Tooling Framework for Instantiating Natural Language Querying System

Manasa Jammi, Jaydeep Sen, Ashish Mittal, Sagar Verma[1], Vardaan Pahuja[2], Rema Ananthanarayanan, Pranay Lohia, Hima Karanam, Diptikalyan Saha, Karthik Sankaranarayanan
{manjammi,jaydesen,arakeshk,pralohia,arema,hkaranam,diptsaha,kartsank}@in.ibm.com, sagar15056@iiitd.ac.in, vardaanpahuja@gmail.com
IBM Research AI, IIIT Delhi[1], University of Montreal[2]

ABSTRACT

Recent times have seen a growing demand for natural language querying (NLQ) interfaces to retrieve information from the structured data sources such as knowledge bases. Using this interface, business users can directly interact with a database without the knowledge of the query language or the data schema. Our earlier work describes a natural language query engine called ATHENA which has several shortcoming around ease of use and compatibility with data stores, formats and flows. In this demonstration paper, we present a tooling framework to address these challenges so that one can instantiate an NLQ system with utmost ease. Our framework makes it easy and practically applicable to all NLIDB scenarios involving different sources of structured data, file formats, and ontologies to enable natural language querying on top of them with minimal human configuration. We present the tool design and the solution to the challenges towards building such a system and demonstrate its applicability in the medical domain.

PVLDB Reference Format:
Manasa Jammi, Jaydeep Sen, Ashish Mittal, Sagar Verma, Vardaan Pahuja, Rema Ananthanarayanan, Pranay Lohia, Hima Karanam, Diptikalyan Saha, Karthik Sankaranarayanan. Tooling Framework for Instantiating Natural Language Querying System. *PVLDB*, 11 (12): 2014-2017, 2018.
DOI: https://doi.org/10.14778/3229863.3236248

1. INTRODUCTION

Despite decades of research in building Natural Language Interface to Database (NLIDB) systems ([11],[9]), such systems are found to be brittle primarily since they aim to achieve 3 lofty goals: (i) they seek to obviate the need for users to learn query languages such as SQL and instead use natural language such as English which is inherently ambiguous, (ii) they look to enable users to query the data without necessarily knowing the exact schema, and (iii) they seek to automatically connect to structured data sources in any format or backend datastore and enable querying with little to no human intervention. Significant progress has been made in recent years towards the first 2 goals with work such as ATHENA [11] which introduces an ontology-driven approach that can simultaneously ground the semantic meaning of terms in the natural language query (NLQ) and provide a layer of abstraction between the logical schema and physical schema of a relational datastore. However, significant amount of work is still necessary to achieve the 3rd goal and make these techniques practically applicable to all NLIDB scenarios involving different sources of structured data, file formats, backend databases, and ontologies. In this paper, we seek to address these challenges towards the 3rd goal and propose a system that can seamlessly connect to any structured datastore and enable natural language querying with minimal human configuration.

In industry settings, structured data could be naturally available from a wide variety of sources - various file formats (such as CSV, HTML or PDF), or data that is already part of an existing structured data stores (such as a relational database, an RDF database, or a Graph database). These are cases where a user would be interested in querying in natural language any of these *existing* data sources. On the other hand, users looking to create and maintain structured data based on industry standards would employ an industry-specific domain ontology as a starting point (for example, FIBO for Finance, or UMLS for Healthcare), and then seek to populate data corresponding to these ontologies. The proposed framework seeks to enable both sets of users to seamlessly query their data: (a) users who bring their data which could be from various file formats or structured backends (i.e. the data-first approach), or (b) users who begin with their ontology of interest and then populate the data accordingly (i.e. the ontology-first approach).

To support both approaches and enable users to directly access their data in natural language across all these scenarios, there are several steps requiring heavy human intervention. Our framework seeks to automate these steps to a large extent by addressing specific problems such as inferring the domain ontology given the structured data, semi-automatic enhancement of vocabulary terms beyond those present in the data, automatically configuring the NLQ pipeline based on query workloads.There have been works which transform a relational database schema to an OWL file [6]. However, a unified framework such as the one proposed in this paper is not present. Further, this framework also extends access to multiple structured backends such as RDF store and Graph

Proceedings of the VLDB Endowment, Vol. 11, No. 12
Copyright 2018 VLDB Endowment 2150-8097/18/8.
DOI: https://doi.org/10.14778/3229863.3236248

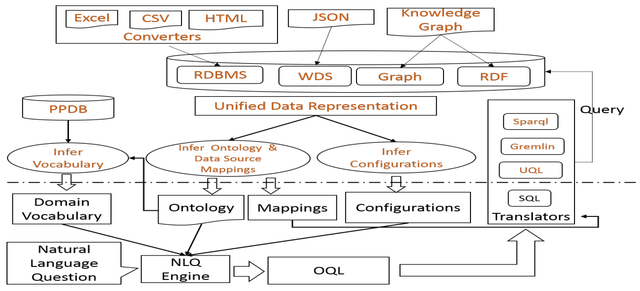


Figure 1: ATHENA and our proposed framework Architecture – Blocks above the dotted line are part of the proposed framework

store, by automatically translating from the intermediate query language (OQL) from ATHENA to respective structured query languages specific to these backend stores such as SPARQL and Gremlin. Below we summarize the contributions in this paper:

- We demonstrate a framework to instantiate an NLQ engine for any data that a user might bring.
- We present solutions for automatic inferring of ontology and the domain vocabulary from the input data.
- We showcase the various flows (data-first and ontology-first flows) that are supported by the proposed framework, and present the corresponding challenges solved in the process.

2. SYSTEM

In this section, we first discuss the background on ATHENA, a state-of-the-art NLIDB system, point out the challenges in building end-to-end data-first or ontology-first flows over multiple datastores on top of ATHENA, and subsequently discuss their solutions.

2.1 Background

We now present a brief review on ATHENA [11]. It needs four inputs, in addition to data, viz., the ontology or the domain schema, the domain vocabulary, the ontology to database mapping, and the configuration. Ontology is constituted of three main ontology elements, *Concepts(C)*, *Properties(P)*, and a set of *Relations(R)* (R$\subseteq$ C X C).

In addition to the ontology, ATHENA needs the domain vocabulary, which consists of *Synonyms*, and the *Translation Index*(TI). Each ontology element can have multiple synonyms which map the NLQ token to an ontology element. For example, the phone number of a person can have multiple synonyms such as *contact*, *mobile*, etc. The *Translation Index* maps every data value in table fields to its corresponding ontology element. For example, *Algorithms* can map to the Property *CourseName* of the Concept *Courses*.

In addition, there are different *Configurations* such as *key*, *default time*, and *default group by* properties that are needed in order to successfully translate the NLQs.

ATHENA also requires a ontology to datastore mapping. This maps concepts to tables, properties to fields, and relations to foreign key - primary key relations, which is used in the target backend query generation. ATHENA follows a two phased approach - the NLQ is first translated into an intermediate query language called Ontology Query Language (OQL) query [11], and in the second phase, this OQL query is translated to SQL using SQL translator. Such architecture enables us to support multiple backends.

2.2 Challenges

We now discuss the challenges necessary to address to achieve the aforementioned goals of the proposed framework.

- **Multiple flows.** ATHENA requires ontology as input which may not be readily available to the user. Instead, the user may want to start with their own structured data (either in raw files or in an existing data store). Therefore, the tool needs to address both the data-first flow and ontology-first flow and cater to these scenarios.
- **Multiple data sources.** The user can have the data in different raw data file formats, or in various backed data stores. The challenge here is to support all such data input formats and translating OQL queries to different target query languages.
- **Ontology derivation from different data stores.** In the data first flow, the ontology needs to be inferred from the existing data store. Specifically, the concepts, its properties and their types, different types of relations such as functional or inheritance, and ARITY of relations need to be inferred.
- **Domain vocabulary enhancement.** ATHENA tries to map the tokens to the ontology elements or data values, which requires synonyms for the same. Users will typically provide a few or no synonyms. The challenge is to therefore derive a comprehensive set of synonyms to achieve robust mapping.
- **Configuration inference.** ATHENA needs to have various configurations specified. Automatically identifying these configurations from the data is a challenging task, which requires semantic understanding of the data model. As these configurations are specific to ATHENA, we do not discuss their inference in this paper.
- **Visualization of the results.** ATHENA fetches the result from the data store as a set of raw tuples. However, the system's user experience can greatly benefit if the results are automatically visualized in the form of the most appropriate graph or a chart.
- **Incremental update in the ontology/data.** After instantiating the entire system for the first time, an user might populate additional data in the data store, or upload another data file to the proposed framework, or change the synonyms during the lifetime of the system. The framework would need to handle such cases seamlessly.

Our framework essentially solves the above problems by extending ATHENA architecture by including components (above the dotted line) as shown in Figure 1.

2.3 Solutions

Handling Multiple Flows and Different Data Sources. In the data first flow, our framework supports a raw data file upload (Excel, CSV, HTML, PDF), as well as a pointer to an existing data store (RDBMS, Graph Data Store, RDF, Document stores like Solr). When a user supplies a raw data file, either they specify a backend datastore type, or an appropriate data store is automatically inferred. A backend database is then created, and the data is populated in that data store.

In case of an Excel file, our framework creates a relational database, where each sheet is converted to a table, and each column in the sheet to the field in the table. The sheet name is taken as the table name. The framework informs the user that the first row of the sheet should contain the column names, otherwise a parse error is produced. The data type of each of these columns are identified by analyzing the data values. The presence of null values and noisy data often makes this task very challenging. We compute the distribution of the data for each column for non-null values and pick the most frequent datatype. If the data is in the form of PDFs, we use [2] to extract the tables from it. We extract tables from HTML using the HTML parser JSoup [8]. User can also have data in an existing data store like RDBMS, RDF, or graph stores.

Once the raw data is stored in a backend data store, an ontology is created from it (discussed later). To support executing queries to different data stores, we have developed OQL to backend data store translators e.g. OQL-SPARQL for an RDF store and OQL-Gremlin for a Graph store, OQL-UQL (Watson Discovery Service document store query language) [4].

The proposed tool also provides another flow where the user can upload an existing domain ontology, an optional associated vocabulary, and existing data. Our framework infers the schema from the existing data and maps it to the pre-enriched ontology using the element names or the associated vocabulary. For example, the framework will map a concept in the pre-enriched ontology called *Compensation* to an Excel sheet named *Salary Statement* having related columns. It informs user on what part of the data could not be mapped to the ontology. This flow enables a user to use any of the publicly available domain ontologies and map it to some existing data.

Ontology Derivation from Multiple Data Stores. We define a two-phase approach for extracting ontology from data stored in different stores. In the first phase, an intermediate data representation (`UDR`) is inferred for each store, subsequently various information of the ontology is inferred from the intermediate format. The advantage of this approach is that the core logic of inferencing ontology from different constraints present in the data is separated out from how these constraints are derived from the data. For example, an inheritance relationship between two concepts, can be inferred based on that key constraint of a property and subset relationship between two properties - a logic which is independent of the format of the data. To the best of our knowledge, such a two phase approach for determining ontology from various forms of data stored in different datastores is novel.

Unified Data Representation (UDR) The main constructs of the UDR are the `Classes`, the `Features` of each `class`, and the `Instances` of the class. A `Class` (C_i) is an abstraction for a table or a concept, and its `Features`, $\{F_i^1, F_i^2, \ldots\}$ are the properties or fields of the class. We also define $I(C_i)$, which is a set of instances of each class C_i, containing the actual data value. The `Features` have various constraints which are used in the ontology inference. Some of these constraints are – (1)*isMarkedasID* which is set to true if this is a marked as an identifying feature in the data (for example, the column is marked as a primary key in the database schema), (2) *isUnique* which is computed to true if there is no duplicate values except null, (3)*isAutoIncremented* which is set to true if the instances of the Feature have contiguous values (applies to only numeric typed Features), (4)*countOfUniqueValues* which returns the number of unique values in the instances of the `Feature`. The constraints are computed by analyzing the instances. Existing relationship constraints (not inferred) is captured by $rel(F_i^j, F_k^l)$.

Owing to the fact that RDBMS has a well-defined schema, the mapping from RDBMS to UDR is simple. All the tables are mapped to `Classes` and the corresponding columns to `Features` of those classes. In JSON, the conversion is done recursively where every key with value of a JSON object is mapped to a class. Every key whose value is a literal becomes the feature of the class. The containment hierarchy in JSON cretes the *rel* constraint.

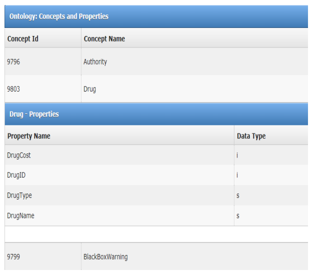
- All the classes in the UDR are mapped to concepts in the ontology. We will be using them interchangeably.
- *Relation Inference:* Ontology supports three types of relations - functional, isA, and union.
  - Identifying Functional Relations: Whenever $I(F_i^j) \subset I(F_k^l) \wedge \neg isAutoIncremented(F_i^j)$ a functional relation is asserted between the two corresponding concepts of $C_i and C_k$. Special check for spurious functional key due to auto-incremented values are also considered.
  - Inferring isA relation: If a foreign key of a table is also it's primary key i.e. $I(F_i^j) \subset I(F_k^l)$ and $(isUnique(F_i^j) \vee isMarkedAsId(F_i^j)) \wedge \neg isAutoIncremented(F_i^j)$ then C_i is inferred as an *ISA* child of the target class C_j.
  - Inferring Union: Union is considered a special case of ISA relations. When multiple child classes referring to the same parent class are mutually exclusive and exhaustive, a union is asserted.
  - Inferring arity of functional relations: For inferring m:n relations, we find classes with exactly two features, and both acting as references to different classes in the UDR. If such a case occurs, the two classes which are being referred are mapped in a m:n relation and the concept corresponding to the join class is removed from the ontology. We identify the relations where the values in the participating features are unique and mark it as 1:1, and the rest as 1:n.
- *Properties:* All features except the ones identified for functional relations map to the data properties of the corresponding concept. The datatype of the properties are also fetched from the UDR. The ontology is stored in OWL2 format.

Domain Vocabulary Enhancement. This consists of two parts – automatic synonym generation and TI variation generation. For the automatic synonym generation, we use different publicly available APIs such as Wordnet [10] and PPDB [1] to generate aliases for all the ontology elements. In case of synonym generation for phrases, we employ an existing technique described in [7]. ATHENA describes automatic synonym generation for certain types of data values like company names and person names. We automatically extend that to add abbreviated or expanded form. For example, the query might contain "Corporation" and the TI might have an instance called "Corp". In such cases, aliases for "Corp" are added which can match the required token.

Visualization of the Results. The proposed framework is also integrated with a visualization engine [5] which takes the result set data and automatically generates a chart which most appropriately represents the data. In order to do this, this technique employs automatic feature identification by looking at the OQL constructs.

Incremental Update in the Ontology/Data. Whenever edits are made to the ontology, the entire pipeline is re-run and all the participating modules are intimated of the change. We take care of data insertions by using database triggers which initiates the refreshing of the pipeline. When more data is added, the TI is also refreshed accordingly.

3. TOOL INTERACTION



Ontology: Concepts and Properties

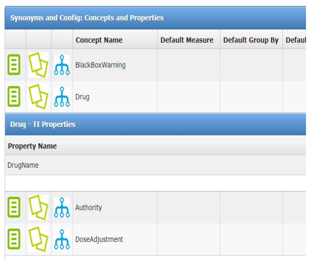
Concept Id	Concept Name
9796	Authority
9803	Drug

Drug - Properties

Property Name	Data Type
DrugCost	i
DrugID	i
DrugType	s
DrugName	s

9799	BlackBoxWarning

(A)



Synonyms and Config: Concepts and Properties

			Concept Name	Default Measure	Default Group By	Defaul
			BlackBoxWarning			
			Drug			

Drug - TI Properties

Property Name
DrugName

			Authority			
			DoseAdjustment			

(B)

Figure 2: (A)Domain Ontology, (B) Vocabulary

In this section present a demo scenario from Healthcare domain. Final demonstration will also include scenarios from Finance and Retail.

Demo Scenario 1: Excel Sheet Data Input. Consider the scenario where a pharmacist has an Excel file which has three sheets, which describes the warning for a set of drugs sent out by different authorities.

- **Drug** (*Drug Id*, *Drug Name*, *Drug Cost*)
- **Authority** (*Auth Id*, *Auth Name*)
- **BlackBox Warning** (*Auth Id*, *Drug Id*, *BBWarning*)

The pharmacist uploads his data to our tool, which is stored in the tool's File system. Next, a relational database is created for the project in the tool's data base, where the data is populated. Next, (1) the ontology is inferred from the database, (2) the domain vocabulary is enriched,(3) and the TI is populated. The ontology, synonyms, and configurations are shown in Figure 2 (A) and Figure 2 (B).

Say the pharmacist tries a query *"How expensive is the medicine Paracetamol?"* In this query, the tool identifies "expensive" as a synonym for "cost", and "medicine" for "drug" (due to automatic vocabulary enhancement). It maps the non contiguous phrase "expensive is the medicine" to *Drug Cost*. However, it does not identify Paracetamol and returns a failure response with the following comments- "I could not understand Paracetamol". The reason for this is that the Excel sheet contains no data value entry called "Paracetamol". The actual data in the sheet is "N-(4 - hydroxyphenyl) ethanamide, N-(4 - hydroxyphenyl) acetamide" which is the IUPAC name for Paracetamol. The tool was not able to automatically populate synonyms for the same because it is very specific to the medical domain, and is not part of the general English dictionary. The pharmacist can then add a synonym for the same using our tool. When the user adds a synonym, the domain vocabulary is refreshed and on the re-run of the query, Paracetamol will now match the entity *Drug Name* of the concept *Drug*. The next query that the pharmacist tries is "Show me the costliest medicine per type". Here, the proposed tool takes the stemmed form of "costliest" - cost, and "medicine" to map it to *Drug Cost*. It also identifies that costliest is a superlative and adds a MAX operator to the query. "Type" is mapped to *Drug Type*, and because there is a "per" in the query, it identifies it as a group by query to produce the SQL *"SELECT MAX(d.DRUGCOST) FROM DRUG d GROUP BY d.DRUGTYPE"*. The result is then fed to the visualization engine which produces the chart as shown in Figure 3.

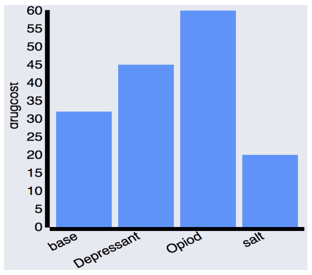


Figure 3: Result Visualization

Demo Scenario 2: Ontology First flow. A user can also upload an ontology file first and can make edits to the ontology using the tool's UI.

The tool creates an empty database from the ontology by mapping concepts to tables, properties to fields, relations to PK-FK.

The rest of the flow is similar to the previous flow. The demo of the tool is available at [3].

4. CONCLUSION

In this paper, we presented a framework which enables easy instantiation of an existing Ontology based NLIDB framework called ATHENA. In addition to ontology flow present in ATHENA, it supports data-first flow where user can input data in various formats. Our framework also handles multiple data stores and automatically creates essential inputs like synonyms and configurations. We demonstrated its practical applicability in the medical domain.

5. REFERENCES

[1] Ppdb: The paraphrase database. `http://www.cis.upenn.edu/~ccb/ppdb/`.

[2] Tabula: Extract tables from pdfs. `http://tabula.technology/`.

[3] Tool demo. `https://youtu.be/9NqdpYdfhhw`.

[4] Watson discovery service. `https://www.ibm.com/watson/services/discovery/`.

[5] R. Ananthanarayanan, P. K. Lohia, and S. Bedathur. Datavizard: Recommending visual presentations for structured data. *arXiv:1711.04971*, 2017.

[6] P. Chujai, N. Kerdprasop, and K. Kerdprasop. On transforming the er model to ontology using protégé owl tool. *International Journal of Computer Theory and Engineering*, 6(6):484, 2014.

[7] N. G. et al. Addressing Practical Challenges for Natural Language Querying in SAP-ERP Platform.

[8] J. Hedley. Jsoup html parser, 2009.

[9] F. Li and H. V. Jagadish. Nalir: An interactive natural language interface for querying relational databases. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pages 709–712. ACM, 2014.

[10] G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[11] D. Saha, A. Floratou, K. Sankaranarayanan, U. Minhas, A. R. Mittal, and F. Özcan. Athena: An ontology-driven system for natural language querying over relational data stores. *PVLDB*, 9(12):1209–1220, 2016.