# DatAgent: The Imminent Age of Intelligent Data Assistants

Antonis Mandamadiotis, Stavroula Eleftherakis, Apostolos Glenis
Dimitrios Skoutas, Yannis Stavrakas, Georgia Koutrika
Athena Research Center
Athens, Greece
{antonismand,seleftheraki,aglenis,dskoutas,ys,georgia}@athenarc.gr

## ABSTRACT

In this demonstration, we present *DatAgent*, an intelligent data assistant system that allows users to ask queries in natural language, and can respond in natural language as well. Moreover, the system actively guides the user using different types of recommendations and hints, and learns from user actions. We will demonstrate different exploration scenarios that show how the system and the user engage in a human-like interaction inspired by the interaction paradigm of chatbots and virtual assistants.

**PVLDB Reference Format:**
Antonis Mandamadiotis, Stavroula Eleftherakis, Apostolos Glenis Dimitrios Skoutas, Yannis Stavrakas, Georgia Koutrika. DatAgent: The Imminent Age of Intelligent Data Assistants. PVLDB, 14(12): 2815 - 2818, 2021.
doi:10.14778/3476311.3476352

## 1 INTRODUCTION

For many, data is considered the $21^{st}$ century's most valuable commodity. Analysts exploring data sets for insight, scientists looking for patterns, and consumers looking for information are just a few examples of user groups that need to dig into data. At the same time, data querying is often a non-trivial and time-consuming process due to users' unfamiliarity with the database contents as well as with query languages such as SQL. Data exploration tools are falling behind in bridging the gap between data and users, making data exploration intended only for the few.

Recent years have witnessed the rise of *human-like interaction tools*, i.e., chatbots and virtual assistants. Chatbots are automated programs that interact with humans via textual or auditory means. They are typically programmed to reply to a limited set of questions or statements. A virtual assistant is a digital software-based agent that, similarly to a personal human assistant, assists users in daily activities like setting clock alarms, scheduling an appointment, and so on. Following the trend for human-like interaction, *the need for human-like data exploration has naturally emerged*. A first wave of change has been brought by natural language interfaces to data (such as industrial ThoughtSpot [9] as well as academic Precis [8], NaLIR [5], and HydraNet [6]) that enable users to query data using natural language instead of SQL. Recent efforts include Amazon QuickSight Q[1], DataChat[2] and Google BigQuery Data QnA[3]. The latter is based on Analyza [1] and returns an English interpretation and the SQL query with the answer for a user NL query.

Inspired by chatbots and virtual assistants, we envision a new generation of human-like data exploration tools, called intelligent data assistants. An *Intelligent Data Assistant*:
(*a*) *converses* with the user in a more natural bilateral interaction,
(*b*) *actively guides* the user through recommendations,
(*c*) *keeps track of the context* and can respond and adapt accordingly,
(*d*) constantly *learns* and improves its behavior.

In this demonstration, we present *DatAgent*, our intelligent data assistant system. DatAgent allows users to ask queries in natural language, can talk back in natural language as well, provides different types of recommendations at various steps taking into account the current context, and learns from user actions. For enabling natural language queries, we are currently leveraging NaLIR [5].

**Challenges**. In DatAgent, we are addressing the following challenges: (*a*) how to enable the system to talk back in natural language, (*b*) how to enable active guidance (*c*) how to determine which actions are relevant given the current exploration context. Furthermore, at the user interface, the challenge is to come up with a design that empowers human-like interactions, departing from the classical form-based interfaces or dashboards.

**Contributions**. DatAgent is the *first intelligent data assistant*. It combines technologies at the intersection of data management, natural language processing (natural language queries and natural language generation), machine learning, query processing, and probability theory (multi-armed bandits). It *contributes novel technology* for *natural language explanations* as well as *novel and diverse query recommendations based on data analysis and user feedback*. Finally, it comes with *an intuitive user interface* that enables a conversational interaction with the user.

**Demonstration**. We will demonstrate different exploration scenarios that show how the system and the user engage in a human-like way to find the data the user is looking for. We will use two real-life data sets: (a) CORDIS[4] containing information about projects funded by the European Union under the Horizon 2020 framework programme, and (b) SDSS[5] containing astrophysics data.

## 2 SYSTEM OVERVIEW

To enable users to ask queries in natural language, we are currently employing NaLIR [5]. NaLIR parses a NL query using the Stanford


Proceedings of the VLDB Endowment, Vol. 14, No. 12 ISSN 2150-8097.
doi:10.14778/3476311.3476352

[1] https://aws.amazon.com/quicksight/q/
[2] https://datachat.ai
[3] https://cloud.google.com/blog/products/data-analytics/introducing-data-qna
[4] https://data.europa.eu/euodp/en/data/dataset/cordisH2020projects
[5] http://skyserver.sdss.org/dr16/en/home.aspx

parser to understand its grammatical structure and maps the various query parts to database elements and SQL query elements.

To enable the system to talk back, we build on our work called Logos [4], which allows generating NL explanations for SQL queries. DatAgent's explanation capabilities are considerably improved in two directions: *query coverage* and *translation quality*. We give an overview of the core concepts and functionality and explain our novel extensions in Section 2.1.

Providing guidance to the user is a challenging problem. What type of guidance could work is shaped by the exploration context: the user, the exploration goal, the domain, the user actions. Recognizing this, we are equipping the system with different recommendation capabilities as described in Sections 2.2 and 2.3.

## 2.1 Natural Language Explanations

**Database and query modeling.** Logos offers textual explanations of SQL queries using a directed graph model, the *database graph*, describing the underlying database. The database graph captures the relationships between the database relations and attributes (nodes of the graph). Its edges are divided into three types: (a) membership edges (attributes to relations) represent attribute projections in queries, (b) selection edges (relations to attributes) represent attributes in predicates, and (c) join edges (attribute to attribute) represent joins. For a given query, Logos detects the part of the database graph to which the query refers to, and (if necessary) it enriches it with additional nodes (e.g., for functions) and edges (e.g., for group-by's) in order to capture the query meaning. The produced graph is called *query graph*.

**Templates and labels.** Natural language explanations are created by traversing the query graph accompanied with a template mechanism [7]. The template mechanism uses labels provided either by the database designer or by the system itself to annotate the query graph. Those created by the designer are stored in special tables in the database called *designer tables*. Template labels are either node- or path-related. A *template label* $l(u)$ or $l((u,v))$ is assigned to a node $u$ or a path $(u,v)$, respectively. For instance, join template labels have the form $l((u,v)) = l(u) + l(u,v) + l(v)$, where the default label $l(u,v)$ by the system is "associated with". However, one may want to change it for a specific pair of tables. For example, for the (people, projects) pair of relations in CORDIS, one could specify that their connecting label is "principal investigators of".

**Novel extensions.** Compared to the earlier Logos, our novel extensions in DatAgent are: (a) grammar-related ones (query parsing and analysis) to cover more query types, and (b) translation-related ones to improve the quality of NL explanations.

In terms of the first category, parser changes have been made for the system to understand (lexical analysis) and also analyze (syntactic analysis) queries containing `SELECT top`, `LIMIT`, `(NOT) IN`, and `(NOT) LIKE` clauses. The parser was further developed in order to generate query-graph parts corresponding to queries with the aforementioned clauses.

Translation-related extensions include the ability of translating lists of values, disjoint queries, the `COUNT(*)` function form, as well as that of translating all of the above mentioned new clauses. Moreover, a mini dictionary has been developed providing the translation process with the plural form of all the attributes and relations. Lastly, bridging tables are excluded from the translation procedure leading to more natural translations.

To depict some of the new capabilities, consider the SQL query:

```
SELECT COUNT(p.title), t.title FROM topics t, project_topics pt, projects p
WHERE t.code = pt.topic AND pt.project = p.unics_id GROUP BY t.title;
```

/* DatAgent explanation */

*Find the titles of topics and the number of projects on these topics grouped by the titles of topics.*

/* Logos explanation */

*Create groups according to the title of topics. Find the number of projects and the title of topics associated with project topics associated with the projects.*

In DatAgent's explanation, the "title" attribute is in plural, the bridging table "project_topics" is excluded from the translation, and the translation of the `GROUP BY` clause is blended in the explanation.

## 2.2 Query Completion Recommendations

The user can start with a query and get recommendations on how to refine the query. Essentially, DatAgent offers recommendations on how to augment the query by adding/altering its `WHERE` clause, each one focusing on some part of the results of the initial query. The user can choose a recommendation and repeat this process until reaching the desired data. In this way, users can progressively zoom in on a part of the data that interests them.

Query recommendations are generated by leveraging attribute correlations and clusters in the data. This approach is based on the observation that people better understand patterns in low-dimensional spaces, and offers recommendations from the data perspective that require no query logs. We integrate our work called PyExplore [2]. Recommendations are generated as follows.

**Finding interesting attributes.** Correlations highlight relationships between attributes of the data set. For example, the month-of-the-year is correlated with the average daily temperature. We compute the correlation of each pair of attributes in the query results. We use *Pearson correlation* for comparison between numerical attributes, *Cramer's V* [6] for categorical ones, and *correlation ratio* for categorical-numerical. To make all correlation metrics in the same range, i.e., [0, 1], we take the absolute value of Pearson Correlation. *Then*, the inverse of the absolute value of the correlation matrix is used as a distance matrix, which is given as input to a clustering algorithm that creates clusters of correlated attributes. In DatAgent, we use *hierarchical clustering with complete linkage* [7], which takes as input the maximum number of attributes per cluster and decides the number of clusters accordingly.

**Generating top-k queries**. For each subset of correlated attributes, the initial query results are clustered using the values of the attributes in the subset. To handle categorical data efficiently, instead of encoding categorical values as dummy variables, we use $k$-modes as extended in [3], which allows clustering objects described by mixed numeric and categorical attributes. The parameter $k$, i.e., the number of clusters, can be specified by the user. We opt for a small number so that few queries will be generated.

[6] Harald Cramér. 1999. Mathematical methods of statistics. Vol. 43. Princeton University Press.

[7] Mahamed Omran, Andries Engelbrecht, and Ayed Salman. 2007. An overview of clustering methods. Intelligent Data Analysis. 11. 583-605. 10.3233/IDA-2007-11602.

Then, for each subset, the resulting cluster labels are fed into a decision tree classifier to produce the split points of the data. The resulting split points are used to create the recommended SQL queries. This is done by traversing the decision tree from the leaves up to the root, and for each path from the starting leaf to the root, the conditions of the `WHERE` clause to be added in the original query are built. These conditions correspond to the cluster boundaries as described by the path in the decision tree.

## 2.3 Identifier Recommendations

In order to assist users in the query formulation process, DatAgent provides *identifier recommendations* (relations and attributes) leveraging a multi-armed bandit approach. The multi-armed bandit problem is a classic reinforcement learning problem where given a fixed number of actions, the algorithm must choose the optimal ones in a way that the expected gain is maximized. Each action has its own probability distribution of success, which the algorithm tries to estimate using its strategy, by selecting each time an action and receiving a reward. The main problem that the algorithm is trying to solve is the dilemma of exploiting the current best known action or exploring other actions to gain more information. In machine learning, this trade-off is known as the exploration vs exploitation dilemma. Bandit algorithms do not require any training data, allowing for continuous learning, making them very popular for recommendation problems.

**Multi-armed bandit algorithm**. The problem of identifier recommendation can be modeled as a multi-armed bandit problem, *which is a novel approach to SQL query recommendations.* The bandit recommends a list of identifiers from a pool of candidates, receives a reward that depicts the user's satisfaction and updates its strategy based on the observed rewards. The goal is to maximize the total reward by choosing the most optimal identifiers, balancing between exploration and exploitation. We focus on SPJ (select-project-join) queries and recommend tables and columns. However, our algorithm can be extended to any SQL query. We use one bandit instance for each clause that we need recommendations for. This divides our problem into smaller sub-problems. In the current version of DatAgent, we focus on: (a) table recommendations (for the FROM clause), and (b) attribute recommendation (for the SELECT clause). We will see in Section 4 how these nicely complement the query completion recommendations for building queries.

**Generating top-k identifier recommendations**. Our system works in a iterative way, based on the feedback received from the user. First, the user receives $k$ table recommendations, and selects one or more tables from the recommendations or types the desired tables manually. Then, the user receives $k$ attribute recommendations for the previously selected tables. The user choices are sent to the system as feedback for the reward of each bandit.

The algorithm generating the recommendations is the *UCB algorithm* (Upper Confidence Bound), which follows the principle of optimism in the face of uncertainty. It uses the average perceived reward of the identifiers, as well as a confidence interval, so that uncertainty is also taken into account when choosing an action. The UCB algorithm selects the items that maximize the following function: $q_a + \sqrt{\frac{c\ ln(t)}{n_a}}$, where $t$ is the current timestamp, $n_a$ is the number of times arm (identifier) $a$ was selected, and $q_a$ is the average reward received for this arm. The first term $q_a$ favors exploitation, while the second term $\sqrt{\frac{c\ ln(t)}{n_a}}$ favors exploration driven by uncertainty. Instead of choosing only the best action at a time, we return the top-$k$ ones, i.e., the ones with the highest UCB values. The reward for the algorithm is a vector $\vec{r_t} \in [0,1]^k$, with the $i^{th}$ value indicating whether the user selected the $i^{th}$ identifier from the recommendation list. This reward is observed once the user has finished choosing the most interesting identifiers.

# 3 USER INTERFACE

The challenge of the user interface is to come up with a design and interaction that empowers human-like interactions. In that vein, we designed our data assistant as follows:

The main user interface (Figure 1) resembles a chat layout, where the user can type a (natural language or SQL) query in the input box and send it to DatAgent ①. The user can switch the target database of a query using the select box located next to the input box ②. The interaction history is maintained and displayed to the user. The user inputs as well as the responses received from DatAgent are shown in the upper part of the interface in the form of a dialog ③. The current state of the dialog between the user and DatAgent defines *a context* which allows us to provide context-dependent actions at each step. In that manner, we help the user interact more effortlessly with our system. These hints are displayed above the user input as blue buttons ④. Observe Figures 1, 2, and 4 for some examples of context-sensitive hints.

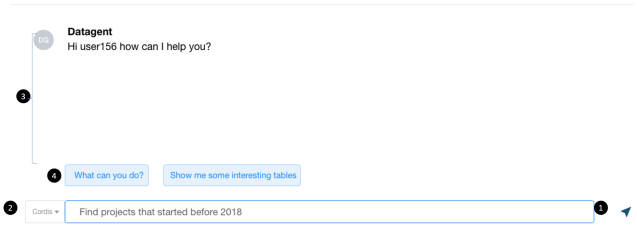


**Figure 1: Main User Interface**

Our system's responses are not in simple text format, instead they may contain SQL code or tabular data. For this reason, we specifically designed three useful components:

- A code block that contains SQL queries with their equivalent NL explanation. For each code block, the user has three options: (a) to execute the query, (b) to edit and execute the new query, and (c) to ask for query recommendations (Figure 2).
- A table containing the results for an executed query (Figure 2).
- An interactive query building tool where recommendations are shown in the form of a button or a menu, and the corresponding query is automatically built in the code block (Figure 4).

# 4 DEMONSTRATION

We will use two databases, CORDIS and SDSS, and we will demonstrate different use cases and show how the user can explore the data in each case with the help of DatAgent. Below, we describe some demonstration examples. DatAgent is highly interactive and

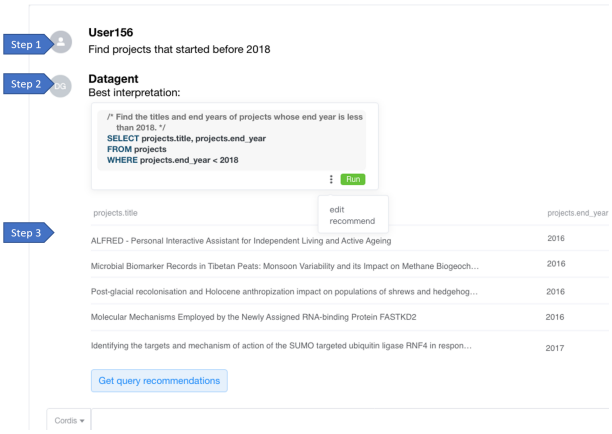


**Figure 2: NL-to-SQL, SQL-to-NL, and results**

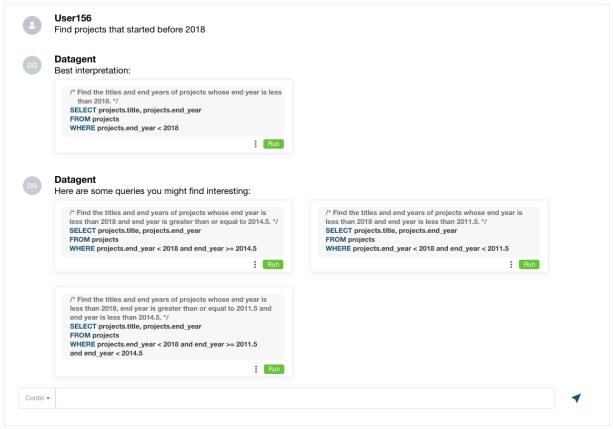


**Figure 3: Query Recommendations**

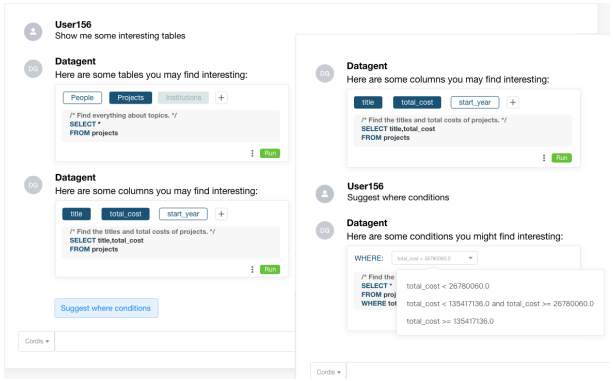


**Figure 4: SQL Builder**

intuitive and naturally lends itself for use by the conference participants that can try out its capabilities for exploring the datasets.

*Demo Example 1: (NL to SQL - SQL to NL)* Let us assume that we have a user that is not an SQL expert. The user wants to query the CORDIS database in order to find out which projects started before 2018. Therefore, as seen in Figure 2, she asks the DatAgent to "Find projects that started before 2018" (step 1). The agent returns the SQL interpretation that matches best the given NL query, along with its corresponding NL explanation (step 2). Assuming that the user is satisfied with the given query, she can choose to execute the query and view the results (step 3). In case the user already knows the exact SQL query which matches her criteria, all steps remain the same except from step 1, where she would directly give the SQL query instead of the NL one.

*Demo Example 2: (Find similar queries)* Continuing with the user of Example 1, we now assume that the user is not satisfied with the currently derived query (Figure 3). To combat this challenge, DatAgent offers useful recommendations that augment the query by adding/altering its `WHERE` clause. In the current example, the user has clicked on the SQL query to open the menu and choose the "Recommend" option (as shown in Figure 2). The user receives three query recommendations from the system.

*Demo Example 3: (Query Builder)* Let us now consider the scenario of an SQL user that has no knowledge about the underlying database schema (or that lacks the experience to derive a starter query), having trouble formulating a query. For this case, our system is able to provide complete step-wise query building assistance. The two recommendation components for identifier and query completion recommendations work complementary for this process.

As seen in the left screenshot of Figure 4, the user first receives table recommendations and selects the most interesting ones. Once the user selects a recommended table, only the joinable tables will be available for selection. Selecting a table will also change the SQL query in the code block to reflect the user's choices. In the second step, the user has decided to retrieve column recommendations for the selected tables. The recommended columns are only from the previously selected tables. Additionally, the user can manually add any table and attribute by typing the name of the desired identifier.

Finally, the user may choose to get suggested `WHERE` conditions. This can be seen in the right screenshot of Figure 4.

## REFERENCES


[1] Kedar Dhamdhere, Kevin S. McCurley, Ralfi Nahmias, Mukund Sundararajan, and Qiqi Yan. 2017. *Analyza: Exploring Data with Conversation.* ACM.

[2] Apostolos Glenis and Georgia Koutrika. 2021. PyExplore: Query Recommendations for Data Exploration without Query Logs. In *SIGMOD Conference 2021.* ACM.

[3] Zhexue Huang. 1998. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data mining and knowledge discovery* 2, 3 (1998), 283–304.

[4] Andreas Kokkalis, Panagiotis Vagenas, Alexandros Zervakis, Alkis Simitsis, Georgia Koutrika, and Yannis Ioannidis. 2012. Logos: a system for translating queries into narratives. In *ACM SIGMOD*, K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman (Eds.). ACM, 673–676.

[5] Fei Li and Hosagrahar V. Jagadish. 2014. Constructing an Interactive Natural Language Interface for Relational Databases. *PVLDB* 8, 1 (Sept. 2014), 73–84.

[6] Qin Lyu, Kaushik Chakrabarti, Shobhit Hathi, Souvik Kundu, Jianwen Zhang, and Zheng Chen. 2020. Hybrid Ranking Network for Text-to-SQL. arXiv:2008.04759 [cs.CL]

[7] Alkis Simitsis, Georgia Koutrika, Yannis Alexandrakis, and Yannis Ioannidis. 2008. Synthesizing structured text from logical database subsets. In *11th international conference on Extending Database Technology.* 428–439.

[8] Alkis Simitsis, Georgia Koutrika, and Yannis Ioannidis. 2008. Précis: from unstructured keywords as queries to structured databases as answers. *The VLDB Journal* 17, 1 (2008), 117–149.

[9] ThoughtSpot 2021. Search & AI-Driven Analytics Platform. https://www.thoughtspot.com/product.