Towards More Realistic Natural Language Queries in Text-to-SQL Benchmarks

Charlotte Dörschner
TU Darmstadt
Darmstadt, Germany
charlotte@doerschner.com

Frank Jäkel
TU Darmstadt
Darmstadt, Germany
jaekel@psychologie.tu-darmstadt.de

Carsten Binnig
TU Darmstadt
Darmstadt, Germany
DFKI
Darmstadt, Germany
hessian.AI
Darmstadt, Germany
carsten.binnig@cs.tu-darmstadt.de

Abstract

Advances in large language models have significantly improved performance on Text-to-SQL benchmarks. However, existing benchmarks collect natural language queries (NLQs) through methods such as paraphrasing SQL queries, resulting in queries that remain strongly SQL-like. While there have been recent trends towards making Text-to-SQL benchmarks more realistic, it is still unclear how users, who may not be proficient in SQL, naturally try to query data. Therefore, in this paper, we present the results of a study with 80 participants, who formulated queries to answer broad questions with the help of a database. The study produced a corpus of 887 NLQs, which we analyzed to identify systematic differences between benchmark-style queries and those posed by our participants. Based on our findings, we discuss implications for the design of more realistic Text-to-SQL benchmarks.

ACM Reference Format:
Charlotte Dörschner, Frank Jäkel, and Carsten Binnig. 2026. Towards More Realistic Natural Language Queries in Text-to-SQL Benchmarks. In *Workshop on Human-In-the-Loop Data Analytics (HILDA '26), May 31–June 05, 2026, Bengaluru, India.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3814573.3814945

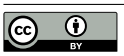
1 Introduction

Text-to-SQL Benchmarks suggest advances. In recent years, driven by large language models (LLMs), significant advances have been made towards enabling database access via natural language. Text-to-SQL benchmarks quantify this progress by testing the ability of natural language interfaces for databases (NLIDBs) to translate natural language queries (NLQs) into SQL queries. Considering the performance of current NLIDBs on the Text-to-SQL task, benchmarks suggest that commercial use of NLIDBs is within reach. For instance, as of today, the highest-ranking system on the BIRD leaderboard achieves an impressive 80% accuracy, not far behind human SQL experts [12].

Benchmarks are under debate. However, concerns have been raised about the validity of Text-to-SQL datasets, suggesting that they do not reflect real-world challenges [e.g. 4, 8, 16, 17]. While some attempts have sought to make benchmarks more realistic, for example, by introducing ambiguity into the NLQs, benchmarks still do not systematically account for what non-expert users actually want to query. Instead, NLQs are collected in ways that are "convenient" to obtain pairs of NLQ/SQL statements to measure the accuracy of translation. For example, benchmarks like WikiSQL [26], Spider 2.0 [11], or BEAVER [2] substitute spontaneous NLQs for descriptions of SQL queries. While this approach prevents NLQs that cannot be translated into SQL, it evokes NLQs that are SQL-like, and thus might be far from how users actually would query data. Meanwhile, other benchmarks introduce specific variations in the method for collecting NLQs to cope with known shortcomings of Text-to-SQL benchmarks. For example, KaggleDBQA [10] masks column names with descriptions to avoid explicit schema references in the NLQs, which were previously criticized [3, 15, 21, 22]. Some other benchmarks, like Ambrosia [18] or AmbiQT [1], meet the demand for ambiguous examples by simulating predefined linguistic phenomena, but it is again unclear whether the resulting NLQs represent real-world queries of how users want to query data.

How do non-experts formulate NLQs? The problem underlying all these uncertainties in benchmark design is that we have little understanding of the problem space for NLIDBs, that is, what NLQs are to be expected in a realistic usage scenario. In particular, while benchmarks have evolved and made NLQs more complex, the NLQs in benchmarks are all somewhat SQL-like. They formulate information needs such as "What is the average duration of stay of a patient". Such queries map well to SQL. But not all information needs do and it is often non-trivial to come up with the right sequence of SQL queries to answer a real-world question. So far, there is hardly any research on NLQs that examines how users without SQL knowledge actually ask such questions.

Our Contributions. This paper presents results from a study by database and cognitive science researchers at TU Darmstadt investigating how users query data, as querying data is fundamentally a cognitive process. For the study, we designed a questionnaire comprising textual database descriptions and associated problem scenarios to elicit NLQs from individuals. In the study that we present, we asked 80 individuals with varying levels of SQL expertise to fill out this questionnaire. In total, we collected 887 queries that allow us to identify typical problems in NLQ formulations. In the following, we present the design and results of this study. Section 2 first provides a brief overview of existing Text-to-SQL benchmarks, highlighting their methodologies and resulting shortcomings. In Sections 3 and 4, we present our study setup and results. Finally, drawing on our new insights, we outline a roadmap for future development of Text-to-SQL benchmarks in Section 5.



HILDA '26, Bengaluru, India

ACM ISBN 979-8-4007-2715-3/26/05
https://doi.org/10.1145/3814573.3814945

Table 1: Overview of approaches of Text-to-SQL benchmarks to elicit benchmark queries. The upper part shows the core benchmarks for NLIDBs, and the lower part shows other benchmarks that extend core benchmarks in several directions.

Benchmark	NLQ Source	Annotation Pipeline	Ambiguity	Invalidity	Ambiguity/Invalidity Source
WikiSQL [26]	human	SQL → NLQ			NA
KaggleDBQA [10]	human	NLQ → SQL	x[a]		NA
BIRD [13]	human	NLQ → SQL			NA
Spider [24]	human	NLQ ↔ SQL	x[b]		NA
Spider 2.0-lite/snow [11]	human	SQL → NLQ			NA
BEAVER [2]	human + LLM	SQL → NLQ			NA
Ambrosia [18]	human	SQL → NLQ	x		predefined
AmbiQT [1]	Spider	NA	x		DB manipulation
CoSQL [23]	human	NLQ → SQL	x	x	spontaneous
SParC [25]	human	NLQ → SQL			spontaneous

[a] according to [4]
[b] according to [7, 21]

2 Benchmarks Overview

Existing Text-to-SQL benchmarks employ different data-collection processes for eliciting NLQ/SQL pairs. In the following, we first discuss how core Text-to-SQL benchmarks collected NLQ/SQL pairs before we discuss other benchmarks that extend the core benchmarks in several directions. Table 1 provides an overview of benchmarks and summarizes the different methods and how they were constructed.

2.1 Core Text-to-SQL Benchmarks

Traditional benchmarks create NLQs with a Text-to-SQL translation task in mind and thus produce NLQs that are highly SQL-like and simplistic in other dimensions, as they use database references (i.e., table names) directly in the NLQs.

Annotation pipelines. To collect NLQ/SQL pairs, different methods have been used. Table 1 summarizes the methods (see column *Annotation Pipeline*) by categorizing them into two major approaches that are employed in practice. In the first approach, NLQ → SQL, the dataset curation method closely corresponds with the Text-to-SQL task. That is, NLQs are collected in a first, independent step and only subsequently get translated into SQL. BIRD and KaggleDBQA are examples of benchmarks that employed this annotation pipeline. In the second approach, SQL → NLQ, the dataset curation method no longer corresponds with Text-to-SQL but rather SQL-to-Text. Here, NLQs are no longer spontaneous expressions of information needs but descriptions of previously collected SQL queries instead. Benchmarks like WikiSQL, Spider 2.0, or BEAVER employ this inverted annotation pipeline to avoid NLQs that cannot be translated into SQL and thus to reinforce the translational correspondence between NLQ/SQL pairs. However, this virtually leaves no ambiguities in NLQs for semantic parsing. Consequently, this annotation pipeline differs from the previous one in that the resulting NLQs are even more SQL-like. The same applies to Spider, where NLQs and SQL queries were generated simultaneously (NLQ ↔ SQL) because the annotators already had the translation into SQL in mind when creating NLQs.

Database descriptions. Another issue of early Text-to-SQL benchmarks was that users who construct benchmark queries are typically equipped with complete information about the database schema. As such, queries contain direct links to schema elements by exact names, as, for example, in WikiSQL and Spider [3, 15, 21, 22]. Newer benchmarks are taking steps to address this issue. For example, KaggleDBQA substitutes original column names in the database descriptions given to users to prevent them from being mentioned directly in NLQs.

2.2 Other Benchmarks & Extensions

Task complexity and contextual dependencies. To make the Text-to-SQL task more challenging, benchmarks use databases with highly normalized schemas or dirty values (e.g., BIRD). As a result, SQL queries become more complex, requiring many joins or data manipulations. Other benchmarks, in turn, attempt to increase complexity via the NLQs. For example, in SParC sequences of NLQs relate to each other, for example, by refining a constraint from a previous NLQ. By introducing contextual dependencies, SParC conceptualizes Text-to-SQL as a multi-turn task. However, given the context of the conversation history, SParC still assumes NLQs that always allow for a translation into SQL. A new generation of benchmarks, on the other hand, even goes so far as to break with this fundamental assumption by introducing ambiguous NLQs. This idea is also reinforced by the fact that ambiguities seem virtually impossible to suppress in NLQs, as revealed by hidden assumptions in corresponding SQL queries [4, 7, 21].

Ambiguous and invalid NLQs. Benchmarks like Ambrosia or AmbiQT map ambiguous NLQs to multiple non-equivalent SQL queries that constitute possible interpretations. For example, Ambrosia relied on crowd workers paraphrasing previously automatically generated examples using an inverted annotation pipeline. AmbiQT, on the other hand, reused the Spider dataset and only manipulated underlying database schemata to introduce ambiguities indirectly. Instead of one-to-many mappings as in ambiguity benchmarks, conversational benchmarks approach ambiguity through clarifying dialogues with the user. For CoSQL, dialogues were produced using a so-called Wizard-of-Oz paradigm, that is, real-time conversation with a human that simulates a perfect system. Interestingly, not only did NLQs require clarifications, but there were also invalid inputs – inputs that cannot be translated into SQL queries.

2.3 Discussion

While several shortcomings of benchmarks have been addressed with the previously named extensions, it remains unclear why benchmarks should reflect what users actually want to ask when

querying databases. In the following, we summarize the findings from the discussions before.

NLQs are aimed for translation to SQL. First, we have already portrayed how the chosen methodology is closely linked to how benchmark creators conceptualize the problem of natural language access to databases. As traditional benchmarks assume a translation task, they view natural language and SQL as interchangeable representations. However, it is unclear if the resulting NLQs are thus realistic, and users who do not have this task in mind would ask questions about databases in a very different way.

Database descriptions and schema references. Second, as NLQs in benchmarks were found to suffer from explicit schema references, benchmark creators masked the names of schema elements to modify annotator knowledge about the database. However, we still lack an understanding of what vocabulary will be used in NLQs if users have no structural information about the database.

Representation of ambiguous and invalid NLQs. Lastly, newer benchmarks set up interesting challenges by featuring ambiguous or invalid NLQs, as presented in Table 1. Nevertheless, it is unclear whether these challenges are also relevant with respect to the real world. For example, ambiguous NLQs are typically only represented with predefined types, and invalidity only occurs in limited form. Not only is there a lack of knowledge about how ambiguity and invalidity manifest in NLQs, but there is also no uniform taxonomy for discussing the representation of different semantic challenges in benchmarks.

3 Study Design

In this section, we present the design of our study to understand how users want to query databases with natural language. Our study comes with important differences from benchmarks: Most importantly, instead of having the task of Text-to-SQL in mind when collecting NLQs, we ask individuals how they attempt to solve problems by asking questions to a database. Moreover, we aim not to provide participants with details about the database schema and involve individuals with varying levels of SQL expertise. In the following, we describe the exact methodology that we used to elicit NLQs in such an arguably more realistic scenario.

3.1 Methodology of Study

Participants. To involve individuals with varying SQL expertise, we recruited 80 participants from students enrolled in a university course on information management. We collected data in two waves, each lasting two weeks. The first wave started at the beginning of the semester where students did not know the relational model and SQL yet, and the second wave followed four weeks later, after the lectures that introduced these topics. Participation was voluntary, and the study was approved by the local ethics board.

Questionnaire. To elicit NLQs, we employed an approach using problem scenarios inspired by Shneiderman [20]. With this approach, a database is briefly described to define the context for a problem. This description only gives individuals a rough idea of what information is contained in the database, but not what the schema looks like exactly. We decided on a company database (Employee domain) and a publications database (Scholar domain) to run two versions of the survey in parallel. This was done to increase the generalizability of our findings. As an example, the Scholar database was described as follows[1]:

> *Imagine you have unrestricted access to a database of scientific publications. The database contains the following information:*
> - *Publication details (title, year, journal)*
> - *The names of all authors*
> - *The names of all scientific journals*
> - *Which publications were written by which authors*
> - *Which publications are cited in which publications*
> - *The keywords of each publication*

Subsequently, participants were instructed to come up with questions for the database that would help to solve a problem. For example, one of the problem scenarios was described as follows[1]. The complete task material may be found in the appendix.

> *You want to collaborate with other authors. You can consider all authors for this purpose. To help you decide who you want to work with, you can examine the authors' publications and previous collaborations by asking questions to the database.*

Given such a problem description, a maximum of ten NLQs could be entered into the online survey form. Overall, we constructed three problem scenarios for each database. Apart from the problem scenarios, our study also involved other tasks, for example, rewriting of given NLQs. However, for this paper, we focus on the NLQs produced for the problem scenarios.

Execution of Study. Participants were pseudo-randomly assigned to one out of the two domains so that they worked on three problem scenarios with a fixed database for context. If a subject participated in both waves, the assigned domain was switched to the other domain in the second survey round. In doing so, the subjects were presented with a new database and new associated tasks in each survey round.

3.2 Corpus Annotation

With the methodology outlined above we assembled a corpus of NLQs that captures (part of) the problem space of NLIDBs. In order to draw conclusions from the collected data, we labeled the corpus with annotations[2] that allow us to categorize NLQs (or spans thereof) on various levels (syntax and semantics as well as the vocabulary used). This allows us not only to study the properties of the NLQs at a wider spectrum, but also to derive a taxonomy of NLQs. Below, we present the result of analyzing these data.

4 Study Results

In total, we collected a corpus of 887 NLQs (n = 491 for Employee and n = 396 for Scholar) from 80 individuals. On average, each problem scenario was solved by 51 subjects (Min = 48, Max = 55) with typically two questions (M = 2.89, SD = 2.35, Mode = 2), resulting in between 125 and 179 NLQs per problem scenario. From the participants at the first survey round (n = 64), 65.6% reported to have prior knowledge of databases and SQL outside of class. Along with the data collection in waves, this self-report suggests that the corpus represents individuals ranging from absolute beginners to intermediate-level learners.

[1] translated from German
[2] enabled by the open-source tool INCEpTION [9]

4.1 Syntax of NLQs

We first analyze the surface characteristics of the NLQs. For this, we used a manual taxonomy-driven annotation of the syntactic aspects of clause structure and speech acts of NLQs. With this syntactic annotation we intend to inspect whether NLQs in our user study differ significantly from the benchmark NLQs in properties such as text length and complexity. Apart from that, we also used this first step to define the annotation spans that constituted a self-contained query. This was necessary as in some cases subjects broke down a query into multiple sentences or input fields in the online survey form. Lastly, we also analyzed whether utterances are grammatically correct or not. For this, we decided to use a relaxed annotation of formal linguistic mistakes (i.e., punctuation was generally ignored, and capitalization was only taken into account if it changed the meaning of a word).

NLQs are short and simple. With 9.74 tokens in our study (whitespace split, SD = 4.27, Min = 1, Max = 35) on average, NLQs can be considered rather short. In contrast, NLQs from Spider or BIRD were reported elsewhere [15] with an average length of well over 10 tokens. Moreover, most NLQs were phrased as interrogatives (89.8%) with a simple clause structure (81.4%). This hints that in reality, NLQs are shorter than in benchmarks, and thus, there might be a significant amount of information missing or implicit in real-world NLQs. For example, queries like "Which departments have the least diversity[?] Name 5[.]" constitute very compact representations of otherwise complex SQL queries.

Many NLQs are ungrammatical. Based on our relaxed annotation of formal linguistic mistakes, we identified 17.5% of NLQs as ungrammatical. Most of them contained spelling errors or grammar errors, but also more severe threats to query understanding like ill-formed sentences (n = 35). For example, the desired output for "Which previous job titles were hold by which employees, whose current are not these?" is hard to grasp. Interestingly, the phenomenon of formal linguistic mistakes has not yet been explicitly considered in Text-to-SQL benchmark design. However, other reports exist that emphasize the prevalence of formal linguistic mistakes in NLQs [e.g. 5]. Text-to-SQL benchmarks could easily introduce such linguistic mistakes with the help of post-hoc dataset perturbations.

4.2 Semantics of NLQs

In Section 2, we introduced recent benchmarking trends involving ambiguous and invalid NLQs. As such, we wanted to better understand if (and which forms of) ambiguity and invalidity appear in real-world NLQs. To approach this, we first divided our corpus into valid, ambiguous, and invalid NLQs. We defined an NLQ as invalid if we find it impossible to come up with a translation into SQL, given the database. We defined an NLQ as ambiguous if there is uncertainty about the expected answer, that is, multiple non-equivalent SQL translations qualify as an answer. An expected answer is a query result that does not contain information beyond what is required to answer a question (e.g., no helpful but unsolicited return columns). Otherwise, an NLQ is considered valid.

Valid NLQs are not the default. Figure 1 shows the results of the annotation of the corpus with NLQ types. Surprisingly, NLQs that are eligible for a translation into SQL only made up a little more than a third of the corpus (36.7% from Employee and 35.4% from Scholar). From the total of 887 collected queries, 566 (63.9%) of instances presented at least one semantic challenge. This finding suggests a significant semantic gap between NLQs and SQL, fundamentally challenging the conceptualization of querying data as a Text-to-SQL translation task. Instead, the NLQs that we observed in our study often represent utterances of complex problems to be solved that involve a sequence of SQL queries.

Consistent findings over domains and time. It is particularly striking that the distribution of NLQ types is highly consistent. As shown in Figure 1, the label counts are very similar between the two domains that we used in our study. Interestingly, even when considering the temporal dimension and splitting the corpus into NLQs collected before (n = 561) vs. after (n = 326) the SQL lectures, the distribution remains consistent. This was contrary to our expectations; the proportion of valid NLQs did not increase at the second survey round (n = 116, 35.6%) compared to the first round (n = 205, 36.5%), although individuals should have improved in SQL expertise through attending a university course. Ambiguous NLQs continued to be predominant (n = 283 pre vs. n = 151 post), and invalid NLQs remained present (n = 73 pre vs. n = 59 post).

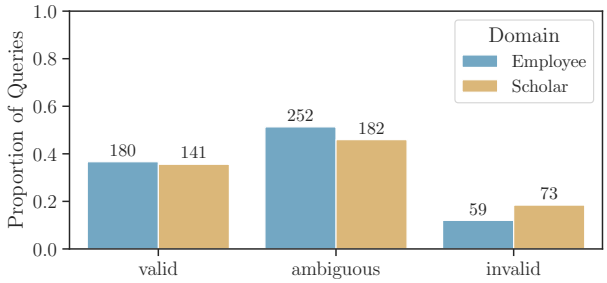


Figure 1: Distribution of NLQ types based on SQL-translatability in the corpus. Query proportions are normalized within each domain.

Taxonomy semantic challenges for Text-to-SQL. Accompanying the annotation of NLQ types, we identified recurring issues hindering Text-to-SQL translation. We did so by following the principles of grounded theory building [see 6]. In this approach, annotation is viewed as an iterative process in which the derivation of categories from the data and the adjustment of the resulting tag-set are performed alternately, until the tag-set can adequately describe the data. We implemented the process so that the first author analyzed the corpus to propose a tag-set, which the team then reviewed using sample data. The resulting tag-set forming a taxonomy of semantic challenges is presented in Table 2. As apparent from the table, seven challenges were related to ambiguous NLQs and four were related to invalid NLQs. As shown in Figure 2, the most frequently annotated challenges were "missing context", "undefined return columns", "unclear filter criteria", and "invalid database reference". In the following, we will discuss these important challenges in more detail.

Most NLQs are ambiguous. With 51.3% from the Employee domain and 46.2% from the Scholar domain, every other NLQ was considered ambiguous. In some cases, individuals assumed information to be stored in a hypothetical NLIDB system about themselves (e.g., preferences) or the current subject of interest. As can be seen from Table 2, we annotated these NLQs with "missing context", albeit being fully aware that the issue must be attributed to our

Table 2: Taxonomy of semantic challenges for Text-to-SQL related to ambiguous and invalid NLQs. The results from reviewing the representation of semantic challenges in existing Text-to-SQL benchmarks are presented in the last column.

Semantic Challenges	Description	Representative Example[a]	Benchmark Occurrence
	Ambiguous Queries		
unclear filter criterion	The query uses vague expressions to define a filter criterion or is a top-k query that does not specify k (unless k=1).	"How many old and new employees are there (date of hire)?"	
ambiguous schema reference	The query contains an ambiguous expression that could refer to multiple elements of the database schema that differ semantically.	"Provide me with a list containing 10 frequently cited sources."	Ambrosia, AmbiQT, KaggleDBQA[b]
missing context	The query refers to a specific entity, value, or previous query result that is not explicitly provided in the query itself and must be inferred from outside information.	"Which journal has publications that match my keywords?"	SParC, CoSQL
undefined aggregation method	The query requires aggregation but it is not clear which aggregation function or grouping level the user intended.	"What are the five top paid departments?"	
abstract concept reference	The query contains an abstract concept that may or may not be represented by the available data, depending on its definition.	"Which journals are scientific?"	KaggleDBQA[b]
undefined return columns	The query requests an entity or a list of entities that could be represented by more than one descriptive column (human-understandable attributes, ID values do not count) but the user does not specify which columns should be returned.	"Which publications were released in 2025?"	Spider
incomprehensible query intent	The query is phrased in such a way that not even its general intent is understandable.	"The number of which in publications cited publication is the highest?"	CoSQL
	Invalid Queries		
invalid database reference	The query mentions entities, relationships, or attributes that definitively cannot be mapped to any database schema element.	"Which author has the best average reader rating?"	CoSQL
insufficient date granularity	The query references a date column that exists but lacks the level of detail required for the intended analysis.	"Which publications have been released in the last 14 days?"	
illogical request	The query assumes impossibilities with respect to the mini world the database is representing, e.g. violates world laws or misuses modifiers.	"Which publication has the highest number of publications?"	
unsupported request	The query requests an output that is not supported by database querying alone, e.g. a qualitative statement.	"What is my personal worth?"	CoSQL

[a]original corpus data translated from German
[b]according to [4]

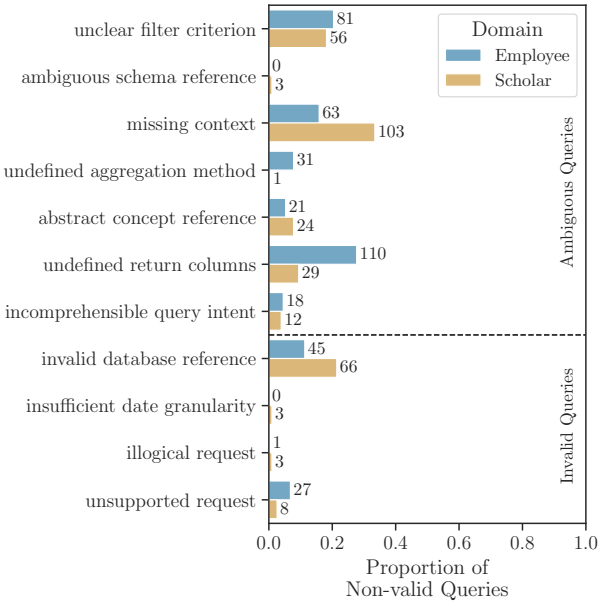


Figure 2: Distribution semantic challenges for Text-to-SQL related to ambiguous and invalid NLQs in the corpus. Query proportions are normalized by the total number of ambiguous and invalid NLQs within each domain. Note that multiple semantic challenges could manifest in a single NLQ.

method of study. Nevertheless, it is interesting to observe that even in the static setting of a questionnaire, subjects display expectations that only conversational NLIDBs could possibly fulfill. Apart from that, we identified two critical sources for ambiguity in NLQs. First, people often omit specifying which columns to return from an entity, which is represented by multiple descriptive columns in the database. As shown in Table 2, we named this challenge "undefined return columns". For example, the database description introduced the entity type "publications" having the attributes "title", "year", and "journal". However, individuals tend to denote a set of entities with "Which publications …" (cf. example in Table 2) rather than mentioning specific attributes, for example, "List the titles of the publications that …". Second, humans use vague expressions to define filter criteria that make it impossible to determine which tuples qualify for the result. As shown in Table 2, we named this challenge "unclear filter criterion". Vague expressions are, for example "old" or "new" to filter for hire date in the Employee database (cf. example in Table 2).

Invalid NLQs occur frequently. A significant proportion of NLQs cannot be translated into a valid SQL query (12.0% from Employee, 18.4% from Scholar). The most prominent reason is the occurrence of invalid database references. It is astonishing how this affects roughly one out of ten queries from our corpus. Typically, participants hallucinate attributes of existing entities or introduce an entirely new entity type, often inspired by what has been mentioned in the problem scenario. For example, participants referred to employed editors, reader reviews, or word counts, which are semantically related to the Scholar domain but were never specified as database content.

Current benchmarks lack NLQ variety. In a final step, we assessed whether the identified semantic challenges related to ambiguous and invalid NLQs are already represented in current benchmarks. We did this by examining how the creators of the benchmarks in Section 2 portrayed their dataset in the associated publications through descriptions of considered phenomena or examples of NLQs. As Table 2 (see column *Benchmark Occurrence*) illustrates, current benchmarks fail to cover the variety of ambiguous NLQs that we observed here. Moreover, currently, completely underrepresented are invalid NLQs, with only CoSQL featuring invalid database references or unsupported requests.

4.3 Vocabulary of NLQs

As mentioned earlier in Section 2, Text-to-SQL benchmarks face the criticism of simplifying schema linking with explicit mentioning of database elements in NLQs. As such, we were interested in the extent to which the database description in our survey was linked to entity references in the NLQs. As discussed before, we avoided having explicit links in the database and the problem description that users had to think about. Based on the user utterances of our study, we performed semi-automatic entity linking[2] on the NLQs to analyze lexical variance in schema references. We focused on the entities as introduced in the database descriptions related to tables. For example, the set of entities for Scholar consisted of "Autor" (author), "Publikation" (publication), "Schlagwort" (keyword), and "Zeitschrift" (journal).

Entity references are biased. Overall, we linked 1558 words ($n = 828$ for Employee and $n = 730$ for Scholar) to database-related entities in the corpus. Based on the word frequencies for entity references, we observed that the participants largely used the words as they occurred in the database descriptions. Even when our chosen terms in the description were unusual, they were rarely substituted. For example, other terms such as "Manager" or "Abteilungsleiter" would have been more typical for everyday language use than our choice of "Führungskraft" to describe the entity of a manager in the Employee domain. Still 84.9% of the entity references were lexemes of "Führungskraft". For the other annotated entities, too, the proportion of lexemes related to the prescribed word never fell below 60%. Based on our findings, we conclude that schema references are biased, and the bias may be controlled by how a database is introduced to annotators.

5 Future Roadmap

In this paper, we explored the problem space of NLIDBs. For this purpose, we collected a corpus of NLQs from individuals who tried to solve problems with limited information about the database schema and varying SQL knowledge. We would like to conclude by providing the database community with specific recommendations for benchmark design based on our findings.

Recommendation 1: Collect NLQs first. Previously, benchmark creators have modified the annotation pipeline liberally, meaning that they substituted spontaneous NLQs for descriptions of SQL queries. However, our findings emphasize that spontaneous NLQs are rather short and simple interrogatives than complex utterances. Thus, our findings suggest that NLQs are by far not equivalent to descriptions of SQL queries but leave out many details. Therefore, we advise benchmark creators to refrain from inverting the annotation pipeline into an SQL-to-Text task during dataset curation. Another advantage of collecting the NLQs in a first independent step is that individuals without SQL expertise can be involved in data collection. After all, individuals without SQL expertise, who might not even understand the general concept of a database, constitute exactly the target user group for NLIDBs.

Recommendation 2: Use different task versions to elicit NLQs. To counteract explicit schema references in NLQs, benchmark creators have substituted names of schema elements with descriptions. However, our findings suggest that individuals tend to promptly adopt the terms as introduced in the database descriptions, resulting in a strong vocabulary bias. Consequently, benchmark creators might be merely substituting the issue of explicit schema references with biased schema references. Thus, we propose that benchmark creators should use different task versions to elicit NLQs, so that the entirety of available annotators is divided into subgroups that each rely on a distinct way to describe a database. Although biases will still occur within a group, they will differ between groups. This constitutes a very simple but effective method for generating more variance in schema references.

Recommendation 3: More focus on semantic challenges. Our findings underline the importance of ambiguous and invalid NLQs for the problem space of NLIDBs. However, many of the semantic challenges we found in our study are not covered in benchmarks as shown in Table 2. Thus, to prepare NLIDBs for real-world challenges, Text-to-SQL benchmarks must focus on ambiguous and invalid NLQs. Especially LLMs, which constitute the foundation for most modern Text-to-SQL models, show difficulties in dealing with ambiguity [4, 19] and invalidity [4]. Yet, handling ambiguity and invalidity should be considered a key competence that distinguishes actual NLIDBs from simple Text-to-SQL translators. And such competences can only be assessed with conversational Text-to-SQL benchmarks. CoSQL has already implemented a gold standard to curate a dataset of realistic dialogues with a Wizard-of-Oz paradigm. However, one issue that remains unresolved is that models must learn not only to identify ambiguous or invalid NLQs, but also to repair them in collaboration with the user. Based on a task from our study other than problem-solving, we can anecdotally report that individuals are generally not able to repair ambiguity or invalidity themselves. Instead, users depend on machine assistance that could be realized via multiple choice questions [e.g. 14] or proposed NLQ reformulations [e.g. 19]. However, such expansions of the traditional Text-to-SQL task require novel techniques for model evaluation that remain a subject for future research.

Conclusion. As demonstrated in our study, natural language queries (NLQs) in real-world settings differ substantially from those found in Text-to-SQL benchmarks. A key finding is that real-world NLQs introduce challenges that are not captured by existing benchmarks. Most notably, they are often not directly translatable into a single SQL query. Instead, NLQs typically express high-level problem statements rather than precise query formulations. Consequently, users must decompose their abstract information needs into a sequence of concrete SQL queries. These observations suggest that future research should move beyond focusing solely on direct natural language–to–SQL translation and instead address the broader cognitive processes involved in solving data-centric problems.

References

[1] Adithya Bhaskar, Tushar Tomar, Ashutosh Sathe, and Sunita Sarawagi. 2023. Benchmarking and Improving Text-to-SQL Generation under Ambiguity. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP '23)*. Association for Computational Linguistics, 7053–7074. doi:10.18653/v1/2023.emnlp-main.436

[2] Peter Baile Chen, Fabian Wenz, Yi Zhang, Devin Yang, Justin Choi, Nesime Tatbul, Michael Cafarella, Çağatay Demiralp, and Michael Stonebraker. 2025. *BEAVER: An Enterprise Benchmark for Text-to-SQL*. arXiv:2409.02038 doi:10.48550/arXiv.2409.02038

[3] Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-Grounded Pretraining for Text-to-SQL. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 1337–1350. doi:10.18653/v1/2021.naacl-main.105

[4] Avrilia Floratou, Fotis Psallidas, Fuheng Zhao, Shaleen Deep, Gunther Hagleither, Wangda Tan, Joyce Cahoon, Rana Alotaibi, Jordan Henkel, Abhik Singla, Alex Van Grootel, Brandon Chow, Kai Deng, Katherine Lin, Marcos Campos, K. Venkatesh Emani, Vivek Pandit, Victor Shnayder, Wenjing Wang, and Carlo Curino. 2024. NL2SQL is a solved problem... Not!. In *14th Conference on Innovative Data Systems Research (CIDR '24)*.

[5] Jonathan Fürst, Catherine Kosten, Farhad Nooralahzadeh, Yi Zhang, and Kurt Stockinger. 2025. Evaluating the Data Model Robustness of Text-to-SQL Systems Based on Real User Queries. In *Proceedings 28th International Conference on Extending Database Technology (EDBT '25)*. OpenProceedings.org, 158–170. doi:10.48786/EDBT.2025.13

[6] Barney G. Glaser and Anselm L. Strauss. 1998. *Grounded theory - Strategien qualitativer Forschung*. Huber, Bern [u.a.]. 270 S. pages.

[7] Moshe Hazoom, Vibhor Malik, and Ben Bogin. 2021. Text-to-SQL in the Wild: A Naturally-Occurring Dataset Based on Stack Exchange Data. In *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog '21)*. Association for Computational Linguistics, 77–87. doi:10.18653/v1/2021.nlp4prog-1.9

[8] Tengjun Jin, Yoojin Choi, Yuxuan Zhu, and Daniel Kang. 2026. Text-to-SQL Benchmarks are Broken: An In-Depth Analysis of Annotation Errors. In *16th Conference on Innovative Data Systems Research (CIDR '26)*.

[9] Jan-Christoph Klie, Michael Bugert, Beto Boullosa, Richard Eckart de Castilho, and Iryna Gurevych. 2018. The INCEpTION Platform: Machine-Assisted and Knowledge-Oriented Interactive Annotation. In *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*. Association for Computational Linguistics, 5–9. http://tubiblio.ulb.tu-darmstadt.de/106270/

[10] Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. KaggleDBQA: Realistic Evaluation of Text-to-SQL Parsers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, Vol. 1: Long Papers. Association for Computational Linguistics, 2261–2273. doi:10.18653/v1/2021.acl-long.176

[11] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin SU, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. In *The Thirteenth International Conference on Learning Representations (ICLR '25)*.

[12] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2025. *BIRD-SQL: A Big Bench for Large-Scale Database Grounded Text-to-SQLs*. Leaderboard - Execution Accuracy (EX). https://bird-bench.github.io/

[13] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Ma Chenhao, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. In *Advances in Neural Information Processing Systems (NeurIPS '23, Vol. 36)*. Curran Associates, Inc., 42330–42357. doi:10.52202/075280-1835

[14] Yuntao Li, Bei Chen, Qian Liu, Yan Gao, Jian-Guang Lou, Yan Zhang, and Dongmei Zhang. 2020. "What Do You Mean by That?" A Parser-Independent Interactive Approach for Enhancing Text-to-SQL. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP '20)*. Association for Computational Linguistics, 6913–6922. doi:10.18653/v1/2020.emnlp-main.561

[15] Anna Mitsopoulou and Georgia Koutrika. 2025. Analysis of Text-to-SQL Benchmarks: Limitations, Challenges and Opportunities. In *Proceedings 28th International Conference on Extending Database Technology (EBDT '25)*. 199–212. doi:10.48786/EDBT.2025.16

[16] Mohammadreza Pourreza and Davood Rafiei. 2023. Evaluating Cross-Domain Text-to-SQL Models and Benchmarks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP '23)*. Association for Computational Linguistics, 1601–1611. doi:10.18653/v1/2023.emnlp-main.99

[17] Cedric Renggli, Ihab F. Ilyas, and Theodoros Rekatsinas. 2025. *Fundamental Challenges in Evaluating Text2SQL Solutions and Detecting Their Limitations*. arXiv:2501.18197 doi:10.48550/arXiv.2501.18197

[18] Irina Saparina and Mirella Lapata. 2024. AMBROSIA: A Benchmark for Parsing Ambiguous Questions into Database Queries. In *Advances in Neural Information Processing Systems (NeurIPS '24, Vol. 37)*. Curran Associates, Inc., 90600–90628. doi:10.52202/079017-2876

[19] Irina Saparina and Mirella Lapata. 2025. Disambiguate First, Parse Later: Generating Interpretations for Ambiguity Resolution in Semantic Parsing. In *Findings of the Association for Computational Linguistics (ACL '25)*. Association for Computational Linguistics, 16825–16839. doi:10.18653/v1/2025.findings-acl.863

[20] Ben Shneiderman. 1978. Improving the human factors aspect of database interactions. *ACM Trans. Database Syst.* 3, 4 (1978), 417–439. doi:10.1145/320289.320295

[21] Alane Suhr, Ming-Wei Chang, Peter Shaw, and Kenton Lee. 2020. Exploring Unexplored Generalization Challenges for Cross-Database Semantic Parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 8372–8388. doi:10.18653/v1/2020.acl-main.742

[22] Semih Yavuz, Izzeddin Gur, Yu Su, and Xifeng Yan. 2018. What It Takes to Achieve 100% Condition Accuracy on WikiSQL. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP '18)*. Association for Computational Linguistics, 1702–1711. doi:10.18653/v1/D18-1197

[23] Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter Lasecki, and Dragomir Radev. 2019. CoSQL: A Conversational Text-to-SQL Challenge Towards Cross-Domain Natural Language Interfaces to Databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP '19)*. Association for Computational Linguistics, 1962–1979. doi:10.18653/v1/D19-1204

[24] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 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 (EMNLP '18)*. Association for Computational Linguistics, 3911–3921. doi:10.18653/v1/D18-1425

[25] Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. SParC: Cross-Domain Semantic Parsing in Context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 4511–4523. doi:10.18653/v1/P19-1443

[26] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. *Seq2SQL: Generating Structured Queries from Natural Language Using Reinforcement Learning*. arXiv:1709.00103 doi:10.48550/arXiv.1709.00103

A Study Material

Supplementary materials for the problem-solving task from the study are provided below. In addition, the database descriptions and the underlying schemas considered for the semantic annotation of the NLQs are presented. Note that all materials have been translated from German.

A.1 Employee Domain

Database Schema.

employees :{[emp_no: integer, birth_date: varchar(255), first_name: varchar(14), last_name: varchar(16), gender: varchar(255), hire_date: varchar(255)]}
departments :{[dept_no: char(6), dept_name: varchar(40)]}
dept_emp :{[emp_no: integer, dept_no: char(6), from_date: varchar(255), to_date: varchar(255)]}
dept_manager :{[emp_no: integer, dept_no: char(6), from_date: varchar(255), to_date: varchar(255)]}
salaries :{[emp_no: integer, from_date: varchar(12), salary: integer, to_date: varchar(255)]}
titles :{[emp_no: integer, title: varchar(20), from_date: varchar(12), to_date: varchar(255)]}

Database Description.

Imagine you have unrestricted access to a database of a large company with several departments. The database contains the following information:

- *Personal data of the employees (name, date of birth, gender, date of hire)*
- *The names of all departments*
- *Current and former departmental affiliation of all employees*
- *Current and former salaries of all employees*
- *Current and former job titles of all employees*
- *Current and former managers from all departments*

Problem-solving Task Instruction.

In the following, you will be presented with various problem scenarios. Your task is to formulate several questions for each scenario that would help solve the problem.
Write down questions in natural language (in German) that can be answered using the database.

Problem Scenarios.

(1) *You are considering accepting a position in one of the company's departments. You can work in any department. To help you make your decision, you can research and compare the departments by asking questions to the database.* [adapted from 20]
(2) *Given the shortage of skilled workers and fierce competition in the job market, you want to do everything you can to retain committed employees in your company. Identify employees who are at increased risk of leaving because they may be dissatisfied with their job by asking questions to the database.*
(3) *As an equal opportunities officer, you strive to promote fairness within the company. To identify existing injustices, you can compare groups of people and departments by submitting queries to the database.*

A.2 Scholar Domain

Database Schema.

paper :{[paperid: int(11), title: varchar(300), year: int(11), journalid: int(11)]}
author :{[authorid: int(11), authorname: varchar(50)]}
journal :{[journalid: int(11), journalname: varchar(100)]}
cite :{[citingpaperid: int(11), citedpaperid: int(11)]}
writes :{[paperid: int(11), authorid: int(11)]}
keyphrase :{[keyphraseid: int(11), keyphrasename: varchar(50)]}
paperkeyphrase :{[paperid: int(11), keyphraseid: int(11)]}

Database Description.

Imagine you have unrestricted access to a database of scientific publications. The database contains the following information:

- *Publication details (title, year, journal)*
- *The names of all authors*
- *The names of all scientific journals*
- *Which publications were written by which authors*
- *Which publications are cited in which publications*
- *The keywords of each publication*

Problem-solving Task Instruction.

In the following, you will be presented with various problem scenarios. Your task is to formulate several questions for each scenario that would help solve the problem.
Write down questions in natural language (in German) that can be answered using the database.

Problem Scenarios.

(1) *You are considering accepting a position as an editor at a journal. You can work at any journal. To help you decide which journal you would like to work at, you can research and compare journals by asking questions to the database.*
(2) *You are to decide which author should receive a newcomer award. Any person can receive the award. To help you decide who should receive the award, you can examine and compare the authors by asking questions to the database.*
(3) *You want to collaborate with other authors. You can consider all authors for this purpose. To help you decide who you want to work with, you can examine the authors' publications and previous collaborations by asking questions to the database.*