Towards Conversational OLAP

Matteo Francia
DISI - University of Bologna
m.francia@unibo.it

Enrico Gallinucci
DISI - University of Bologna
enrico.gallinucci@unibo.it

Matteo Golfarelli
DISI - University of Bologna
matteo.golfarelli@unibo.it

ABSTRACT

The democratization of data access and the adoption of OLAP in scenarios requiring hand-free interfaces push towards the creation of smart OLAP interfaces. In this paper, we envisage a conversational framework specifically devised for OLAP applications. The system converts natural language text in GPSJ (Generalized Projection, Selection and Join) queries. The approach relies on an ad-hoc grammar and a knowledge base storing multidimensional metadata and cubes values. In case of ambiguous or incomplete query description, the system is able to obtain the correct query either through automatic inference or through interactions with the user to disambiguate the text. Our tests show very promising results both in terms of effectiveness and efficiency.

1 INTRODUCTION

Nowadays, one of the most popular research trends in computer science is the democratization of data access, analysis and visualization, which means opening them to end users lacking the required vertical skills on the services themselves. Smart personal assistants [10] (Alexa, Siri, etc.) and auto machine learning services [20] are examples of such research efforts that are now on corporate agendas [1].

In particular, interfacing natural language processing (either written or spoken) to database systems opens to new opportunities for data exploration and querying [13]. Actually, in the area of data warehouse, OLAP (On-Line Analytical Processing) itself is an *"ante litteram"* smart interface, since it supports the users with a "point-and-click" metaphor to avoid writing well-formed SQL queries. Nonetheless, the possibility of having a conversation with a smart assistant to run an OLAP session (i.e., a set of related OLAP queries) opens to new scenarios and applications. It is not just a matter of further reducing the complexity of posing a query: a conversational OLAP system must also provide feedback to refine and correct wrong queries, and it must have memory to relate subsequent requests. A reference application scenario for this kind of framework is *augmented business intelligence* [9], where hand-free interfaces are mandatory.

In this paper, we envisage a conversational OLAP framework able to convert a natural language text into a GPSJ query. GPSJ [11] is the main class of queries used in OLAP since it enables Generalized Projection, Selection and Join operations over a set of tables. Although, some natural language interfaces to databases have already been proposed, to the best of our knowledge this is the first proposal specific to OLAP systems.

In our vision, the desiderata for an OLAP smart interface are:

- #1 it must be automated and portable: it must exploit cubes metadata (e.g. hierarchy structures, role of measures, attributes, and aggregation operators) to increase its understanding capabilities and to simplify the user-machine interaction process;
- #2 it must handle OLAP sequences rather than single queries: in an OLAP sequence the first query is fully described by the text, while the following ones are implicitly/partially described by an OLAP operator and require the system to have memory of the previous ones;
- #3 it must be robust with respect to user inaccuracies in using syntax, OLAP terms, and attribute values, as well as in the presence of implicit information;
- #4 it must be easy to configure on a DW without a heavy manual definition of the lexicon.

More technically, our text-to-SQL approach is based on a grammar parsing natural language descriptions of GPSJ queries. The recognized entities include a set of typical query lexicons (e.g. group by, select, filter) and the domain specific terms and values automatically extracted from the DW (see desiderata #1 and #4). Robustness (desiderata #3) is one of the main goals of our approach and is pursued in all the translation phases: lexicon identification is based on a string similarity function, multi-word lexicons are handled through *n*-grams, and alternative query interpretations are scored and ranked. The grammar proposed in Section 4.3 recognizes full queries only, thus desiderata #2 is not covered by the current work. To sum up, the main contributions of this paper are:

(1) a list of features and desiderata for an effective conversational OLAP system;
(2) an architectural view of the whole framework;
(3) an original approach to translate the natural language description of an OLAP query in a well-formed GPSJ query (Full query module in Figure 1).
(4) a set of tests to verify the effectiveness of our approach.

The remainder of the paper is organized as follows: in Section 2 the functional architecture and the modules of a conversational OLAP framework are sketched; Section 3 discusses related works; Section 4 describes in detail the query generation process; in Section 5 a large set of effectiveness and efficiency tests are reported; finally, Section 6 draws the conclusions and discusses the system evolution.

2 SYSTEM OVERVIEW

Figure 1 sketches a functional view of the architecture. Given a DW, that is a set of multidimensional cubes together with their metadata, the *offline* phase is aimed at extracting the DW specific terms used by user to express the queries. Such information are stored in the system knowledge base (KB). Noticeably, this phase runs only once for each DW unless it undergoes modifications. More in detail, the Automatic KB feeding process extracts from the cubes categorical attribute values and metadata (e.g. attribute and measure names, hierarchy structures, aggregation operators). With reference to the cube represented through a DFM schema in Figure 2, the KB will store the names of the measures (e.g. StoreSales, StoreCost) together with their default aggregation operators (e.g. sum, avg). Additional synonyms can be automatically extracted from open data ontologies (Wordnet [16] in our implementation); for example, "client" is a synonym for Customer. The larger the number of synonyms, the wider the

language understood by the systems. Besides the domain specific terminology, the KB includes the set of standard OLAP terms that are domain independent and that do not require any feeding (e.g. group by, equal to, select). Further enrichment can be optionally carried out manually (i.e., by the KB enrichment module) when the application domain involves a non-standard vocabulary (i.e., when the physical names of tables and columns do not match the words of a standard vocabulary).

The *online* phase runs every time a query is issued to the system. The spoken query is initially translated to text by the Speech-to-text module. This task is out of scope in our research and we exploited the Google API in our implementation. The uninterpreted text is then analyzed by the Interpretation module that actually consists of two sub-modules: Full query is in charge of interpreting the texts describing full queries, which typically happens when an OLAP session starts. Conversely, OLAP operator modifies the latest query when the user states an OLAP operator along an OLAP session. On the one hand, understanding a single OLAP operator is simpler since it involves less elements. On the other hand, it requires to have memory of previous queries (stored in the Log) and to understand which part of the previous query must be modified.

Due to natural language ambiguity, user errors and system inaccuracies, part of the text can be misunderstood. The role of the Disambiguation module is to solve ambiguities by asking appropriate questions to the user. The reasons behind the misunderstandings are manyfold, including (but not limited to): ambiguities in the aggregation operator to be used; inconsistency between attribute and value in a selection predicate; (3) identification of relevant elements in the text without full comprehension.

The output of the previous steps is a data structure (i.e., a parsing tree) that models the query and that can be automatically turned in a SQL query exploiting the DW structure stored in the KB. Finally, the obtained query is run on the DW and the results are reported to the user by the Execution & Visualization module. Such module could exploit a standard OLAP visualization tool or it could implement voice-based approaches [21] to create an end-to-end conversational solution.

3 RELATED WORKS

Conversational business intelligence can be classified as a *natural language interface* (NLI) to *business intelligence* systems to drive analytic sessions. Despite the plethora of contributions in each area, to the best of our knowledge, no approach lies at their intersection.

NLIs to operational databases enable users to specify complex queries without previous training on formal programming languages (such as SQL) and software; a recent and comprehensive survey is provided in [2]. Overall, NLIs are divided into two categories: question answering (QA) and dialog (D). While the former are designed to operate on single queries, only the latter are capable of supporting sequences of related queries as needed in OLAP analytic sessions. However, to the best of our knowledge, no dialog-based system for OLAP sessions has been provided so far. The only contribution in the dialog-based direction is [14], where the authors provide an architecture for querying relational databases; with respect to this contribution we rely on the formal foundations of the multidimensional model to drive analytic sessions (e.g., according to the multidimensional model it is not possible to group by a measure, compute aggregations of categorical attributes, aggregate by descriptive attributes, ensure drill-across validity). Also differently from [14], the results we provide are supported by extensive effectiveness and efficiency performance evaluation that completely lack in [14]. Finally, existing dialog systems, such as [17], address the exploration of linked data. Hence they are not suitable for analytics on the multidimensional model. As to question answering, existing systems are well understood and differentiate for the knowledge required to formulate the query and for the generative approach. Domain agnostic approaches solely rely on the database schema. NaLIR [13] translates natural language queries into dependency trees [15] and brute-forcefully transforms promising trees until a valid query can be generated. In our approach we rely on n-grams instead of dependency trees [15] since the latter cannot be directly mapped to entities in the knowledge base (i.e., they require tree manipulation) and are sensible to the query syntax (e.g., *"sum unit sales"* and *"sum the unit sales"* produce two different trees with the same meaning). SQLizer [22] generates templates over the issued query and applies a "repair" loop until it generates queries that can be obtained using at most a given number of changes from the initial template. Domain specific approaches add semantics to the translation process by means of domain-specific ontologies and ontology-to-database mappings. SODA [5] uses a simple but limited keyword-based approach that generates a reasonable and executable SQL query based on the matches between the input query and the database metadata, enriched with domain-specific ontologies. ATHENA [18] and its recent extension [19] map natural language into an ontology representation and exploit mappings crafted by the relational schema designer to resolve SQL queries. Analyza [6] integrates the domain-specific ontology into a "semantic grammar" (i.e., a grammar with placeholders for the typed concepts such as measures, dimensions, etc.) to annotate and finally parse the user query. Additionally, Analyza provides an intuitive interface facilitating user-system interaction in spreadsheets. Unfortunately, by relying on the definition of domain specific knowledge and mappings, the adoption of these approaches is not plug-and-play as an ad-hoc ontology is rarely available and is burdensome to create.

In the area of business intelligence the road to conversation-driven OLAP is not paved yet. The recommendation of OLAP sessions to improve data exploration has been well-understood [3] also in domains of unconventional contexts [9] where hand-free interfaces are mandatory. Recommendation systems focus on integrating (previous) user experience with external knowledge to suggest queries or sessions, rather than providing smart interfaces to BI tools. To this end, personal assistants and conversational interfaces can help users unfamiliar with such tools and SQL language to perform data exploration. However, end-to-end frameworks are not provided in the domain of analytic sessions over multidimensional data. QUASL [12] introduces a QA approach over the multidimensional model that supports analytical queries but lacks both the formalization of the disambiguation process (i.e., how ambiguous results are addressed) and the support to OLAP sessions (with respect to QA, handling OLAP sessions requires to manage previous knowledge from the Log and to understand whether the issued sentence refines previous query or is a new one). Complementarily to our framework, [21] recently formalized the vocalization of OLAP results.

To summarize, the main differences of our approach to the previous works are the following.

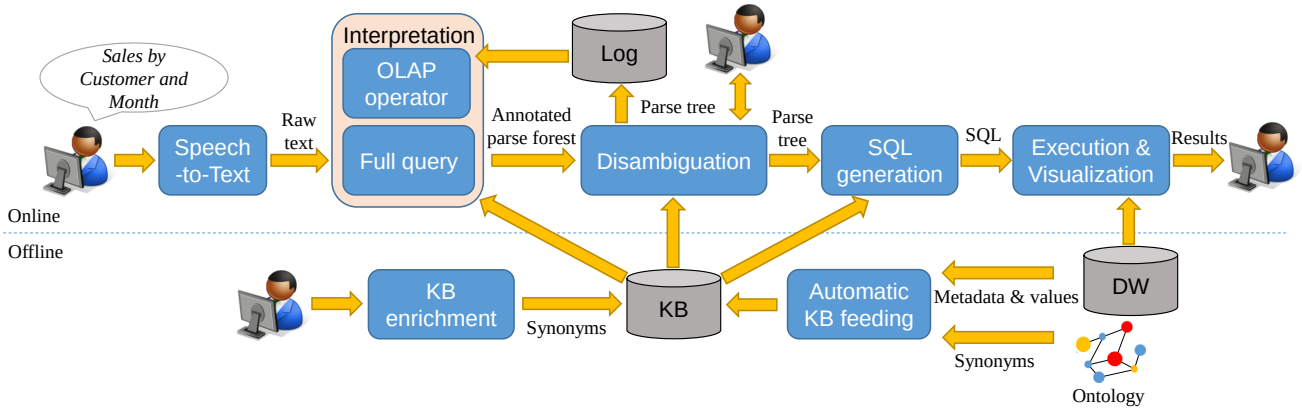


Figure 1: A functional architecture for a conversational OLAP framework.

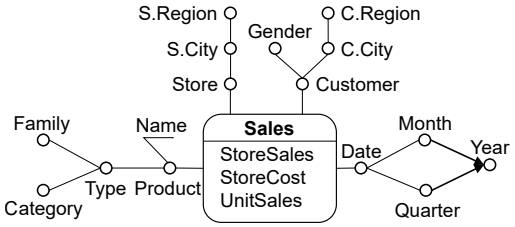


Figure 2: Simplified DFM representation of the Foodmart Sales cube.

(1) The envisioning of an end-to-end general-purpose *dialog-driven* framework supporting OLAP sessions.
(2) The definition of the framework's functional architecture and the formalization of its modules.
(3) The enforcement of multidimensional constraints by the means of formal grammar supporting GPSJ queries.
(4) A plug-and-play implementation of the core modules on top existing data warehouses: our framework does not impact existing schemata and the integration with external knowledge is supported but not mandatory.

4 QUERY GENERATION

In the next subsections, we describe in detail the interpretation process for a single full query. Figure 3 shows the necessary steps in the online phase. We remind that, with reference to Figure 1, modules Speech-to-Text, OLAP operator and, Execution & Visualization are out of scope here. Conversely, the offline phase process and the related modules are implemented but not reported.

4.1 The Knowledge Base

The system Knowledge Base (KB in Figure 1) stores all the information extracted from the DW that is necessary to support the translation phases. Information can be classified in DW *Entities* and *Structure*. More in detail, the DW structure enables consistency checks on parse trees and, given a parse tree, allows the SQL generation. It includes:

- **Hierarchy structure**: the roll-up relationships between attributes.
- **Aggregation operators**: for each measure, the applicable operators and the default one.
- **DB tables**: the structure of the database implementing the DW including tables' and attributes' names, primary and foreign key relationships.

The set of DW entities includes:

- **DW element names**: measures, dimensional attributes and fact names.
- **DW element values**: for each categorical attribute, all the values are stored.

Several synonyms can be stored for each entity, enabling the system to cope with slang and different shades of the text. Both DW entities and DW structure are automatically collected by querying the DW and its data dictionary and are stored in a QB4OLAP [8] compatible repository. Besides DW entities, that are domain specific, the knowledge base includes those keywords and patterns that are typically used to express a query:

- **Intention keywords**: express the role of the subsequent part of text. Examples of intention keywords are group by, select and filter.
- **Operators**: include logic (e.g. and, or), comparison (e.g. greater, equal) and aggregation operators (e.g. sum, average).
- **Patterns of dates and numbers**: used to automatically recognize dates and numbers in the raw text.

Overall, the entities defined in the system are defined as $\mathcal{E} = \{E_1, E_2, ..., E_m\}$; note that each entity can be a multi-word that can be modeled as a sequence of tokens $E = \langle t_1, ..., t_r \rangle$.

4.2 Tokenization and Mapping

A raw text T can be modeled as a sequence of tokens (i.e., single words) $T = \langle t_1, t_2, ..., t_z \rangle$. The goal of this phase is to identify in T the known entities that are the only elements involved in the Parsing phase.

Turning a text into a sequence of entities means finding a mapping between tokens in T and $\mathcal{E}$. More formally:

Definition 4.1 (Mapping & Mapping function). A mapping function $M(T)$ is a partial function that associates sub-sequences[1] from T to entities in $\mathcal{E}$ such that:

- sub-sequences of T have length n at most;
- the mapping function determines a partitioning of T;

[1]The term n-gram is used as a synonym of sub-sequence in the area of text mining.

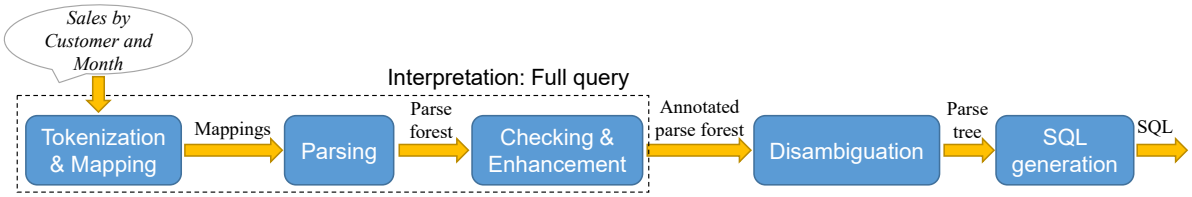


Figure 3: Query generation steps for a text describing a complete query. The KB is involved in all the steps.

- a sub-sequence $T' = \langle t_i, ..., t_l \rangle \in T$ with $(l \leq n)$ is associated to an entity E if and only if $E \in TopN_{E \in \mathcal{E}} Sim(T', E)$ and $Sim(T', E) > \alpha$.

The output of a mapping function is a sequence $M = \langle E_1, ..., E_l \rangle$ on $\mathcal{E}$ that we call a *mapping*. Given the mapping M the similarities between each entity E_i and the corresponding tokens $Sim(T', E_i)$ are also retained and denoted with $Sim_M(E_i)$. A mapping is said to be *valid* if the fraction of mapped tokens in T is higher than a given threshold β. We call $\mathcal{M}$ the set of valid mappings.

Several mapping functions (and thus several mappings) may exists between T and $\mathcal{E}$ since Definition 4.1 admits sub-sequences of variable lengths and retains, for each sub-sequence, the top similar entities. This increases interpretation robustness, but it can lead to an increase in computation time. Generating several different mappings increases robustness since it allows to choose, in the next steps, the *best text interpretation* out of a higher number of candidates. Generated mappings differ both in the number of entities involved and, in the specific entities mapped to a token. In the simple case where multi-token mappings are not possible (i.e., $n = 1$ in Definition 4.1) the number of generated mappings for a raw text T, such that $|T| = z$, is:

$$\sum_{i=\lceil z \cdot \beta \rceil}^{z} \binom{z}{i} \cdot N^i$$

The formula counts the possible configurations of sufficient length (i.e., higher or equal to $\lceil z \cdot \beta \rceil$) and, for each length, count the number of mappings determined by the top similar entities. Since the number of candidate mappings is exponential, we consider only the most significant ones through α, β, and N: α imposes sub-sequence of tokens to be very similar to an entity; N further imposes to consider only the N entities with the highest similarity; finally, β imposes a sufficient portion of the text to be mapped.

The similarity function $Sim()$ is based on the Levenshtein distance and keeps token permutation into account to make similarity robust to token permutations (i.e., sub-sequences $\langle P., Edgar \rangle$ and $\langle Edgar, Allan, Poe \rangle$ must result similar). Given two token sequences T and W with $|T| = l, |W| = m$ such that $l \leq m$ it is:

$$Sim(\langle t_1, ..., t_l \rangle, \langle w_1, ..., w_m \rangle) =$$

$$\max_{D \in Disp(l,m)} \frac{\sum_{i=1}^{l} sim(t_i, w_{D(i)}) \cdot \max(|t_i|, |w_{D(i)}|)}{\sum_{i=1}^{l} \max(|t_i|, |w_{D(i)}|) + \sum_{i \in \hat{D}} |w_i|}$$

where $D \in Disp(l, m)$ is an l-disposition of $\{1, ..., m\}$ and $\hat{D}$ is the subset of values in $\{1, ..., m\}$ that are not present in D. Function $Sim()$ weights token similarity based on their lengths (i.e., $\max(|t_i|, |w_{D(i)}|)$ and penalizes similarities between sequences of different lengths that implies unmatched tokens (i.e., $\sum_{i \in \hat{D}} |w_i|$).

Example 4.2 (Token similarity). Figure 4 shows some of the possible token dispositions for the two token sequences $T = \langle P., Edgar \rangle$ and $W = \langle Edgar, Allan, Poe \rangle$. The disposition determining the highest similarity is surrounded by a dashed rectangle; the similarity is 0.46 and it is calculated as

$$Sim(T, W) = \frac{sim(P., Poe)|Poe| + sim(Edgar, Edgar)|Edgar|}{|Poe| + |Edgar| + |Allan|}$$

□

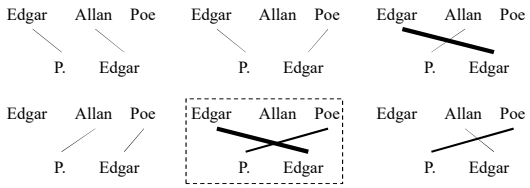


Figure 4: Token dispositions: arcs denote the correspondence of tokens for a specific disposition (the bolder the line, the higher the similarity). The disposition determining the maximum similarity is shown in a dashed rectangle.

We assume the best interpretation of the input text to be the one where (1) all the entities discovered in the text are included in the query (i.e., all the entities are parsed through the grammar) and, (2) each entity discovered in the text is perfectly mapped to one sub-sequence of tokens (i.e., $Sim_M(E_i) = 1$). The two previous statements are modeled through the following score function. Given a mapping $M = \langle E_1, ..., E_m \rangle$, we define its score as

$$Score(M) = \sum_{i=1}^{m} Sim_M(E_i) \quad (1)$$

The score is higher when M includes several entities with high values of Sim_M. Although at this stage it is not possible to predict if a mapping will be fully parsed, it is apparent that the higher the mapping score, the higher its probability to determine an optimal interpretation. As it will be explained in the Section 4.4, ordering the mapping by descending score also enables pruning strategies to be applied.

Example 4.3 (Tokenization and mapping). Given the set of entities $\mathcal{E}$ and a tokenized text $T = \langle medium, sales, in, 2019, by, the, region \rangle$, examples of mappings M_1 and M_2 are:

$M_1 = \langle$avg, UnitSales, where, 2019, group by, region$\rangle$

$M_2 = \langle$avg, UnitSales, where, 2019, group by, *Regin*$\rangle$

where the token *the* is not mapped and the token *region* is mapped to the attribute Region in M_1 and to the value *Regin* in M_2 (where *Regin* is a value of attribute Customer that holds a sufficient similarity). □

Table 1: Generic and domain specific syntactic categories.

Category	Entity	Synonym samples
Int	select	return, show, get
Whr	where	in, such that
Gby	group by	by, for each, per
Cop	=, <>, >, <, ≥, ≤	equal to, greater than
Agg	sum, avg	total, medium
Cnt	count, count distinct	number, amount
Fct	*Facts*	Domain specific
Mea	*Measures*	Domain specific
Att	*Attributes*	Domain specific
Val	*Categorical values*	Domain specific
	Dates and numbers	-

4.3 Query Parsing

Parsing a valid mapping means searching in the text the complex syntax structures (i.e., *clauses*) that build-up the query. Given a mapping M, the output of a parser is a *parse tree* PT_M, i.e. an ordered, rooted tree that represents the syntactic structure of a string according to a given grammar. To the aim of parsing, entities (i.e., terminal elements in the grammar) are grouped in syntactic categories (see Table 1). The whole grammar is described in Figure 5. As a GPSJ query consists of 3 clauses (measure, group by and selection), in our grammar we identify four types of derivations[2]:

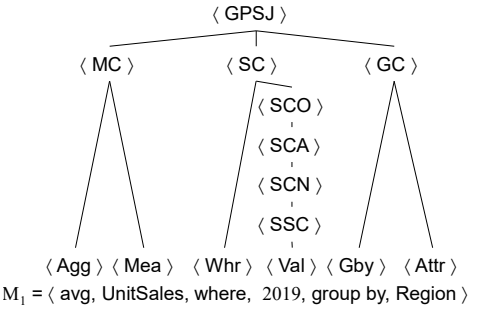
- **Measure clause** ⟨MC⟩: this derivation consists of a list of measure/aggregation operator pairs. If the operator is omitted (i.e., ⟨MC⟩ ::= ⟨Mea⟩ applies) the default one specified in the KB will be applied to the measure during the Checking & Enhancement step.
- **Group by clause** ⟨GC⟩: this derivation consists of a sequence of attribute names preceded by an entity of type ⟨Gby⟩.
- **Selection clause** ⟨SC⟩: this derivation consists of a Boolean expression of simple selection predicates ⟨SSC⟩; follows standard SQL operator priority (see ⟨SCO⟩, ⟨SCA⟩, and ⟨SCN⟩ derivations).
- **GPSJ query** ⟨GPSJ⟩: this derivation assembles the final query. Only the measure clause is mandatory since a GPSJ could aggregate a single measure with no selections. The order of clauses is irrelevant; this implies the proliferation of derivations due to clause permutations.

The GPSJ grammar is LL(1)[3] [4], is not ambiguous (i.e., each mapping admits a single parsing tree PT_M) and can be parsed by a LL(1) parser with linear complexity [4]. If the input mapping M can be fully parsed, PT_M will include all the entities as leaves. Conversely, if only a portion of the input belongs to the grammar, an LL(1) parser will produce a partial parsing, meaning that it will return a parsing tree including the portion of the input mapping that belongs to the grammar. Remaining entities can be either singleton or complex clauses that were not possible to connect to the main parse tree. We will call *parse forest* PF_M the union of the parsing tree with residual clauses. Obviously, if all the entities are parsed, it is $PF_M = PT_M$. Considering the whole forest rather than the simple parse tree enables disambiguation and errors to be recovered during the Disambiguation phase.

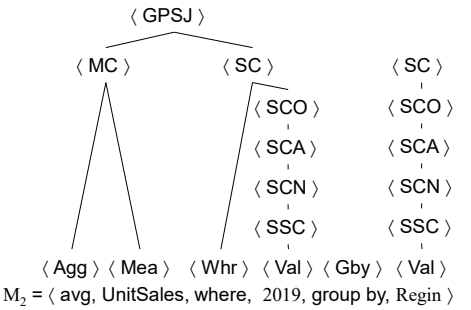
[2]A derivation in the form ⟨X⟩ ::= e represent a substitution for the non-terminal symbol ⟨X⟩ with the given expression e. Symbols that never appear on the left side of ::= are named terminals. Non-terminal symbols are enclosed between ⟨⟩.
[3]The rules presented in Figure 5 do not satisfy LL(1) constraints for readability reasons. It is easy to turn such rules in a LL(1) complaint version, but the resulting rules are much more complex to be read and understood.

⟨GPSJ⟩ ::= ⟨MC⟩⟨GC⟩⟨SC⟩ | ⟨MC⟩⟨SC⟩⟨GC⟩ | ⟨SC⟩⟨GC⟩⟨MC⟩
| ⟨SC⟩⟨MC⟩⟨GC⟩ | ⟨GC⟩⟨SC⟩⟨MC⟩ | ⟨GC⟩⟨MC⟩⟨SC⟩
| ⟨MC⟩⟨SC⟩ | ⟨MC⟩⟨GC⟩ | ⟨SC⟩⟨MC⟩ | ⟨GC⟩⟨MC⟩
| ⟨MC⟩
⟨MC⟩ ::= (⟨Agg⟩⟨Mea⟩ | ⟨Mea⟩⟨Agg⟩ | ⟨Mea⟩ | ⟨Cnt⟩⟨Fct⟩
| ⟨Cnt⟩⟨Attr⟩)+
⟨GC⟩ ::= ⟨Gby⟩⟨Attr⟩+
⟨SC⟩ ::= ⟨Whr⟩⟨SCO⟩
⟨SCO⟩ ::= ⟨SCA⟩"*or*"⟨SCO⟩ | ⟨SCA⟩
⟨SCA⟩ ::= ⟨SCN⟩"*and*"⟨SCA⟩ | ⟨SCN⟩
⟨SCN⟩ ::= "*not*"⟨SSC⟩ | ⟨SSC⟩
⟨SSC⟩ ::= ⟨Attr⟩⟨Cop⟩⟨Val⟩ | ⟨Attr⟩⟨Val⟩ | ⟨Val⟩⟨Cop⟩⟨Attr⟩
| ⟨Val⟩⟨Attr⟩ | ⟨Val⟩

Figure 5: Backus-Naur representation of the grammar.



(a) The entire mapping is parsed (i.e., $PF_M = PT_M$).



(b) ⟨Gby⟩⟨Val⟩ cannot be parsed.

Figure 6: Parsing forests from Example 4.3.

Example 4.4 (Parsing). Figure 6 reports the parsing outcome for the two mappings in Example 4.3. M_1 is fully parsed, thus its parsing forest correspond to the parsing tree (i.e., $PT_{M_1} = PF_{M_1}$). Conversely, in M_2 the last token is wrongly mapped to the attribute value *Regin* rather than to the attribute name *Region*. This prevents the full parsing and the parsing tree PT_M does not include all the entities in M.

Annotation type	Gen. derivation sample	Example
Ambiguous Attribute	⟨SSC⟩ ::= ⟨Val⟩	*"sum unit sales for Salem"*, but *Salem* is member of City and StoreCity
Ambiguous Agg. Operator	⟨MC⟩ ::= ⟨Mea⟩	*"unit sales by product"*, but sum and avg are valid aggregations
Attribute-Value Mismatch	⟨SSC⟩ ::= ⟨Attr⟩⟨Cop⟩⟨Val⟩	*"sum unit sales for product New York"*, but *New York* is not a Product
MD-Meas Violation	⟨MC⟩ ::= ⟨Agg⟩⟨Mea⟩	*"sum prices per store"*, but Price is not addictive
MD-GBY Violation	⟨GC⟩ ::= ⟨Gby⟩⟨Attr⟩+	*"average prices by name"*, but Product is not in ⟨GC⟩
Unparsed clause	–	*"average unit sales by Regin"*, but *Regin* is not an attribute

Table 2: Annotation types.

4.4 Knowledge-based Parse Forest Checking and Enhancement

The Parsing phase verifies adherence of the mapping to the GPSJ grammar, but this does not guarantee the executability of the corresponding SQL query due to implicit elements, ambiguities and errors. The following list enumerates the cases that require further processing.

- **Ambiguous attribute**: the ⟨SSC⟩ clause has an implicit attribute and the parsed value belongs to multiple attribute domains.
- **Ambiguous aggregation operator**: the ⟨MC⟩ clause has an implicit aggregation operator but the measure is associated with multiple aggregation operators.
- **Attribute-value mismatch**: the ⟨SSC⟩ clause includes a value that is not valid for the specified attribute, i.e. it is either outside or incompatible with the attribute domain.
- **Violation of a multidimensional constraint on a measure**: the ⟨MC⟩ clause contains an aggregation operator that is not allowed for the specified measure.
- **Violation of a multidimensional constraint on an attribute**: the ⟨GC⟩ clause contains a descriptive attribute without the corresponding dimensional attribute[4].
- **Implicit aggregation operator**: the ⟨MC⟩ clause has an implicit aggregation operator and the measure is associated with only one aggregation operator or a default operator is defined.
- **Implicit attribute**: the ⟨SSC⟩ clause has an implicit attribute and the parsed value belongs to only one attribute domain.
- **Unparsed clause**: The parser was not able to fully understand the mapping and the returned parse forest includes one or more dandling clauses.

Each bullet above determines a possible annotation of the nodes in the parsed forest. In case of implicit information, the entities in the KB represented by the implicit nodes are automatically added to the parse tree; thus no user action is required. All the other cases cannot be automatically solved and the nodes are annotated with a specific error type that will trigger a user-system interaction during the Disambiguation phase. Table 2 reports annotation examples.

A textual query generates several parsing forests, one for each mapping. In our approach, only the most promising one is proposed to the user in the Disambiguation phase. This choice comes from two main motivations:

- Proposing more than one alternative queries to the user can be confusing and makes very difficult to contextualize the disambiguation questions.
- Proposing only the most promising choice may incur in missing the optimal derivation but it makes easier to create a baseline query. This one can be improved by adding or removing clauses through further interactions enabled by the OLAP operator module.

[4]According to DFM a *descriptive attribute* is an attribute that further describes a dimensional level (i.e., it is related one-to-one with the level), but that can be used for aggregation only in combination with the corresponding level.

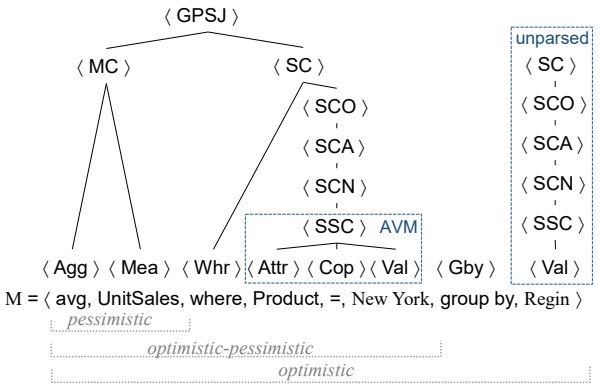


Figure 7: Portion of the parsing forest contributing to its score depending on the adopted scoring criterion.

Definition 4.5 (Parsing Forest Score). Given a mapping M and the corresponding parsing forest PF_M, we define its score as $Score(PF_M) = Score(M')$ where M' is the sub-sequence of M belonging to the parsing tree PT_M.

The parsing forest holding the highest score is the one proposed to the user. This ranking criterion is based on an *optimistic-pessimistic* forecast of the outcome of the Disambiguation phase. On the one hand, we optimistically assume that the annotations belonging to PT_M will be positively solved in the Disambiguation phase and the corresponding clauses and entities will be kept. On the other hand, we pessimistically assume that non-parsed clauses belonging to PF_M will be dropped. The rationale of our choice is that an annotated clause included in the parsing tree is more likely to be a proper interpretation of the text. As shown in Figure 7, a totally pessimistic criterion (i.e., exclude from the score all the annotated clauses and entities) would carry forward a too simple, but non-ambiguous, forest; conversely, a totally optimistic criterion (i.e., consider the score of all the entities in PF_M) would make preferable a large but largely non-parsed forest. Please note that the bare score of the mapping (i.e., the one available before parsing) corresponds to a totally optimistic choice since it sums up the scores of all the entities in the mapping.

The ranking criterion defined above enables the pruning of the mappings to be parsed as shown by Algorithm 1 that reports the pseudo-code for the Full query module. Reminding that mappings are parsed in descending score order, let us assume that, at some step, the best parse forest is $PF_{M'}$ with score $Score(PF_{M'})$. If the next mapping to be parsed, M'', has score $Score(M'') <$

Algorithm 1 Parsing forest selection

Require: $\mathcal{M}$: set of valid mappings
Ensure: PF^*: best parsing forest
1: $\mathcal{M} \leftarrow sort(\mathcal{M})$ ▷ sort mappings by score
2: $PF^* \leftarrow \varnothing$
3: **while** $\mathcal{M} \neq \varnothing$ **do** ▷ while mapping space is not exhausted
4: $M \leftarrow head(\mathcal{M})$ ▷ get the mapping with highest score
5: $\mathcal{M} \leftarrow \mathcal{M} \setminus \{M\}$ ▷ remove it from $\mathcal{M}$
6: $PF_M \leftarrow parse(M)$ ▷ parse the mapping
7: **if** $score(PF_M) > score(PF^*)$ **then**
▷ if current score is higher than previous score
8: $PF^* \leftarrow PF_M$ ▷ store the new parsing forest
9: $\mathcal{M} \leftarrow \mathcal{M} \setminus \{M' \in \mathcal{M}, score(M') \leq score(PF^*)\}$
▷ remove mappings with lower scores from $\mathcal{M}$
return PF^*

$Score(PF_{M'})$, we can stop the algorithm and return $PF_{M'}$ since the optimistic score of M'' is an upper bound to the score of the corresponding parse forest.

Algorithm 1 works as follows. At first, mappings are sorted by their score (Line 1), the best parse forest is initialized, and the iteration begins. While the set of existing mappings is not exhausted (Line 3), the best mapping is picked, removed from the set of candidates, and its parsing forest is generated (Lines 4–6). If the score of the current forest is higher than the score of the stored one (Line 7), then the current forest is stored (Line 8) and all the mappings with a lower score are removed from the search space (Line 9) as the pruned mappings cannot produce parsing forests with a score greater than what has been already parsed.

4.5 Parse Tree Disambiguation

Each annotation in the parse forest requires a user choice to restore query correctness. User-system interaction takes place through a set of questions, one for each annotation. Obviously, each answer must be parsed following the same step sequence discussed so far. Table 3 reports the question templates for each annotation type. Each question allows to either provide the missing information or to drop the clause. Templates are standardized and user choices are limited to keep interaction easy. This allows also unskilled users to obtain a baseline query. Additional clauses can be added through the OLAP operator module that implements OLAP navigation.

At the end of this step, the parse forest is reduced to a parse tree since ambiguous clauses are either dropped or added to the parsing tree.

4.6 SQL Generation

Given a parsed tree PT_M, the generation of its corresponding SQL requires to fill in the `SELECT`, `WHERE`, `GROUP BY` and `FROM` statements. The SQL generation is applicable to both star and snowflake schemata and is done as follows:

- `SELECT`: measures and aggregation operators from ⟨MC⟩ are added to the query selection clause together with the attributes in the group by clause ⟨GC⟩;
- `WHERE`: predicate from the selection clause ⟨SC⟩ (i.e., values and their respective attributes) is added to the query predicate;
- `GROUP BY`: attributes from the group by clause ⟨GC⟩ are added to the query group by set;
- `FROM`: measures and attributes/values identify, respectively, the fact and the dimension tables involved in the query. Given these tables, the join path is identified by following the referential integrity constraints (i.e., by following foreign keys from dimension tables imported in the fact table).

Example 4.6 (SQL generation). Given the GPSJ query *"sum the unit sales by type in the month of July"*, its corresponding SQL is:

```
SELECT Type, sum(UnitSales)
FROM Sales s JOIN Product p ON (s.pid = p.id)
             JOIN Date d ON (s.did = d.id)
WHERE Month = "July"
GROUP BY Type
```

□

5 EXPERIMENTAL TESTS

In this section, we evaluate our framework in terms of effectiveness and efficiency. Tests are carried out on a real-word benchmark of analytics queries [7]. Since queries from [7] refers to private datasets, we mapped the natural language queries to the Foodmart schema[5]. The Automatic KB feeding populated the KB with 1 fact, 39 attributes, 12337 values and 12449 synonyms. Additionally, only 50 synonyms where manually added in the KB enrichment step (e.g., *"for each"*, *"for every"*, *"per"* are synonyms of the group by statement). While mapping of natural language queries to the Foodmart domain, we preserved the structure of the original queries (e.g., word order, typos, etc.). Overall, 75% of the queries in the dataset are valid GPSJ queries, confirming how general and standard GPSJ queries are. The filtered benchmark includes 110 queries. We consider token sub-sequences of maximum length $n = 4$ (i.e., the [1..4]-grams) as no entity in the KB is longer than 4 words. Each sub-sequence is associated to the top N similar entities with similarity at least higher than α such that at least a percentage β of the tokens in T is covered. The value of β is fixed to 70% based on an empirical evaluation of the benchmark queries. Table 4 summarizes the parameters considered in our approach.

5.1 Effectiveness

The effectiveness of our framework quantifies how well the translation of natural language queries meets user desiderata. Effectiveness is primarily evaluated as the parse tree similarity $TSim(PT, PT^*)$ between the parse tree PT produced by our system and the correct one PT^*, which is manually written by us for each query in the benchmark. Parse tree similarity is based on the tree distance [23] that keeps into account both the number of correctly parsed entities (i.e., the parse tree leaves) and the tree structure that codes the query structure (e.g., selection clauses *"(A and B) or C"* and *"A and (B or C)"* refers to the same parsed entities but underlie different structures). In this subsection, we evaluate how our framework behaves with(out) disambiguation.

Let's consider first the system behavior without disambiguation. Figure 8 depicts the performance of our approach with respect to variations in the number of retrieved top similar entities (i.e., $N \in \{2, 4, 6\}$). Values are reported for the top-k trees (i.e., the k trees with the highest score). We remind that only one parse forest is involved in the disambiguation phase; nonetheless, for testing purposes, it is interesting to see if best parse tree belongs

[5] A public dataset about food sales between 1997 and 1998 (https://github.com/julianhyde/foodmart-data-mysql).

Annotation type	Description	Action
Ambiguous Attribute	⟨Val⟩ is member of these attributes [...]	Pick attribute / drop clause
Ambiguous Agg. Operator	⟨Mea⟩ allows these operators [...]	Pick operator / drop clause
Attribute-Value Mismatch	⟨Attr⟩ and ⟨Val⟩ domains mismatch, possible value are [...]	Pick value / drop clause
MD-Meas Violation	⟨Mea⟩ does not allow ⟨Agg⟩, possible operators are [...]	Pick operator / drop clause
MD-GBY Violation	It is not allowed to group by on ⟨Attr⟩ without ⟨Attr⟩	Add attribute / drop clause
Unparsed ⟨GC⟩ clause	There is a dangling grouping clause ⟨GC⟩	Add clause to ⟨GPSJ⟩ / drop clause
Unparsed ⟨MC⟩ clause	There is a dangling measure clause ⟨MC⟩	Add clause to ⟨GPSJ⟩ / drop clause
Unparsed ⟨SC⟩ clause	There is a dangling predicate clause ⟨SC⟩	Add clause to ⟨GPSJ⟩ / drop clause

Table 3: Templates and actions required to solve misunderstandings in textual queries.

Symbol	Range	Meaning
N	$\{2, 4, 6\}$	Num. of top similar entities
α	$\{0.4, 0.5, 0.6\}$	Token/entity minimum similarity
β	70%	Sentence coverage threshold
n	4	Maximum sub-sequence length

Table 4: Parameter values for testing.

to the top-k ranked ones. Effectiveness slightly change varying N and it ranges in $[0.88, 0.91]$. Effectiveness is more affected by the entity/token similarity threshold α and ranges in $[0.83, 0.91]$. In both cases, the best results are obtained when more similar entities are admitted and more candidate mappings are generated. Independently of the chosen thresholds the system results very stable (i.e., the effectiveness variations are limited) and even considering only one query to be returned its effectiveness is at the state of the art [13, 18, 22]. This confirms that (1) the choice of proposing only one query to the user does not negatively impact on performances (while it positively impacts on interaction complexity and efficiency) and (2) our scoring function properly ranks parse tree similarity to the correct interpretation for the query since the best ranked is in most cases the most similar to the correct solution. As the previous tests do not include disambiguation, only 58 queries out of 110 are not ambiguous and produce parse trees that can be fed *as-is* to the generation and execution phases. This means that 52 queries — despite being very similar to the correct tree, as shown by the aforementioned results — are not directly executable without disambiguation (we recall the ambiguity/error types from Table 2). Indeed, of these 52 queries, 38 contains 1 ambiguity annotation, 12 contains two ambiguities annotations, and 2 contains three or more ambiguities annotations. Figure 10 depicts the performance when the best parse tree undergoes iterative disambiguation (i.e., an increasing number of correcting action is applied). Starting from the best configuration from the previous tests (for $N = 6$ and $\alpha = 0.4$), by applying an increasing number of correcting actions the effectiveness increases from 0.89 up to 0.94. Unsolved differences between PT and PT^* are mainly due to missed entities in the mappings.

Although our approach shows effectiveness comparable to state-of-art proposals [13, 18, 22], it was not possible to run a detailed comparison against them because the implementations are not available and the provided descriptions are far from making them reproducible. Furthermore, despite the availability of some natural language datasets, the latter are hardly compatible with GPSJ queries and are not based on real analytic workloads.

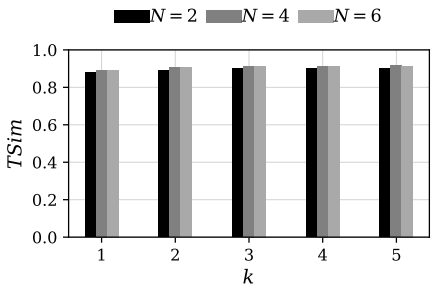


Figure 8: Effectiveness varying the number of retrieved top similar entities N and the number of $Top\text{-}k$ queries returned (with $\alpha = 0.4$).

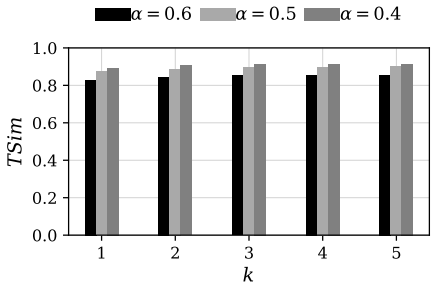


Figure 9: Effectiveness varying the similarity threshold α and the number of $Top\text{-}k$ queries returned (with $N = 6$).

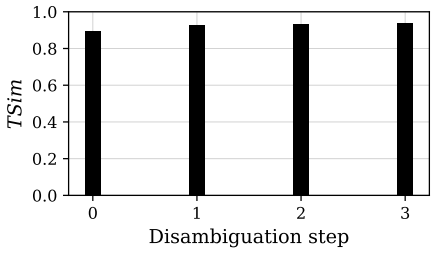


Figure 10: Effectiveness as a function of the number of disambiguation steps (i.e., application of correcting actions; with $k = 1$, $N = 6$ and $\alpha = 0.4$).

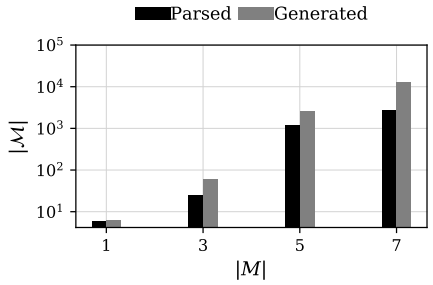


Figure 11: Number of "Generated" and "Parsed" mappings varying the number $|M|$ of entities in the optimal tree (with $N = 6$ and $\alpha = 0.4$).

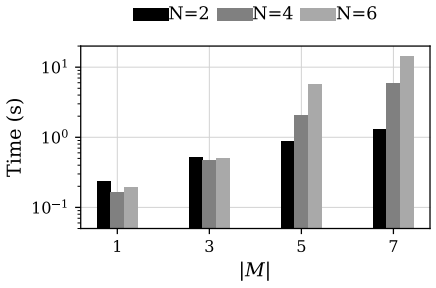


Figure 12: Execution time varying the number $|M|$ of entities in the optimal tree and the number of top similar entities N (with $\alpha = 0.4$).

5.2 Efficiency

We ran the tests on a machine equipped with Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz CPU and 8GB RAM, with the framework implemented in Java.

In Section 4.2 we have shown that the mapping search space increases exponentially in the text length. Figure 11 confirms this result showing the number of generated mappings as a function of the number of entities $|M|$ included in the optimal parse tree PT^*, $|M|$ is strictly related to $|T|$. Note that, with reference to the parameter values reported in Table 4, the configuration analyzed in Figure 11 is the worst case since it determines the largest search space due to the high number of admitted similar entities. Noticeably, pruning rules strongly limit the number of mappings to be actually parsed.

Figure 12 shows the average execution time by varying $|M|$ and the number of allowed top similar entities N. The execution time increase with the number of entities included in the optimal parse tree, as such also the number of top similar entities impacts the overall execution time. We recall that effectiveness remains high also for $N = 2$ corresponding to an execution time of 1 second, raising to 10^1 seconds in the worst case. We emphasize that the execution time corresponds to the time necessary for the translation, and not to the time to actually execute them. Queries are executed against the enterprise data mart and their performance clearly depends on the underlying multidimensional engine.

6 CONCLUSIONS AND FUTURE WORKS

In this paper, we proposed a conversational OLAP framework and an original approach to translate a text describing an OLAP query in a well-formed GPSJ query. Tests on a real case dataset have shown state-of-the-art performances. More in detail, we have shown that coupling a grammar-based text recognition approach with a disambiguation phase determines a 94% accuracy. We are now working in many different directions to complete the framework by: (a) supporting an OLAP navigation session rather than a single query, as this requires to extend the grammar and to have memories of previous queries in order to properly modify them; (b) designing a proper visual representation of the interpreted text to improve the system effectiveness, which can be obtained by defining a proper graphical metaphor of the query on top of a conceptual visual formalism such as DFM; (c) testing the whole framework on real users to verify its effectiveness from points of view that go beyond accuracy, e.g. usability, immediacy, memorability; (d) exploiting the Log to improve the disambiguation effectiveness and to learn synonyms not available a-priori.

REFERENCES

[1] [n. d.]. Hype Cycle for Artificial Intelligence, 2018. http://www.gartner.com/en/documents/3883863/hype-cycle-for-artificial-intelligence-2018. ([n. d.]). Accessed: 2019-06-21.

[2] Katrin Affolter, Kurt Stockinger, and Abraham Bernstein. 2019. A comparative survey of recent natural language interfaces for databases. *The VLDB Journal* 28, 5 (2019), 793–819.

[3] Julien Aligon, Enrico Gallinucci, Matteo Golfarelli, Patrick Marcel, and Stefano Rizzi. 2015. A collaborative filtering approach for recommending OLAP sessions. *Decision Support Syst.* 69 (2015), 20–30.

[4] John C. Beatty. 1982. On the relationship between LL(1) and LR(1) grammars. *J. ACM* 29, 4 (1982), 1007–1022.

[5] Lukas Blunschi, Claudio Jossen, Donald Kossmann, Magdalini Mori, and Kurt Stockinger. 2012. SODA: Generating SQL for Business Users. *PVLDB* 5, 10 (2012), 932–943.

[6] Kedar Dhamdhere, Kevin S. McCurley, Ralfi Nahmias, Mukund Sundararajan, and Qiqi Yan. 2017. Analyza: Exploring Data with Conversation. In *IUI*. ACM, 493–504.

[7] Krista Drushku, Julien Aligon, Nicolas Labroche, Patrick Marcel, and Verónika Peralta. 2019. Interest-based recommendations for business intelligence users. *Inf. Syst.* 86 (2019), 79–93.

[8] Lorena Etcheverry and Alejandro A. Vaisman. 2012. QB4OLAP: A Vocabulary for OLAP Cubes on the Semantic Web. In *COLD (CEUR Workshop Proceedings)*, Vol. 905. CEUR-WS.org.

[9] Matteo Francia, Matteo Golfarelli, and Stefano Rizzi. 2019. Augmented Business Intelligence. In *DOLAP (CEUR Workshop Proceedings)*, Vol. 2324. CEUR-WS.org.

[10] Ramanathan V. Guha, Vineet Gupta, Vivek Raghunathan, and Ramakrishnan Srikant. 2015. User Modeling for a Personal Assistant. In *WSDM*. ACM, 275–284.

[11] Ashish Gupta, Venky Harinarayan, and Dallan Quass. 1995. Aggregate-Query Processing in Data Warehousing Environments. In *VLDB*. Morgan Kaufmann, 358–369.

[12] Nicolas Kuchmann-Beauger, Falk Brauer, and Marie-Aude Aufaure. 2013. QUASL: A framework for question answering and its Application to business intelligence. In *RCIS*. IEEE, 1–12.

[13] Fei Li and H. V. Jagadish. 2016. Understanding Natural Language Queries over Relational Databases. *SIGMOD Record* 45, 1 (2016), 6–13.

[14] Gabriel Lyons, Vinh Tran, Carsten Binnig, Ugur Çetintemel, and Tim Kraska. 2016. Making the Case for Query-by-Voice with EchoQuery. In *SIGMOD Conference*. ACM, 2129–2132.

[15] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *ACL (System Demonstrations)*. The Association for Computer Linguistics, 55–60.

[16] George A. Miller. 1995. WordNet: A Lexical Database for English. *Commun. ACM* 38, 11 (1995), 39–41.

[17] Amrita Saha, Mitesh M. Khapra, and Karthik Sankaranarayanan. 2018. Towards Building Large Scale Multimodal Domain-Aware Conversation Systems. In *AAAI*. AAAI Press, 696–704.

[18] Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R. Mittal, and Fatma Özcan. 2016. ATHENA: An Ontology-Driven System for Natural Language Querying over Relational Data Stores. *PVLDB* 9, 12 (2016), 1209–1220.

[19] Jaydeep Sen, Fatma Ozcan, Abdul Quamar, Greg Stager, Ashish R. Mittal, Manasa Jammi, Chuan Lei, Diptikalyan Saha, and Karthik Sankaranarayanan.

2019. Natural Language Querying of Complex Business Intelligence Queries. In *SIGMOD Conference*. ACM, 1997–2000.

[20] Radwa El Shawi, Mohamed Maher, and Sherif Sakr. 2019. Automated Machine Learning: State-of-The-Art and Open Challenges. *CoRR* abs/1906.02287 (2019).

[21] Immanuel Trummer, Yicheng Wang, and Saketh Mahankali. 2019. A Holistic Approach for Query Evaluation and Result Vocalization in Voice-Based OLAP. In *Proc. of the 2019 Int. Conf. on Manage. of Data, SIGMOD Conf. 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. 936–953.

[22] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. SQLizer: query synthesis from natural language. *PACMPL* 1, OOPSLA (2017), 63:1–63:26.

[23] Kaizhong Zhang and Dennis E. Shasha. 1989. Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems. *SIAM J. Comput.* 18, 6 (1989), 1245–1262.