2023 IEEE 39th International Conference on Data Engineering (ICDE) | 979-8-3503-2227-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICDE55515.2023.00278

GENSQL: A Generative Natural Language Interface to Database Systems

Yuankai Fan[1], Tonghui Ren[1], Zhenying He[1,2], X.Sean Wang[3,1], Ye Zhang[4], Xingang Li[4]
[1]*School of Computer Science, Fudan University, Shanghai, China*
[2]*Shanghai Key Laboratory of Data Science, Shanghai, China*
[3]*Zhejiang Lab, Hangzhou, China*
[4]*China UnionPay Merchant Services Co.,LTD., Shanghai, China*
ykfan19@fudan.edu.cn, thren22@m.fudan.edu.cn, zhenying@fudan.edu.cn
xywangCS@fudan.edu.cn, yezhang@chinaums.com, xgli2@chinaums.com

***Abstract*—To make databases more accessible to a much broader audience of non-technical users, many applications, such as chatbots and search engines, have developed a natural language (NL) interface for the underlying databases (NLIDB). With the advances in machine learning techniques, most recent research employs language translation models to build NLIDB systems. In this demonstration, we introduce GENSQL, a generative NLIDB system that enables users to query databases using NL. Unlike most existing NLIDB systems that attempt to leverage a generalized language translation model to convert NL to SQL queries (NL2SQL) for any database, GENSQL utilizes a set of sample queries to capture the specific structure and semantics of a given database and thus to provide more accurate translation results. The underlying NL2SQL model in GENSQL is a novel generate-and-rank model named GAR designed by the authors, which first generates a set of generalized SQL queries with corresponding NL expressions from the given sample queries, and ranks the NL expressions to get the best matching one, and hence the SQL query. This demonstration shows the effectiveness of GENSQL, especially in answering complex queries, which proves its utility in practice.**

***Index Terms*—NLIDB, NL2SQL, Generate-and-rank**

I. INTRODUCTION

As the database user base is shifting toward a much broader audience of non-technical users, designing natural language (NL) interfaces to databases (NLIDB) has become increasingly important in the last few decades. These NLIDB systems enable users without specialized knowledge of a query language (e.g., SQL) to access the data stored in databases by typing queries expressed via NL.

Recently, due to the maturity of language translation techniques, a substantial amount of NLIDB systems have been built upon language translation models and mainly focused on increasing the accuracy of translating NL to SQL queries (NL2SQL) [1]–[5]. Despite the significant gains in terms of translation accuracy, the generalized language translation models may fail on complex queries that require an understanding of the structure and semantics specific to a database, and thus cannot provide reliable performance for real use.

In this demonstration, we introduce GENSQL, an NL2SQL system that provides user-friendly query interfaces for users to interact with databases in spoken language and can be plugged into any existing RDBMSs, such as MySQL. Unlike existing NLIDB systems, GENSQL utilizes the information from the given sample queries (e.g., from query logs) of a database to enhance the overall NL2SQL translation performance. The backbone technique in GENSQL is a practical generate-and-rank approach named GAR [6] designed by the authors, which first attempts to generate all possible SQL queries from the given samples, then synthesizes the corresponding NL expressions called dialects for each generalized SQL query, and last ranks to get the best matching dialect, and hence the SQL query. As a bridge between users and the underlying databases, GENSQL can help users with a non-technical background construct data operations and significantly improve the efficiency of interacting with databases. In addition, since GENSQL learns from sample queries of a given database as a starting point, GENSQL can easily improve supportability on more complex queries if more query varieties we have in the sample set. Hence, we expect that GENSQL will inspire more research on integrating an NL querying interface into existing RDBMSs to improve database usability.

II. RELATED WORK

Natural Language Interface to Databases. Multiple lines of research have developed NL interfaces for formulating database queries in data management and natural language processing (NLP) communities [7]. Early works [8]–[11], are rule-based approaches, which use handcrafted grammar and rules to map NL queries to SQL queries specific to a certain database. TEMPLAR [10] also proposes using database query logs, but the approach is rule-based and only focuses on keyword mapping and join path inference tasks. With the recent success of neural machine translation, many machine learning-based approaches, such as RAT-SQL [1], SMBOP [3], and PICARD [5], have been proposed to build NLIDB systems, which treat the NL2SQL problem as a translation task and employ the encoder-decoder architecture to tackle the problem. In addition, the development of large annotated datasets has catalyzed progress in the field, where a commonly-used public NLIDB dataset is named SPIDER [12].

Query Interpretation. Research has also explored methods to translate SQL queries to NL (SQL2NL). Earlier attempts [13], [14] explicitly study the problem of translating small databases under certain constraints. [15] proposes to employ a sequence-to-sequence (Seq2Seq) neural network to represent the SQL query and NL query jointly, while [16] employs a template-based method to synthesize the NL expressions. Snowball [17] employs an iterative training procedure by recursively augmenting the training set to generate the text.

Learning-to-rank. The framework of learning-to-rank (LTR) has been successfully applied in multiple areas, such as question answering, recommendation, and document retrieval. With the recent advances in pre-training for text, many recent works in this field have been proposed [18], [19] by utilizing the pre-trained language models.

III. System Architecture

In this section, we first present the GenSQL system overview, introduce the NL2SQL method used in GenSQL, and then give an analysis of its performance.

A. *GenSQL Overview*

GenSQL is a database querying system that can be plugged into any existing RDBMS to provide easy access to the data stored in the database for users[1]. Fig. 1 presents the system overview of our GenSQL system. As can be seen, the architecture of GenSQL can be roughly divided into the following three layers:

- The *Data Layer* is mainly responsible for query data preparation, which includes two components - the **Query Collector** collects the sample queries either from user input or from query logs from the databases, and a **SQL Generator** generalizes the samples in an attempt to generate all possible SQL queries.
- The *Model Layer* provides the backbone techniques for NL2SQL translation in the GenSQL system, where the **Dialect Builder Model** first utilizes SQL2NL technique to synthesize the dialects for each generalized SQL query from the Data Layer, and then the **Ranking Models** are used for a given NL query to retrieve the best dialect and hence the resulting SQL query.
- The *Interface Layer* provides rich features to enable users to interact with GenSQL system. Serving different user bases, GenSQL integrates two kinds of querying interfaces - (i) **User Interface** enables ordinary users to query the underlying databases by typing NL input, and GenSQL reacts with the execution results with detailed information provided; (ii) **Admin Interface** provides additional features for expert users like database administrators to manage the system, including underlying DBMS selection, data upload, database schema lookup, sample queries check, coverage testing, etc.

B. *The Generate-and-Rank Approach in GenSQL*

In GenSQL, We formulate the NL2SQL task as a semantic matching problem and utilize a generate-and-rank approach to achieve the NL query to dialect, hence to SQL query

[1]Currently, GenSQL supports SQLite RDBMS, and next, we will include the supportability of other mainstream RDBMSs.

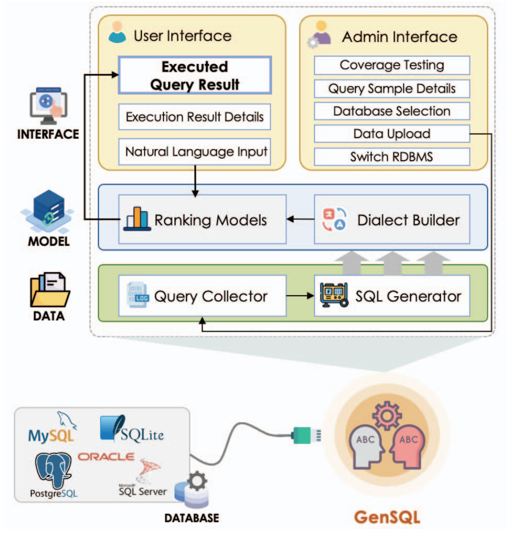


Fig. 1: System architecture of GenSQL

translation. The high-level view of our proposed Gar approach is illustrated in Fig .2. Given a set of sample SQL queries, Gar first uses the following data preparation process to generate dialect expressions during an offline initialization phase: generalization and SQL2NL. After the data preparation process, an LTR model is used to rank the generated dialect expressions for the final translation.

Generalization. The step in Fig. 2-① uses a set of generalization rules to generalize the sample queries to provide a good coverage for similar queries while limiting the resulting set to a manageable size.

SQL2NL. The step in Fig. 2-② uses a template-based SQL2NL method [16]. Each clause of a SQL query is mapped to an NL phrase mechanically with the help of a parsed query tree. Then the phrases are combined into a sentence, which we call a dialect expression.

Learning-to-Rank. The learning-to-Rank (LTR) model in Fig. 2-③ follows the LTR approach in information retrieval that trains a neural network in a supervised manner. In Gar, the LTR model learns to rank the semantic similarities between NL queries and dialect expressions (explained below). The model is used to rank the dialect expressions and then find the best-matching one for a given NL query, leading to the translation result.

C. *Performance Analysis*

To evaluate GenSQL, We compare the translation accuracy with that of four state-of-the-art machine learning-based translation methods, namely Bridge [2], Rat-SQL [1], Gap [4] and Smbop [3]. Here, all the four baseline models are Seq2Seq-based NL2SQL translation models. We adopt the metrics *Exact Match Accuracy* and *Execution Match Accuracy* introduced in [12] as the main indicators, where the former is defined as syntactic equivalence, and the latter is defined as

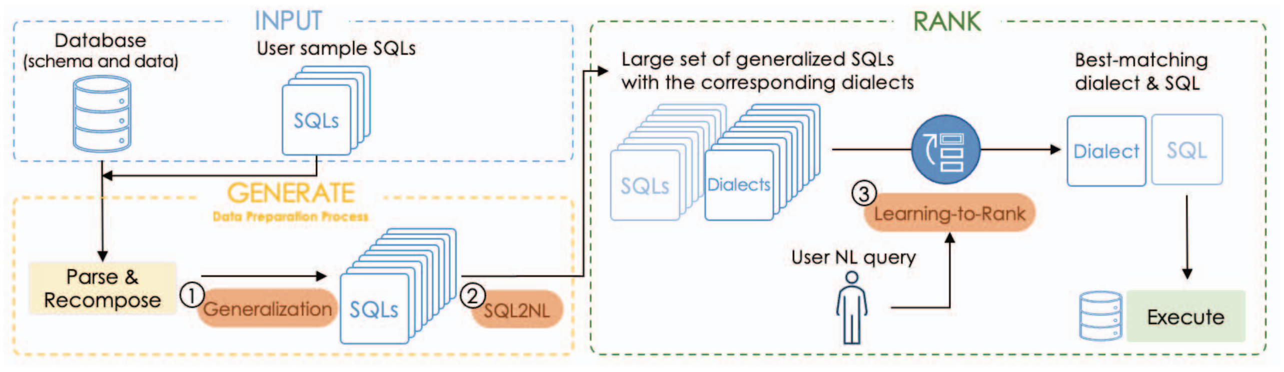


Fig. 2: Overview of our proposed GAR approach

execution result equivalence. All the neural network models in all the methods (including GENSQL) are trained on the same training set from the SPIDER benchmark. Note that since GENSQL leverages sample queries to capture the specific semantics of an underlying database, we obtain the sample queries as follows. We first use the SQL queries of the SPIDER validation set to generate generalized query sets, then we rule out all the ground truth queries from the generalized query sets and use the sets as the sample queries.

Table 1 presents the results of all these models on the validation set of the SPIDER benchmark. Since GAR masks out values during the generalization process and leverages a post-processing step to specify values for the translation results, GAR attains 72.6% execution match accuracy, slightly lower than SMBOP by 2.6%. Notably, GAR achieves 78.5% exact match accuracy, outperforming all the other four models, further showing the effectiveness of the novel generate-and-rank approach for the NL2SQL problem.

TABLE 1: Performance of five different NL2SQL models

NLIDB Model	Exact Match Acc.	Execution Match Acc.
GAR	**0.785**	0.726
SMBOP	0.737	0.752
BRIDGE	0.687	0.680
GAP	0.727	0.349
RAT-SQL	0.694	0.341

IV. DEMONSTRATION OVERVIEW

This demonstration aims to provide an interactive demo for users and database administrators to learn the working mechanisms of our proposed GENSQL system introduced above. We chose the database *concert_singer* from the validation set of the SPIDER benchmark and select to use the NL query *"How many singers are from each country?"* to present this demonstration. Fig. 3 gives the screenshots of the two interfaces (i.e., user interface and admin interface) provided in the GENSQL system, which describe the main functionalities that the two interfaces include.

A. User Interface

To maximize the usability of the databases for ordinary users, GENSQL provides a chat-based dialog interface for users to interact with the system. As can be found in Fig. 3a, the user interface mainly includes the following three functionalities.

① **NL Query Input.** Given a database, users types the NL queries in the input box, and wait for the response from the GENSQL system.

② **Executed Query Result.** When GENSQL receives the input from the user, the system employs the backbone generate-and-rank model GAR to translate the NL query to the corresponding SQL query, and then triggers the execution of the SQL query on the selected database. After the execution, GENSQL displays the executed result in the chat dialog.

③ **Execution Result Details.** To help users validate the correctness of the executed result, GENSQL allows users to check the additional execution details by clicking on the ▶ button. The information includes the following four parts: (i) the SQL query executed to retrieve the result. (ii) the top-ranked dialects that GAR selects for the given NL query; (iii) the corresponding top-ranked SQL queries; (iv) the sample database schema. In addition, GENSQL also allows users to provide feedback after checking the executed SQL query, which can help GENSQL further improve its performance.

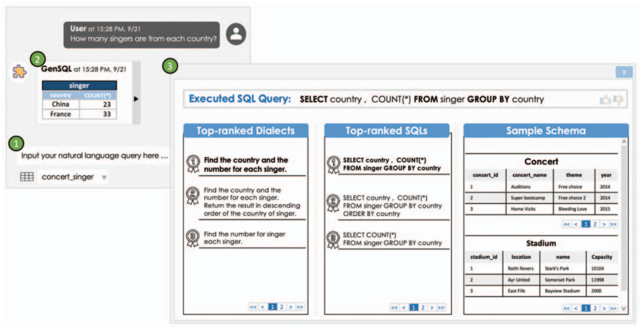
B. Admin Interface

GENSQL provides rich features for expert users (e.g., database administrators) to manage the system. Fig. 3b presents the overview of the admin interface, which mainly includes the following five functionalities.

① **Switch RDBMS.** GENSQL is designed to be plugged into any existing mainstream RDBMSs. Administrators first selects the RDBMS that they want to operate on.

② **Data Upload.** Once the RDBMS is determined, the system requires the administrators to upload the databases the user wants to query. In addition, since GENSQL leverages sample queries to capture specific semantics for an underlying database, a set of sample queries also requires to be uploaded. Here, the sample queries can be exported from the query logs of existing databases.

③ **Database Selection.** Administrators select the database and use the schema overview section to check the schema of current database.



User at 15:28 PM, 9/21
How many singers are from each country?

singer	
country	COUNT(*)
China	23
France	33

Executed SQL Query: SELECT country , COUNT(*) FROM singer GROUP BY country

1. Find the country and the number for each singer.
2. Find the country and the number for each singer. Return the result in descending order of the country of singer.
3. Find the number for singer each singer.

1. SELECT country , COUNT(*) FROM singer GROUP BY country
2. SELECT country , COUNT(*) FROM singer GROUP BY country ORDER BY country
3. SELECT COUNT(*) FROM singer GROUP BY country

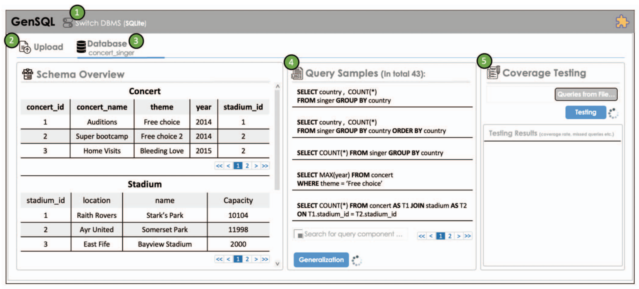
Concert

concert_id	concert_name	theme	year
1	Auditions	Free choice	2014
2	Super bootcamp	Free choice 2	2014
3	Home Visits	Bleeding Love	2015

Stadium

stadium_id	location	name	Capacity
1	Raith Rovers	Stark's Park	10104
2	Ayr United	Somerset Park	11998
3	East Fife	Bayview Stadium	2000

(a) The user interface in GENSQL



Concert

concert_id	concert_name	theme	year	stadium_id
1	Auditions	Free choice	2014	1
2	Super bootcamp	Free choice 2	2014	2
3	Home Visits	Bleeding Love	2015	2

Stadium

stadium_id	location	name	Capacity
1	Raith Rovers	Stark's Park	10104
2	Ayr United	Somerset Park	11998
3	East Fife	Bayview Stadium	2000

```
SELECT country , COUNT(*)
FROM singer GROUP BY country

SELECT country , COUNT(*)
FROM singer GROUP BY country ORDER BY country

SELECT COUNT(*) FROM singer GROUP BY country

SELECT MAX(year) FROM concert
WHERE theme = 'Free choice'

SELECT COUNT(*) FROM concert AS T1 JOIN stadium AS T2
ON T1.stadium_id = T2.stadium_id
```

(b) The admin interface in GENSQL

Fig. 3: The two querying interfaces of our proposed GENSQL

④ **Query Samples Details.** GENSQL allows administrators to check the details of the sample queries that the current database has. This functionality includes the following three features: (i) Lookup each sample query; (ii) Search specific query components that the administrator is interested in; (iii) If the administrator satisfies the sample queries, trigger the generalization process to get all candidate SQL queries for the translation purpose of the backbone GAR model.

⑤ **Coverage Testing.** To evaluate if the generalization has a good coverage, the system allows administrators to do some coverage testing. By choosing the testing queries from a local file and clicking the testing button, the system will generate a report for the coverage over the testing queries, which roughly includes an overall coverage rate, the missing queries, and other testing statistics.

V. CONCLUSION

In this demonstration, we presented a practical NLIDB system prototype named GENSQL to achieve database querying via natural language. GENSQL is designed for usability, providing a chat-based dialog interface for ordinary users to simplify the interaction between the system and the user. On the other hand, GENSQL also provides rich features for expert users like database administrators to manage the system properly. The underlying idea for GENSQL is to use a set of sample queries to capture the specific structure and semantics of the underlying database, and to employ a novel generate-and-rank NL2SQL model GAR. The prototype system is still evolving to build more ability to the translation process for dealing with more complex queries and databases.

VI. ACKNOWLEDGEMENTS

The authors would like to thank all the anonymous reviewers for their insightful comments and suggestions. This work was supported by NSFC (Grant No. 61732004, 62272106) and the Zhejiang Lab (Grant No. 2021PE0AC01).

REFERENCES

[1] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson, "RAT-SQL: relation-aware schema encoding and linking for text-to-sql parsers," in *ACL*, 2020.
[2] X. V. Lin, R. Socher, and C. Xiong, "Bridging textual and tabular data for cross-domain text-to-sql semantic parsing," in *EMNLP*, 2020.
[3] O. Rubin and J. Berant, "Smbop: Semi-autoregressive bottom-up semantic parsing," in *NAACL*, 2021.
[4] P. Shi, P. Ng, Z. Wang, H. Zhu, A. H. Li, J. Wang, C. N. dos Santos, and B. Xiang, "Learning contextual representations for semantic parsing with generation-augmented pre-training," in *AAAI*, 2021.
[5] T. Scholak, N. Schucher, and D. Bahdanau, "PICARD: parsing incrementally for constrained auto-regressive decoding from language models," in *EMNLP*, 2021.
[6] Y. Fan, Z. He, T. Ren, D. Guo, C. Lin, R. Zhu, G. Chen, Y. Jing, K. Zhang, and X. Wang, "Gar: A generate-and-rank approach for natural language to sql translation," in *International Conference on Data Engineering, ICDE*, 2023.
[7] H. Kim, B. So, W. Han, and H. Lee, "Natural language to SQL: where are we today?" *Proc. VLDB Endow.*, vol. 13, no. 10, pp. 1737–1750, 2020. [Online]. Available: http://www.vldb.org/pvldb/vol13/p1737-kim.pdf
[8] A. Simitsis, G. Koutrika, and Y. E. Ioannidis, "Précis: from unstructured keywords as queries to structured databases as answers," *PVLDB*, pp. 17(1):117–149, 2008.
[9] F. Li and H. V. Jagadish, "Constructing an interactive natural language interface for relational databases," *PVLDB*, pp. 8(1):73–84, 2014.
[10] C. Baik, H. V. Jagadish, and Y. Li, "Bridging the semantic gap with SQL query logs in natural language interfaces to databases," in *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019*. IEEE, 2019, pp. 374–385. [Online]. Available: https://doi.org/10.1109/ICDE.2019.00041
[11] D. Saha, A. Floratou, K. Sankaranarayanan, U. F. Minhas, A. R. Mittal, and F. Özcan, "ATHENA: an ontology-driven system for natural language querying over relational data stores," *PVLDB*, pp. 9(12):1209–1220, 2016.
[12] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. R. Radev, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task," in *EMNLP*, 2018.
[13] A. Simitsis and G. Koutrika, "Comprehensible answers to précis queries," in *CAiSE*, 2006.
[14] A. Simitsis, G. Koutrika, Y. Alexandrakis, and Y. E. Ioannidis, "Synthesizing structured text from logical database subsets," in *EDBT*, 2008.
[15] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, "Summarizing source code using a neural attention model," in *ACL*, 2016.
[16] G. Koutrika, A. Simitsis, and Y. E. Ioannidis, "Explaining structured queries in natural language," in *ICDE*, 2010.
[17] C. Shu, Y. Zhang, X. Dong, P. Shi, T. Yu, and R. Zhang, "Logic-consistency text generation from semantic parses," in *ACL*, 2021.
[18] Z. Dai and J. Callan, "Deeper text understanding for IR with contextual neural language modeling," in *SIGIR*, 2019.
[19] S. Han, X. Wang, M. Bendersky, and M. Najork, "Learning-to-rank with BERT in tf-ranking," *CoRR*, 2020.