Building Natural Language Interfaces for Databases in Practice

Claude Lehmann
claude.lehmann@zhaw.ch
Zurich University of Applied Sciences
Winterthur, Switzerland

Dennis Gehrig
dennis.gehrig@zhaw.ch
Zurich University of Applied Sciences
Winterthur, Switzerland

Stefan Holdener
stefan@veezoo.com
Veezoo AG
Zurich, Switzerland

Carlo Saladin
carlo@veezoo.com
Veezoo AG
Zurich, Switzerland

João Pedro Monteiro
jp@veezoo.com
Veezoo AG
Zurich, Switzerland

Kurt Stockinger
kurt.stockinger@zhaw.ch
Zurich University of Applied Sciences
Winterthur, Switzerland

ABSTRACT

Natural language interfaces to databases have recently made substantial progress due to advances in machine learning. Users no longer need technical knowledge to search for insights in their database. However, research is largely focused on increasing the one-shot accuracy, instead of building systems that interact with and guide a user's search. In this demo, we present Veezoo, an AI-powered data analytics platform that enables users to directly talk to their databases.

CCS CONCEPTS

• **Information systems** → **Search interfaces**.

KEYWORDS

Natural language interfaces, user interfaces, database querying

ACM Reference Format:
Claude Lehmann, Dennis Gehrig, Stefan Holdener, Carlo Saladin, João Pedro Monteiro, and Kurt Stockinger. 2022. Building Natural Language Interfaces for Databases in Practice. In *34th International Conference on Scientific and Statistical Database Management (SSDBM 2022), July 6–8, 2022, Copenhagen, Denmark.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3538712.3538744

1 INTRODUCTION

Due to significant improvements in building natural language-to-SQL systems (NL2SQL) over the last few years, interacting with data has become much simpler and more effective for a wide range of users [1]. While data exploration used to require a technical skill set, a much broader audience of non-technical users can now interact with previously inaccessible data [2].

Previous work on NL2SQL systems mainly focused on increasing the accuracy of translating natural language questions to SQL [3, 4, 7, 8]. However, there has been very little focus on developing intelligent user interfaces that help users in reformulating natural language questions that cannot be answered by the NL2SQL system. Traditional papers report that a certain number of queries cannot be answered by their systems, but they do not provide a recipe for how these hard queries can be answered. Incidentally, these hard queries often yield more interesting results.

In this paper we introduce *Veezoo, an NL2SQL system that enables users to interact with data in a dialog in case the system does not understand the users' questions.* In addition, we *perform an experiment to show Veezoo's effectiveness in answering hard queries* by guiding the users with relevant feedback and by suggesting how to reformulate the original user questions.

2 RELATED WORK

In recent years, the Spider dataset [12] has become the de facto standard for measuring NL2SQL performance. The official leaderboard[1] differentiates between *exact set match without values* and *execution with values*. While the former focuses on the prediction accuracy, the latter is more directly anchored in real-world use cases, by also executing generated SQL statements and evaluating the returned values. As we are interested in the application of such systems with real users and businesses, we are only interested in the second leaderboard. The goal is to generate a valid SQL query, execute it on the database and find the same set of values as contained in the list of gold values. Currently the top 3 entries all use transformer-based architectures. These are `T5-3B+PICARD` [8] with 75.1% execution accuracy, `RATSQL+GAP+NatSQL` [4] with 73.3% and `SmBoP+GraPPa` [7] with 71.1%. The aforementioned leaderboard entries are composed of multiple methods combined to yield even stronger results. We will give an overview of the most prominent methods and ideas in the following paragraph.

Predictions of an NL2SQL system must be a valid SQL statement, thus it is difficult to take off-the-shelf pre-trained language models as-is. A possible solution is to use abstract syntax trees to guide the SQL generation [9, 10], which makes sure that no syntactically invalid SQL is generated and to also help the flow of information in the neural network. However, this approach requires the addition of specialized control tokens or changing the model architecture.

The `PICARD` method [8] incrementally increases the beam size of the beam search (i.e. the number of generated results) and chooses only valid candidates among them, though this comes at the cost

[1] https://yale-lily.github.io/spider

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SSDBM 2022, July 6–8, 2022, Copenhagen, Denmark
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9667-7/22/07…$15.00
https://doi.org/10.1145/3538712.3538744

of additional computational resources. RATSQL [9] adds the representation of relations to the encoder's attention mechanism of the transformer architecture. NatSQL [4] is an intermediate representation layer between the natural language query and the final SQL statement (similar to SemQL [5]), allowing to learn a representation that is detached from SQL dialects. NatSQL is built as a replacement of SemQL. Their authors note a compatibility with a larger variety of SQL queries, the simplification of the query structure and a reduction in the number of predicted items in the schema. GraPPa [11] is an approach for pre-training, where new synthetic examples are generated from a synchronous context-free grammar (SCFG), making it a data-efficient learning approach. SmBoP [7] combines GraPPa with RATSQL. In addition, it changes the traditional top-down search through the abstract syntax tree by a bottom-up approach, enabling parallelization to significantly speed up training and inference.

3 THE VEEZOO NL2SQL SYSTEM

3.1 Veezoo System Architecture

Veezoo is a commercial data analytics software that uses NL2SQL technology to provide easy access to information for business users. Its architecture is roughly divided into the following parts:

- A *Knowledge Graph* that helps match keywords to database concepts, such as tables, columns and distinct values. This Knowledge Graph is created in an automatic way based on the database schema and can be manually edited by the user. Furthermore, it contains business logic and allows to fine-tune question understanding.
- A *parser* that maps natural language utterances into an unambiguous intermediate representation, called a *logical form*.
- A *query processor* that transforms logical forms into SQL queries and executes them.
- A *visualization engine* that displays the queried data in the most appropriate chart for easy understanding.

We will focus the description of the underlying system on the parser component, since it is the most relevant for the discussion in this paper. Veezoo's parser component first uses techniques in the areas of Entity Linking, Relation Extraction and Temporal Expression Parsing to recognize relevant passages in the user's utterance and to generate possible understandings — including entities, numbers and dates — of each passage. The identified understandings are then extended and combined using a series of well-defined rules into multiple candidate logical forms, each one representing a possible interpretation of the utterance. The system makes use of a machine learning model to score these logical forms, choosing thus the most likely interpretation of the utterance for further processing with the query processor.

3.2 Veezoo UI

The area where Veezoo excels is its user interface. When first interacting with a database, users generally lack the knowledge about the contents of the database. Veezoo solves this by presenting its users the Knowledge Graph, a summary of the underlying database schema. Figure 1 shows both the list of entities and attributes of Spider's *singer* database on the left side as a quick reference, as well as a visual representation of the relationships between entities and attributes on the right side. Both can be accessed easily and quickly in Veezoo's sidebar. In addition to helping users understand the database, the interface gives the users customization options such as manually adding synonyms.

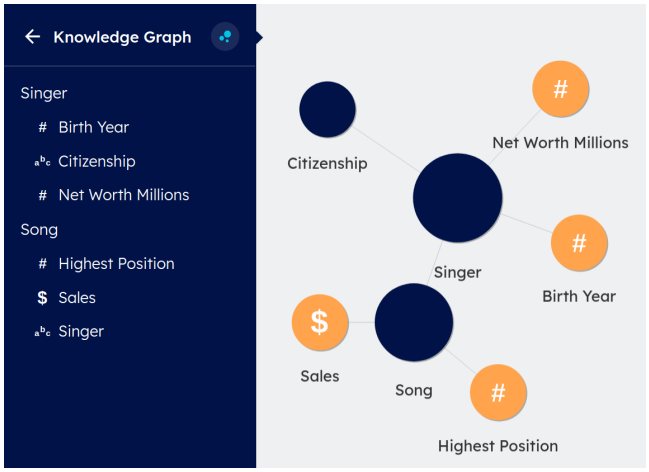

Figure 1: Veezoo's Knowledge Graph of the Spider database *singer*. The figure shows a list of entities and attributes on the left side. The visual knowledge graph is shown on the right side. Icons identify different data types.

The most important innovation of Veezoo, however, is the *dialog interface*, which enables users to have a direct conversation with the system. Users can ask Veezoo questions about the content of a database and Veezoo will transform the natural language question into a SQL statement, query the database and provide an answer.

Consider this question from the Spider [12] development set: *What are the names of the singers who are not French citizens?* Initially, Veezoo has trouble understanding some words in the question (see Figure 2). Using the FollowUp-feature, the users are able to further refine their questions, staying within the context of their first question. The words "citizen" and "French" are not recognized, but Veezoo suggests the term "France - Citizenship" while typing. The correct result is returned once the follow-up question is asked.

Let us now investigate the different areas of Veezoo's interface. Figure 2 shows the example of a question that Veezoo does not fully understand. In such cases, a note will be displayed to give additional information about the issue and some helpful tips to avoid them in the future. The most important features available to the user are: (a) the question bar where users formulate their questions, (b) a previously asked question, (c) a feedback note, where Veezoo indicates issues or gives hints to the user, (d) the answer to a question, (e) auto-complete suggestions for what the user is typing, based on what Veezoo finds in the knowledge graph, (f) question suggestions that could be interesting for the user, (g) asking a follow-up question, drilling down into this particular answer, and finally (h) a manual column selection to show additional information.

Note, that for a successful question, the feedback note (c) would be missing. All other features are always available, irrespective of the success of the question. In addition, the data returned by the

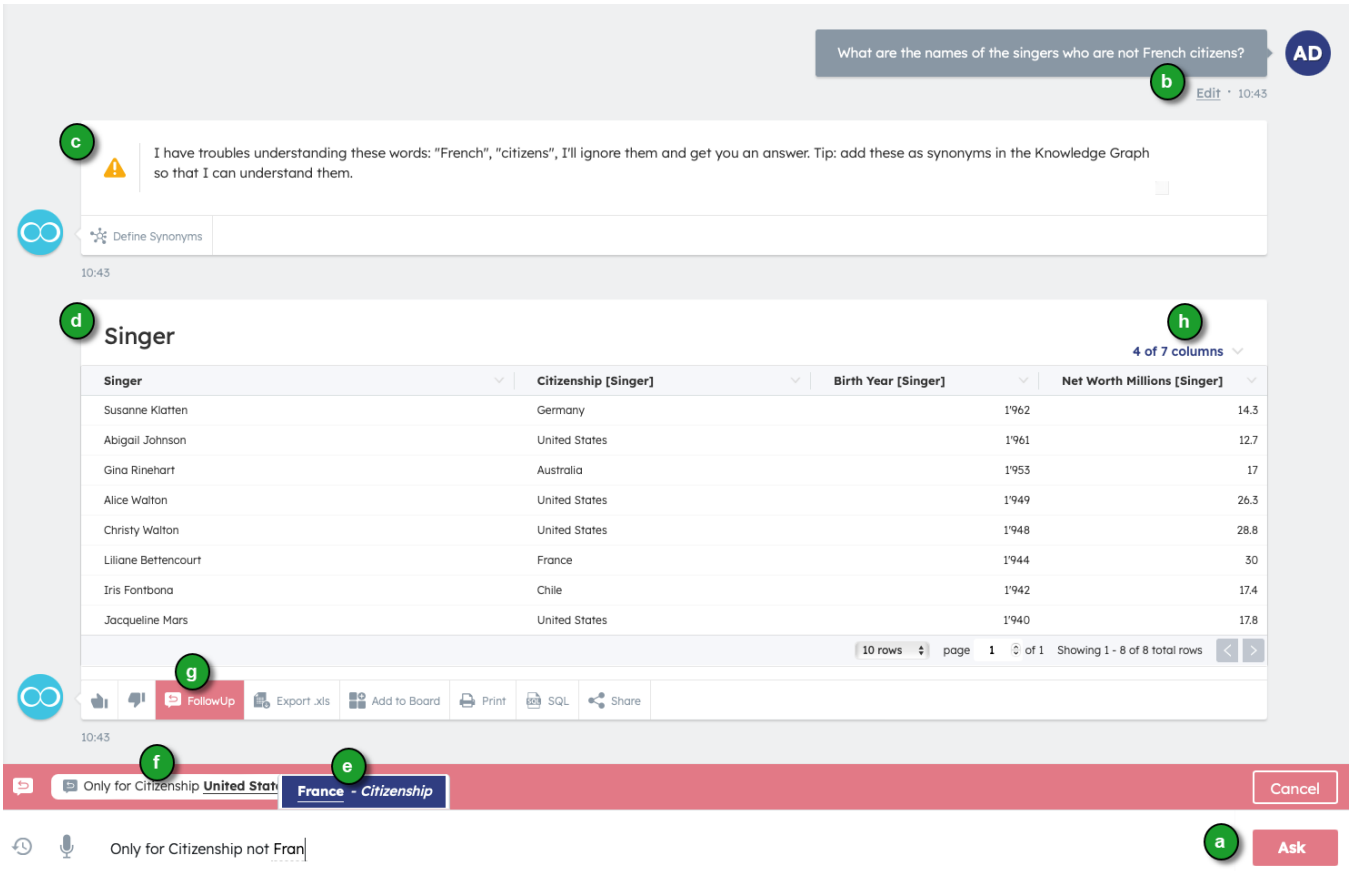

Figure 2: The interface for asking Veezoo a question, showing the case when Veezoo is not confident about its answer. In the middle, the result to a previous question is shown. At the bottom, users can adjust their questions, ask a different question, or even drill down with a *FollowUp* request.

query is displayed in various types of visualizations, if appropriate. This rich feature set allows users to fully engage with their data, exploring it, fixing problems in understanding and finally gaining previously hidden insights. Veezoo automatically offers additional information outside of what was asked, for example, if asked about the names of singers that were born before 1980, Veezoo will show the `Birth Year` among other columns. A user might not have asked to see this column, but the additional information helps to familiarize the user with the data in the database. A follow-up question could then ask "What is their net worth", enriching the answer with the content of the "Net Worth Millions" column.

Recently, Veezoo has also been extended to enable *natural language questions for recommender system queries* such as "Which films should I suggest to user X" [6].

4 DEMONSTRATION AND EVALUATION

In order to evaluate Veezoo's UI, we carried out an experiment assessing Veezoo's feedback mechanism. The goal of the experiment was to measure how many interactions between the user and Veezoo are required to arrive at the correct answer, given that Veezoo is not able to answer a question correctly. In order to do that, we constructed the worst case scenario, namely using an unoptimized Knowledge Graph in Veezoo (e.g. no added synonyms or cleaned up column names), having users that are unfamiliar with Veezoo's interface participate in the experiment, and only choosing questions where we know Veezoo returns an incorrect result. In many business scenarios, however, users settle on a coherent company vocabulary and denominate concepts by the same name, as has been shown in practice by Veezoo's customers.

For the experiment, we chose ten databases from the Spider development set. Next, we picked questions that Veezoo did not answer correctly. Every user was given ten questions from three different databases and was then tasked to find the correct answer from Veezoo, using any of the available features. Users were selected from various IT backgrounds (software engineers, researchers and students, primarily), but not necessarily well-versed in natural language topics. Typical corrections or user interactions include (1) rewriting the question, (2) reformulating certain words, (3) using the suggestions proposed by Veezoo or (4) inspecting the Knowledge Graph to find correct table and column names.

The results of our 16 test users $U1$ through $U16$ are shown in Figure 3. The figure shows each user's performance across their ten

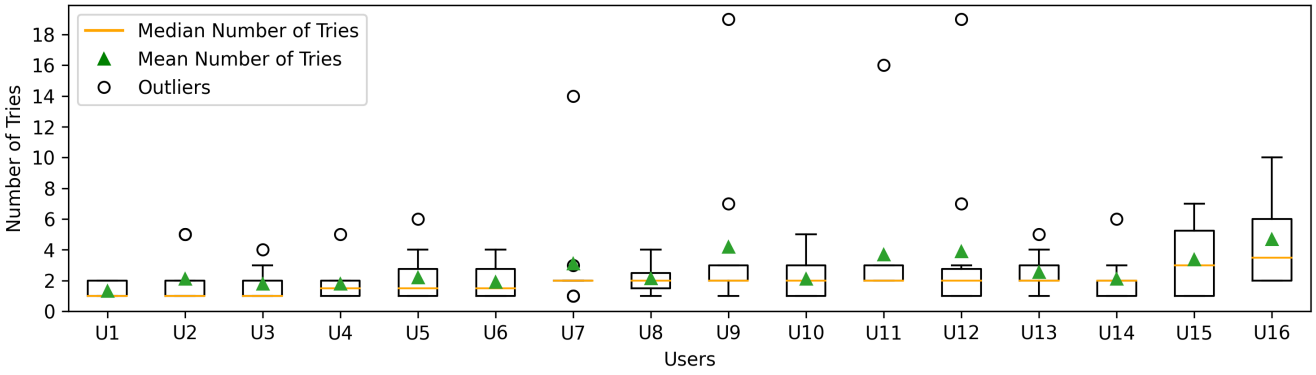

Figure 3: User performance in the experiment. Users are ranked by the median number of tries to get a correct answer from an initially incorrectly answered question.

questions. Their answers were manually checked for correctness. The x-axis shows the different users, ordered by increasing median number of tries, the y-axis shows the number of tries. A try consists of any query to Veezoo that differs from the given question.

Except for users $U15$ and $U16$, all users have a median number of tries of 2, indicating that for most questions, Veezoo's answer was not far off and untrained users were quickly able to figure out how to correct the question and satisfy their information need. The figure exhibits a similar picture for the mean number of tries, where 9 out of the 16 users could answer their question on average in 2.5 or less tries. Even so, we can clearly identify a few outliers, e.g. user $U15$ needed up to 7 tries to get a satisfying answer, user $U16$ required 10 tries for multiple questions, while users $U9$ and $U12$ needed 19 tries for correctly answering a single question.

The question *Find the role, street, city and state of the professionals living in a city that contains the substring 'West'* from the `dog_kennels` database posed a challenge for many users, because asking for the substring "West" was not yet easily accessible in Veezoo, depending on the configuration of the underlying knowledge graph. Six users had to deal with that question. $U11$ (16 tries) and $U14$ (6 tries) managed to get a correct answer from Veezoo by explicitly selecting the cities over Autocomplete. Users $U2$, $U13$ and $U15$ gave up after less than 5 tries. User $U9$ only gave up after 19 tries thus increasing their average immensely. For all other questions, the users found a satisfying answer.

5 CONCLUSION

We demonstrated the usefulness of a natural language interface to databases that is designed for usability, enabling quick information access and leading users toward their desired information. While the choice of the underlying NL2SQL system is an important one, we argue that the characteristics of the user interface — independent of the NL2SQL system — also play an important role. Veezoo has been shown to be very effective for large-scale customers in real-world business settings. Furthermore, our evaluation demonstrated that even in an adversarially designed scenario, Veezoo can guide users to find their answers effectively by engaging them in a dialog between system and user — inspired by a dialog between humans.

ACKNOWLEDGMENTS

The work was funded by Innosuisse as an innovation project under the project number `34223.1 IP-ICT` and by the European Union's Horizon 2020 research and innovation program under grant agreement No 863410.

REFERENCES

[1] Katrin Affolter, Kurt Stockinger, and Abraham Bernstein. 2019. A Comparative Survey of Recent Natural Language Interfaces for Databases. *The VLDB Journal* 28, 5 (2019).
[2] Sihem Amer-Yahia, Georgia Koutrika, et al. 2021. INODE: Building an End-to-End Data Exploration System in Practice. *SIGMOD Record* (2021).
[3] Ursin Brunner and Kurt Stockinger. 2021. ValueNet: A Natural Language-to-SQL System that Learns from Database Information. In *International Conference on Data Engineering (ICDE)*.
[4] Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R. Woodward, John H. Drake, and Qiaofu Zhang. 2021. Natural SQL: Making SQL Easier to Infer from Natural Language Specifications. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.
[5] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.
[6] Yasamin Klingler, Claude Lehmann, João Pedro Monteiro, Carlo Saladin, Abraham Bernstein, and Kurt Stockinger. 2022. Evaluation of Algorithms for Interaction-Sparse Recommendations: Neural Networks don't Always Win. In *International Conference on Extending Database Technology (EDBT)*.
[7] Ohad Rubin and Jonathan Berant. 2021. SmBoP: Semi-autoregressive Bottom-up Semantic Parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics*.
[8] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.
[9] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.
[10] Pengcheng Yin and Graham Neubig. 2018. TRANX: A Transition-based Neural Abstract Syntax Parser for Semantic Parsing and Code Generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.
[11] Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. 2021. GraPPa: Grammar-Augmented Pre-Training for Table Semantic Parsing. In *Proceedings of the 2021 International Conference on Learning Representations*.
[12] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.