My Database User Is a Large Language Model

Eduardo R. Nascimento[1][a], Yenier T. Izquierdo[1][b], Grettel M. García[1][c],
Gustavo M. C. Coelho[1][d], Lucas Feijó[1][e], Melissa Lemos[1][f], Luiz A. P. Paes Leme[2][g] and
Marco A. Casanova[1,3][h]

[1]*Instituto Tecgraf, PUC-Rio, Rio de Janeiro, 22451-900, RJ, Brazil*
[2]*Instituto de Computação, UFF, Niterói, 24210-310, RJ, Brazil*
[3]*Departamento de Informática, PUC-Rio, Rio de Janeiro, 22451-900, RJ, Brazil*

Keywords: Text-to-SQL, GPT, Large Language Models, Relational Databases.

Abstract: The leaderboards of familiar benchmarks indicate that the best text-to-SQL tools are based on Large Language Models (LLMs). However, when applied to real-world databases, the performance of LLM-based text-to-SQL tools is significantly less than that reported for these benchmarks. A closer analysis reveals that one of the problems lies in that the relational schema is an inappropriate specification of the database from the point of view of the LLM. In other words, the target user of the database specification is the LLM rather than a database programmer. This paper then argues that the text-to-SQL task can be significantly facilitated by providing a database specification based on the use of LLM-friendly views that are close to the language of the users' questions and that eliminate frequently used joins, and LLM-friendly data descriptions of the database values. The paper first introduces a proof-of-concept implementation of three sets of LLM-friendly views over a relational database, whose design is inspired by a proprietary relational database, and a set of 100 Natural Language (NL) questions that mimic users' questions. The paper then tests a text-to-SQL prompt strategy implemented with LangChain, using GPT-3.5 and GPT-4, over the sets of LLM-friendly views and data samples, as the LLM-friendly data descriptions. The results suggest that the specification of LLM-friendly views and the use of data samples, albeit not too difficult to implement over a real-world relational database, are sufficient to improve the accuracy of the prompt strategy considerably. The paper concludes by discussing the results obtained and suggesting further approaches to simplify the text-to-SQL task.

1 INTRODUCTION

The *Text-to-SQL* task is defined as "*given a relational database D and a natural language (NL) sentence S that describes a question on D, generate an SQL query Q over D that expresses S*" (Katsogiannis-Meimarakis and Koutrika, 2023)(Kim et al., 2020). A text-to-SQL tool provides a straightforward way to create an NL interface to a database. The user submits an NL question *S*, and the tool translates *S* to an SQL query *Q* in such a way that the execution of *Q* over the database *D* returns an answer to *S*.

Numerous tools have addressed this task with relative success (Affolter et al., 2019)(Katsogiannis-Meimarakis and Koutrika, 2023)(Kim et al., 2020) over well-known benchmarks, such as Spider – Yale Semantic Parsing and Text-to-SQL Challenge (Yu et al., 2018) and BIRD – BIg Bench for LaRge-scale Database Grounded Text-to-SQL Evaluation (Li et al., 2023). The leaderboards of these benchmarks point to a firm trend: the best text-to-SQL tools are all based on Large Language Models (LLMs).

However, when run over real-world databases, the performance of LLM-based text-to-SQL tools is significantly less than that reported in the Spider and BIRD Leaderboards (Nascimento et al., 2024). One of the reasons is that real-world databases have large

[a] https://orcid.org/0009-0005-3391-7813
[b] https://orcid.org/0000-0003-0971-8572
[c] https://orcid.org/0000-0001-9713-300X
[d] https://orcid.org/0000-0003-2951-4972
[e] https://orcid.org/0009-0006-4763-8564
[f] https://orcid.org/0000-0003-1723-9897
[g] https://orcid.org/0000-0001-6014-7256
[h] https://orcid.org/0000-0003-0765-9636

schemas, whereas these benchmarks have a large number of databases whose schemas are quite small, as detailed in Section 2.1. But there is a simpler reason: the relational schema is often an inappropriate specification of the database from the point of view of an LLM. In other words, *the target user of the relational database specification should be the LLM, rather than a database programmer*.

From a broader perspective, the text-to-SQL task involves translating an NL question *S*, which is expressed in the end user's vocabulary and (hopefully) follows the NL grammar, into an SQL query *Q*, which uses the database metadata and data vocabulary. Thus, for the LLM to succeed in the text-to-SQL task, it should, first of all, be able to match the user and the database vocabularies. This matching task is sometimes called *schema linking* and accounts for a large percentage of the errors.

Consider, for example, the NL sentence *S*: "*What is the installation with the largest number of open maintenance orders?*". Sentence *S* uses the end user's terms "*installation*", "*open*", and "*maintenance order*" which, in the best scenario, would match the database table names "Installation" and "Maintenance_Order", and the column name "Situation", which has "open" as a value. However, in a practical scenario, the relational schema may induce a quite different vocabulary, such as the table names "TB_IN" and "TB_MO", and the column name "MO_ST", and the database may use "1" as a value of "MO_ST" to indicate that the order is *open*.

An LLM trained for the text-to-SQL task would translate the expression "*the largest number*" to the correct SQL constructs, which is not a trivial feat (compare it to the treatment of aggregations in earlier approaches reported in (Affolter et al., 2019)). It would then produce the correct SQL query under the best scenario but it would fail in the practical scenario, due to the use of database terms which are thoroughly inappropriate to the LLM.

This paper then argues that the text-to-SQL task can be greatly facilitated by a database specification that provides:

- *LLM-friendly views* that map (fragments of) the database schema to terms close to the terms users frequently adopt and that try to pre-define frequently used joins.

- *LLM-friendly descriptions* of the database values.

LLM-friendly views are nothing but the familiar concept of views, designed to present (fragments of) the relational schema (that is, database metadata) to the LLM. As such, they can be implemented with the usual DBMS mechanisms, within the database. LLM-friendly descriptions refer to a set of constructs that

try to capture the data semantics. They may be defined as a set of *prompt completion pairs*, such as (in OpenAI GPT syntax[1]):

```
{"prompt": "the order is open",
"completion": "Situation='open'"},
```

used to fine-tune the LLM, or to expand the LLM with database-specific knowledge using Retrieval-Augmented Generation (Lewis et al., 2020). A third strategy would be to include *data samples* in the LLM prompt, as in Section 4.1.

To argue in favor of the proposed approach, the paper first introduces a benchmark dataset consisting of a database, three sets of LLM-friendly views, and 100 NL questions and their translation to SQL. The database was inspired by a real-world asset integrity management database, in production at an energy company, which features a relational schema with 27 tables, 585 columns, and 30 foreign keys (some of which are multi-column); the largest table has 81 columns. Thus, it has nearly 640 objects. The NL questions mimic those submitted by end users, and their SQL translations were manually created by experts. The set of NL questions contains 33 classified as simple, 33 as medium, and 34 as complex.

The paper then investigates the performance of a text-to-SQL prompt strategy, implemented with LangChain[2] using GPT-3.5 and GPT-4, over the test dataset. The prompt strategy includes data samples to help the LLM capture the data semantics. The results suggest that specifying a set of LLM-friendly views and data samples is sufficient for the prompt strategy to achieve good performance on the text-to-SQL task, which is significantly better than the performance obtained over the original relational schema.

The paper concludes with a discussion of the results obtained and suggests further approaches to improve the performance of the text-to-SQL task over real-world databases, following the guidelines that *the target user of the database specification is the LLM*.

The paper is organized as follows. Section 2 covers related work. Section 3 describes the benchmark dataset. Section 4 details the experiments. Finally, Section 5 contains the conclusions.

---

[1]https://platform.openai.com/docs/guides/fine-tuning

[2]https://python.langchain.com/docs/use_cases/qa_structured/sql

2 RELATED WORK

2.1 Text-to-SQL Datasets

The Spider – Yale Semantic Parsing and Text-to-SQL Challenge (Yu et al., 2018) defines 200 datasets, covering 138 different domains, for training and testing text-to-SQL tools.

For each database, Spider lists 20–50 hand-written NL questions and their SQL translations. An NL question S, with an SQL translation Q, is classified as easy, medium, hard, and extra-hard, where the difficulty is based on the number of SQL constructs of Q – `GROUP BY`, `ORDER BY`, `INTERSECT`, nested subqueries, column selections, and aggregators – so that an NL query whose translation Q contains more SQL constructs is considered harder. The set of NL questions introduced in Section 3.3 follows this classification, but does not consider extra-hard NL questions.

Spider proposes three evaluation metrics: *component matching* checks whether the components of the prediction and the ground truth SQL queries match exactly; *exact matching* measures whether the predicted SQL query as a whole is equivalent to the ground truth SQL query; *execution accuracy* requires that the predicted SQL query select a list of gold values and fill them into the right slots. Section 4.2 describes the metric used in the experiments reported in this paper, which is a variation of execution accuracy.

Most databases in Spider have very small schemas: the largest five databases have between 16 and 25 tables, and about half of the databases have schemas with five tables or fewer. Furthermore, all Spider NL questions are phrased in terms used in the database schemas. These two limitations considerably reduce the difficulty of the text-to-SQL task. Therefore, the results reported in the Spider leaderboard are biased toward databases with small schemas and NL questions written in the schema vocabulary, which is not what one finds in real-world databases.

Spider has two interesting variations. Spider-Syn (Gan et al., 2021a) is used to test how well text-to-SQL tools handle synonym substitution, and Spider-DK (Gan et al., 2021b) addressed testing how well text-to-SQL tools deal with domain knowledge.

BIRD – BIg Bench for LaRge-scale Database Grounded Text-to-SQL Evaluation (Li et al., 2023) is a large-scale cross-domain text-to-SQL benchmark in English. The dataset contains 12,751 text-to-SQL data pairs and 95 databases with a total size of 33.4 GB across 37 domains. However, BIRD still does not have many databases with large schemas: of the 73 databases in the training dataset, only two have more than 25 tables, and, of the 11 databases used for de-velopment, the largest one has only 13 tables. Again, all NL questions are phrased in the terms used in the database schemas.

Finally, the `sql-create-context`[3] dataset also addresses the text-to-SQL task, and was built from WikiSQL and Spider. It contains 78,577 examples of NL queries, SQL `CREATE TABLE` statements, and SQL Queries answering the questions. The `CREATE TABLE` statement provides context for the LLMs, without having to provide actual rows of data.

Despite the availability of these benchmark datasets for the text-to-SQL, and inspired by them, Section 3 describes a test dataset tuned to the problem addressed in this paper. The test dataset consists of a relational database, whose design is based on a real-world database, three sets of LLM-friendly views, specified as proposed in this paper, and a set of 100 test NL questions, that mimic those posed by real users, and their ground truth SQL translations.

2.2 Text-to-SQL Tools

The Spider Web site[4] publishes a leaderboard with the best-performing text-to-SQL tools. At the time of this writing, the top 5 tools achieved an accuracy that ranged from an impressive 85.3% to 91.2% (two of the tools are not openly documented). Four tools use GPT-4, as their names imply. The three tools that provide detailed documentation have an elaborate first prompt that tries to select the tables and columns that best match the NL question. This first prompt is, therefore, prone to failure if the database schema induces a vocabulary which is disconnected from the NL question terms. This failure cannot be fixed by even more elaborate prompts that try to match the schema and the NL question vocabularies, but it should be addressed as proposed in this paper.

The BIRD Web site[5] also publishes a leaderboard with the best-performing tools. At the time of this writing, out of the top 5 tools, two use GPT-4, one uses CodeS-15B, one CodeS-7B, and one is not documented. The sixth and seventh tools also use GPT-4, appear in the Spider leaderboard, and are well-documented.

The Awesome Text2SQL Web site[6] lists the best-performing text-to-SQL tools on WikiSQL, Spider (Exact Match and Exact Execution) and BIRD (Valid Efficiency Score and Execution Accuracy).

---

[3]https://huggingface.co/datasets/b-mc2/sql-create-context

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

[5]https://bird-bench.github.io

[6]https://github.com/eosphoros-ai/Awesome-Text2SQL

The DB-GPT-Hub[7] is a project exploring how to use LLMs for text-to-SQL. The project contains data collection, data preprocessing, model selection and building, and fine-tuning weights, including LLaMA-2, and evaluating several LLMs fine-tuned for text-to-SQL.

Finally, LangChain[8] is a generic framework that offers several pre-defined strategies to build and run SQL queries based on NL prompts. Section 4.1 uses LangChain to create a text-to-SQL prompt strategy.

3 A TEST DATASET FOR THE TEXT-TO-SQL TASK

This section describes a test dataset to help investigate how LLM-friendly views affect the text-to-SQL task. The dataset consists of a relational database, three sets of LLM-friendly views, and a set of 100 test NL questions, and their SQL *ground truth* translations. It should be stressed that this dataset was designed exclusively for testing text-to-SQL tools, in the context of this paper; it was not meant for training such tools.

In general, a *benchmark dataset* to test text-to-SQL tools is a pair $B = (D, \{(L_i, G_i)/i = 1, ..., n\})$, where D is a database and, for $i = 1, ..., n$, L_i is an NL question over D, and G_i is an SQL query over D that translates L_i.

3.1 The Relational Database

The selected database is a real-world relational database (in Oracle) that stores data related to the integrity management of an energy company's industrial assets. The relational schema of the adopted database contains 27 relational tables with, in total, 585 columns and 30 foreign keys (some multicolumn), where the largest table has 81 columns.

Table and column names in the relational schema do not follow a specific vocabulary. They are assigned using mnemonic terms based on an internal company specification for naming database objects, but this rule is not always followed. This scenario implies that users who do not know the relational schema have difficulty understanding the semantics of the stored data and need to turn to database specialists when retrieving data related to maintenance and integrity management processes, even if there is a description for the tables and their columns.

Also, some column values are not end-user-friendly. For example, coding values and combina-

tions of different values hide semantic information and terms relating to the business process that are not explicitly stored in the database. To overcome this situation, database experts often create SQL Functions that contain the logic to represent the semantics hidden in the hardcoded values.

In this context, it is hard for end-users, including non-human users such as LLMs, to retrieve information directly from the relational schema. Therefore, creating views over the relational schema that reflect users' terms is a task frequently performed by the company's DBAs.

3.2 The Sets of Views

To verify how the proposed approach affects the text-to-SQL task, the test dataset introduces three sets of LLM-friendly views of increasing complexity:

- *Conceptual Schema Views*: define a one-to-one mapping of the relational schema to end users' terms; the views basically rename tables and columns.

- *Partially Extended Views*: extend the conceptual schema views with new columns that pre-define joins that follow foreign keys, as well as other selected columns.

- *Fully Extended Views*: combine several conceptual schema views into a single view; the set may optionally include some conceptual schema views.

This paper argues that adopting LLM-friendly views for minimizing schema-linking problems is far simpler than creating elaborate prompt strategies.

3.3 The Set of Test Questions and Their Ground Truth SQL Translations

The test dataset contains a set of 100 NL questions, $L = \{L_1, ..., L_{100}\}$, that consider the terms and questions experts use when requesting information related to the maintenance and integrity processes.

The ground truth SQL queries, $G = \{G_1, ..., G_{100}\}$, were manually defined over the conceptual schema views so that the execution of G_i returns the expected answer to the NL question L_i. The use of the conceptual schema views facilitated this manual task, since these views use a vocabulary close to that of the NL questions.

An NL question L_i is classified into *simple*, *medium*, and *complex*, based on the complexity of its ground truth SQL query G_i, as in the Spider benchmark (extra-hard questions were not considered). The

---

[7]https://github.com/eosphoros-ai/DB-GPT-Hub
[8]https://python.langchain.com

set L contains 33 simple, 33 medium, and 34 complex NL questions.

Note that the NL questions classification is anchored on the conceptual schema views. But, since these views map one-to-one to the tables of the relational schema, a classification anchored on the relational schema would remain the same. The classification is maintained for the other sets of views, even knowing that the definition of these other sets of views might simplify the translation of some NL questions (which was one of the reasons for considering these sets of views, in the first place).

4 EXPERIMENTS

4.1 Experimental Setup

The experiments used a text-to-SQL implementation based on LangChain's SQLQueryChain[9], which automatically extracts metadata from the database, creates a prompt with the metadata and passes it to the LLM. This chain greatly simplifies creating prompts to access databases through views since it passes a view specification as if it were a table specification.

Figure 1 illustrates the prompt implemented: (A) contains instructions for the LLM; (B) defines the output format; (C) partly illustrates how the `maintenance_order` view is passed to the LLM as a `CREATE TABLE` statement; (D) shows 3 data samples from the `maintenance_order` view; and (E) passes the NL question.

The experiments tested the LangChain-based strategy with GPT-3.5-turbo-16k and GPT-4 against the 100 questions introduced in Section 3.3, separately for the database relational schema of Section 3.1 and each of the three sets of views outlined in Section 3.2.

4.2 Evaluation Procedure

Let L_i be an NL question, G_i be the corresponding SQL ground truth query, and P_i be the SQL query predicted by the text-to-SQL strategy. Let PT_i and GT_i be the tables that P_i and G_i return when executed over the database, called the *predicted* and the *ground truth* tables, respectively.

The experiments first used an automated procedure that tests if PT_i and GT_i are similar. The notion of similarity adopted neither requires that PT_i and GT_i have the same columns, nor that they have the same rows. The procedure goes as follows.

---
[9]https://docs.langchain.com

For each column of GT_i, the most similar column of PT_i is computed. The similarity of GT_i and PT_i was computed as their Jaccard coefficient; since GT_i and PT_i are sets of values, the similarity is, therefore, based on sets of values, and not on the syntactical similarity of the column names. This step induces a partial matching M from columns of GT_i to columns of PT_i. If the fraction of the number of columns of GT_i that match some column of PT_i is below a given threshold, the procedure signals that P_i is *incorrect*.

The *adjusted ground truth table* AGT_i is constructed by dropping all columns of GT_i that do not match any column of PT_i, and the *adjusted predicted table* APT_i is constructed by dropping all columns of PT_i that are not matched and permuting the remaining columns so that PC_k is the kth column of APT_i iff GC_k, the kth column of AGT_i, is such that $M(GC_k) = PC_k$.

Then, AGT_i and APT_i are compared. If their similarity is above a given threshold tq, then the procedure signals that P_i is *correct*; otherwise, it signals that P_i is *incorrect*.

The similarity of AGT_i and APT_i was computed as their Jaccard coefficient (recall that tables are sets of tuples), and the threshold tq was set to 0.95. Thus, AGT_i and APT_i need not have the same rows but, intuitively, P_i will be incorrect if APT_i contains only a small subset of the rows in AGT_i, or APT_i contains many rows not in AGT_i.

Finally, the results of the automated procedure were manually checked to eliminate false positives and false negatives.

The *accuracy* of a given text-to-SQL strategy over the benchmark B is the number of correct predicted SQL queries divided by the total number of SQL queries, as usual.

This evaluation procedure is entirely based on column and table values, not column and table names. Therefore, a text-to-SQL tool may generate SQL queries over the relational schema or any set of views, and the resulting SQL queries may be compared with the ground truth SQL queries based on the data the queries return from the underlying database.

4.3 Results

Table 1 shows the results of running the LangChain prompt for GPT-4 and GPT-3.5-turbo-16k only once over the relation schema and the three sets of LLM-friendly views, all with data samples. Columns under "**#Questions correctly translated**" show the number of NL questions per type, correctly translated to SQL (recall that there are 33 simple, 33 medium, and 34 complex NL questions, with a total of 100); columns under "**Accuracy**" indicate the accuracy results per

Figure 1: Example of a prompt.

Table 1: Results for GPT-4 and GPT-3.5 over the relation schema and the sets of LLM-friendly views, with data samples.

Model		Experimental setup	#NL questions correctly translated				Accuracy			
			Simple	Medium	Complex	Total	Simple	Medium	Complex	Overall
GPT-4	1	Relacional schema	22	11	8	41	0,67	0,33	0,24	0,41
	2	Conceptual schema views	32	18	15	65	0,97	0,55	0,44	0,65
	3	Partially extended views	30	25	19	74	0,91	0,76	0,56	*0,74*
	4	Fully extended views	28	20	18	66	0,85	0,61	0,53	0,66
GPT-3.5	1	Relacional schema	24	8	4	36	0,73	0,24	0,12	0,36
	2	Conceptual schema views	27	16	5	48	0.78	0.18	0.18	0,48
	3	Partially extended views	26	18	8	52	0,79	0,55	0,24	*0,52*
	4	Fully extended views	28	13	6	47	0,85	0,39	0,18	0,47

NL question type, and the overall accuracy.

Overall, the accuracy results with GPT-4 were much better than those with GPT-3.5-turbo-16k; if we compare the best accuracy results (the gray cells), GPT-4 achieved an overall accuracy 22% better than GPT-3.5-turbo-16k.

Let us concentrate on the accuracy results with GPT-4. A comparison between the results of Experiments 1 and 2 indicates that the overall accuracy achieved with the conceptual schema views was 24% better than that achieved with the relational schema. This means that simply renaming the tables and columns to terms closer to the end-user vocabulary sufficed to improve accuracy substantially.

Now, a comparison between the results of Experiments 1 and 3 captures a more subtle improvement. The partially extended views simplify the text-to-SQL task by eliminating joins in certain medium and complex questions. The overall accuracy achieved with these views was substantially better (33% better) than that achieved with the relational scheme.

Also, note that Experiment 3 failed to translate two more simple NL questions than Experiment 2. One explanation is that LLMs are non-deterministic; if the experiments were repeated several times, Table

1 could report slightly different accuracy results for Experiments 2 and 3.

A comparison between the results of Experiments 3 and 4 shows a decrease of 8%. Indeed, the fully extended views save more joins, facilitating the text-to-SQL task, but they require passing much larger view specifications in the prompt. Furthermore, the definition of a fully extended view, which combines several views, requires renaming several columns, which may create columns with similar names. In conjunction – views with many columns and similar column names – confuse the LLM, leading to ambiguous matches with an NL question.

In summary, the results suggest that the partially extended views, with just a few extra columns that pre-define joins, is a better alternative than fully extended views, that combine several views. These views also proved to be a much better alternative than using the relational schema or the set of conceptual schema views. From a broader perspective, the accuracy increases when one moves from prompting the LLM with the relational schema to prompting the LLM with LLM-friendly views and data samples, corroborating the position argued in this paper.

5 CONCLUSIONS

This paper argued that the target user of a database specification should be viewed as the LLM, when improving the performance of a text-to-SQL strategy.

From the point of view of metadata, this position quite simply asks to create a database specification that defines a vocabulary close to that of the NL questions to be submitted for translation to SQL. This specification can be easily implemented with familiar views. As for the data, this position requires creating a set of constructs that try to capture the data semantics. This can be far more complex and would require knowledge of the LLM API capabilities if one wants to go beyond providing data samples. Fortunately, LLMs are "few-show learners", that is, they can learn to perform a new language task from only a few examples (Brown et al., 2020). Thus, providing a few data samples per table helps.

To help convince the reader of the soundness of the position, the paper introduced a test dataset, with three sets of LLM-friendly views of increasing complexity, and 100 NL questions and their translation to SQL. Using the benchmark dataset, the experiments suggested that there is a dramatic increase in accuracy when one moves from prompting the LLM with the relational schema to prompting the LLM with LLM-friendly views and data samples, as argued in the paper.

Views also help reduce the SQL query complexity by including additional columns with pre-defined joins. However, the larger the view, the more tokens its definition would consume, and LLMs typically limit the number of tokens passed. Also, the LLM may get lost when the views have many columns.

Finally, there is room for further improvement. For example, the LLM-friendly views used in the experiments were created by inspecting the database documentation and by mining a log of user questions. Albeit this process was tedious but not too difficult, further work will focus on a tool that automatically creates views on the fly, depending on the NL question submitted, along the lines of the tool described in (Nascimento et al., 2023).

ACKNOWLEDGEMENTS

REFERENCES

Affolter, K., Stockinger, K., and Bernstein, A. (2019). A comparative survey of recent natural language interfaces for databases. *The VLDB Journal*, 28.

Brown, T. B. et al. (2020). Language models are few-shot learners. In *Proc. Advances in Neural Information Processing Systems 33*. doi:10.48550/arXiv.2005.14165.

Gan, Y., Chen, X., Huang, Q., Purver, M., Woodward, J. R., Xie, J., and Huang, P. (2021a). Towards robustness of text-to-sql models against synonym substitution. *CoRR*, abs/2106.01065.

Gan, Y., Chen, X., and Purver, M. (2021b). Exploring underexplored limitations of cross-domain text-to-sql generalization. In *Conference on Empirical Methods in Natural Language Processing*.

Katsogiannis-Meimarakis, G. and Koutrika, G. (2023). A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, 32(4):905–936.

Kim, H., So, B.-H., Han, W.-S., and Lee, H. (2020). Natural language to sql: Where are we today? *Proc. VLDB Endow.*, 13(10):1737–1750.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474.

Li, J. et al. (2023). Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *arXiv preprint arXiv:2305.03111*.

Nascimento, E. R., Garcia, G. M., Feijó, L., Victorio, W. Z., Lemos, M., Izquierdo, Y. T., Garcia, R. L., Leme, L. A. P., and Casanova, M. A. (2024). Text-to-sql meets the real-world. In *Proc. 26th Int. Conf. on Enterprise Info. Sys.*

Nascimento, E. R., Garcia, G. M., Victorio, W. Z., Lemos, M., Izquierdo, Y. T., Garcia, R. L., Leme, L. A. P., and Casanova, M. A. (2023). A family of natural language interfaces for databases based on chatgpt and langchain. In *Proc. 42nd Int. Conf. on Conceptual Modeling – Posters&Demos*, Lisbon, Portugal.

Yu, T. et al. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proc. 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921.