow Lim,
Min Wang
IBM T.J. Watson Research Center
{liplim,akement,min}@us.ibm.com

ABSTRACT

We present Linkage Query Writer (LinQuer), a system for generating `SQL` queries for semantic link discovery over relational data. The LinQuer framework consists of (a) `LinQL`, a language for specification of linkage requirements; (b) a web interface and an API for translating `LinQL` queries to standard `SQL` queries; (c) an interface that assists users in writing `LinQL` queries. We discuss the challenges involved in design and implementation of a declarative and easy to use framework for discovering links between different data items in a single data source or across different data sources. We demonstrate different steps of the linkage requirements specification and discovery process in several real world scenarios and show how the LinQuer system can be used to create high-quality linked data sources.

1. INTRODUCTION

Discovering links between different entities in data sources is a challenging task and an attractive research area. Existence of links add value to data sources, enhance data access and information discovery, and allow or enhance many increasingly important data mining tasks [4]. When data sources are not linked, they resemble *islands of data* (or *data silos*), where each island maintains only part of the data necessary to satisfy a user's information needs. Penetrating these silos to both understand their contents and understand potential semantic connections is a daunting task. What users and data publishers need is automated support for creating referential links between data that reside in different sources and that are semantically related. Finding such links often requires the use of approximate matching (to overcome syntactic representational differences and errors) and semantic matching (to find specific semantic relationships). Furthermore, both types of matching must be tightly integrated to accommodate for the tremendous heterogeneity found in the data that reside in today's information systems.

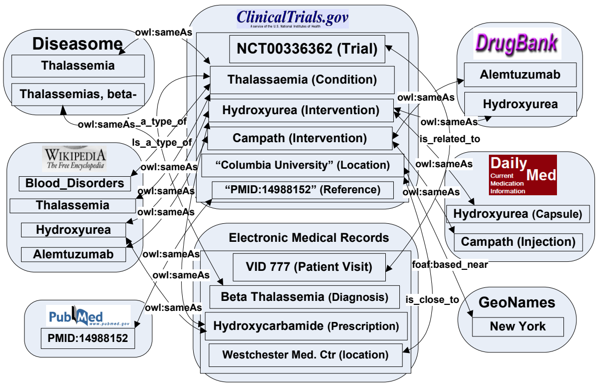


Figure 1: Sample entities from a clinical trials database interlinked with several related data sources.

In spite of their importance, research in finding such semantic links has mainly focused on a more restricted version of the problem, namely, on *entity resolution*, i.e., the identification of entities in disparate sources that represent the same *real-world* entity (see [3] and references therein). More recently, there has been interest in finding other types of semantics links. The general problem investigated in LinQuer is the discovery of links between entities that may refer to the same or semantically related *real-world* entities. Consider, for example, a scenario in which a data publisher wants to publish an online data source of clinical trials with links to other online medical data sources, and a related scenario where a clinician needs to link patient records from an Electronic Medical Record (EMR) database to the related clinical trials. Figure 1 shows a sample of the entities in the data sources in these scenarios as well as several typed links that could be found between these entities.

The importance of finding such links is also highlighted by the recent efforts of the Linking Open Data (LOD) community project at W3C [1]. In the context of the LOD project, a number of tools and frameworks have been developed that allow for the generation and publication of linked data from relational databases [7]. Although these frameworks simplify the process of publishing linked data, the number of links between existing LOD data sources is three orders of magnitude less than the number of base data triples. The majority of the links are either a result of existing links and unique identifiers in the relational data (e.g., links between two data items that are both derived from a Wikipedia page, both having the same URL), or a laborious implementation of a semi-automatic and domain-specific linkage algorithm.

VLDB '09, August 24-28, 2009, Lyon, France

In the absence of globally unique identifiers, and without automated support for link discovery, users must experiment with a myriad of different linkage methods to find one that suits their needs. In Figure 1, matching the patient visit with the clinical trial requires matching "Beta Thalassemia" with "Thalassemia" using the semantic knowledge from a medical source (such as the popular NCI thesaurus) that "Beta Thalassemia" *is a type of* "Thalassemia" along with a semantic matching technique such as Oracle's ontology-based matching [2], and then matching "Thalassemia" with "Thalass*a*emia" using approximate string matching. Furthermore, the locations "Westchester Med. Ctr" from the patient visit and "Columbia University" from the trial can be matched based on additional geographical information (possibly using a source of geographical information such as GeoNames[1]) that states that both locations are in the same state (New York) or within a certain distance.

To overcome these challenges, we have developed Linkage Query Writer (LinQuer), a generic and extensible framework for integrating link discovery methods. Our goal is to facilitate experimentation and help users find and tune the link discovery methods that will work best for their domain and application. Our framework permits the discovery of links between autonomous relational sources. We introduce LinQL, an extension of SQL that integrates querying with link discovery methods. A query written in LinQL identifies (a) a set of *source* data; (b) a set of *target* data; and (c) a set of methods(s) to use in finding links between the specified source and target data. Our framework includes a variety of native link discovery methods and is extensible to additional methods, written as user-defined functions (UDF). This permits users to interleave declarative queries with interesting combinations of link discovery requests. The link discovery methods may be syntactic (approximate match or similarity functions), semantic (using ontologies or dictionaries to find specific semantic relationships), or a combination of both.

In this demonstration, we seek to show several key aspects of the LinQuer framework.

- We show that by integrating *ad hoc* querying and a rich collection of link discovery methods, our framework supports interactive application and evaluation of link discovery. A common way to use our framework would be to declaratively specify a portion of the data of interest and invoke one or more link discovery methods. The results can be evaluated by a user or automated technique to help in refining or comparing different methods.

- We show how our declarative specification is translated into SQL code. Often, such programs are implemented using programming languages like Java or C by third-party developers, and are automatically invoked with arguments defined through the declarative specification. As such, for the data publishers these programs act as *black-boxes* that sit outside the data publishing framework and whose modification requires the help of these developers. We address these shortcomings by providing a native SQL implementation for a number of link discovery algorithms. Such native, declarative implementations make the methods highly extensible and customizable.

- In order to show the effectiveness of our framework, we have applied our framework to finding links between a real clinical trial data source and several other data sources including patient data and Wikipedia articles. We show the summary of the results of extensive experimentation we have performed for this application. The result of applying the LinQuer system is also showcased in the context of the Linked Clinical Trials (LinkedCT) project[2], the first linked data source of clinical trials, which contains a large number of links to other data sources of medical information.

```
linkspec_stmt:= CREATE LINKSPEC linkspec_name
                AS link_method opt_args opt_limit;
linkindex_stmt:= CREATE LINKINDEX opt_idx_args
                 linkindex_name ON table(col)
                 USING link_method;
link_method:= native_link | link_clause_expr | UDF;
native_link:= synonym | hyponym | stringMatch;
link_clause_expr:= link_clause AND link_clause_expr
                   | link_clause OR link_clause_expr
                   | link_clause;
link_clause:= LINK source WITH target
              USING link_terminal opt_limit;
link_terminal:= native_link | UDF | linkspec_name
opt_limit:= LINKLIMIT number;
```

Figure 2: The LinQL grammar (main components)

2. THE LINQUER SYSTEM

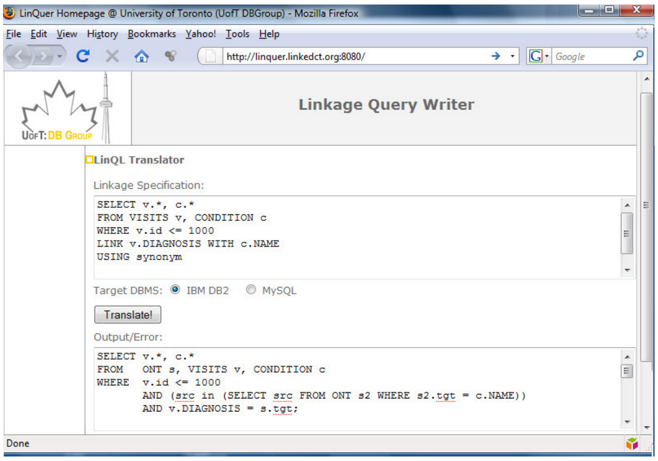
The LinQL Language We introduce the syntax (grammar) and semantics of the LinQL declarative link specification language. We discuss the characteristics of the link finding primitives required by a flexible framework that is capable of effective link discovery in many real world scenarios, and show how LinQL supports such primitives. Although the linkage specification elements we discuss below are expressible in a number of different languages and notations (e.g., RDF/XML, N3, NTriples), in our framework we choose for consistency an SQL-like syntax.

A *link specification*, or *linkspec* for short, defines the conditions that two given values must satisfy before a link can be established between them. In more detail, a linkspec is defined using the grammar of Figure 2 (we omit here the full grammar and only present its main components). As shown in the figure, a CREATE LINKSPEC statement defines a new linkspec and accepts as parameters the name of the defined linkspec and the names of the relation columns whose values need to be linked. To create such links, our framework provides several *native* (or *built-in*) methods including synonym, hyponym, and a variety of string matching methods. The native string matching methods are based on state-of-the-art string similarity predicates that can be implemented in SQL (refer to the web interface for a list of all the supported similarity predicates). Such native methods can be used as such or they can be customized by setting their parameters.

A link specification can also be defined in terms of *link clause expressions*. Link clause expressions are boolean combinations of link clauses, where each link clause is semantically a boolean condition on two columns and is specified using either (a) a native method; (b) a user-defined function (UDF); or (c) a previously defined linkspec. Note the mutual

[1] http://www.geonames.org.

[2] http://linkedct.org



LinQuer Homepage @ University of Toronto (UofT DBGroup) - Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help
http://linquer.linkedct.org:8080/
Google
UofT: DB Group
Linkage Query Writer
LinQL Translator
Done

Linkage Specification:

```
SELECT v.*, c.*
FROM VISITS v, CONDITION c
WHERE v.id <= 1000
LINK v.DIAGNOSIS WITH c.NAME
USING synonym
```

Target DBMS: ● IBM DB2 ○ MySQL

Translate!

Output/Error:

```
SELECT v.*, c.*
FROM   ONT s, VISITS v, CONDITION c
WHERE  v.id <= 1000
       AND (src in (SELECT src FROM ONT s2 WHERE s2.tgt = c.NAME))
       AND v.DIAGNOSIS = s.tgt;
```

Figure 3: A snapshot of the LinQL query translator interface at http://linquer.linkedct.org.

recursive nature of linkspecs and link clauses. A linkspec can be defined using link clauses while a link clause can be specified using previously defined linkspecs. The LINKLIMIT allows the users to limit the number of links and only consider k results, or the *top-k* where ordering is possible.

Similar to SQL index definitions, a *link index* can be defined to speed up the linkage query execution. When a link index is defined, the output SQL query will contain a set of *preprocessing* queries that should be run prior to the other queries. These queries basically materialize the views that are needed in order to run linkage queries. This is particularly useful when the data in a table is static where preprocessing can significantly reduce the query execution time.

From LinQL to SQL Irrespective of the language used, a declarative specification must be translated into some form of a program that finds the links between the source and target data. As stated earlier, we provide native SQL implementation for the link discovery algorithms within our framework. This approach has several advantages including the ability to (a) easily implement this framework on existing relational data sources with minimum effort and without any need for externally written program code; (b) take advantage of the underlying DBMS optimizations in the query engine while evaluating the SQL implementations of the link finding algorithms; and (c) use specific efficiency and functionality enhancements to improve the efficiency of these algorithms. Our implementation is built upon the extension and combination of our work on approximate selection predicates [5] and ontology-based keyword search [6].

Web Interface and API We provide an easy-to-use web interface and API for translation of LinQL queries to standard SQL queries. The interface is available at http://linquer.linkedct.org. Our current implementation generates SQL output for IBM DB2 and MySQL, although the output queries can easily be modified to support any relational DBMS. We also provide an interface that assists users in writing LinQL queries. Figure 3 shows a snapshot of a simple LinQL query translation on the web interface.

3. DEMONSTRATION USE CASES

In this demonstration, we will go through different steps of link discovery in several interesting real world application scenarios and report the results of applying our framework in an important health care domain. The main goal is to show the applicability of our framework in a variety of linkage needs in our example scenarios and prove the need for the functionality of all the components in our framework. These scenarios are built around an online database of clinical trials published on ClinicalTrials.gov. This database is a registry of federally and privately supported clinical trials conducted in research centers all around the world. It contains detailed information about the trials, including information about the conditions associated with the trials, their eligibility criteria and locations. The main clinical trials database in our experiments contains information about 61,920 trials. This database was obtained on September 2008 from the online website in XML format. Using the functionality of DB2 as a hybrid relational-XML DBMS, we stored all the data in relational tables.

Other datasets that we use in our linkage scenarios include a database of patient visits from Electronic Medical Records (EMRs), the list of Wikipedia entries (article titles) about disease and drugs, DailyMed[3] (published by the National Library of Medicine, providing high quality information about marketed drugs), Diseasome[4] (containing information about 4,300 disorders and disease genes linked by known disorder-gene associations for exploring known phenotype and disease gene associations) and DrugBank[5] (a repository of roughly 5,000 FDA-approved drugs). We also use the National Cancer Institute (NCI)'s thesaurus as a source of semantic information about medical terms. Due to privacy issues associated with EMR records, our patient visits database is synthetic, generated using a data generator that resembles real EMR records in a hospital. The diagnosis and prescription values are randomly picked by the data generator from NCI terms. The data generator also generates an additional column with a small random string error in the diagnosis field. The string error injected in the string resembles real errors and typos occurred in string databases, e.g., replacing a character with another character close to it on a standard keyboard, or swapping two characters or word tokens.

Scenarios We show the application of the LinQuer system in the following scenarios: Linking **(1)** to trial conditions from the diagnosis in the patient visits, **(2)** from trial conditions to diseases in Wikipedia, **(3)** from trial conditions to diseases in Diseasome, **(4)** to trial interventions from the prescriptions in the patient visits, **(5)** from trial interventions to drugs in Wikipedia, **(6)** from trial interventions to drugs in DrugBank, **(7)** from trial interventions to drugs in DailyMed, and **(8)** finding trials related to each other based on similarity in the title of their publications.

Link Discovery Steps In our demonstration, we will illustrate a few ways in which LinQuer could be used in designing a link discovery solution for the scenarios above. Consider our first scenario in which the goal is to find links to clinical trials from patient visits. In prototyping a solution, a user may select a subset of patient visits (in our example, 1,000 visits) to use in evaluating different link alternatives. Assume that pateient visits are stored in table visits where its column diagnosis contains the diagnosis associated with the particular patient's visit, the table condition stores each trial's conditions in its column name,

[3] From http://dailymed.nlm.nih.gov.

[4] From http://www.ncbi.nlm.nih.gov/omim.

[5] From http://drugbank.ca.

and the table `ont` stores the synonym information from the NCI thesaurus with column `src` containing concept IDs and column `tgt` containing the terms. Two terms are synonyms if they have the same concept ID. One way of using LinQuer is described below.

- The records that match with a simple exact matching can be obtained using a simple SQL query that matches `visits.diagnosis` with `condition.name`. This query returns only 33 matches in this case, linking 2 out of 1,000 patient visit records to the trials. This is due to the string errors in `visits.diagnosis` values.
- The next step a user could try is to use an approximate string matching predicate with a low similarity threshold to observe the results. A low threshold will give high recall (but possibly low precision) on the discovered links. The user then manually inspects a subset of the links returned. The user can judge the quality of the links by a partial knowledge of the type of the links required. If there are too many unwanted links, she can increase the threshold. If she sees missing links, she can lower the threshold or try a new type of link method to see if there are links that are not being found. In this case, using the weighted Jaccard [5] native string matching method, and inspecting 100 linked records, the accuracy (percentage of the links between diagnosis and conditions that are considered correct matches) was 91% for threshold 0.7 and 40% for threshold 0.4. Interactively, the user can change the threshold to retrieve a high number of links she considers relevant (without having too many incorrect links).
- Let's assume that the user has found a threshold for string matching that gives a good accuracy on the user's test data. (S)he may notice from the inspection of the discovered links, that the string matching is not finding links based on synonyms. The next step then may be to use the semantic information in NCI to improve the matching using synonym and hyponym matching. The semantic matching that is only based on synonyms results in 147 links to 104 distinct trials, out of which 69 links to 24 distinct trials could not be found using exact or string matching. Repeating the above query with semantic matching based on hyponyms of depth 2 from NCI results in 68 additional links to 21 distinct trials.
- Continuing the analysis, the user notices that some synonyms are still not being found because they contain errors. Hence, (s)he decides to combine the approximate string matching with the semantic link discovery by writting the following LinQL code.

  ```
  CREATE LINKSPEC mixmatch
  AS LINK src WITH tgt
     USING synonym;
     AND
     LINK src WITH ont.tgt
     USING weightedJaccard

  SELECT v.*, c.*
  FROM   visits v, condition c
  WHERE  LINK v.diagnosis WITH c.name
         USING mixmatch
  ```

  Using combined string matching and semantic matching results in 173 links to 120 distinct trials, 26 more links to 16 more distinct trials when compared with matching based on synonyms only.
- Now, depending on the results of the above steps, the user can write a single query for the linkage needs specific to this application. Here we choose to combine exact matching, string matching, semantic matching based on synonyms and hyponyms, and mixed semantic matching allowing string errors. This can all be expressed using a single LinQL query. The combined approach results in 1,255 links from 383 visit records to the related clinical trials.

Summary of the results The table below shows the number of links found using exact matching, string matching, semantic matching, and a combination of string and semantic matching methods in the scenarios described above using the LinQuer framework. Our demonstration will walk through a similar scenario and allow users to experiment with different LinQL queries for tailoring the link discovery.

Scenario	Exact	String	Semantic	Combined
1	33	1,102	173	1,255
2	164	333	173	342
3	232	778	301	830
4	318	806	4225	6630
5	8,442	9,716	10,630	11,527
6	9,867	11,865	12,127	23,493
7	14,257	24,461	27,685	39,396
8	11	2,074	—	2,074

4. CONCLUSION

In this demonstration, we will present a declarative extensible framework for link discovery from relational data. We will show how a simple specification language, LinQL, can be used for specification of linkage requirements in a variety of user and domain needs, and how LinQL queries can be translated into standard SQL queries. We will demonstrate the effectiveness of our approach in several link discovery scenarios in a real world health care application. Our focus has been on developing efficient techniques that can handle large data sets, but also on usability. We illustrate how a user can interactively experiment with and customize different link methods to better understand what are the most effective methods for her domain. We show how our framework can significantly enhance the process of publishing a high-quality data source with links to other data sources on the web.

5. REFERENCES

[1] C. Bizer, T. Heath, T. Berners-Lee, Linked data: Principles and state of the art, in: W3C Track @ WWW2008.

[2] S. Das, E. I. Chong, G. Eadon, J. Srinivasan, Supporting ontology-based semantic matching in RDBMS, in: VLDB '04.

[3] A. K. Elmagarmid, P. G. Ipeirotis, V. S. Verykios, Duplicate record detection: A survey, IEEE Transactions on Knowledge and Data Engineering 19 (1) (2007) 1–16.

[4] L. Getoor, C. Diehl, Link mining: A survey, SigKDD Explorations Special Issue on Link Mining 7 (2) (2005) 3–12.

[5] O. Hassanzadeh, Benchmarking declarative approximate selection predicates, Master's thesis, Univ. of Toronto (2007).

[6] A. Kementsietsidis, L. Lim, M. Wang, Supporting ontology-based keyword search over medical databases, in: Proceedings of the AMIA 2008 Symposium.

[7] S. S. Sahoo, W. Halb, S. Hellmann, K. Idehen, T. T. Jr, S. Auer, J. Sequeda, A. Ezzat, A survey of current approaches for mapping of relational databases to RDF, Tech. rep., W3C RDB2RDF incubator group (2009).