Question Answering on OLAP-like Data Sources

Nadine Steinmetz
Technische Universität Ilmenau
Germany
nadine.steinmetz@tu-ilmenau.de

Samar Shahabi-Ghahfarokhi
Technische Universität Ilmenau
Germany
samar.shahabi-ghahfarokhi@tu-ilmenau.de

Kai-Uwe Sattler
Technische Universität Ilmenau
Germany
kus@tu-ilmenau.de

ABSTRACT

Today, knowledge is mostly stored in structured data sources and requires to be queried by formal query languages. Natural language interfaces to these data sources enable users to query the data without knowledge about the technical details of the query language which is particularly useful in analytics and decision support. In this paper, we present an approach to query data that is stored in an OLAP database only using natural language. We describe the preliminaries and transformation process from a question to the formal SQL query on a snowflake schema. The approach takes into account synonyms to identify the relevant fact table and uses mapping techniques to appoint the correct attributes and dimension tables within the schema. As an example, we utilized the Foodmart data source to implement the first prototype and prepared over 40 different questions to assess our approach and provide a benchmark for further studies.

1 INTRODUCTION

The developments in the fields of speech recognition and natural language processing (NLP) over the last few years have fostered many novel applications. Apart from the now nearly ubiquitous virtual assistants like Amazon Alexa or Apple Siri in smartphones, smart speakers, and other consumer devices, research in the field of question answering (QA) has made a significant progress since IBM's DeepQA system Watson won the quiz show Jeopardy! in 2011. QA systems – or in the special form of Natural Language Interfaces to Databases (NLIDB) [2] – provide a user-friendly way to access any type of knowledge base simply by entering a question formulated in natural language (NL). To answer such a question, the type of the question, objects and predicates as well as entities have to be identified by parsing and analyzing the sentence. Then, the question has to be mapped to a query, e.g. in SQL or SPARQL, on a database or knowledge base providing the necessary facts. Particularly, RDF-based knowledge bases such as DBpedia [6, 9, 10] or databases enriched by ontologies [8] have been used successfully.

OLAP databases are used to answer multidimensional analytical queries in the business intelligence context. Typically, OLAP is based on a multidimensional data model distinguishing between facts or measures and dimensions forming a so-called hypercube. In such a hypercube, dimensions span the data space and provide information about the facts represented by the cells of the cube. OLAP cubes are usually queried either via visual interfaces (e.g. Tableau) which generate and execute database queries, or simply by standard query languages, such as SQL or MDX. While the latter provides the most powerful access, particularly SQL with advanced grouping features and OLAP functions is too difficult for non-database experts. On the other hand, visual interfaces are easily to use (e.g. for drill down/rollup operations) even for business users but limited in their expressiveness. Therefore, combining question answering with OLAP seems to be a natural solution to provide access to analytical databases. QA for (relational) OLAP means

- working with star or snowflake schemas consisting of fact and dimension tables,
- answering questions referring to facts and dimensions/dimension values.

The goal of our work is an approach to answer basic questions for facts such as "What were the total sales of beverages in March 2018?" but also more complex questions requiring calculations, sorting and ranking or other more advanced operations like "What were the total sales per product class in May 2018 compared to May 2017?".

The focus of our work is the processing of textual questions – a speech recognition interface is out of the scope of our work but can be easily added by using external cloud services such as Alexa Voice Service or Google Speech-to-Text. The main contributions of our work are twofold. We present an approach for mapping NL questions to SQL queries on a snowflake schema. And we describe a schema-agnostic technique to derive such mappings. In addition, we provide a set of NL questions as benchmark for the evaluation of QA systems for OLAP.

The remainder of this paper is organized as follows. In Sect. 2 we discuss related approaches from NLIDB and QA. The processing of questions and the mapping approach to SQL queries is described in Sect. 3. We have implemented this approach in a prototype which is evaluated using a set of questions on the Foodmart cube. The results of this evaluation presented in Sect. 4 demonstrate that our QA approach is capable of answering even complex questions. Finally, we conclude the results in Sect. 5 and point out to future work.

2 RELATED WORK

Li et al. introduced an interface (NaLIR) that transforms NL to SQL [5]. The interface is interactive and requests feedback from the user. Hereby, the user interacts with the interface during the query construction process at two different stages. First, the NL phrase is parsed into a lexical tree. Afterwards, the parse tree is analyzed and specific nodes are identified which can be mapped parts of the underlying SQL schema information. At this stage, if the mapping fails, the first feedback is requested by the user. Otherwise, the successful mappings are presented to the user. After all nodes are interpreted correctly (with the help of the user), the parse tree is adapted according to the their system. If necessary, implicit nodes can be inserted and the user is able to give feedback on that process. The query tree – verified by the user – is transformed to a SQL query containing joins, aggregate functions etc. The system is designed to answer rather simple questions. The authors claim, the question *Return the author who*

has the most publications in database area is hard to answer. Also, the system relies on a high amount of feedback by the user.

The authors of the approach described above reference the field of NLIDB for their scientific solutions as the interfaces are designed to query (relational) databases in general. In addition, the research field of *question answering* has emerged, especially since new types of knowledge bases have been established, such as graph databases or triple stores. Our approach is designed to transform a NL question to SQL to be able to query a schema stored in a database. However, we also relate our approach to the research field of question answering, as it is based on OLAP. A star or snowflake schema constitutes a specific and powerful way of storing data respectively knowledge.

Naeem et al. presented an approach to generate OLAP queries based on NL [7]. The question is analyzed for Part-of-Speech, linguistic dependencies and semantic roles. They utilize SBVR which is a vocabulary specific to business use cases. Different parts of the NL question are mapped to different pre-defined semantic roles. The SQL query is then generated according to these assigned roles. The approach seems to be able to process rather simple questions, but more complicated queries containing comparisons, grouping and other functions are not considered.

Sen et al. just recently presented an approach to build complex SQL queries using NL [8]. The authors focus on queries that include subqueries and comparisons between different subqueries. Their approach is based on very few rules, but tailored to a specific benchmark containing finance data. The heart of their system is an ontology containing over 150 concepts and several hundred properties. Input questions are mapped to the ontology and then the SQL query is constructed based on the ontological elements. As already stated above, our approach is designed to be as agnostic as possible regarding the underlying data source. Therefore, a change of the data source is enabled without changing (parts of) the system.

There are several further approaches to NLIDBs published in recent years. In addition to the analytical approaches as described above, there are systems that transform natural language to SQL using neural networks. But as we also chose an analytical approach, these systems are out of scope in terms of related work.

Just recently, a survey has been published comparing NLIs for databases [1]. The authors compare 24 interfaces that transform NL to a formal query language (namely SQL or SPARQL). The comparison is based on a set of 10 NL questions that have been constructed by means of different challenges that developers are facing when transforming the input into the respective query. Unfortunately, none of the presented systems are based on OLAP-like data sources, but some are transforming NL to SQL queries - as we describe those systems above. For the evaluation of our approach, we adopt the categorization of questions by means of query challenges, as described in detail in Section 4.

In terms of Business Intelligence (BI) scenarios, Tableau is an established software tool[1]. It focusses on querying business data and visualizing the data in a very intuitive way. The user can easily manipulate the visualization and create different perspectives. The user can query the data using NL, but an input is immediately mapped to the underlying data source respectively pre-defined terms and the user is provided with concrete items of the data source – functions, relations, attributes or concrete data terms. After choosing one of the suggestions, the user can complete or refine the input query. In contrast to that, we focus on complete NL questions that could also originate from a speech interface. In our approach, the processing and mapping is performed on a complete sentence/question.

In recent years, question answering is mostly focussed on answering questions regarding an underlying semantic knowledge base. NL questions are transferred to a SPARQL query based on the knowledge structured as RDF triples. The challenge Question Answering over Linked Data (QALD) has been established in 2011 as part of the Extended Semantic Web Conference (ESWC). The latest challenge took place as part of the 18th International Semantic Web Conference (ISWC) [3]. In addition to many concurring systems that participated in recent years, the organizers of the challenge published all datasets containing at least 100 questions and the corresponding SPARQL queries. As we are focussing on data sources stored in star/snowflake schema, the comparison to the systems and datasets is not applicable for our approach presented in this paper.

Table 1: Comparison of related approaches to our approach

	NaLIR [5]	Naeem et al. [7]	Sen et al. [8]	Tableau	Our Approach
SQL	✓	✓	✓	✓	✓
OLAP		✓		✓	✓
Schema agnostic				✓	✓
NL question	✓	✓	✓		✓
Complex queries			✓	✓	✓

In terms of a better understanding the contribution of our approach, we compare related systems using the following characteristics:

- data source is OLAP-like and the approach constructs SQL queries based on facts and dimensions
- the approach is agnostic regarding the schema of the data source
- NL questions are supported
- complex queries containing grouping, subqueries are supported

Table 1 shows the comparison of our approach to related approaches regarding the defined characteristics. Obviously, our approach is bridging the gap between OLAP-like data sources and NL questions in a schema agnostic way – and is also able to answer complex questions.

3 METHOD

The proposed approach considers any given knowledge base structured as a snowflake schema having one or more fact tables and several dimension tables. Only one table of a dimension is directly connected to the fact table. Thereby, the length of the longest path between a fact table and the deepest dimension table can be long and at least $p_{max} > 1$.

For our approach, we utilized the Foodmart data source[2] – as described further in Section 4. In the schema of the Foodmart data source the longest path from the fact table to a dimension

[1] https://www.tableau.com/

[2] https://github.com/rsim/mondrian_demo/blob/master/db/foodmart.sql

table is $p_{max} = 2$. Therefore, and also for simplification reasons in describing our approach, we assume a snowflake schema with $p_{max} = 2$. Hence, we utilize the term *first level dimension table* for the tables directly connected to the fact table ($p = 1$) and *second level dimension table* for dimension tables that are connected via one dimension table in between ($p = 2$).

However, our approach is also able to handle snowflake schemas having $p_{max} > 2$.

There are certain preliminaries for the analysis of the underlying data. We extract information from the schema and use it as pre-knowledge for the queries. In addition, we consider further knowledge about the specific data and general language transformation issues. The NL questions are processed based on this preparative information and as a result a formal SQL query is created to retrieve the requested data from the snowflake schema and present it to the user. More details about our approach are described in the following sections.

For our approach, some information about the underlying data source is required to be able to transform the NL question to a formal query. Most of the required information might be provided by the data owner in database constraints, such as foreign keys, fact and dimension table names. Our approach is able to handle unknown data sources and extract the required information from the given tables. In this way, we are able to handle data sources given as a set of csv files or an undocumented sql database provided by an URL. In the following sections we describe the required process steps.

3.1 Preliminaries

The first and foremost prerequisite for our approach is having the data in a snowflake schema. In addition, we require the information about which tables in a given database are the fact tables. Obviously, all other tables are assumed to be dimension tables. If no schema containing foreign key constraints is given, we identify the constraints using an automatic detection algorithm, such as described in [4]. For the mapping of NL phrases to the database, the tables and attributes require to have descriptive names – in the best case. For synthetically created data sources this might not be true. We identified several solutions for that challenge:

- automatic detection of type of data based on the containing data – this is effective for common data, such as person names, locations or dates
- analysis of sample queries and respective NL questions – this requires user input
- manual annotation of table names and attributes – this also requires user input

For the description of our QA approach, we assume to have descriptive names for tables and attributes – as it is the case for popular benchmarks, such as TPC-H or TPC-DS, but also for the Foodmart data source. The next section describes the knowledge we extract from an unknown database to satisfy our prerequisites.

3.2 Knowledge about Data

We extract the following information required to map NL to a snowflake schema:

- information about the schema – names of fact and dimension tables, join attributes and fact attributes, plus synonyms
- mapping list to identify NL phrases for SQL operators
- thesaurus for general synonyms

Besides the actual data source containing the facts and dimensions, we only utilize an additional data source having stored the synonyms, extracted schema information and SQL vocabulary. The concrete extraction of information and further knowledge required for processing NL questions is described in detail in the following sections.

3.2.1 Schema information. Provided that the name(s) of the fact table(s) are known, we extract and store further required information from the information schema of the database. This includes:

- names of dimension tables
- join attributes
- names of fact attributes
- synonyms for table and attribute names

Information about foreign keys is either given by the database constraints or automatically detected by an algorithm, such as [4]. In this way, the join attributes for fact and dimension tables are detected. All remaining attributes from the fact table(s) are assumed to be fact attributes containing the actual measures and numbers.

In addition, we analyze and clean the attribute and table names and store them ready to be mapped by terms of the NL question. However, the NL question might not mention the exact (annotated) name of the affected attribute or fact table. Therefore, we utilize the datamuse API[3] to add synonyms for attributes and tables.

3.2.2 SQL operators and functions. The required application of SQL operators and functions in the formal query mostly can be identified by certain keywords in the NL question of any domain. These keywords are assumed to be static throughout questions of different domains. Therefore, we utilize a list of potential keywords to be able to map them to the correct SQL operator resp. function. For instance, the keyword *total* indicates the use of the SQL function SUM for the requested attribute. We provide the complete mapping list as download[4].

3.2.3 Thesaurus. Time information can be stored in various ways. In return, time information can be referenced in various ways in NL. For instance, the question *How much bread was sold in the first quarter of 2010?* is targeted on all sales in the first three months – namely January, February, and March. In this case, we either have to filter for dates having 1, 2 or 3 in the month field of the date or the date is stored having the month information separately and we have to filter on this attribute. In addition, the time information contains a separate field containing the respective quarter. In either way, we have to analyze the underlying data source and how date information is stored. Furthermore, it is required to add synonymous information of how dates might be referenced in NL. Therefore, we maintain a thesaurus as a mapping table for date information. Dependent on the underlying data source, terms like *first quarter* are mapped to the data source, as e.g. to *Q1*.

3.3 Processing the Natural Language

The NL question is preprocessed and mapped to our thesaurus and the underlying data source. After preprocessing and extraction of specific terms, the following checks are executed:

- which SQL operator – "total" -> SUM, "average"->AVG

[3] http://www.datamuse.com/api/

[4] https://dbgit.prakinf.tu-ilmenau.de/code/qa-data/blob/master/sql_operators.tsv

Table 2: POS Tags that are taken into account for the extraction of terms from the NL question

Tag(s)	Type	Example
CD	cardinal numeral	1998
FW	foreign word	les
JJ*	adjective	total
NN*	noun	sales, costs
PDT	predeterminer	*both* sales and costs
POS	possessive ending	January*'s*
RB*	adverb	good, better, best
VB*	verb	sold
WRB	wh-adverb	when, how

- which fact table – e.g. "sales"-> sales_fact
- which dimension table – "which store [...]?", "In which city [...]?"
- which attribute from fact table – "How many units where ordered [...?]" -> units_ordered from inventory_fact
- is there a term from a dimension table? – "How much bread [...]?" -> "Bread" is found as data term in the field for product categories within the product_class dimension

After these checks, the NL question is presented as an intermediate formal description. This formal description is then transformed to the actual SQL query[5].

3.3.1 Preprocessing. The NL question is tagged for Part-of-Speech (POS) using the NLTK library[6] and the Brown tagset[7]. Subsequently, the respective terms are extracted from the NL question according to a predefined white list of POS tags. Table 2 shows the white list of POS tags and respective sample words for each tag.

If there are continuous words having a noun tag (`NN, NNS, NNP, NNPS`) these words are extracted as coherent terms. Words marked with other tags are extracted as single words. Words with tags not contained in the white list are dismissed. In addition, further word combinations are identified, such as *third month*, *last month*, *how many*.

For instance, for the question

`How many paper wipes were sold in the first month of 1997?`

the following terms are extracted: `[how many],[paper wipes], [were], [sold], [first month], [1997]`.

Subsequently, the list of extracted terms is checked for terms contained in our thesaurus. For our example, the term *first month* could be translated to *January* or *1* depending on how dates are stored in our data source.

After this preprocessing step, the question is represented as a list of n tuples – the question description *desc*. Each tuple contains the extracted term (respectively its translation) and the POS tag(s).

$$desc = (t_1, t_2, ..., t_n) \tag{1}$$

with n = number of extracted terms,

$$t_i = (k_i, (POS_1, ..., POS_m)) \tag{2}$$

with k being an extracted term which can consist of one or more words – which results in one or more POS tags for each extracted term.

In the next steps, these terms are checked for query-related operators and the schema of the data source. The list of tuples is checked item per item and if a check is positive, the respective tuple is replaced with a new tuple depending on the different checks as described in detail in the following sections.

3.3.2 Operators. Based on what the question asks for, several SQL operators might be relevant to be used in the query. We utilize a mapping list of manually identified NL phrases that indicate the use of a SQL operator. For instance, the NL phrase *maximum* suggests to use the `MAX` operator in the `SELECT` clause or *different* indicates the use of `DISTINCT`. If a term from the NL question is matching a term from our mapping list, the term is replaced with the respective SQL operator in the intermediate representation *desc* of the question. The tuple t_i (containing term and POS tag(s)) in *desc* is replaced by the operator/function s_i as a string:

$$s_i \in S \tag{3}$$

with S being the set of all SQL operators and functions.

3.3.3 Fact Tables. As a preliminary we know the names of the fact tables in the snowflake schema. To identify the correct fact table (in case there are more than one fact table in the schema) we check the remaining terms in the tuples if they are matching a fact table's name. As it is not very likely having the exact fact tables' name in the question, we utilize the datamuse API (as described in Section 3.2.1) to map a term and identify the correct fact table. For our example, the datamuse API provides the term *sold* as one of the synonyms for *sales*.

The information about the correct fact table is stored separately in addition to the intermediate representation *desc* of the question.

3.3.4 Dimension Tables. In some cases a question might refer directly to a dimension of the schema. For instance, a data source with localized information one dimension could be the region. In that case, a question *Which region locates the most stores?* directly refers to the dimension *region*. The tuple t_i of the matching term is replaced with a new tuple d:

$$d_i = (\text{''}, \text{''}, dim_j, (key_{j_1}, ..., key_{j_n})) \tag{4}$$

with '' being empty strings, dim_j the name of the dimension table, $(key_{j_1}, ..., key_{j_n})$ the list of join attributes between the dimension table and the fact table and potential second level dimension tables. The first empty string in the tuple denotes an unspecified data constraint, in terms of not having a concrete data term found in the data source, such as Bread or January. The second empty string stands for an unassigned attribute for the corresponding table – there is no specific attribute affected.

3.3.5 Attributes. Questions might also refer directly to attributes from the fact or dimension tables. For instance, a data source about sales for different products might include a dimension table listing all available products and their properties. In that case, a possible question could be *What is the net weight of [...]?*. The net weight for all products is stored in a field in a dimension table. The respective term in *desc* is mapped to the attribute and the tuple t is replaced with a tuple a:

$$a_i = (\text{''}, attr_j, dim_j, (key_{j_1}, ..., key_{j_n})) \tag{5}$$

[5] A detailed transformation process of a sample NL question to a SQL query is illustrated here: https://dbgit.prakinf.tu-ilmenau.de/code/qa-data/blob/master/sample_transformation.pdf

[6] http://www.nltk.org/

[7] http://clu.uni.no/icame/manuals/BROWN/INDEX.HTM

with '' being an empty string, $attr_j$ the name of the attribute mapped to the term, dim_j the name of the dimension table, $(key_{j_1}, ..., key_{j_n})$ the list of join attributes between the dimension table and the fact table and potential second level dimension tables. Similar to d_i, the empty string denotes an unspecified data constraint.

3.3.6 Term. Concluding, for the remaining tuples t of the list it is checked, if the term is stored as data in the underlying data source. In this way, the information for the `WHERE` clause constraining attributes to concrete data is retrieved. For all dimension tables views are created having the concrete data in one field and the corresponding attribute name in a second field. These views are created for all dimension tables separately. Remaining terms from the NL question are queried regarding the data fields in these views. In case of a match, a tuple is created containing the matching term, the attribute name, the name of the dimension table and the join attribute for this dimension table:

$$w_i = (term, attr_j, dim_j, (key_{j_1}, ..., key_{j_n})) \quad (6)$$

with $term$ being the data term found in the data source, $attr_j$ the attribute name where the data term is stored in the data source, dim_j the dimension table where the data term is stored, $(key_{j_1}, ..., key_{j_n})$ the list of join attributes between the dimension table and the fact table and potential second level dimension tables.

The previous tuple containing the matching term in the description *desc* is replaced by the newly created tuple w_i.

3.4 Construction of the SQL Query

After processing all checks as described above, *desc* consists of processed tuples/strings only. All initial tuples that could not be processed/mapped to the data source are dismissed. In the next step, *desc* is transformed to the actual SQL query.

All tuples of *desc* are analyzed and the containing distinct dimension tables are extracted and stored as a list for the creation of the joins in the query. As the schema might also contain second level dimension tables, the list is analyzed if second level dimension tables are contained and the corresponding first level dimension table is missing. If so, the first level dimension table is added to the list of tables.

As we identified the correct fact table from the NL question, we have to check if the question can actually be answered based on the retrieved dimension tables. Although we are able to handle different fact tables in one schema, it is possible that a fact table is not connected to a dimension table identified in the question. If that is the case, we have to exit the transformation process and give feedback to the user that the question cannot be answered based on the underlying data source.

3.4.1 SELECT clause. The `SELECT` clause is created based on the tuples in *desc* that contain at least one empty string at the beginning (of type *a* and *d*) - resulting from steps 3.3.4 and 3.3.5. As already described above, these two types of tuples contain empty string(s) at the first (and second) position. The first empty string in these tuples means that no data constraint is identified. The second empty string denotes an unassigned attribute for the corresponding table. Hence, these two types of tuples are considered to add the attributes to the `SELECT` clause. If only the first string in the tuple is empty, the attribute set on second position in the tuple ($attr_j$) is directly used in the `SELECT` clause. If also the second string is empty, the tuple contains the affected table on position three (dim_j) and the join attribute of this dimension table to the fact table is used in the `SELECT` clause. Needless to mention, if *desc* contains more than one tuple of these type, all attributes are added and separated by a comma. As described in Section 3.3.2, our approach is able to add SQL operators and functions to the query. In that case, the operator/function is part of *desc* at the relative position as mentioned in the SQL query. For instance, the question *How many different products [...]?* requires the functions `COUNT` (for *How many*) and `DISTINCT` (for *different*). Both mentions are followed directly by the concrete reference in the data source. Therefore, SQL functions are assigned to the attribute referenced in the tuple directly following the function in *desc*. Another special case is the existence of a compare operator in *desc*. In that case, the reference that the NL question asks for a comparative query is stored in *desc* as string *compare*. The string is set in *desc* at the position between the tuples stating the comparative data constraints or before all tuples relevant for the comparison. In the NL question such a type of query is identified by NL phrases like *compared to* or *comparing a and b*. For instance, the question *What were [...] in January 1997 compared to March 1998?* requires two subqueries – one for January 1997 and one for March 1998. In this case, the number of required subqueries is counted and respective aliases created. For each alias, attributes as previously described are added to the `SELECT` clause having the alias as prefix. As described in Section 3.3.2 we identify NL phrases that refer to SQL functions/operators, such as `SUM` or `AVG`, and store as a string s_i in *desc*. When creating the `SELECT` clause, this function is added to attributes contained in the following tuple of s_i within *desc*. In case there are more than one attribute added to the `SELECT` clause, the remaining attributes (including table references) are added to a separate list to be processed later when the `GROUP BY` case is checked.

3.4.2 FROM clause. The `FROM` clause includes the previously identified fact table combined with `INNER JOIN`s to the affected (first and second level) dimension tables. As described in Section 3.2.1, the join attributes for fact and dimension tables are provided and utilized for the joins in the `FROM` clause of the query. For the special case of comparative subqueries, the `FROM` clause contains the subqueries for each constraint from the NL question using the previously created alias. For each subquery a tuple w_i exists in *desc*. w_i contains the table, the attribute and the data term required for creating the SQL query containing a constraint on an attribute.

3.4.3 WHERE clause. The `WHERE` clause is constructed based on the tuples *w* contained in *desc*. All consecutive tuples of type *w* (having the first string set to a concrete data term) are added to the `WHERE` clause separated by `AND`. Each tuple *w* contains the table (position 3), the attribute (position 2) and the concrete data term (position 1) required to construct the constraint: w[2].w[1]='w[0]'.

Concluding, the existence of hints for the use of the `GROUP`, `ORDER` and `LIMIT` clauses are checked.

3.4.4 Further operators. For the `GROUP` clause a separate list is created during the creation of the `SELECT` clause. If there is an aggregation function used in the `SELECT` clause, the remaining attributes identified for selection are added to this separate list. This list is checked for containing items and if so, these attributes are added to the end of the query surrounded by the grouping function. The `ORDER` clause is utilized in case there is an aggregate function in the `SELECT` clause. If so, the attribute used for the

aggregate function is also used for ordering the results. For the `LIMIT` clause the NL question is analyzed for references to only present a limited amount of results. For instance, the phrases *first*, *last*, *top* followed by a number are hints to use a *LIMIT* clause following the corresponding number at the end of the SQL query.

3.5 Potential Extensions

In the current version, our approach is able to answer a various number of different question types – described more in detail in Section 4. Nevertheless, we identified several issues we already considered as a future concept but did not integrate a solution, yet.

3.5.1 User Feedback. A NL interface might be expected to act (almost) like a human when it comes to answering questions. As already stated above, there are questions that cannot be answered directly. In the current version, our prototype would simply exit the process and give a feedback like *This question cannot be answered based on the current data source.* But, we prefer to design the interface in a more conversational way. If the process comes across a problem, the interface should communicate the problem to the user and ask for advice/help. So far, we identified two different issues when a user's feedback would be required:

- Ambiguity
- Missing data

The first case refers to data terms within the data source that occur multiple times. This might be the case for different cities having the same name (ambiguity for the same type, e.g. cities) or data terms that occur in different attributes in different tables (ambiguity over different types - stored in different tables, e.g. a person and a city having the same name). In the current version, our approach groups all results when the term occurs multiple times in the same field (as for different cities with the same name) – different cities with the same name are treated as if they are the same. In the second case, our approach uses the first occurrence of the data term in data source when searching for table and attribute. In both cases, the best way of solving the ambiguity would be to call for the user's help. For the first case, the approach is required to take additional information from the data source and present it to the user when asking for advice which term is the correct one. For instance, when a question asks for something in Richmond the user should be called back: *Do you mean the Richmond in San Francisco district or in Vancouver district?*. In the second case, the schema information of the data source must be taken into account. The different tables and corresponding attributes must be presented to the user, so a selection can be communicated. For instance, the user might be asked *Do you mean the region with city Camacho or the employee with last name Camacho?*. Obviously, this call back could be avoided as we should be aware that with Camacho a place could be referred just by analyzing the NL question more in detail. But, our approach is as agnostic as possible regarding the data source. Although, we might know what a place is in the NL question, we are not aware which dimension tables store localized information.

For the case of missing data, the NL question is assumed to ask for terms that are not contained in the data source or a term from a dimension table that is not connected to the fact table. In the first case, an incorrectly written term in the NL question might be the cause for the issue. Therefore, in the first step we will integrate a spell-checker to be able to automatically correct words. In the second case, the NL question contains data source information and reference to a fact table, but the tables are not connected and a potential query does not present any result. The user might be presented with an alternative question by simplifying the question to an answerable query. For instance, the question *What were the costs for bread in January 1997?* refers to expenses regarding bread for specific time period. When a fact table about expenses is not connected to specific products, the user might be called back *We cannot provide the costs for bread. Would you rather be presented with the costs for all products for January 1997?*.

3.5.2 Rules. As mentioned, our approach is considered agnostic regarding the actual underlying data source. There are several prerequisites required for our transformation process, but in general we extract the essential information from the (information schema of the) data source and add additional information by using APIs, such as datamuse. However, there are questions that require a more detailed knowledge about the data source. For instance, the question *How many products were in stock [...]?* might be answered directly from the data source if the information of items in stock are stored explicitly. But, there could also be the case, that this information must be calculated from the existing information. For this example, the data source might provide the information of items ordered and items sold. Items in stock are calculated by subtracting the number of items sold from the number of items ordered.

To solve this type of issue, we consider explicit rules similar to MDX (Multidimensional expressions – a query language for OLAP data sources, based on SQL). Thereby, so-called calculated members can be created using various expressions and operators. A member can be stored in the OLAP cube to be available in further queries.

We are aware of the fact, that this type of rules cannot be retrieved from the (unknown) data source following our agnostic approach. Therefore, this extension builds upon the extension on user feedback as described above. We consider two different ways of retrieving additional rules:

(1) A user provides additional information about the data source when the data source is initialized. This requires the user to have further knowledge about the information schema and the contained data.
(2) The rules are retrieved as part of the user feedback when a question cannot be answered. The unknown term could be presented to the user together with (parts of) the information schema.

In either case, the user feedback could be retrieved in a visual way. The user might mark parts of the schema and add operators or comments. Thereby, the user is not required to have detailed knowledge about SQL or the desired rule language.

4 EVALUATION

4.1 Domain Application

Data stored in an OLAP-like structure is especially useful for BI scenarios. A (filtered) fact can be queried and the additional dimensions allow the user to drill up or down along the (hierarchical) structured data. Thus, NLIs for OLAP databases are an effective instrument for data scientists in marketing or other business contexts. We therefore initially apply our approach to a business scenario and utilized data provided by Foodmart and structured in a snowflake schema[8]. A similar data source is the

[8] https://github.com/rsim/mondrian_demo/blob/master/db/foodmart.sql

TPC-H benchmark[9]. It also comprises several fact and dimension tables including speaking attribute and table names. Our approach is therefore directly applicable to this benchmark. But, due to the size of the dataset and for illustration purposes, we utilized the Foodmart data source[10]. To be able to test and evaluate our system, we created a set of questions containing several challenging SQL operators or functions. The benchmark is described more in detail in the next section.

4.2 Benchmark

We are not aware of a benchmark containing questions and queries based on an OLAP data source. Hence, we are not able to present a qualitative evaluation in terms of accuracy or precision in comparison to other approaches. Though, we are eager to create and provide such a benchmark for further research approaches. Therefore, we utilized the Foodmart data source where data is stored in a snowflake schema with first and second level dimensions. The data source contains three different fact tables – holding data about sales, inventory and expense facts – and 17 dimension tables – holding data about additional information, such as stores, regions, customers, products etc. Each fact table is connected to at least four dimension tables directly and several dimension tables are connected in second level via other dimension tables.

We identified different question types and levels of difficulty for a NL interface to OLAP. There are simple questions with restrictions on one dimension table or complicated questions requiring calculations, subqueries and/or several SQL functions/operators. For the development process and assessment, we collected a number of over 40 different questions and grouped them together according to difficulty levels. To further explain the different levels, we adopt the approach presented by Affolter et al .[1]. The authors classify different questions by the required usage of operators and functions for the respective formal query. The following operators are applicable for our use case: **J**oin, **F**ilter, **A**ggregation, **O**rdering, **S**ubquery, **C**oncept. Concept denotes a pre-defined term, as e.g. "great movie" or "in stock" (when this information must be calculated and is not stored in the data source explicitly). Using these challenges, we identified four different levels:

Level 1 refers to rather simple questions with a simple constraint of one or more dimensions like *How many different products were sold in October 1998?*. Here, only a `COUNT` on one attribute and a constraint in one dimension table is required. The queries contain the challenges (J, A, F).

Level 2 refers to questions containing aggregations and grouping like *Show for each brand the total sales in 1998*. The word *each* indicates the use of the an aggregation in the `SELECT` clause and a grouping function for the respective attribute associated with brand. The queries contain the challenges (J, A, F, O).

Level 3 refers to queries containing subqueries for compared results or comparison using <, >, == like *In which month less bread was sold than in January 1997?*. For this type of question, a subquery containing the data for January 1997 is constructed and it is compared using the operator < and the `HAVING` clause. The queries contain the challenges (J, A, S, F, O).

Level 4 refers to complicated queries containing calculations on attributes or pre-defined constraints of data terms, such as *How many units by Jeffers were in stock in the first month of 1997?* – in case the term *in stock* cannot be mapped directly to an attribute of the data source. In this case, *in stock* must be calculated from available fields, such as *ordered* and *shipped* or similar attributes. This type of challenge is defined as Concept. Therefore, the queries contain the challenges (J, A, F, C) and optionally (O, S).

Table 3 shows sample questions and the respective challenges for each difficulty level. The full dataset containing over 40 different questions based on the Foodmart data source is available as download[11]. The dataset includes three columns, with the NL question in the first column, the difficulty level in the second column, and the corresponding SQL query in the third column. The dataset contains 14 questions of level 1, 12 questions of level 2, 11 questions of level 3 and 4 questions of level 4.

In the current version, our prototype is able to process questions of level 1-3 directly and questions of level 4 for specific cases, such as calculating *in stock* from the attributes *units_ordered* and *units_shipped* in the Foodmart data source. Regarding the benchmark, this means that our approach is able to answer 93% of the questions. Of course, we are aware of the fact that these results are not statistically firm – neither in terms of the number of questions nor regarding the subjective nature of the benchmark. Unfortunately, none of the competing systems presented in Section 2 is available as API or other source. Therefore, we are not able to compare the quality of the results of our approach to other approaches. We aim to encourage other researchers to utilize and expand our benchmark. Thereby, a qualitative evaluation can be enabled in the future. But, the current results show that our approach is able to construct SQL queries for a good number of NL questions with different difficulty levels facing several challenges.

As described in Section 3.5, we plan to integrate rules to be able to do calculations and constraints as required by complicated questions of level 4. Several concepts might be identified automatically or added manually by the respective user. This applies for concepts like *important*, *in stock* or *great* (in terms of good reviews). Another special case is *What are the total sales in the business year 1997/98?*. Here, *business year* might refer to a time period from July 1997 to June 1998. The data source does not contain sales information in terms of business years. Therefore, the query must contain constraints from pre-defined rules where concrete terms are mapped to data item constraints.

4.3 Portability of the Benchmark's Domain to other Domains

Initially, we applied our approach to an existing data source structured in a snowflake schema from the domain of business data. This type of data design makes sense when you need insights into your data and receive metrics or ratios, as e.g. mostly questions that start with how many/much or what were the total *xy* for *ab*. For this type of questions, it is a reasonable approach to structure the respective data in fact and dimension tables and make the data source accessible via a QA application. In this way, business analysts or marketing strategists are able to get insight into their key data without being forced to have knowledge about any formal query language. Although, we utilized the Foodmart data source for our first prototype, the data source and the data's domain can be easily replaced respectively changed. For instance, similar questions as collected in our benchmark are applicable

[9] http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf

[10] The queries described in Section 2.4 of the TPC-H specification might reflect real-world business questions, but cannot be summarized in one NL question - which our approach is aiming at.

[11] https://dbgit.prakinf.tu-ilmenau.de/code/qa-data/blob/master/queryDataset.tsv

Table 3: Question examples for the Foodmart data source

Level	Question	Challenges
1	How many different customers were in the first quarter of 1997?	J(sales_fact, customer, time_by_day) A(COUNT(customer_id)) F(quarter='Q1', year='1997')
2	How much bread was sold per city in 1997?	J(sales_fact, product, product_class, store, time_by_day) A(COUNT(product_id), GROUP BY(store_city)) O(COUNT(product_id)) F(product_category='Bread', year='1997')
3	In which month less bread was sold than in January 1997?	J(sales_fact, time_by_day, product, product_class) A(COUNT(product_id) S(HAVING(COUNT(product_id)) < F(month='January', year='1997', product_category='Bread')) F(month='January', year='1997', product_category='Bread') O(COUNT(product_id))
4	How many units by Jeffers were in stock in the first month of 1997?	J(inventory_fact, product, time_by_day) A(SUM(units_ordered, SUM(units_shipped))) F(brand_name='Jeffers', month='January', year='1997') C(in stock = units_ordered - units_shipped)

for data about issue trackers and software projects, such as *How many bug fixes were committed in January 2019 per developer?*.

5 SUMMARY & OUTLOOK

In this paper, we presented our approach in the research field of NLIDB specifically for OLAP data sources. We designed our prototype system to be as agnostic as possible regarding the underlying data source. Required information is extracted from the schema and the datamuse API is utilized to add synonyms to the data that is used to map phrases from the NL question to terms of the data source. In our approach, the NL question analyzed and an intermediate representation of the question is created. After several processing steps, this representation contains different types of tuples which are utilized in the subsequent construction of the SQL query. Depending on the type of the tuple it is used to create the `SELECT`, `FROM`, or `WHERE` clause. Specific terms identified in the NL question indicate the use of SQL operators and functions, such as SUM, MAX, GROUP BY, or LIMIT.

With this approach, we are able to answer questions of different difficulty levels. In default of an OLAP benchmark, we created a dataset containing more than 40 questions based on the Foodmart data source. The benchmark is available for download and we are planning to collect more questions to be able to evaluate our approach and compare it to other approaches.

Future work includes the implementation of the extensions described in Section 3.5. The first and foremost prerequisite for these extensions is the design and development of a graphical user interface (GUI). On the one hand, our intended user frontend is targeted on being interactive regarding the presented results from the data source. On the other hand, further developments aim at an conversational interface. The user is either presented with concrete results from the data source, or asked for feedback to give a relevant answer for an initially unclear question. For further data insights, the user is enabled to request more information by asking questions that build upon the preceding questions. Furthermore, the user frontend should provide the user with a facility to add an own data source and ask questions on it. The data source might be provided as a bunch of csv files or an SQL file or as an URL to a distant database. The data source is analyzed regarding the required information for our approach. Eventually the user is involved to extract the required information or give more descriptive hints about the schema, if essential parts are missing (such as speaking attribute or table names). The implementation of rules – as described in Section 3.5.2 – is a desired enhancement of our approach. This enables the construction of even more complex queries and give answers to questions of the highest difficulty level.

Overall, our approach is already able to construct SQL queries for rather complex NL questions. The addressed GUI will make the prototype available for users and the enhancements will complete our approach in terms of complexion of queries.

6 ACKNOWLEDGEMENTS

This work was partially funded by the German Research Foundation (DFG) under grant no. SA782/26.

REFERENCES

[1] Katrin Affolter, Kurt Stockinger, and Abraham Bernstein. 2019. A comparative survey of recent natural language interfaces for databases. *The VLDB Journal* 28, 5 (01 Oct 2019), 793–819. https://doi.org/10.1007/s00778-019-00567-8

[2] Ion Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. 1995. Natural language interfaces to databases - an introduction. *Natural Language Engineering* 1, 1 (1995), 29–81.

[3] Key-Sun Choi, Luis Espinosa Anke, Thierry Declerck, Dagmar Gromann, Jin-Dong Kim, Axel-Cyrille Ngonga Ngomo, Muhammad Saleem, and Ricardo Usbeck (Eds.). 2018. *Joint Proceedings of ISWC 2018 Workshops SemDeep-4 and NLIWOD-4. Workshop on Semantic Deep Learning (SemDeep-2018).* Vol. 2241. CEURS.

[4] Lan Jiang and Felix Naumann. 2019. Holistic primary key and foreign key detection. *Journal of Intelligent Information Systems* (2019), 1–23.

[5] F. Li and H. V. Jagadish. 2014. Constructing an Interactive Natural Language Interface for Relational Databases. *Proc. VLDB Endow.* 8, 1 (Sept. 2014), 73–84. https://doi.org/10.14778/2735461.2735468

[6] Giuseppe M. Mazzeo and Carlo Zaniolo. 2016. Answering Controlled Natural Language Questions on RDF Knowledge Bases. In *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016.* 608–611.

[7] M. Asif Naeem and Imran Sarwar Bajwa. 2012. Generating OLAP Queries from Natural Language Specification. In *ICACCI '12*. ACM, 768–773. https://doi.org/10.1145/2345396.2345522

[8] Jaydeep Sen, Fatma Ozcan, Abdul Quamar, Greg Stager, Ashish Mittal, Manasa Jammi, Chuan Lei, Diptikalyan Saha, and Karthik Sankaranarayanan. 2019. Natural Language Querying of Complex Business Intelligence Queries. In *SIGMOD '19*. ACM, New York, NY, USA, 1997–2000. https://doi.org/10.1145/3299869.3320248

[9] Nadine Steinmetz, Ann-Katrin Arning, and Kai-Uwe Sattler. 2019. From Natural Language Questions to SPARQL Queries: A Pattern-based Approach. In *Datenbanksysteme für Business, Technologie und Web (BTW 2019).* 289–308.

[10] Lei Zou, Ruizhe Huang, Haixun Wang, Jeffrey Xu Yu, Wenqiang He, and Dongyan Zhao. 2014. Natural Language Question Answering over RDF: A Graph Data Driven Approach. In *SIGMOD '14*. 313–324.