Natural Language Querying of Complex Business Intelligence Queries

Jaydeep Sen, Fatma Özcan, Abdul Quamar, Greg Stager*, Ashish Mittal, Manasa Jammi, Chuan Lei, Diptikalyan Saha, Karthik Sankaranarayanan
IBM Research AI, *IBM Canada

ABSTRACT

Natural Language Interface to Database (NLIDB) eliminates the need for an end user to use complex query languages like SQL by translating the input natural language statements to SQL automatically. Although NLIDB systems have seen rapid growth of interest recently, the current state-of-the-art systems can at best handle point queries to retrieve certain column values satisfying some filters, or aggregation queries involving basic SQL aggregation functions. In this demo, we showcase our NLIDB system with extended capabilities for business applications that require complex nested SQL queries without prior training or feedback from human in-the-loop. In particular, our system uses novel algorithms that combine linguistic analysis with deep domain reasoning for solving core challenges in handling nested queries. To demonstrate the capabilities, we propose a new benchmark dataset containing realistic business intelligence queries, conforming to an ontology derived from FIBO and FRO financial ontologies. In this demo, we will showcase a wide range of complex business intelligence queries against our benchmark dataset, with increasing level of complexity. The users will be able to examine the SQL queries generated, and also will be provided with an English description of the interpretation.

1 INTRODUCTION

With the omnipresence of mobile devices coupled with recent advances in automatic speech recognition capabilities, there has been a growing demand for Natural Language Interfaces to Databases (NLIDB). The reason behind the rapid increase of popularity of NLIDB systems is due to the fact that they do not require the users to learn a complex query language, such as SQL, or understand the exact schema of the data, or how it is stored, making it very easy to explore complex data sets, beyond simple keyword search queries.

Natural language interfaces to databases have become an increasingly important research field during the last decade [6, 8, 9]. Most prominent among state-of-the-art includes our previous work ATHENA [9] and its predecessor NALIR [6], both of which essentially are rule-based systems. While NALIR supports few specific types of basic nested queries using natural language parsing and human in-the-loop feedback, it could not handle complex nested queries, and the need of an expert feedback became a bottleneck. On the other hand, ATHENA improves over NALIR by avoiding user dependence with ontologies capturing domain semantics, but it falls short of handling nested queries as well.

A different category of works [10, 12] employs machine learning approaches to tackle the problem. However, the main drawback with this approach is the limited amount of domain-specific data to train with, making systems like Seq2SQL [12] to handle only single table select and project queries. More recently, DBPal [11] claims to handle join queries and a selective set of nested queries where the corresponding query phrasings have to be available as part of the training data.

In both segments of research, nested query handling remains an open challenge, because none of the systems are equipped with the deep domain reasoning capability, which is essential for interpreting BI queries and producing correct nested queries. In particular, the challenges in handling nested queries are as follows.

• **Detecting a NL query which requires nesting.** Whether a natural language (NL) query requires nesting depends on multiple convoluted factors, including operations involved, operands in use, domain semantics, and SQL query semantics.

• **Forming the correct subqueries.** To form the right subqueries (inner and outer), we need to accurately identifying which tokens from the NL query are to be considered for each subquery, which is often non-trivial in natural language phrasing.

• **Forming the correct join condition between subqueries.** Finding the correct join conditions between subqueries depends not only on SQL query semantics, but often requires deep domain reasoning.

SIGMOD'19, June 2019, Amsterdam, Netherlands

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

In this demonstration, we will showcase our NLIDB system, which is an extension of our earlier work on ATHENA. The system is capable of handling a wide range of BI queries including complex join and nested queries. To address the above challenges, our system uses novel algorithms that combine linguistic analysis with deep domain reasoning without prior training or feedback from human in-the-loop. In our previous work [5], we demonstrated how to use ATHENA in querying financial knowledge bases curated from unstructured content. In this demonstration, we enrich the *Finance* domain ontology beyond [5], with transaction data, resulting in a new benchmark *FIBEN* with more concepts and relations. This new benchmark emulates data marts, and hence allows more complex BI queries. In particular, we integrate two widely known financial domains (SEC [3] and TPoX [7]) to create *FIBEN*, and curate meaningful data conforming to it. We demonstrate the capabilities of our system with real life BI queries and complex analytic queries on *FIBEN* to derive crucial business insights from underlying data, with increasing level of complexity. The users will be able to examine the SQL queries generated, and also will be provided with an English description of the interpretation[1].

2 A FINANCE BENCHMARK DATASET

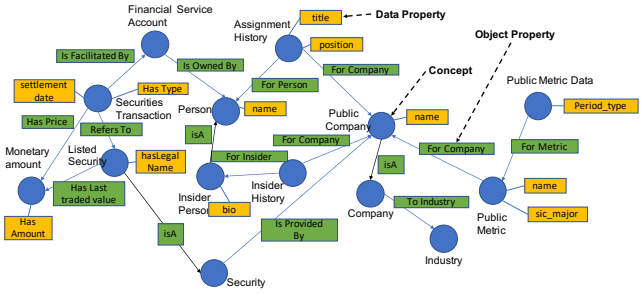


Figure 1: Snapshot of FIBEN Ontology

FIBEN is a complex business intelligence benchmark dataset that emulates a real-world data mart. To create *FIBEN*, we combined data from two different financial sub-domains.

• **SEC dataset** [3] is a publicly available dataset extracted from public Securities and Exchange Commission (SEC) filings submitted as XBRL documents. The filing data is curated using an enrichment flow [5], and provides information about public companies, their officers, and financial metrics reported over a period of time. This dataset is a source of archived financial data reported to the SEC over the last 15 years, and lends itself to provide rich business insights regarding the financial health and performance of public companies in a variety of different industry sectors.

• **TPoX dataset** [7] is a transaction processing benchmark, which describes financial transactions over holdings and securities provided by public companies. We used the TPoX data generator to generate dynamic data in terms of financial transactions. Each financial transaction is linked to a customer account. The customer account describes the customer's portfolio in terms of the securities held, and is associated with the buying or selling of securities such as stocks, bonds and mutual funds of publicly traded companies in different financial markets over a period of time. The TPoX schema is shown in Figure 2.

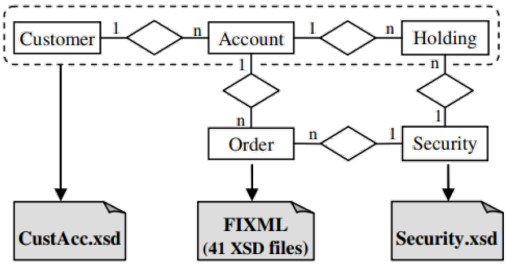


Figure 2: TPoX Schema

We mapped the two datasets through an extensive set of data transformations to the relevant subsets of two standard finance ontologies: Finance Industry Business Ontology (FIBO) [1] and Finance Report Ontology (FRO) [2]. FIBO is the de-facto industry standard defined by the enterprise data management (EDM) council to represent business concepts and information in the finance domain. FRO is a formal report ontology of an XBRL based financial report which captures the financial metric data reported by public companies to SEC. Figure 1 shows the combined FIBO-FRO ontology describing the schema of our *FIBEN* dataset.

To provide a standard benchmark for testing BI applications, we integrate the two datasets through public companies. Securities held and traded by customers in the TPoX dataset are provided by public companies that are available in the SEC dataset. The combined *FIBEN* dataset enables a rich set of analytical queries including complex joins and nested queries across both datasets.

3 SYSTEM OVERVIEW

The system architecture is depicted in Figure 3 and is extended from the architecture presented in ATHENA [9]. The components specific to nested query handling include Nested Query Detector, Subquery Formation, and Subquery Join Condition. While the complete details of each of the components is beyond the scope of a demonstration paper, we will use an example query flow to explain these extended components. Consider the query "who bought more IBM stocks than they sold".

Evidence Annotator. This follows the same process as detailed in ATHENA [9], where the Evidence Annotator scans through all the words (a.k.a. tokens) and identifies "stock" to be a mention of the ontology concept "ListedSecurity", "IBM" as a value in the database column corresponding to

[1] See https://youtu.be/GR55C-NWQwY for a demo video.

"ListedSecurity.name". Similarly, "bought" and "sold" are values of *"Transaction.type"*, and "who" is an indication to include *"Persons"* in the select clause of the target SQL query. The tokens that are mapped to some ontology elements are called entities.

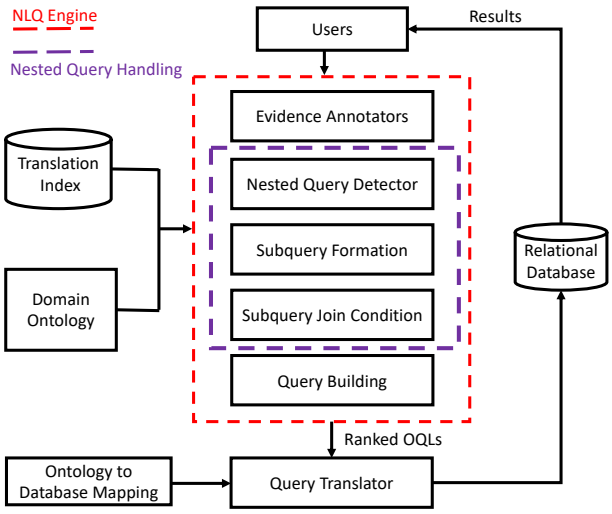


Figure 3: System Architecture

Nested Query Detector. Multiple aspects of a natural language query along with the domain semantics can indicate if it requires nesting. The different aspects include linguistic analysis of the NL query and semantic annotators which maps specific tokens from NL query to domain terms. There is a reasoning submodule that works on the outputs of linguistic analyzer and semantic annotators to detect a possible nested query. In this example, linguistic analysis identifies a comparison of type "more than" with argument "stocks" and Semantic Annotators maps "stock" to a concept. The reasoning submodule identifies that the operand "stocks" of "more than" operation must be expanded into a subquery in order to be compared using a numeric operator like "more than".

Subquery Formation. Once a query is detected to be a nested query, subquery formation module builds two subqueries for the inner and outer queries, by identifying the right set of tokens associated with each part. Note that the subqueries do not need to have a disjoint set of tokens and often the subqueries share tokens without any hard boundary on their positions in the NL query. The right set of tokens for each subquery is found by using a set of rules applicable to the query, where each rule considers the annotator outputs and domain elements. In the example query, the right set of tokens for the outer query are {Who, Bought, IBM, Stock}, and for inner query are {Who, Sold, IBM, Stock}, where "Who", "IBM", "Stock" tokens are shared between both subqueries. The set of rules applied to determine the shared tokens for this specific question are as follows.

Rule 1. Any argument("Stock") of a numeric comparison ("more than") will be shared across subqueries.

Rule 2. Any instance value ("IBM") from the database having a functional relation with an argument ("Stock") of a comparison argument will be shared.

Rule 3. An entity in the outer Query ("who") that is co-referenced in the inner query ("they") will be shared with the inner query as well.

Subquery Join Condition. Once the logical subqueries are formed, the join condition needs to be identified between the subqueries to produce the complete query. The join condition depends on the linguistic analysis as well as the domain reasoning. In the example question, linguistic analysis maps "more than" to ">" operator, and identifies that it compares with subquery results. Domain analysis finds the argument "stock" is not non-numeric, and hence it cannot be the operand of the comparison. It is then left to domain reasoning to identify that every stock has an associated "count" for each transaction and "count" is a numeric entity. Join condition is thus derived as ">" applied over "SUM(Transaction.hasCount')".

4 DEMONSTRATIONS

As mentioned earlier, our work focus on handling a wide range of complex nested queries without the need for human-in-the-loop feedback or prior training. We will demonstrate this through three different tasks.

Familiarization with FIBEN Ontology. The complete *FIBEN* ontology consists of 152 concepts, 664 data properties and 159 object properties (i.e., relations). Such large-scale ontology emphasizes why an end user needs to get familiarized with the domain semantics in order to appreciate the insightful analytic queries in this domain. We plan to showcase the complete ontology and the underlying database schema to the audiences so that they understand the various actors in the finance domain and the different roles that those actors are playing. They can use tools like WebVOWL [4] to navigate the ontology themselves.

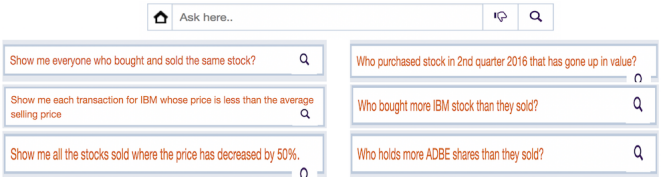


Figure 4: UI screen with pre-populated queries

Nested Query Answering. For the main part of our demonstration, we will focus on the wide range of complex nested queries that our system can handle. Instead of a story script, we would show a sequence of queries with increasing order

of their complexity, and pre-populate the UI to have such sample complex business intelligence queries. An example snapshot of such a UI screen is shown in Figure 4. As can be seen, the queries cover a wide range of complexity, ranging from queries that do not need nesting and can be expressed using having clauses, to queries with different join conditions combining inner and outer query blocks in the nested query. For example, the query "Show me everyone who bought and sold the same stock" needs equality check (for "same") among non-numeric fields. Whereas a query like "Who has bought more IBM stocks than they sold" is even more complex because it involves numeric comparison between two aggregation (i.e., sum of stock counts) results.

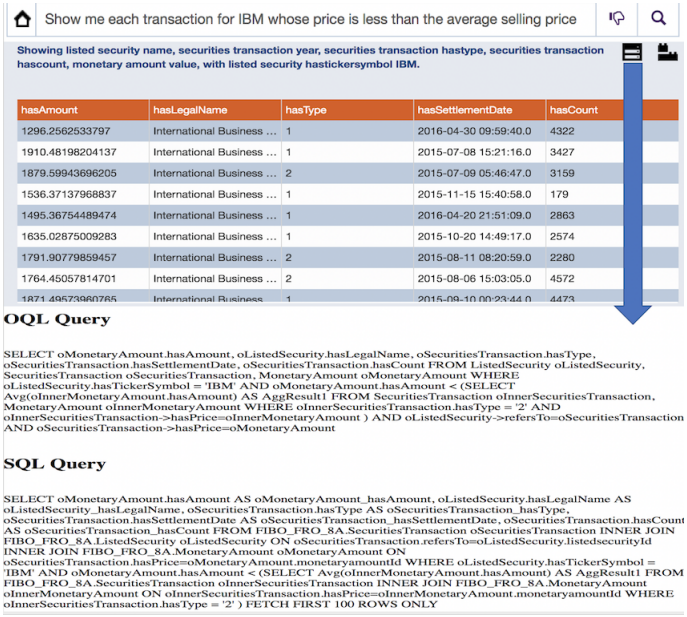
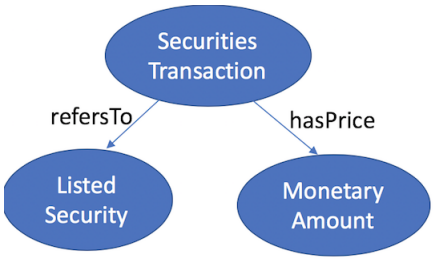


Figure 5: An example of a complex query and answer

Interactive UI. The same UI with pre-populated interesting business intelligence queries (Figure 4) will be used to let the audiences interact with the system. The audiences can choose to run such pre-populated queries, or frame their own queries over the *FIBEN* ontology for an unscripted demonstration. Figure 5 shows the visualization with an example question and its answer. For every query, we will also show the English description of how the system interpreted the user statement, and the generated SQL query which is executed against the data store to produce the query result. Our demo will run on DB2®[2]. Figure 6 shows the part of the ontology graph which was relevant for answering the query "Show me each transaction for IBM whose price is less than the average selling price". It also shows the specific subgraphs which were created for both *outer query* and *inner query*. We will include these details for the audiences to easily correlate the query interpretation with the underlying domain semantics.



Outer Query:
Nodes: [ListedSecurity, SecuritiesTransaction, MonetaryAmount]
Edges: [SecuritiesTransaction<->MonetaryAmount}(hasPrice), SecuritiesTransaction<->ListedSecurity}(refersTo)]

Inner Query:
Nodes: [SecuritiesTransaction, MonetaryAmount]
Relations: [SecuritiesTransaction<->MonetaryAmount}(hasPrice)]

Figure 6: Subgraph and subqueries for an example query

[2]DB2 is a registered trademark of IBM Corporation

REFERENCES

[1] 2018. FIBO. https://spec.edmcouncil.org/fibo/. (Oct. 2018).
[2] 2018. FRO. http://xbrl.squarespace.com/financial-report-ontology/. (Oct. 2018).
[3] 2019. SEC Financial Statement Data Sets. https://www.sec.gov/dera/data/financial-statement-data-sets.html. (Jan. 2019).
[4] 2019. WebVOWL: Web-based Visualization of Ontologies. http://vowl.visualdataweb.org/webvowl.html. (Jan. 2019).
[5] Shreyas Bharadwaj, Laura Chiticariu, Marina Danilevsky, et al. 2017. Creation and Interaction with Large-scale Domain-Specific Knowledge Bases. *PVLDB* 10 (2017), 1965–1968.
[6] Fei Li and H. V. Jagadish. 2014. Constructing an Interactive Natural Language Interface for Relational Databases. *Proc. VLDB Endow.* 8, 1 (2014), 73–84.
[7] Matthias Nicola, Irina Kogan, and Berni Schiefer. 2007. An XML transaction processing benchmark. In *SIGMOD Conference*.
[8] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a Theory of Natural Language Interfaces to Databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces (IUI '03)*. ACM, New York, NY, USA, 149–157. https://doi.org/10.1145/604045.604070
[9] Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, et al. 2016. ATHENA: an ontology-driven system for natural language querying over relational data stores. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1209–1220.
[10] Alane Suhr, Srinivasan Iyer, and Yoav Artzi. 2018. Learning to Map Context-Dependent Sentences to Executable Formal Queries. In *NAACL-HLT*.
[11] Prasetya Utama, Nathaniel Weir, Fuat Basik, et al. 2018. An End-to-end Neural Natural Language Interface for Databases. *arXiv preprint arXiv:1804.00401* (2018).
[12] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *arXiv preprint arXiv:1709.00103* (2017).