Conversational OLAP in Action

Matteo Francia
DISI — University of Bologna, Italy
m.francia@unibo.it

Enrico Gallinucci
DISI — University of Bologna, Italy
enrico.gallinucci@unibo.it

Matteo Golfarelli
DISI — University of Bologna, Italy
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 demonstration we present COOL, a tool supporting natural language *COnversational OLap* sessions. COOL interprets and translates a natural language dialogue into an OLAP session that starts with a GPSJ (Generalized Projection, Selection and Join) query. The interpretation relies on a formal grammar and a knowledge base storing metadata from a multidimensional cube. COOL is portable, robust, and requires minimal user intervention. It adopts an n-gram based model and a string similarity function to match known entities in the natural language description. In case of incomplete text description, COOL can obtain the correct query either through automatic inference or through interactions with the user to disambiguate the text. The goal of the demonstration is to let the audience evaluate the usability of COOL and its capabilities in assisting query formulation and ambiguity/error resolution.

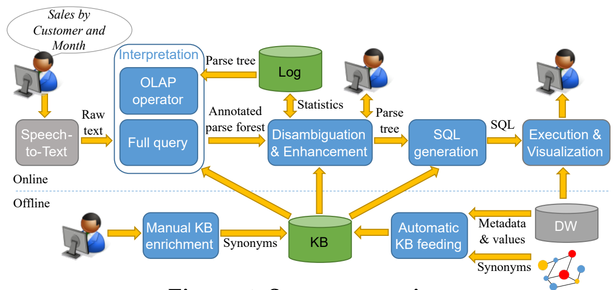


Figure 1: System overview.

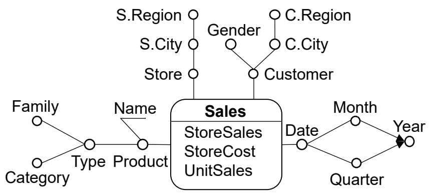


Figure 2: Sales cube in the DW.

1 INTRODUCTION

Following the spreading of analytic tools, a heterogeneous plethora of data scientists is accessing data. However, the gap between data scientists and analytic skills is growing since different types of data require to learn specialized metaphors and formal tools (e.g., SQL language to query relational data). Natural language interfaces are a promising bridge towards the democratization of data access [13]. Rather than demanding vertical skills in computer science and data architectures, natural language is a native "tool" to organize and provide meaningful questions/answers. Interfacing natural language processing (either written or spoken) to database systems opens to new opportunities for data exploration and querying [9]. Actually, in the area of data warehouse, OLAP (On-Line Analytical Processing) 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 is augmented business intelligence [6], where hand-free interfaces are mandatory.

In this demo paper, we propose COOL to convert natural language into *COnversational OLap* sessions composed of GPSJ queries and analytic operators. GPSJ [8] 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 [3], this is the first proposal addressing a full-fledged implementation for OLAP analytical sessions that is:

- *Automated and portable*: by reading metadata (e.g., hierarchy structures, measures, attributes, and aggregation operators) from a ROLAP engine, COOL automatically builds the minimal lexicon involved in the translation.
- *Robust* to user inaccuracies in syntax, OLAP terms, and attribute values, by exploiting metadata and implicit information.
- *Extendable* and easily configurable on a data warehouse (DW) without a heavy manual definition of the lexicon. The minimal lexicon is extendable by importing known ontologies in the Knowledge Base.

COOL's initial proposal in [4] has been now extended by (1) implementing a full-fledged application that supports a complete OLAP session rather than a single query; and (2) providing a visual metaphor based on the Dimensional Fact Model (DFM) [7] to guide user interaction (Figure 2 conceptualizes a multidimensional cube with the DFM formalism). Noticeably, the user interface's effectiveness has been assessed with 40 users, including data scientists and master students with varying skill levels.

The goal of the demonstration is to let the audience evaluate the usability of COOL and its capabilities in assisting query formulation and ambiguity/error resolution. The system is publicly available at https://big.csr.unibo.it/cool.

2 SYSTEM OVERVIEW

Figure 1 sketches a functional view of the architecture. Given a set of multidimensional cubes (DW), we distinguish between an offline phase (to initialize and configure the system) and an online phase (to enable the user interaction). We refer the user to [7] for an explanation of the DW terminology.

2.1 The offline phase

The *offline* phase automatically extracts the set of entities $\mathcal{E}$, i.e., the DW-specific terms used to express the queries. Such information is stored in the knowledge base (`KB`), which relies on the DFM expressiveness [7]. This phase runs *only* when the `DW` undergoes modification (e.g., cube schemas or data instances) and extracts all multidimensional metadata. A cube modeled as a *star schema* in a ROLAP engine consists of dimension tables (DTs: the cube hierarchies) and a fact table (FT: the cube). The `Automatic KB feeding` extracts measures (FT columns not included in the primary key), attributes (DT columns), values (distinct instances of the DT columns), and hierarchies (either coded in a specific table or inferred [10]). These elements represent the lexicon necessary to translate natural language into conversational sessions. COOL supports lexicon extension with external synonyms that can be automatically imported from open data ontologies (e.g., Wordnet [11]) to widen the understood language. Besides the domain-specific terminology, the `KB` also includes the set of standard terms that are domain independent and that do not require any feeding (e.g., group by, where, select). Further enrichment can be optionally carried out manually when the application domain involves a non-standard vocabulary (e.g., the physical names of tables and columns do not match a standard vocabulary).

2.2 The online phase

The *online* phase runs every time a natural language query is issued. In a hand-free scenario (e.g., [5]), the spoken query is initially translated to text by the `Speech-to-text` software module. Since this task is out of our research scope, we exploited the public Web Speech API (https://wicg.github.io/speech-api/). The uninterpreted text is then analyzed by the `Interpretation` step, refined in the `Disambiguation & Enhancement` step, translated by the `SQL generation` step, and finally executed and visualized by the `Execution & Visualization` step.

2.2.1 Interpretation. `Interpretation` consists of two alternative steps. `Full query` interprets the texts describing full queries (which happens when an analytic session starts). `OLAP operator` modifies the latest query when the user states an OLAP operator along a session (i.e., roll-up, drill-down, and slice&dice). The switch between the two steps to manage the conversation (i.e., a dialog between the user and COOL) is modeled by two states: *engage* and *navigate*.

- *Engage*: this is the initial state, in which the system expects a full query to be issued and whose interpretation is demanded to `Full query`. When COOL achieves a successful interpretation (i.e., it is able to run the query) it switches to the *navigate* state.
- *Navigate*: the dialogue evolves by iteratively applying OLAP operators that refine the query (e.g., by aggregating by different levels or narrowing the query selectivity). The management of these steps is demanded to `OLAP operator` until the user resets the session, making COOL return to the *engage* state.

`Full query` and `OLAP operator` follow these steps: (i) `Tokenization & Mapping`, (ii) `Parsing`, and (iii) `Checking & Annotation`.

Tokenization & Mapping. A raw text T is a sequence of single words $T = \langle t_1, ..., t_z \rangle$. The goal of this step is to identify the entities in T, i.e., the only elements that will be involved in the `Parsing` step. Turning a text into a sequence of entities means finding a *mapping* between words in T and $\mathcal{E}$.

Definition 2.1 (Mapping & Mapping function). A mapping function $M(T)$ is a partial function that associates sub-sequences (or *n-grams*)[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;
- a sub-sequence $T' = \langle t_i, ..., t_l \rangle \in T$ (with $|T'| \leq n$) is associated to an entity E if and only if $Sim(T', E) > \alpha$ (where $Sim()$ is a similarity function, later defined) and $E \in TopN(\mathcal{E}, T')$ (where $TopN(\mathcal{E}, T')$ is the set of N entities in $\mathcal{E}$ that are the most similar to T' according to $Sim(T', E)$).

The output of a mapping function is a sequence $M = \langle E_1, ..., E_l \rangle$ on $\mathcal{E}$ that we call a *mapping*.

The similarity function $Sim()$ is based on the Levenshtein distance and keeps token permutation into account to make similarity robust to token permutations (e.g., sub-sequences $\langle P., Edgar \rangle$ and $\langle Edgar, Allan, Poe \rangle$ must result similar).

Several mappings might exist between T and $\mathcal{E}$ since Definition 2.1 admits sub-sequences of variable lengths (corresponding to different partitionings of T) and associates the top similar entities to each sub-sequence. This increases the interpretation robustness since COOL chooses the best mapping through a scoring function. Given a mapping $M = \langle E_1, ..., E_m \rangle$, its score $Score(M)$ (i.e., the sum of entity similarities) is higher when M includes several entities with high similarity values. Intuitively, the higher the mapping score, the higher the probability to determine an optimal interpretation.

Parsing. Parsing validates the syntactical structure of a mapping against a formal grammar and outputs a data structure called *parse tree* that is later used to translate a mapping into SQL.

In `Full query`, `Parsing` is responsible for the interpretation of a complete GPSJ query stated in natural language. A GPSJ query contains a measure clause (MC) and optional group-by (GC) and selection (SC) clauses. Parsing a full query means searching in a mapping the complex syntax structures (i.e., clauses) that build-up the query. Given a mapping M, the output of the parser is a parse tree PT_M, i.e., an ordered tree that represents the syntactic structure of a mapping according to the grammar from Figure 3. To the aim of parsing, entities are terminal elements in the grammar.

In `OLAP operator`, `Parsing` is responsible for searching the syntactic structures of the OLAP operators that build-up the conversation. The grammar is described in Figure 4. Our conversation steps are inspired to well-known OLAP visual interfaces (e.g., Tableau[2]). To apply an OLAP operator, COOL must be in the state *navigate*, i.e., a full GPSJ query has been already successfully interpreted and translated into a parse tree PT_C that acts as a context for the operator. The output of the parser is a parse tree PT_M that is used to update PT_C (see Section 2.2.3).

Both GPSJ grammars are LL(1)[3] [2], not ambiguous (i.e., each mapping admits, at most, a single parse tree PT_M), and can be parsed by an LL(1) parser with linear complexity [2]. If the input mapping M is fully parsed, PT_M includes all the entities as leaves. Conversely, if only a portion of the input belongs to the grammar, an LL(1) parser produces a partial parsing, meaning that it returns a parse tree including the portion of the input mapping

[1]The term *n*-gram is used as a synonym of sub-sequence in the area of text mining.
[2]https://www.tableau.com/
[3]The rules presented in Figure 3 do not satisfy LL(1) constraints for readability reasons. It is easy to turn such rules in an 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⟩ | ⟨Fct⟩⟨Cnt⟩
| ⟨Cnt⟩⟨Attr⟩ | ⟨Attr⟩⟨Cnt⟩)+

⟨GC⟩ ::= "*group by*" ⟨Attr⟩+

⟨SC⟩ ::= "*where*" ⟨SCA⟩

⟨SCA⟩ ::= ⟨SCN⟩ "*and*" ⟨SCA⟩ | ⟨SCN⟩

⟨SCN⟩ ::= "*not*" ⟨SSC⟩ | ⟨SSC⟩

⟨SSC⟩ ::= ⟨Attr⟩⟨Cop⟩⟨Val⟩ | ⟨Attr⟩⟨Val⟩ | ⟨Val⟩⟨Cop⟩⟨Attr⟩ | ⟨Val⟩⟨Attr⟩ | ⟨Val⟩

⟨Cop⟩ ::= " = " | " <> " | " > " | " < " | " ≥ " | " ≤ "

⟨Agg⟩ ::= "*sum*" | "*avg*" | "*min*" | "*max*" | "*stdev*"

⟨Cnt⟩ ::= "*count*" | "*count distinct*"

⟨Fct⟩ ::= *Domain-specific facts*

⟨Mea⟩ ::= *Domain-specific measures*

⟨Attr⟩ ::= *Domain-specific attributes*

⟨Val⟩ ::= *Domain-specific values*

Figure 3: Backus-Naur representation of the `Full query` grammar. Entities from the `KB` are terminal symbols.

⟨OPERATOR⟩ ::= ⟨DRILL⟩ | ⟨ROLLUP⟩ | ⟨SAD⟩ | ⟨ADD⟩ | ⟨DROP⟩ | ⟨REPLACE⟩

⟨DRILL⟩ ::= "*drill*" ⟨Attr⟩$_{from}$ "*to*" ⟨Attr⟩$_{to}$ | "*drill*" ⟨Attr⟩

⟨ROLLUP⟩ ::= "*rollup*" ⟨Attr⟩$_{from}$ "*to*" ⟨Attr⟩$_{to}$ | "*rollup*" ⟨Attr⟩

⟨SAD⟩ ::= "*slice*" ⟨SSC⟩

⟨ADD⟩ ::= "*add*" (⟨MC⟩ | ⟨Attr⟩ | ⟨SSC⟩)

⟨DROP⟩ ::= "*drop*" (⟨MC⟩ | ⟨Attr⟩ | ⟨SSC⟩)

⟨REPLACE⟩ ::= "*replace*" (⟨MC⟩$_{old}$ "*with*" ⟨MC⟩$_{new}$ | ⟨Attr⟩$_{old}$ "*with*" ⟨Attr⟩$_{new}$
| ⟨SSC⟩$_{old}$ "*with*" ⟨SSC⟩$_{new}$)

Figure 4: Backus-Naur representation of the `OLAP operator` grammar. We omit ⟨MC⟩, ⟨Attr⟩, ⟨SSC⟩ in common with Figure 3.

that belongs to the grammar (i.e., the *PT* rooted in ⟨GPSJ⟩). The remaining entities can be either singleton or complex clauses that could not be connected to the main parse tree. We will call *parse forest* PF_M the union of the parse 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 in the `Disambiguation & Enhancement` step. Henceforth, we refer to the parser's output as a parse forest independently of the presence of residual clauses.

2.2.2 Disambiguation & enhancement. Due to natural language ambiguities, speech-to-text inaccuracies, and wrong query formulations, parts of the text can be misunderstood. The reasons behind the misunderstandings are manifold, including (but not limited to) a wrong usage of aggregation operators (e.g., summing non-additive measures), inconsistencies between attributes and values in selection predicates (e.g., filtering on product *"New York"*), or grouping by a descriptive attribute. Such parts of the parse forest are annotated as ambiguities. The `Disambiguation & Enhancement` step solves ambiguities (if any) automatically whenever possible (by exploiting implicit information) or by asking appropriate questions to the user. Through disambiguation, the parse forest PF_M is reduced to a single parse tree PT_M.

2.2.3 SQL generation. `SQL generation` translates a full-query parse tree into an executable SQL query. If an OLAP operator has been submitted, the context parse tree PT_C must be updated according to the OLAP operator parse tree PT_M. All the OLAP operators can be implemented atop the addition/removal of new/existing nodes in PT_C. We apply a depth-first search algorithm to retrieve the clauses interested by the OLAP operator. We recall that adding a new clause to ⟨GPSJ⟩ (e.g., "add city" requires to add the attribute City) requires to append the new clause to the existing ⟨MC⟩/⟨GC⟩/⟨SC⟩ (and to create it if it does not exist in ⟨GPSJ⟩). Given a full query parse tree PT_M, the generation of its corresponding SQL requires to fill in the `SELECT`, `WHERE`, `GROUP BY` and `FROM` statements. The SQL generation applies to both star and snowflake schemas [7] and is done as follows:

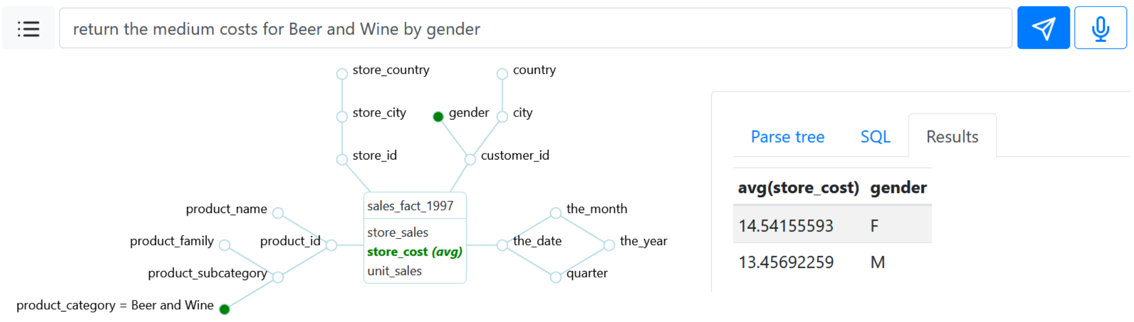
- `SELECT`: measures and aggregation operators from ⟨MC⟩ and attributes in the group by clause ⟨GC⟩;
- `WHERE`: predicates from the selection clause ⟨SC⟩;
- `GROUP BY`: attributes from the group by clause ⟨GC⟩;
- `FROM`: measures, attributes, and values identify fact and dimension tables. The join path is identified by following the referential integrity constraints.

3 VISUALIZATION METAPHOR

The obtained query is run on the `DW` and the results are reported to the user by the `Execution & Visualization` software module. The visual interaction relies on the DFM [7] (Figures 5 and 6), which natively provides a graphical representation for multidimensional cubes and queries: such representation is conceptual and user-oriented, and its effectiveness is confirmed by its adoption in commercial tools (e.g., https://www.indyco.com/) for both modeling and descriptive purposes. With reference to Figure 2, the DFM explicitly represents the cube as a rectangle (i.e., Sales) including the measure names (e.g., StoreSales) surrounded by hierarchies organized in many-to-one acyclic graphs (e.g., Products are grouped in Types). This representation includes all the elements necessary to formulate a GPSJ query, namely measure clauses, group-by clauses, and selection clauses on both measures and dimensional attributes.

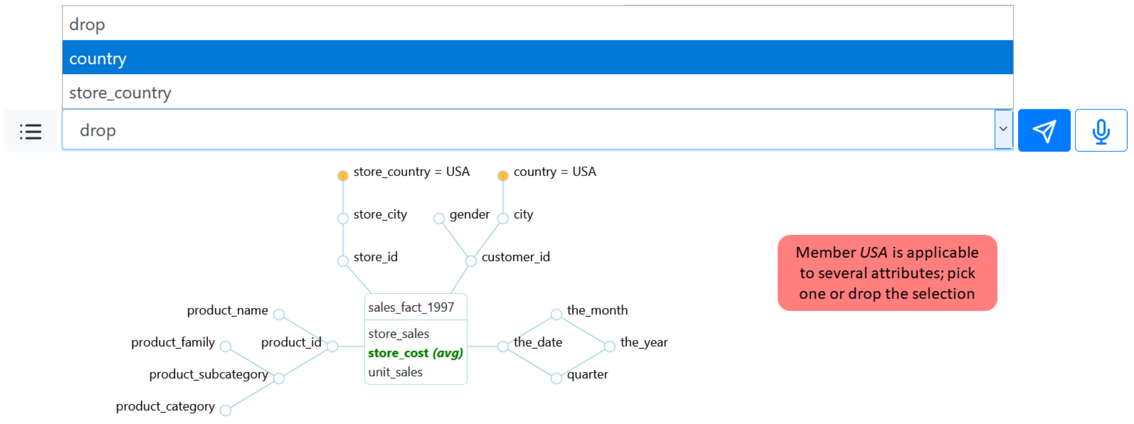
At first, users are asked to submit the natural language description of a full query (e.g., *"return the medium costs for Beer and Wine by gender"*). Once COOL interprets the natural language query, it shows in green the entities that are fully-understood (`product_category=Beer and Wine`, `store_cost(avg)`, and `gender` in Figure 5) and, if no ambiguities exist, it also returns the query result. If some ambiguities exist, COOL shows in yellow the ambiguous elements on the DFM one by one. For instance, when users ask for *"average costs in the USA"* (Figure 6), COOL shows that `store_cost(avg)` is correctly understood, while `USA` is ambiguous since it belongs to both attributes `store_country` and `country`. In the case of ambiguities, besides color codes, COOL notifies the user of the encountered ambiguity and enables multiple resolutions (e.g., either picking the correct attribute or dropping the clause). COOL also shows the parse tree on request, enabling more skilled users to understand how natural language is interpreted. After issuing a full query, the conversational session proceeds by describing further analytic steps (e.g., *"generalize product subcategory to category"* or more briefly *"generalize product subcategory"*).

We tested COOL against the Foodmart [1] cube with 40 users, mainly master students in data science, with basic or advanced knowledge of business intelligence and data warehousing. On a scale from 1 (very poor) to 5 (very high), on average, users scored 3.60 ± 0.7 their familiarity with the English language, and 3.28 ± 1.1 their familiarity with the OLAP paradigm. Users evaluated the responsiveness and user-friendliness of COOL as 4.09 ± 0.85, and the overall user experience (e.g., the perceived translation accuracy) as 3.82 ± 0.91, confirming a good — or even


return the medium costs for Beer and Wine by gender

avg(store_cost)	gender
14.54155593	F
13.45692259	M

Figure 5: COOL returns the results for a fully-interpreted query.


drop

Member *USA* is applicable to several attributes; pick one or drop the selection

Figure 6: COOL shows a hint to disambiguate the *member* "USA".

optimal — experience. Finally, the average response time is less than 1 second for mappings including up to 9 entities.

4 DEMO PROPOSAL

In the demonstration, we develop an experience to showcase the translation of natural language dialogues into analytical sessions. In particular, we consider the usability criteria of a visual interface from [12]. Usability is assessed by *learnability* (how easily novel users accomplish basic tasks), *efficiency* (how quickly users can perform tasks), *error recovery* (how many errors users make and how easily can they recover from these errors), and *satisfaction* (how pleasant it is to use the interface). Two distinct scenarios are proposed to assess these functionalities.

In the *guided* scenario, we drive the formulation of analytic sessions on the Sales cube. In particular, we provide users with three formal definitions of GPSJ queries in the format $GPSJ = \{\{GC\}, \{SC\}, \{MC\}\}$; users interact with the system by formulating such queries in natural language. Once the query description is submitted (spoken or written), COOL drives users in the resolutions of ambiguities, if any, through a question-answer approach. Then, further analytics steps are shown to the user, who is required to abstract and describe the operators necessary to accomplish such steps. In such a way, we address *learnability* and *efficiency*, allowing users to approach COOL in a user-friendly manner.

In the following *unguided* scenario, users are to freely interact with COOL starting only with a generic analytic task (e.g., *"Investigate sales drop. This might depend on examining products sales over time"*). This requires to formulate custom analytic goals and sessions toward such goals (e.g., sales drop for the category "Beer and Wine" might be caused by a drop in "Wine" sales). As we expect users to encounter ambiguities while freely navigating the cube (e.g., in case of homonyms and synonyms), we address the robustness of COOL through its *error recovery* capability. Also, we assess user *satisfaction* by understanding how easy it is for the user to accomplish their analytic goal (i.e., how many steps it takes).

REFERENCES

[1] [n.d.]. Foodmart. https://github.com/julianhyde/foodmart-data-mysql. Accessed: 18/01/2021.
[2] John C. Beatty. 1982. On the relationship between LL(1) and LR(1) grammars. *J. ACM* 29, 4 (1982), 1007–1022.
[3] Kedar Dhamdhere, Kevin S. McCurley, Ralfi Nahmias, Mukund Sundararajan, and Qiqi Yan. 2017. Analyza: Exploring Data with Conversation. In *Proc. IUI*. ACM, New York, NY, USA, 493–504.
[4] Matteo Francia, Enrico Gallinucci, and Matteo Golfarelli. 2020. Towards Conversational OLAP. In *Proc. DOLAP@EDBT/ICDT (CEUR Workshop Proceedings)*, Vol. 2572. CEUR-WS.org, Copenhagen, Denmark, 6–15.
[5] Matteo Francia, Matteo Golfarelli, and Stefano Rizzi. 2019. Augmented Business Intelligence. In *Proc. DOLAP@EDBT/ICDT (CEUR Workshop Proceedings)*, Vol. 2324. CEUR-WS.org, Lisbon, Portugal, 1–10.
[6] Matteo Francia, Matteo Golfarelli, and Stefano Rizzi. 2020. A-BI$^+$: A framework for Augmented Business Intelligence. *Inf. Syst.* 92 (2020), 101520.
[7] Matteo Golfarelli, Dario Maio, and Stefano Rizzi. 1998. The Dimensional Fact Model: A Conceptual Model for Data Warehouses. *Int. J. Cooperative Inf. Syst.* 7, 2-3 (1998), 215–247.
[8] Ashish Gupta, Venky Harinarayan, and Dallan Quass. 1995. Aggregate-Query Processing in Data Warehousing Environments. In *Proc. VLDB*. Morgan Kaufmann, San Francisco, CA, USA, 358–369.
[9] Fei Li and H. V. Jagadish. 2016. Understanding Natural Language Queries over Relational Databases. *SIGMOD Record* 45, 1 (2016), 6–13.
[10] Heikki Mannila and Kari-Jouko Räihä. 1994. Algorithms for Inferring Functional Dependencies from Relations. *Data Knowl. Eng.* 12, 1 (1994), 83–99.
[11] George A. Miller. 1995. WordNet: A Lexical Database for English. *Commun. ACM* 38, 11 (1995), 39–41.
[12] Jakob Nielsen. 1993. *Usability engineering*. Academic Press.
[13] Yu Su. 2018. *Towards Democratizing Data Science with Natural Language Interfaces*. Ph.D. Dissertation. UC Santa Barbara.