**Journal of Information Systems Engineering and Management**
2026, 11(1s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

**Research Article**

# Natural-Language-to-SQL Systems with Safe Guardrailing Mechanisms: Architecture, Challenges, and Future Directions

Rahul Jain
Cisco Systems Inc., USA

**ARTICLE INFO**

Received: 05 Dec 2025
Revised: 17 Jan 2026
Accepted: 27 Jan 2026

**ABSTRACT**

Natural-Language-to-SQL (NL2SQL) systems have become revolutionary in the process of democratizing access to data stored in databases, allowing users without formal SQL knowledge to work with structured data via conversational interfaces. Transformer-based architectures and large language models have greatly enlarged the potential and availability of database querying interfaces due to the development of parsers of rules into more useful or enhanced structures. Enterprise implementation, however, poses significant problems such as the complexity of schema grounding, the need for optimization of queries, multilingual support, and the need for high security governance. Best practices of NL2SQL need to be designed with a hybrid architecture that provides flexibility of neural generation and limited flexibility of decoding that would guarantee syntactic consistency and schema conformity. Guardrail mechanisms are imperative elements that offer levels of protection by sanitizing SQL, disambiguating intent, implementing prompting strategies, and implementing role access control. The current architectures are based on decomposed processing pipelines, which separate schema linking and query generation to allow the architecture to be more robust on more complex enterprise schemas. Multi-agent collaborative architectures and lakehouse data platforms are also promising future advances in the development of NL2SQL functionality, which can be used to support iterative reasoning, self-correction, integration of querying different data types, and ensure that metadata governance and security protection are maintained throughout the analytical processes.

**Keywords:** Text-To-SQL, Query Generation, Database Interfaces, Safety Guardrails, Lakehouse Architecture

## 1. Introduction

The natural-language interface has become a revolutionary feature within contemporary data ecosystems and allows individuals to query structured databases using conversational commands instead of the standard SQL language. The recent, more extensive surveys on Text-to-SQL parsing have recorded the history of these systems, starting with early rule-based systems, through neural sequence-to-sequence models, to modern large language models implementations, and the history of each paradigm shift has increased the accessibility and performance of database querying interfaces [1]. Such Natural-Language-to-SQL (NL2SQL) systems transform human linguistic expression into the accuracy needed to operate a relational database system to provide an opportunity to analysts, business users, and operational personnel who may not have formal SQL training to extract insights out of organizational data resources.

Nonetheless, the installation of NL2SQL systems in the business world creates a lot of complications despite these developments. The organizations need severe control over the access to the data, deterministic execution of queries, and integrity of distributed data sources. Natural language is also fundamentally ambiguous, and even large language models can produce syntactically correct but semantically erroneous queries, computationally inefficient queries, or even queries that are security-threatening. The Spider benchmark has provided a baseline of cross-domain semantic parsing and has provided a large human-labeled scale dataset specifically tailored to evaluate more complex and cross-domain text-to-SQL capabilities, providing a systematic way of comparing the performance of systems across a wide range of database schemas and depths of query complexity [2]. This paper explores the underlying design concepts, architectural designs, guardrail designs, and research designs that form the basis of safe and robust NL2SQL systems on an enterprise scale, and outlines new directions in multi-agent AI systems and self-refining query pipelines that are likely to push the field further.

## 2. Fundamentals of Natural-Language Query Systems

The working principle of NL2SQL systems is: the unstructured natural-language queries are converted to a structured SQL query that correctly reflects the user intent. This metamorphosis needs numerous strata of semantic interpretation, such as linguistic parsing, named entity recognition, schema mapping, and intent recognition. Not only does the system have to decode the surface meaning of a query, but it also has to be able to decode the nuanced dependencies between conditions, temporal predicates, aggregate functions, and contextual predicates that affect query processing. The key here is schema encoding, which is database organization in a form that can be effectively processed by the brain.

RAT-SQL framework came up with relation-aware schema encoding and linking schemes, which greatly improved the method used in the field of relating natural language to the database structure [3]. In this architecture, it is acknowledged that good NL2SQL parsing needs explicit modelling of relationships between schema components, such as relationships between tables by using foreign keys, by membership of columns to tables, and relationships between question tokens and schema components. Representing the database schema as a graph structure with nodes and relationships defined by edges, RAT-SQL revealed that the relationship-aware encoding is much more effective in the ability of the model to produce the right SQL, especially when using complex queries with multiple table joins and nested subqueries.

Schema grounding refers to the correspondence between the user language and the database schema that forms one of the most difficult parts of NL2SQL systems. Enterprise database design is often very complex, with the production environment containing many tables, nested relationships in the form of foreign keys, and domain-related naming conventions that do not always conform to the language of everyday life. It is not only a lexical matching problem; systems have to realize that when a user types in customer purchases, he may be referring to a table called transactions that has an association with a table called clients using an intermediate table called orders.

The BIRD benchmark extended the assessment framework by establishing a large-scale database-based text-to-SQL dataset that directly fills the gap between academic benchmarking and the complexity of real-world databases [4]. Contrary to previous benchmarks, which tended to use simplified schemas and clean data, BIRD uses databases with problematic properties such as ambiguous column names, noisy or incomplete data values, domain-specific terminology, and more complex multi-table relationships that resemble production database environments. This benchmark demonstrated that even advanced large language models can do much worse when they are faced with the realistic conditions of databases, which makes the robustness of schema understanding and external knowledge incorporation crucial to deploy the NL2SQL in practice.

In the absence of stable schema grounding, models can give false column mappings, produce incomplete result sets, or produce queries that are in violation of relational constraints like referential integrity or domain constraints. Contemporary methods solve the problem by means of retrievalaugmented generation, in which the relevant schema components are retrieved dynamically according to the input query and given as context to the language model. The BIRD benchmark in particular showed that external knowledge sources and understanding of database content to enhance model performance on realistic queries is, in fact, a requirement for future systems, where pure schema structure should be replaced with semantic knowledge of actual data values and domain conventions.

| Approach | Encoding Method | Key Strength |
|---|---|---|
| RAT-SQL | Relation-aware graph | Multi-table join handling |
| BIRD | Database-grounded | Real-world schema complexity |
| Embedding-based | Vector similarity | Dynamic retrieval |
| Template-based | Structured patterns | Deterministic mapping |

Table 1: Schema Encoding Approaches [3, 4]

The difficulty of disambiguation is another important aspect of the fundamental design of NL2SQL. In cases where the user uses ambiguous terminology that might correspond to more than one element of the schema, the system needs to either clarify the ambiguity by using context or have the user clarify the ambiguity in a dialogue. Complex systems keep records of conversation history and user profiles to guide such disambiguation choices, based on the experience of earlier interactions, of which interpretations are compatible with particular user roles and analysis patterns.

## 3. Architectures for NL2SQL Systems

Various paradigms have been developed over the years, with the latest systems shifting more to a hybrid paradigm that combines the flexibility of neural generation with high-level constraints to guarantee syntactic validity and compliance with the schema. The structural issue in question is the necessity to strike the right balance between the open-ended generative abilities of language models and the demands of a specific SQL syntax, as well as the limitations of a target database schema.

Proposed by the PICARD framework, is a paradigm-shifting framework on constrained auto-regressive decoding, which resolves one of the fundamental reliability issues of neural text-to-SQL systems [5]. Instead of letting language models freely generate SQL tokens and then verifying generated outputs afterward, PICARD also uses incremental parsing as part of the generation process. Each decoding step has a system that checks candidate tokens against SQL grammar rules and schema constraints and rejects those that would result in syntactically invalid queries or schema-inconsistent queries before becoming part of the output. This strategy will guarantee that no invalid SQL will ever be generated and will only refer to tables and columns that exist in the target database, significantly lowering the errors that are common with unconstrained generation strategies.

The incremental parsing algorithm that PICARD uses is based on the idea that it maintains a partial state of the parsing that establishes the grammatical context of the SQL query being built. Upon the language model proposing a next token, the parser tries to continue the partial parse, and if the extension fails due to the token either being inappropriate according to the rules of the SQL syntax or by pointing to a non-existent schema element, the token is masked, and the model must propose instead. This limited decoding strategy showed substantial gains on the benchmark tests by removing complete classes of errors pertaining to syntax errors and schema hallucination, which afflict unconstrained generation systems.

**Journal of Information Systems Engineering and Management**
2026, 11(1s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

The study carried out on the premises of constrained generation developed the DIN-SQL framework, proposed decomposed in-context learning with self-correction facilities to further enhance architectural refinement [6]. This strategy acknowledges that intricate text-to-SQL translation can be gained through partitioning the task into solvable subtasks instead of end-to-end generation. DIN-SQL splits the translation process into separate stages, such as schema linking, query classification, SQL generation, and self-correction, and each stage is addressed by more specialized prompting strategies that direct the attention of the language model to a particular part of the problem.

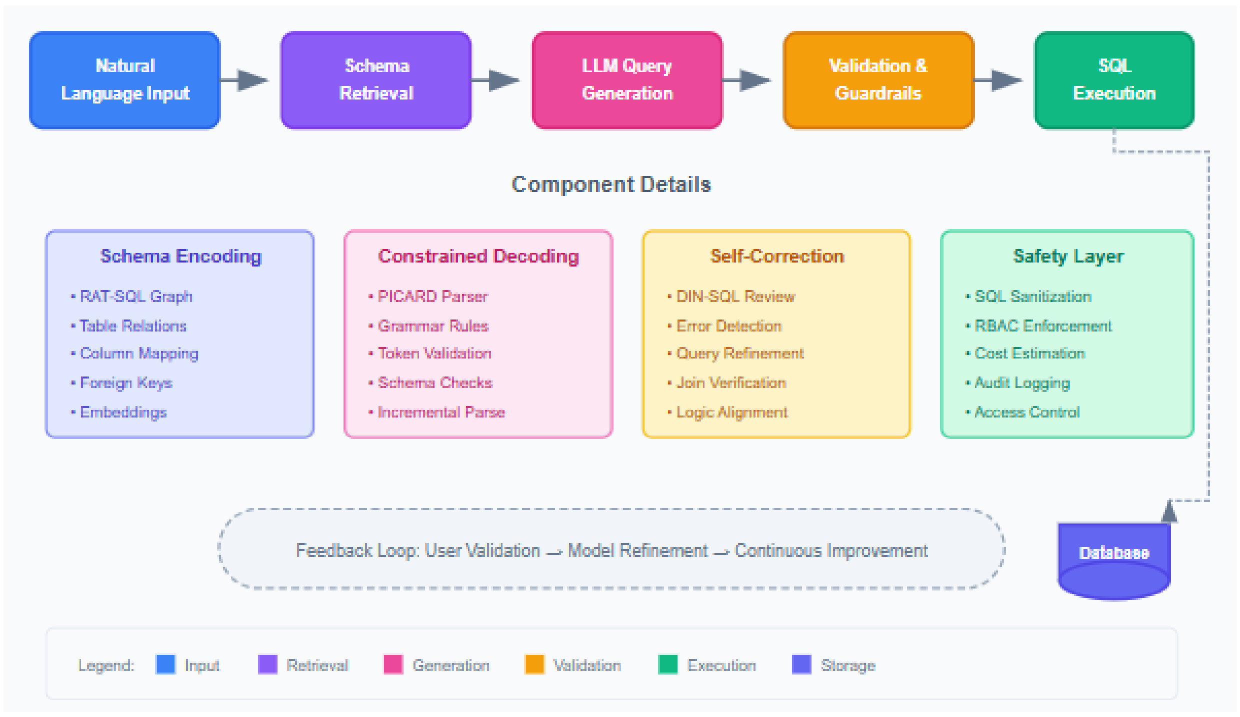

Fig 1: NL2SQL Architecture Pipeline [5, 6]

The self-correction mechanism of DIN-SQL is one such innovation in architecture that is of particular significance. Following initial SQL generation, the system asks the language model to check its own output for possible errors, taking into account whether all the entities mentioned are well-represented, whether the join conditions are properly defined, and whether the logic of the query reflects the original semantics of the question. This self-criticism model allows the model to detect and fix errors that otherwise could be transmitted to an execution to enhance the overall system reliability, without the need to have an external validation infrastructure.

The modern production systems are generally designed with Multi-stage pipelines that break the NL2SQL task down into smaller parts: natural-language interpretation, schema retrieval and linking, SQL generation, and validation. This breakdown enables every step to be optimized independently of both accuracy and robustness, as well as computational performance. The schema retrieval phase can use embedding-based similarity search to find the relevant tables and columns, and the generation phase will involve the assembly of syntactically sound SQL that includes the schema elements that were retrieved. Validation phases undertake syntax verification, semantic verification, and safety verification, and then the queries are sent to execution.

Another architectural factor that implies performance and security is the execution environment. Deployments of many execute generated SQL in sandboxed database connections, which impose constraints on resource access and accessibility. Connection pooling, limit of query time, and result set size give further protection at the infrastructure layer. Observability and monitoring abilities record telemetry information on query latency, metrics on accuracy, based on user feedback, and error rates, which guide continuous improvement.

## 4. Guardrail and Safety Mechanisms

Guardrails are the key feature of enterprise NL2SQL systems, which offer many levels of safeguarding against misplaced, unprotected, or even harmful query generation. The history of natural language database interface offers valuable background to the knowledge of the present safety requirements. Initial studies into habilitation in natural language interfaces provided the principles underpinning the limitations of system ability and the significance of effective communication with users on what the system can and cannot process [7]. Recognition of this groundbreaking work was that natural language interfaces have to consider the expectations of the user and gracefully handle queries that are beyond the ability of the system to handle, a principle that still drives the design of guardrails today.

Habitability is a term used to describe how easily users can become educated to formulate queries using the limitations of a natural language interface. Early systems learned that users soon form mental models of system capabilities and adjust their language to those capabilities; however, only happens when the system can respond consistently and understandably to them. This understanding guides the contemporary guardrail design which cannot just avoid the detrimental queries, but also inform the users when their requests cannot be answered and refer them to the effective reformulations. The systems that deny the queries and do not provide reasons or act in a way that is not consistent are a contravention of user trust and uptake.

SQL sanitization layers are a major type of guardrail in modern applications that check the generated queries against suspicious operations and then run them. This layer detects potentially dangerous patterns such as Cartesian products caused by omission of join conditions, full table scans on large tables not properly filtered by appropriate predicates, unlimited time windows that might fetch too much historical data, and writes in environments that are set to read-only access. Some models that estimate query cost can examine generated SQL to forecast resource usage, block, or need approval of a query that exceeds set limits.

The C3 exemplified the way zero-shot text-to-SQL applications via ChatGPT might be improved with the help of a carefully thought-out prompting strategy, which takes into account the aspect of safety [8]. This work proposed Clear Prompting, Calibration with Hints, and Consistent Output mechanisms that all enhance the accuracy and reliability of the generated SQL. Calibration strategy is specifically applicable to guardrailing, since it entails giving the model clear instructions on how it should not be constrained by the schema and data type, and the scope of queries that are not supposed to be violated. C3 exemplified that the incorporation of safety constraints into the prompting strategy directly aimed at ensuring safety constraints could be enforced at the generation stage, not just by means of post-generation validation.

Another important safety issue that is considered in the consistent output mechanism in C3 is that it guarantees deterministic behavior in repeated queries. Enterprise usage requires the ability to generate a semantically equivalent query, on a semantically identical query, with a few word variations in phrasing. Unpredictable output not only serves to confuse the users, but it also generates problems of data integrity, where varying query formulations give slightly different results. The output consistency approach of the C3 framework gives a template or a framework of guardrail mechanisms that offer predictability of behavior as well as its correctness.

| Mechanism | Function | Implementation Layer |
|---|---|---|
| SQL Sanitization | Block risky operations | Post-generation |
| RBAC Enforcement | Access control validation | Query layer |
| C3 Calibration | Schema constraint hints | Prompt design |
| Intent Disambiguation | User clarification | Pre-generation |
| Audit Logging | Activity tracking | Execution layer |

Table 2: Guardrail Mechanisms [7, 8]

Another guardrail strategy is intent disambiguation workflows, which are especially useful in situations where user input does not give enough specificity to create a specific query. Instead of giving assumptions that might yield wrong answers, advanced systems have the user involved in a clarification conversation and give a choice of the ambiguous terms or demand further information before acting. This human-in-the-loop solution accepts the natural shortcomings of automated interpretation but keeps the user in control of query semantics.

Authorization of data policies assigned to a requesting user is maintained through role-based access control at the query layer to ensure that the generated SQL complies with data authorization policy. The system can also block queries that involve tables or columns to which the user is not authorized to access and may error out the query or will automatically impose row-level or column-level security filters. Audit logging and compliance monitoring give retrospective access to the activity of NL2SQL systems, including the original natural language input, generated SQL, and execution output and the corresponding user identity of every interaction, facilitating security investigation and compliance needs.

## 5. Key Challenges in Enterprise Environments

The real-world deployment of NL2SQL systems is more complex than the complexity involved during benchmarking or limited scope systems. The real-world environments have heterogeneity in several aspects, such as a multitude of database management systems, various SQL flavors, distributed data sources, varying data quality characteristics, and intricate data access control requirements.

Benchmarks of text-to-SQL models leveraging large language models showed substantial differences in performance when compared from a research environment to a real-world corporate setup [9]. This study systematically analyzed a host of state-of-the-art models on a variety of benchmarks and introduced insights on how aspects such as schema complexity, query difficulty, and domain influence text-to-SQL model performance. From these results, it is clear that large language models are highly proficient at evaluating easy queries and handling a neatly designed schema, but fare poorly when facing the unclear vocabulary terminology, complex JOINS, and domain-specific notation found in corporate databases.

It's also worth noting that the benchmark test showed the value of using accuracy measures based on execution, rather than measures based on a comparison of matches alone. In the enterprise domain, the best measure of success is getting the correct result sets from the executed SQL, rather than just matching the executed SQL with a comparison string. Various SQL queries can yield the same result, and measures should be designed with this consideration in mind. Similarly, queries with a comparable structure can still generate incorrect results because of slight logical mistakes.

Query optimization is an area that is highly challenging when the SQL generated from the models may be grammatically correct and semantically correct, but inefficient from a performance perspective, specifically when the enterprise data may contain numerous rows and naive query patterns may perform well on smaller data but may time out or use too many resources when run on enterprise data sizes. Query optimization stacks need to examine the generated SQL to look for inefficient patterns and rewrite the SQL to utilize available indexes, remove redundancies, move predicates closer to the data sources, and stabilize the execution plans.

RESDSQL framework handled the problem of text-to-SQL at an enterprise level by proposing a decoupled framework where schema linking is handled separately from skeleton parsing [10]. It is assumed in the framework design that the error in schema linking affects the whole process of SQL generation. Therefore, by ranking schema components before predicting the SQL structure of the problem question, the RESDSQL framework handled questions more robustly, requiring a correct link to schema tables and columns.

The decoupling approach used in the RESDSQL system will be especially beneficial in the context of enterprise systems, which often involve schemas with many tables. In this scenario, the number of possible combinations of tables and columns grows combinatorially and becomes too cumbersome to efficiently explore from an end-to-end modeling point of view. By first focusing the context of the schema search on a set of very informative points, the SQL search space also shrinks, making the search more accurate and efficient. Additionally, this method also allows the updating of the schema in phases while retraining the ranking function to include a new table.

An environment of multiple languages and industry-related vocabularies makes semantic interpretation even more difficult for enterprise applications. Enterprises functioning across multiple global locations will get queries in several languages, and the NL2SQL model may require multiple language support or translation processes before querying. The industry-related taxonomies face semantic interpretation challenges when industry-specific vocabularies are not present in the general training dataset or when different industry contexts have different meanings for the same term. The ongoing change management of the base schemas is a serious operational issue that requires constant enterprise database updates with new tables, renamed columns, and modified relationships.

## 6. Future Research Directions: Autonomous Agents and Secure Querying

Newer NL2SQL systems are now incorporating autonomous agent architectures with iterative reasoning, self-correction, and multi-step query-planning that goes beyond single-turn question answering. These agents break down the complex analytical problems into sets of simpler questions, combine intermediate answers, and optimize interpretations based on the results of executions. The development of agentic AI systems is a fundamental change from reactive query translation to proactive analytical support.

The MetaGPT architecture proposed new paradigms of meta programming of multi-agent collaborative systems that provide captivating architectural designs of next-generation NL2SQL platforms [11]. This framework illustrates how various specialized agents may collaborate via well-organized communication protocols to undertake the complex tasks that cannot be achieved by individual models. Within the framework of NL2SQL, these architectures may include intent interpretation, schema analysis, query generation, optimization, and safety validation agents, the output of which is then specialized with specialized knowledge to the overall system output.

The multi-agent design overcomes a number of limitations of monolithic NL2SQL designs. Multifaceted analytical questions can involve the ability to cut across multiple areas: linguistic skills to deconstruct user intent, database skills to traverse schema structures, optimization skills to execute efficiently, and security knowledge to impose access control. There is no effective model that is successful on all of these dimensions, yet agent systems can combine specialized capabilities by means of organized coordination. The strategy of encoding standard operating procedures and communication protocols designed in MetaGPT is a model of how to organize these types of collaborations in enterprise NL2SQL scenarios.

Another research direction that is currently underway is the integration of NL2SQL-based components with wider data platform elements. The systems include natural language querying with data catalog exploration, automated visualization generation, and insight summarization, which offer more comprehensive analytical processes. Users can start with exploratory inquiries, navigate down into the particular data units with follow-up inquiries, and demand visualization or a narrative summary of the results, without changing tools or circumstances. These combined experiences demand synchronization among a variety of AI functions and control in data interpretation and access control enforcement across the workflow.

Lakehouse architecture paradigm provides a base on which single data platform supports higher NL2SQL capability in various data formats and processing trends [12]. This architectural vision is an integrated solution that brings together the data warehousing and advanced analytics on open platforms that provide direct access to data that is stored in cloud object store with the reliability and performance features that were only available on traditional data warehouses. The lakehouse model is a solution to long-term conflict between data warehouse governance and data lake flexibility and offers a common base of NL2SQL systems, able to query structured transactional data as well as semi-structured analytical data.

The architecture of lakehouses implements features that are specifically useful in the development of NL2SQL, such as ACID transaction processing, schema enforcement, schema evolution, and time-travel queries, which can be used to analyse historical data. These allow the set of questions that can be answered using the NL2SQL systems to increase in the number of questions that require answers about the data as it was at a particular point in time, comparisons of how the data was in the past and the present, and analysis of changes that can help track how a metric changed over a specific time. A convergence of architectures toward lakehouse platforms indicates that future NL2SQL systems will be used on more and more integrated data landscapes where metadata and governance models are similar.

| **Framework** | **Core Innovation** | **Primary Benefit** |
|---|---|---|
| PICARD | Incremental parsing | Syntax guarantee |
| DIN-SQL | Decomposed learning | Self-correction |
| RESDSQL | Decoupled linking | Schema robustness |
| C3 | Zero-shot prompting | No fine-tuning |
| MetaGPT | Multi-agent coordination | Task specialization |

Table 3: NL2SQL Framework Comparison [4, 5, 7, 8]

The secure querying system is still advancing with the innovations of a policy-driven AI system. Attribute-level access control. Fine-grained attribute-level access control allows systems to allow queries that access some of the columns and automatically redact or aggregate any sensitive attributes according to the level of user authorization. Differential privacy integration adds noise to query outcomes, so that it is possible to perform an aggregate analysis, but not possible to identify the individual records. Self-learning guardrails are a future direction of research in which safety controls automatically determine new schemas, user behavior patterns, and new security risks, and can adjust their rules accordingly without rules being updated manually.

## Conclusion

Natural-Language-to-SQL systems have become an important contribution to the idea of conversational access to enterprise databases, the intersection between human language and structured query execution. To perform well at scale needs careful architectural design that includes strong schema encoding schemes, multi-phase generation schemes with limited decoding, and extensive validation layers that are syntactically correct and semantically accurate. The safety mechanisms should respond to various risk dimensions such as query correctness, enforcement of access control, and protection of computational resources via layered guardrails such as SQL sanitization, intent disambiguation, and role-based security integration. Enterprise deployment is not limited to the conditions of academic benchmarks, where schema heterogeneity, large-scale data optimization, and constant response to changes in database structure need to be considered. Architectures that decouple these two functions, schema comprehension and query skeleton production, could provide a viable solution to the challenge of dealing with the complexity inherent in enterprise-scale models and remain accurate. The combination of NL2SQL functionality with autonomous multi-agent systems and integrated lakehouse solutions puts natural language as a more practical universal interface to organizational data access and allows more people to be involved in data-driven decision-making processes without losing the governance requirements and security limits fundamental to business functions.

## References

[1] Panos Ipeirotis and Haotian Zheng, "Natural Language Interfaces for Databases: What Do Users Think?," arXiv:2511.14718v1, 2025. [Online]. Available: https://arxiv.org/html/2511.14718v1

[2] Tao Yu et al., "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task," arXiv:1809.08887, 2019. [Online]. Available: https://arxiv.org/abs/1809.08887

[3] Bailin Wang et al., "RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers," arXiv:1911.04942, 2021. [Online]. Available: https://arxiv.org/abs/1911.04942

[4] Jinyang Li et al., "Can LLM Already Serve as A Database Interface? A Big Bench for Large-Scale Database Grounded Text-to-SQLs," arXiv:2305.03111, 2023. [Online]. Available: https://arxiv.org/abs/2305.03111

[5] Torsten Scholak et al., "PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models," arXiv:2109.05093, 2021. [Online]. Available: https://arxiv.org/abs/2109.05093

[6] Mohammadreza Pourreza and Davood Rafiei, "DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction," arXiv:2304.11015, 2023. [Online]. Available: https://arxiv.org/abs/2304.11015

[7] Androutsopoulos et al., "Natural Language Interfaces to Databases – An Introduction," arXiv:cmp-lg/9503016, 1995. [Online]. Available: https://arxiv.org/abs/cmp-lg/9503016v2

[8] Xuemei Dong et al., "C3: Zero-shot Text-to-SQL with ChatGPT," arXiv:2307.07306, 2023. [Online]. Available: https://arxiv.org/abs/2307.07306

[9] Dawei Gao et al., "Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation," arXiv:2308.15363, 2023. [Online]. Available: https://arxiv.org/abs/2308.15363

[10] Haoyang Li et al., "RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL," arXiv:2302.05965, 2023. [Online]. Available: https://arxiv.org/abs/2302.05965

[11] Sirui Hong et al., "MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework," arXiv:2308.00352, 2024. [Online]. Available: https://arxiv.org/abs/2308.00352

[12] Michael Armbrust et al., "Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics," CIDR, 2021. [Online]. Available: https://people.eecs.berkeley.edu/~matei/papers/2021/cidr_lakehouse.pdf