LLM Prompting for Text2SQL via Gradual SQL Refinement

Anonymous ACL submission

Abstract

Recent studies have shown that prompting large language models (LLMs) for Text2SQL can achieve promising performance. However, the task still remains very challenging due to the difficulty of aligning complex natural language semantics with database schema. In this paper, we present a novel prompting approach for Text2SQL via Gradual SQL Refinement (GSR). It consists of three sequential prompting steps: 1) Clause Decomposition, which breaks down a complex natural language question into simpler clauses to facilitate natural language interpretation; 2) SQL-driven Schema Linking, which improves schema linking by targeted schema information retrieval based on the preliminary SQL generated in the first step; 3) SQL Execution Refinement, which further refines the SQL generated in the second step based on the results of SQL execution. GSR is a gradual prompting approach in that it begins with only one SQL and then gradually refines the SQL based on SQL analysis and execution at each of the following steps. We have validated the efficacy of GSR by an empirical study on the benchmark datasets. Our experiments show that its execution accuracy on BIRD and Spider are 69.26% and 87.7% respectively when using GPT-4o. With only a few prompts, GSR is ranked 11th on the BIRD benchmark, considerably outperforming the existing single-candidate alternatives. Its performance is even highly competitive compared with the existing approaches based on model fine-tuning or multiple-candidate generation, which requires considerably more prompts and token consumption.

1 Introduction

The goal of Text2SQL is to translate natural language questions into corresponding SQL queries (Deng et al., 2021). It enables users to interact with databases using simple natural language queries without requiring any knowledge of SQL, thus lowering the barrier for non-technical users to access relational databases. However, inherent differences between natural language and SQL usually make this task particularly challenging. Natural language tends to be ambiguous, with context-dependent semantic information, whereas SQL queries must be syntactically precise and aligned with database schema, specifically the tables, columns, values and relationships involved. Therefore, generating correct SQL queries not only requires understanding a user's intent but also ensuring that the generated SQL aligns with the underlying database schema.

In recent years, leveraging the powerful understanding and generation capabilities of large language models (LLMs) (Achiam et al., 2023) has become a key approach to improve Text2SQL performance (Rajkumar et al.), with prompt engineering emerging as a mainstream technical strategy (Nan et al.). Some studies aimed to enhance the reasoning ability of LLMs by incorporating various contextual learning techniques, such as chain-of-thought prompts (Tai et al.; Wei et al.), question decomposition (Eyal et al., 2023; Pourreza and Rafiei, 2024, 2023; Wang et al., 2024), and self-consistency (Gao et al., 2023; Sun et al., 2024). Other studies instead focused on schema linking (Dong et al., 2023; Wang et al., 2024, 2020a; Talaei et al., 2024; Lee et al., 2024), which aimed to improve performance by providing LLM models with more specific database schema. However, generally speaking, Text2SQL still remains very challenging due to the difficulty of accurately aligning natural language semantics with database schema.

In this paper, we introduce a novel LLM prompting approach for Text2SQL, named as Gradual SQL Refinement (GSR), which can gradually refine a SQL by a sequence of targeted prompts. Supposed to perform Text2SQL with only a few prompts, it consists of the following three sequential prompting steps:

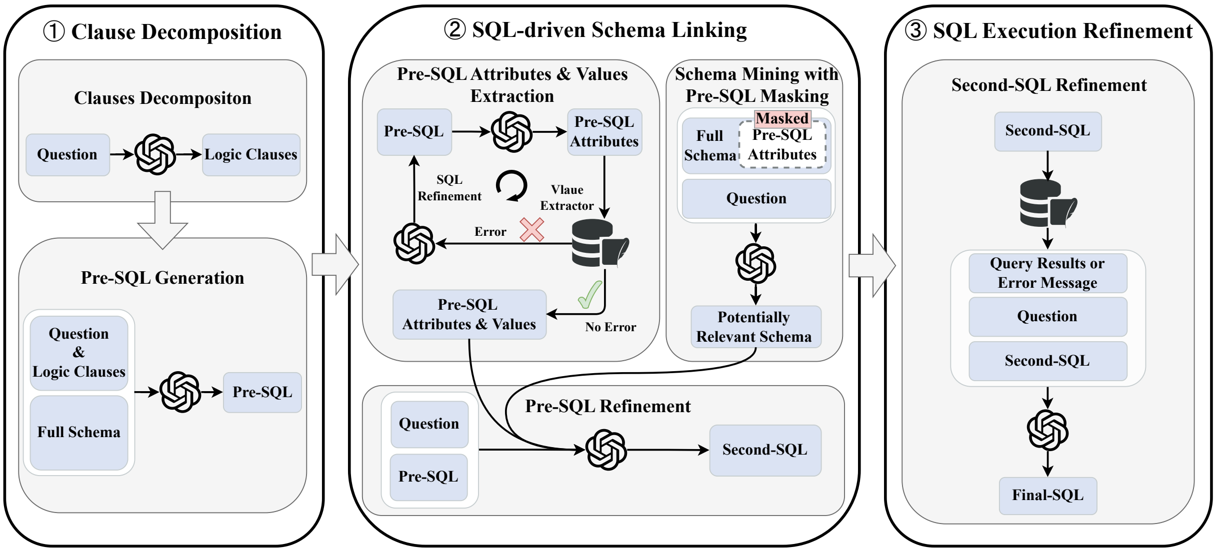

Figure 1: The Gradual SQL Refinement (GSR) framework for Text2SQL: 1) the 1st step decomposes a natural language question into multiple simpler clauses, and leverages them as well as the original question to generate a preliminary SQL (Pre-SQL); 2) the 2nd step performs SQL-driven schema linking to identify and correct schema misalignment errors in the Pre-SQL, resulting in Second-SQL; 3) the 3rd step further refines the Second-SQL based on its execution results and generates the final SQL (Final-SQL).

1. **Clause Decomposition:** it translates a complex natural language question into a sequence of simpler clauses with equivalent semantic meaning; by prompting LLMs with these simpler clauses as well as the original question, GSR can effectively improve the accuracy of natural language interpretation;

2. **SQL-driven Schema Linking:** it performs schema linking by targeted schema information retrieval based on the preliminary SQL generated by the first step (Pre-SQL). Instead of providing LLMs with blanket database schema information, it only provides some sample values of attributes present in the preliminary SQL and other potentially useful schema information missing in Pre-SQL. By feeding these SQL-tailored schema information as well as the original question to LLMs, GSR can more precisely identify and correct schema misalignment errors in the Pre-SQL;

3. **SQL Execution Refinement:** it further refines the SQL generated by the second step (Second-SQL) based on its execution results. Specifically, GSR executes the Second-SQL on the database to obtain the query results or execution error information, which are then leveraged to assess the validity of the Second-SQL and its alignment with the original natural language question.

We have sketched the framework of GSR in Figure 1. As far as we know, even though there already exist many prompting approaches for Text2SQL, the proposed GSR is the first gradual SQL-driven approach in that it begins with only one SQL, and then gradually refines the SQL based on SQL analysis and execution at each of the following steps. It is noteworthy that unlike the existing approaches of question-driven schema linking (Wang et al., 2020a; Dong et al., 2023; Pourreza and Rafiei, 2023; Talaei et al., 2024; Lee et al., 2024), which provide instructions based on a given question to identify relevant schema information, the proposed schema linking of GSR is SQL-driven in that it extracts potentially useful schema information based on a SQL as well as the original question and leverages them for schema alignment. As a crucial step in the iterative process of SQL refinement, SQL-driven schema linking can more precisely detect and correct schema misalignment errors present in a SQL. On the other hand, the existing technique of question decomposition usually decomposes a question into multiple sub-questions and then invokes LLMs to construct sub-SQLs for sub-questions, which are finally fused to generate the final SQL (Wang et al., 2024). In contrast, the purpose of clause decomposition in GSR is not to generate sub-SQLs, but to provide simple yet effective instructions for Text2SQL translation.

The major contributions of our work can be summarized as follows:

- We propose a novel gradual SQL-driven prompting approach of GSR for Text2SQL, which begins with a preliminary SQL, and then iteratively performs schema refinement by targeted SQL analysis and execution;
- We present the specific techniques of clause decomposition, SQL-driven schema linking and SQL execution refinement to enable the implementation of GSR;
- We empirically validate the efficacy of the proposed GSR. Our experiments show that GSR achieves the execution accuracy of 69.26% and 87.7% on the BIRD and Spider benchmarks respectively when using GPT-4o. With only a few prompts, GSR is ranked 11th on the BIRD benchmark, considerably outperforming the existing single-candidate approaches. Its performance is even highly competitive compared with the existing approaches based on model fine-tuning or multi-candidate generation, which requires considerably more prompts and token consumption.

2 Related Work

Early rule-based (Thompson and Thompson, 1983; Tang and Mooney, 2001) and template-based (Zelle and Mooney, 1996; Wang et al., 2011) approaches for Text2SQL were highly domain-specific and heavily dependent on handcrafted rules, making them unsuitable for complex queries or diverse databases. With the widespread adoption of deep learning (Vaswani et al., 2017), the research in Text2SQL has undergone significant transformations, enabling end-to-end learning and enhanced contextual understanding (Sutskever et al., 2014). Early neural network-based methods, such as SQLNet (Xu et al., 2017) and Seq2SQL (Zhong et al., 2017), framed SQL generation as a sequence-to-sequence (Seq2Seq) learning problem. These models laid the foundation for neural approaches but struggled to incorporate schema-specific information while handling complex queries. Schema-aware models addressed this limitation by explicitly modeling the relationships between queries and database schema elements (Yu et al., 2018a; Guo et al., 2019). The transformer architecture significantly enhanced the capabilities of Text2SQL systems by capturing complex relationships between schema elements and natural language queries, e.g., RAT-SQL (Wang et al., 2020b), T5 (Raffel et al., 2023), BART (Lewis et al., 2019) and PICARD (Scholak et al., 2021). While these models achieved remarkable results, they also faced challenges, such as noise from large schema and inefficiencies in handling complex multi-table queries, highlighting the need for further optimization.

With the emergence of LLMs, an increasing number of studies have explored their potential for Text2SQL. Some studies introduced various chain-of-thought prompt strategies to improve SQL generation performance, such as ACT-SQL (Zhang et al., 2023), COE-SQL (Zhang et al., 2024) and TA-SQL (Qu et al., 2024). In contrast, DTS-SQL (Pourreza and Rafiei, 2024), DEA-SQL (Xie et al., 2024) and DIN-SQL (Pourreza and Rafiei, 2023) used the strategy of task decomposition strategy to improve prompting accuracy. Some other studies, e.g., MAC-SQL (Wang et al., 2024), leveraged both of the above strategies for Text2SQL. It is also noteworthy that while some studies, e.g., DAIL-SQL (Gao et al., 2023) and PET-SQL (Li et al., 2024c) focused on optimizing prompt design, others, e.g., Codes (Li et al., 2024b) and SuperSQL (Li et al., 2024a), instead focused on improving SQL generation by fine-tuning a pre-trained model.

There are also some work specifically focused on schema linking. For instance, C3-SQL (Dong et al., 2023) exploited zero-shot prompts of self-consistency for schema linking. MCS-SQL (Lee et al., 2024) leveraged zero-shot chain-of-thought reasoning and schema order shuffling for table and column selection. E-SQL (Caferoğlu and Özgür Ulusoy, 2025) aimed to improve schema linking through question enrichment. RSL-SQL (Cao et al., 2024) used bidirectional schema linking to mitigate risks of incomplete recall and noise. CHESS (Talaei et al., 2024), on the other hand, presented a context-based schema selection method. It is noteworthy that these existing approaches of schema linking are mainly question-driven, analyzing natural language questions to filter schema information. In contrast, the proposed schema linking in GSR is SQL-driven, analyzing SQL to retrieve relevant schema information.

More recently, some studies have employed the Multi-Candidate Strategy (MCS) to improve SQL generation accuracy. For instance, MCS-SQL (Lee et al., 2024) generates multiple SQL candidates using diverse prompts and filters them based on confidence scores. It uses a separate LLM to select the final SQL. CHESS (Talaei et al., 2024) similarly generates multiple SQL candidates through a multi-agent framework, and refines them iteratively with an LLM if execution errors occur. It requires a Unit Tester agent to evaluates candidates and selects the highest-scoring SQL. CHASE-SQL (Pourreza et al., 2024) also introduced a multi-path reasoning framework to generate multiple SQL candidates. It selects the best SQL through pairwise comparisons with a fine-tuned binary-candidates selection LLM. XiYan-SQL (Li et al., 2023) employed a multi-generator ensemble strategy to enhance candidate generation. It combines prompt engineering and the SFT method to enhance SQL quality and diversity. These studies have shown that the multi-candidate strategy can effectively enhance execution accuracy. However, this strategy usually requires considerably more prompts, thus significantly increasing token consumption. In comparison, the proposed GSR generates only one candidate SQL and requires much less token consumption.

3 Methodology

In this section, we detail the technical solutions for the three essential steps of GSR, i.e., clause decomposition, SQL-driven schema linking and SQL execution refinement.

3.1 Clause Decomposition

To enhance question interpretation, GSR adopts a new prompting strategy called "Clause Decomposition". It breaks down a natural language question into multiple simpler logical units, which if fused, would have the same semantic meaning with the original question. By segmenting a question into more manageable parts, clause decomposition can effectively simplify relationship between descriptive terms and their corresponding entities, thus enabling more precise natural language interpretation. It is noteworthy that clause decomposition is supposed to be automatically performed by LLMs. An illustrative example of clause decomposition is shown in Figure 2.

Then, GSR feeds the resulting clauses as well as the original question and the full database schema to a LLM, and generates a corresponding SQL query, which is referred to as the preliminary SQL (Pre-SQL). The generated Pre-SQL usually contain errors, particularly in involved attributes and selection conditions, which need to be refined in the following steps.

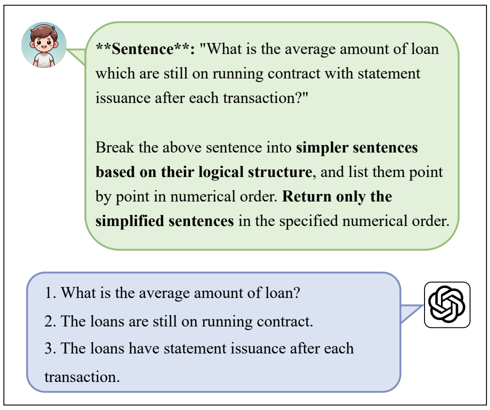

Figure 2: An illustrative example prompt for clause decomposition.

3.2 SQL-driven Schema Linking

Without access to actual data in a database, the Pre-SQL usually contains errors involving attributes and selection conditions. This necessitates schema linking. Due to the large number of columns in a database, feeding LLMs with all the schema information would result in excessively long input contexts, potentially filled with a significant amount of irrelevant and redundant information, which may reduce SQL accuracy as well as increasing token cost.

To overcome this limitation, GSR adopts a SQL-driven schema linking strategy that leverages the Pre-SQL as well as the original natural language question to retrieve relevant schema information. Specifically, the process of SQL-driven schema linking is composed of two stages: 1) Pre-SQL attributes & values extraction, which retrieves the attribute and value information present in the Pre-SQL; 2) schema mining with Pre-SQL masking, which retrieves additional schema information potentially relevant to the natural language question by missing in the Pre-SQL.

Pre-SQL Schema Extraction: in the first stage, GSR extracts the tables and their columns involved in the Pre-SQL, and then constructs an individual query statement for each table and column combination to retrieve actual attribute values from the database. This process ensures that the extracted data reflect the real contents of the database, providing a precise reference for subsequent refinement on Pre-SQL.

Specifically, GSR constructs a SQL value condition checker to determine whether a column in the Pre-SQL is involved in value condition, and uses it to optimize value retrieval. If a column is involved in value conditions, GSR extracts its condition value, and retrieves the top-five values with the highest similarities to the condition value as value samples of this column. Otherwise, if a column is not involved in any value condition, GSR randomly retrieves three distinct values as value samples.

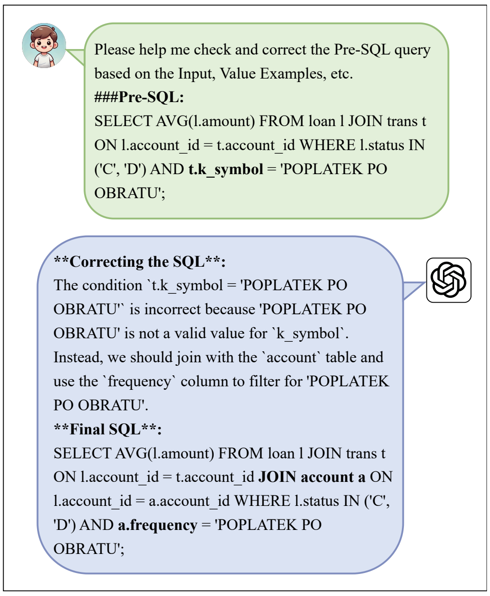

Figure 3: An illustrative example prompt for SQL-driven schema linking.

It is noteworthy that a Pre-SQL may contain some column names not existing in the given database schema. As a result, the execution of value retrieval may receive error information from a database. In this case, GSR would feed the error message as well as the Pre-SQL to a LLM and ask it for column correction. The column correction would be repeatedly invoked until no error information is returned. However, to ensure the efficiency of value retrieval, GSR limits the maximum iterations of column correction at 3.

Schema Mining with Pre-SQL Masking: the second stage aims to mine tables and columns potentially relevant to the original query, but not present yet in the Pre-SQL, thereby constructing a schema that is comprehensive but not redundant. Towards this aim, GSR first masks the tables and columns involved in the Pre-SQL within the full database schema, resulting in a masked schema denoted as $Schema_m$, and then asks a LLM to select the tables and columns within $Schema_m$ that are potentially relevant to the original natural language query. We denote the schema obtained after schema mining with Pre-SQL masking as $Schema_p$, which represents a subset of potentially relevant schema not present yet in the Pre-SQL.

Prompt Instruction for SQL-driven Schema Linking: after the two stages, GSR obtains not only Pre-SQL's tables, columns and their sample values, but a small set of potentially relevant tables and columns information not present yet in the Pre-SQL. Then, GSR feeds the retrieved schema information as well as the original query and Pre-SQL to a LLM and instructs it to refine the Pre-SQL, resulting in a new SQL (Second-SQL). An illustrative example of instruction prompt for SQL-driven schema linking is shown in Figure 3.

3.3 SQL Execution Refinement

The Second-SQL may still contain some minor errors, e.g., incorrect values in the condition part and the misguided use of keywords, both of which would result in the SQL query returning an empty result. The typical errors in the Second-SQL include: 1) the case sensitivity of the condition values; 2) incorrect data types involved in arithmetic operations; 3) misused SQL keywords, e.g., **distinct**, **group by**, and **order by**.

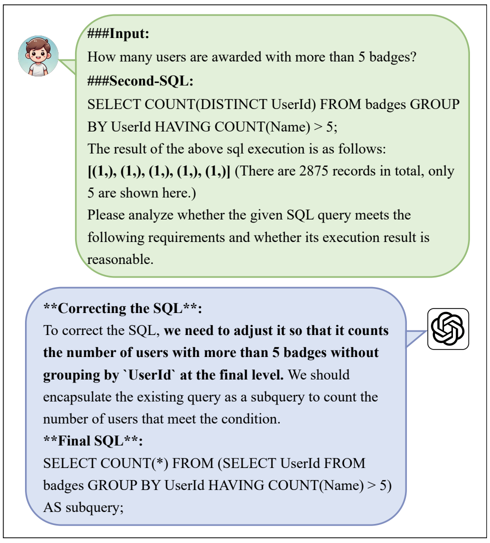

Figure 4: An illustrative example prompt for SQL execution refinement.

Therefore, in the final step, GSR executes the Second-SQL and then leverages the returned results to detect and correct the potential misalignment between the SQL query results and the expected output of the original question. Specifically, GSR instructs the large model to refine the Second-SQL by Requirement Check (RC) and Result Reasonableness Check (RRC). The RC focuses on having the model check if the conditions and the use of values in the Second-SQL align with the question's requirements, while the RRC is supposed to examine issues such as case sensitivity or data types in arithmetic operations. Due to the context length limitation of LLMs, only a portion of the execution results from the Second-SQL, along with the total record count of the query, will be provided as input. The SQL obtained after execution refinement from the Second-SQL is the Final-SQL. An illustrative example of instruction prompt for SQL execution refinement is shown in Figure 4.

4 Empirical Study

4.1 Experimental Setting

Datasets: we conduct our experiments on the benchmark datasets of BIRD (Li et al., 2023) and Spider (Yu et al., 2018b). The Spider dataset is a large-scale, cross-domain Text-to-SQL task. Its primary challenge lies in accurately selecting the correct columns from multiple tables within a given database schema, as well as effectively managing the join relationships between these tables. The Spider is a valuable resource for evaluating the generalization capabilities of LLM models. On the Spider dataset, the training, development, and test sets include 146, 20, and 40 databases respectively; they contain 8659, 1034, and 2147 examples respectively. In comparison, the BIRD's primary challenge lies in handling both database values and external knowledge, requiring LLM models to effectively integrate external information with the database content. Furthermore, the BIRD emphasizes the efficiency of SQL query generation. These characteristics make the BIRD notably more complex than the Spider. The training, development, and test sets of Spider include 9428, 1534, and 1789 examples and 69, 11, and 15 databases respectively.

Implementation: we have employed the latest proprietary model, GPT-4o, as the backbone of our experiments. We set the temperature of GPT-4o to 0.2. Additionally, we used the text-embedding-3-small model to perform vector encoding of database values.

Evaluation Metrics: we evaluate model performance using the metrics of Execution Accuracy (EX), Reward-based Valid Efficiency Score (R-VES) and Soft F1-score, as defined by the BIRD benchmark. Specifically, EX measures the correctness of the predicted SQL queries by comparing their execution results with those of the ground-truth SQLs. R-VES is an adjusted version of the previously proposed Valid Efficiency Score (VES). R-VES is used to evaluate the efficiency of valid SQL queries generated by the model. It not only considers whether the generated SQL query returns the correct results but also takes into account the resource consumption and execution time during query execution. The soft F1 score provides a more flexible evaluation by reducing the impact of minor discrepancies in the table output, such as column reordering or missing values. Since our experiments are conducted on the commercial LLM of GPT-4o and its performance is very stable, the reported results are single-run, which is also the default practice in most existing work.

4.2 Evaluation Results

We have compared GSR with 16 existing alternatives, and presented the comparative results in Table 1. It is noteworthy that some of the compared alternatives require either **M**ulti-**C**andidate **S**trategy (**MCS**) or **M**odel **F**ine-**T**uning (**MFT**), while the proposed GSR requires neither of them. It can be observed that using the GPT-4o model, GSR achieves the execution accuracy of 69.26% on the test set of BIRD, ranked 11th on the BIRD leaderboard. It performs considerably better than the existing alternatives requiring neither **MCS** and **MFT**, and even outperforms some alternatives leveraging either MCS or MFT, e.g., MCS-SQL and E-SQL. We have also reported the performance of the existing approaches ranked higher than GSR on the BIRD leaderboard and having technical reports or project links. It can be observed that all of them require MCS or MFT; the majority of them even leverage both. By gradually refining a single SQL, GSR can achieve competitive performance with much less prompts and token consumption. For instance, GSR requires averagely only 7 LLM calls, while CHESS, to achieve good performance, typically needs to generate up to 20 candidate SQL queries and invoke at least 30 LLM calls, which require considerably more token consumption. CHASE-SQL similarly uses three different methods to generate totally 21 SQL candidate queries, which requires at least 21 LLM calls, not

Method	MCS	MFT	Model	Date	R-VES	dev EX	test EX
DIN-SQL	No	No	GPT-4	Sep 2023	53.07	50.72	55.90
DAIL-SQL	No	No	GPT-4	Sep 2023	54.02	54.76	57.41
MAC-SQL	No	No	GPT-3.5	Dec 2023	-	50.56	55.90
MAC-SQL	No	No	GPT-4	Dec 2023	57.60	57.56	59.59
DTS-SQL	No	Yes	DeepSeek-7B	Feb 2024	-	55.80	60.31
Codes	No	Yes	Codes-15B	Feb 2024	56.73	58.47	60.37
MCS-SQL	Yes	No	GPT-4	May 2024	61.23	63.36	65.45
TA-SQL	No	No	GPT-4	May 2024	-	56.19	59.14
SuperSQL	No	No	GPT-4	Jul 2024	-	58.50	62.66
E-SQL	Yes	No	GPT-4o-mini	Sep 2024	55.64	61.60	59.81
E-SQL	Yes	No	GPT-4o	Sep 2024	62.43	65.58	66.29
RSL-SQL	Yes	No	DeepSeek	Oct 2024	-	63.56	65.51
GSR(ours)	**No**	**No**	**GPT-4o**	**Jan 2025**	**64.41**	**66.88**	**69.26**
CHESS	Yes	No	Proprietary	Nov 2024	66.53	68.31	**71.10**
Distillery	No	Yes	GPT-4o	Jul 2024	67.41	67.21	**71.83**
CHASE-SQL	Yes	Yes	Gemini	Nov 2024	66.53	74.46	**74.79**
XiYan-SQL	Yes	Yes	GPT-4o	Dec 2024	66.53	73.34	**75.63**

Table 1: Comparison of execution accuracy and reward-based valid efficiency score on the BIRD benchmark: 1) the column of **MCS** denotes whether it uses the **M**ulti-**C**andidate **S**trategy, and the column of **MFT** denotes whether it requires **M**odel **F**ine-**T**uning (for the **MCS** approach, it would be marked as **MFT** if it fine-tunes a SQL selection model; 2) we highlight the results of GSR and the existing approaches ranked higher than GSR on the BIRD test; those ranked higher than GSR require either multi-candidate strategy or model fine-tuning.

the mention the cost of fine-tuning SQL selection model. XiYan-SQL instead fine-tunes 4 SQL generation models and 1 SQL selection model, invoking considerable model fine-tuning cost.

Complexity Level	EX	Soft F1
Simple	77.13	78.24
Moderate	65.77	67.56
Challenging	49.82	52.27
Overall	69.26	70.79

Table 2: The evaluation results on the BIRD using GPT-4o across query complexity levels.

In Table 2, we have also presented a detailed performance breakdown across different query complexity levels on the BIRD test set. It demonstrates that GSR can generate correct SQLs for the majority of simple and moderate queries. Although its execution accuracy for challenging queries is only 49.82%, its overall execution accuracy remains relatively stable. Without relying on multi-candidate SQL generation or model fine-tuning, GSR achieves outstanding results with lower computational overhead by gradually refining the generated SQL.

We have also conducted experiments on the Spider test set to evaluate the generalization capability of the proposed GSR approach. The detailed comparative results have been presented in Table 3.

Method	Model	Date	EX
DIN-SQL	GPT-4	Sep 2023	85.3
DAIL-SQL	GPT-4	Sep 2023	86.6
MAC-SQL	GPT-3.5	Dec 2023	75.5
MAC-SQL	GPT-4	Dec 2023	82.8
DTS-SQL	DeepSeek-7B	Feb 2024	84.4
DEA-SQL	GPT-4	Feb 2024	87.1
TA-SQL	GPT-4	May 2024	85.0
PET-SQL	GPT-4	Jun 2024	87.6
MSC-SQL	Gemma-2-9B	Oct 2024	84.7
RSL-SQL	DeepSeek	Oct 2024	87.5
CHASE-SQL	Gemini	Nov 2024	87.6
GSR(ours)	**GPT-4o**	**Jan 2025**	**87.7**
MCS-SQL	GPT-4	May 2024	**89.6**
RSL-SQL	GPT-4o	Oct 2024	**87.9**
CHESS	Proprietary	Nov 2024	87.2
XiYan-SQL	GPT-4o	Dec 2024	**89.6**

Table 3: The evaluation results in terms of execution accuracy on the Spider test set.

GSR achieves the execution accuracy of 87.7% on the Spider test set, outperforming most of the existing approaches. It is noteworthy that all the three approaches, which perform better than GSR, i.e.,

Method	Simple	Moderate	Challenging	Overall
GSR	72.86	58.62	55.17	**66.88**
w/o Clause Decomposition	72.86	57.76	53.79	$66.49_{\downarrow 0.39}$
w/o SQL-driven Schema Linking	70.05	53.23	53.10	$63.36_{\downarrow 3.52}$
w/o Execution Refinement	71.03	55.82	48.97	$64.34_{\downarrow 2.54}$

Table 4: The evaluation results of ablation study on the BIRD Dev set.

MCS-SQL, RSL-SQL and XiYan-SQL, requires either multi-candidate strategy or model fine-tuning. These results validate the efficacy and strong generalization capability of the GSR.

4.3 Ablation Study

To evaluate the efficacy of each component in the proposed GSR framework, we have conducted an ablation study by systematically removing individual components and measuring the incremental impact of each component in terms of execution accuracy. The evaluation results on the BIRD development set are presented in Table 4 and Table 5, where Table 4 reports the experimental results of ablated GSR after removing each individual component and Table 5 illustrates the impact of each component as they are incrementally incorporated into the solution.

Step	EX
baseline	59.26
+Clause Decomposition	$60.76_{\uparrow 1.50}$
+SQL-driven Schema Linking	$64.34_{\uparrow 3.58}$
+Execution Refinement	$66.88_{\uparrow 2.54}$

Table 5: The evaluation results of incremental contribution in terms of execution accuracy on the BIRD Dev set.

From Table 4, we can observe that without clause decomposition, the performance of GSR remains stable on the simple queries, but drops by 0.86% and 1.38% on the moderate and challenging queries respectively. It clearly demonstrates the efficacy of clause decomposition on complex queries. In contrast, without SQL-driven schema linking, the performance of GSR drops on all the three query categories, with the respective margins of 2.81%, 5.39% and 2.07% on the simple, moderate and challenging queries. It demonstrates the challenge of schema linking even on the simple queries and the efficacy of the proposed SQL-driven approach. The results w.r.t SQL execution refinement are similar. Without the final execution refinement, the performance of GSR consistently drops on all the three query categories.

The incremental evaluation results, as shown in Table 5, also demonstrate that each of the three components can effectively improve the performance of GSR, by the incremental margins of 1.50%, 3.58% and 2.54% respectively. It is noteworthy that SQL-driven schema linking achieves the biggest performance boost, illustrating the central role schema linking plays in Text2SQL.

In summary, our ablation study demonstrates that SQL-driven Schema Linking contributes the most to execution accuracy, playing a critical role in ensuring precise table and column mappings. Execution Refinement, particularly valuable for complex queries, can effectively improve SQL correctness by addressing execution issues such as value mismatches and syntax errors. Clause Decomposition can also provides additional benefits to complex query interpretation. By integrating these three components, the proposed GSR achieves high execution accuracy while minimizing token consumption.

5 Conclusion

In this paper, we propose a novel gradual prompting approach of GSR for Text2SQL, which begins with a preliminary SQL and then iteratively refines it based on SQL analysis and execution. We have presented corresponding techniques for clause decomposition, SQL-driven schema linking and SQL execution refinement to enable the implementation of GSR. Our empirical study on two benchmark datasets have also demonstrated that with only a few prompts , GSR outperforms the existing single-candidate alternatives. Its performance is even highly competitive compared with the existing approaches based on model fine-tuning or multiple-candidate strategy, which require considerably more prompts and token consumption.

Limitations

Our work has the several limitations, which may inspire future research:

- The performance of the proposed GSR is to a large extent dependent on the quality of the preliminary SQL generated in the first step. Even though the proposed clause decomposition can to some extent enhance natural language interpretation for complex queries, the generation of the preliminary SQL may be worthy of further investigation;
- Our current work focus on SQL refinement, but not on multi-candidate SQL generation and selection. Even though the proposed techniques, e.g. SQL-driven schema linking, can be straightforwardly incorporated in the existing MCS solution, a systematic MCS solution based on GSR needs to be further investigated in future work;
- Our current work doesn't investigate model fine-tuning, which may be necessary considering the challenge of Text2SQL and the existing performance gap between LLMs and human beings. However, the general idea of iterative SQL-driven refinement may inspire new fine-tuning strategies for Text2SQL.

Ethics Statement

The datasets and models utilized in this paper, and the implementation of the code and the resulting models, are not associated with any ethical concerns.

References

OpenAI:Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, FlorenciaLeoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mo Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, HyungWon Chung, Dave Cummings, and Jeremiah Currier. 2023. Gpt-4 technical report.

Hasan Alp Caferoğlu and Özgür Ulusoy. 2025. E-sql: Direct schema linking via question enrichment in text-to-sql. *Preprint*, arXiv:2409.16751.

Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. 2024. Rsl-sql: Robust schema linking in text-to-sql generation. *Preprint*, arXiv:2411.00073.

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*.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, lu Chen, Jinshu Lin, and Dongfang Lou. 2023. C3: Zero-shot text-to-sql with chatgpt. *Preprint*, arXiv:2307.07306.

Ben Eyal, Amir Bachar, Ophir Haroche, Moran Mahabi, and Michael Elhadad. 2023. Semantic decomposition of question and sql for text-to-sql parsing.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *Preprint*, arXiv:2308.15363.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. *Preprint*, arXiv:1905.08205.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2024. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation. *Preprint*, arXiv:2405.07467.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *Preprint*, arXiv:1910.13461.

Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024a. The dawn of natural language to sql: Are we fully ready? *Proceedings of the VLDB Endowment*, 17(11):3318–3331.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024b. Codes: Towards building open-source language models for text-to-sql. *Preprint*, arXiv:2402.16347.

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. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Preprint*, arXiv:2305.03111.

Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, and Hangyu Mao. 2024c. Pet-sql: A prompt-enhanced two-round refinement of text-to-sql with cross-consistency. *Preprint*, arXiv:2403.09732.

Linyong Nan, Yilun Zhao, Weijin Zou, Narutatsu Ri, Jaesung Tae, Ellen Zhang, Arman Cohan, Dragomir Radev, Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, JaredD Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, ArielHerbert Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam Mccandlish, Alec Radford, Ilya Sutskever, Dario Amodei, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Oliveira Pinto, JaredKa Plan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Michael Petrov, Heidy Khlaaf, GirishSas Try, Pamela Mishkin, Brooke Chan, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Philippe Tillet, FelipePetroski Such, DaveCum Mings, Matthias Plappert, Fotios Chantzis, ElizaBeth Barnes, Ariel Herbert-Voss, WilliamHebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, AndrewN Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, and Bob Mcgrew. Enhancing few-shot text-to-sql capabilities of large language models: A study on prompt design strategies.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O. Arik. 2024. Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. *Preprint*, arXiv:2410.01943.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Preprint*, arXiv:2304.11015.

Mohammadreza Pourreza and Davood Rafiei. 2024. Dts-sql: Decomposed text-to-sql with small large language models.

Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024. Before generation, align it! a novel and effective strategy for mitigating hallucinations in text-to-sql generation. *Preprint*, arXiv:2405.15307.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the limits of transfer learning with a unified text-to-text transformer. *Preprint*, arXiv:1910.10683.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. Evaluating the text-to-sql capabilities of large language models.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. *Preprint*, arXiv:2109.05093.

Ruoxi Sun, Sercan Ö. Arik, Alex Muzio, Lesly Miculicich, Satya Gundabathula, Pengcheng Yin, Hanjun Dai, Hootan Nakhost, Rajarishi Sinha, Zifeng Wang, and Tomas Pfister. 2024. Sql-palm: Improved large language model adaptation for text-to-sql (extended). *Preprint*, arXiv:2306.00739.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. *Preprint*, arXiv:1409.3215.

Chang-You Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. Exploring chain-of-thought style prompting for text-to-sql.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. Chess: Contextual harnessing for efficient sql synthesis. *Preprint*, arXiv:2405.16755.

Lappoon R. Tang and Raymond J. Mooney. 2001. *Using Multiple Clause Constructors in Inductive Logic Programming for Semantic Parsing*, page 466–477.

Bozenn H. Thompson and Frederick B. Thompson. 1983. Introducing ASK, a simple knowledgeable system. In *First Conference on Applied Natural Language Processing*, pages 17–24, Santa Monica, California, USA. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, AidanN. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Neural Information Processing Systems,Neural Information Processing Systems*.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020a. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020b. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2024. Mac-sql: A multi-agent collaborative framework for text-to-sql. *Preprint*, arXiv:2312.11242.

Chang Wang, Jialu Fan, Aditya Kalyanpur, and David Gondek. 2011. Relation extraction with relation topics. *Empirical Methods in Natural Language Processing,Empirical Methods in Natural Language Processing*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models.

Yuanzhen Xie, Xinzhou Jin, Tao Xie, MingXiong Lin, Liang Chen, Chenyun Yu, Lei Cheng, ChengXiang Zhuo, Bo Hu, and Zang Li. 2024. Decomposition for enhancing attention: Improving llm-based text-to-sql through workflow paradigm. *Preprint*, arXiv:2402.10671.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *Preprint*, arXiv:1711.04436.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018a. Typesql: Knowledge-based type-aware neural text-to-sql generation. *Preprint*, arXiv:1804.09769.

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. 2018b. 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*.

JohnM. Zelle and RaymondJ. Mooney. 1996. Learning to parse database queries using inductive logic programming. *National Conference on Artificial Intelligence,National Conference on Artificial Intelligence*.

Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. 2023. Act-sql: In-context learning for text-to-sql with automatically-generated chain-of-thought. *Preprint*, arXiv:2310.17342.

Hanchong Zhang, Ruisheng Cao, Hongshen Xu, Lu Chen, and Kai Yu. 2024. Coe-sql: In-context learning for multi-turn text-to-sql with chain-of-editions. *Preprint*, arXiv:2405.02712.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *Preprint*, arXiv:1709.00103.

A Prompt Details

A.1 Clause Decomposition

1. Generate Clause

Sentence: Ẃhat is the average amount of loan which are still on running contract with statement issuance after each transaction?´

Break the above sentence down into simple sentences and list them point by point in numerical order, returning only the simple sentences listed in numerical order.

2. Generate Pre-SQL

You are a database expert. Based on the following sections: ###Database Schema, ###Input, ###Hint, and ###Logic Clause, generate the SQL query that meets the requirements of ###Input. Each section provides specific information:

###Database Schema: Details the structure of the database, including tables and columns.
###Input: Specifies the data the user wants to query, including required columns and conditions.
###Hint: Provides additional context or constraints related to the ###Input. Some reference information for you to complete ###Input.
###Logic Clause: Offers further explanation to clarify the query requirements.

Goal: 1. Correctly understand the requirements of ###Input based on ###Logic Clause.
2. Be sure to use the hints given in ###Hint, then determine which part of ###Input the hints are used to complete, and write SQL that combines the contents of ###Hint and ###Input, and do not write anything that is not mentioned in ###Input.
3. Using SQLite syntax, write a single-line SQL query that selects only the columns required by ###Input.

Output Format:

Only return the SQL statement as a single line, following this format:

###SQL: SELECT song_name , song_release_year FROM singer ORDER BY age LIMIT 1; ###END

###Database schema:
financial contains tables such as account, card, client, disp, district, loan, order, trans.
-Table: account:
    -Column: account_id
        -Column_description: the id of the account
    -Column: district_id
        -Column_description: location of branch
    -Column: frequency
        -Column_description: frequency of the acount
    -Column: date
        -Column_description: the creation date of the account
    -Primary Key: account_id
    -Foreign Keys: district_id -> district.(district_id)
-Table: card:
    -Column: card_id
        -Column_description: id number of credit card
    -Column: disp_id
        -Column_description: disposition id
    -Column: type

-Column_description: type of credit card
-Column: issued
-Column_description: the date when the credit card issued
-Primary Key: card_id
-Foreign Keys: disp_id -> disp.(disp_id)
......

###Input:
What is the average amount of loan which are still on running contract with statement issuance after each transaction?

###Hint:
status = 'C' stands for running contract, OK so far; status = 'D' stands for running contract, client in debt. 'POPLATEK PO OBRATU' stands for issuance after transaction

###Logic Clause:
1. What is the average amount of loan?
2. The loans are still on running contract.
3. The loans have statement issuance after each transaction.

A.2 SQL-driven Schema Linking

1. Table Column Extractor

Please help me extract the tables and columns involved in the following SQL statement, then list them. When listing, do not use aliases, and the column names should be enclosed in double quotes. Here are some examples, please follow the format of the examples for output.

###Example 1:
Input:
SELECT MAX("Free Meal Count (K-12)" * 1.0 / "Enrollment (K-12)") AS highest_eligible_free_rate FROM frpm WHERE "County Name" = 'Alameda';
Output:
{Table frpm:
columns:"Free Meal Count (K-12)","Enrollment (K-12)","County Name"}

###Example 2:
Input:
SELECT COUNT(*) FROM satscores s JOIN schools sch ON s.cds = sch.CDSCode WHERE s.AvgScrMath > 400 AND sch.Virtual = 'F';
Output:
{Table satscores:
columns:"cds","AvgScrMath"},
{Table schools:
columns:"CDSCode","Virtual"}

Input:
SELECT AVG(l.amount) FROM loan l JOIN trans t ON l.account_id = t.account_id WHERE l.status IN ('C', 'D') AND t.k_symbol = 'POPLATEK PO OBRATU';

Output:

2. Masked SQL Schema Extractor

You are a database expert. Your task is to help me extract the tables and columns related to the ###Input from the ###Database Schema, based on the following components: ###Database Schema, ###Input, ###Hint.

Each section provides specific information:
###Database Schema: Details the structure of the database, including tables and columns.
###Input: Specifies the data the user wants to query, including required columns and conditions.
###Hint: Provides additional context or constraints related to the ###Input.

Please follow the steps below and write down each step of the process:
1. You need to understand exactly what ###Input needs.
2. Please based on the column_description of the columns of each table, I need you to help me find the columns related to ###Input as per the requirement. For each table, you need to find 1 to 3 columns that may be related to ###Input. Note that each table is required.
3. Please list the columns that you think are related to the ###Input in the format below. For each table, you need to list 1 to 3 columns that may be relevant, even if they are not. Please do not use another format, return only what is in the format below, no additional information. Format:
###Related Schema
{Table satscores:
columns:"cds","AvgScrMath"},
{Table schools:
columns:"CDSCode","Virtual"}
###END

###Database schema:
financial contains tables such as account, card, client, disp, district, loan, order, trans.
-Table: account:
    -Column: account_id
        -Column_description: the id of the account
    -Column: district_id
        -Column_description: location of branch
    -Column: frequency
        -Column_description: frequency of the acount
    -Column: date
        -Column_description: the creation date of the account
    -Primary Key: account_id
    -Foreign Keys: district_id -> district.(district_id)
-Table: card:
    -Column: card_id
        -Column_description: id number of credit card
    -Column: disp_id
        -Column_description: disposition id
    -Column: type
        -Column_description: type of credit card
    -Column: issued
        -Column_description: the date when the credit card issued
    -Primary Key: card_id
    -Foreign Keys: disp_id -> disp.(disp_id)
......

###Input:
What is the average amount of loan which are still on running contract with statement issuance after each transaction?

###Hint:
status = 'C' stands for running contract, OK so far; status = 'D' stands for running contract, client in debt. 'POPLATEK PO OBRATU' stands for issuance after transaction

###Logic Clause:
1. What is the average amount of loan?
2. The loans are still on running contract.
3. The loans have statement issuance after each transaction.

3. Refine Pre-SQL

You are a database expert. Please help me check the Pre-SQL based on ###Input, ###Hint, ###Pre-SQL and ###Value Examples. Please follow the steps below:
1. Pay close attention to the column_description (if provided) for each column in the ###Value Examples. Explicitly write out the column_description, analyze them, and check if the correct columns are being used in the current SQL.
2. Pay close attention to the value_description (if provided) and the value_sample for each column. Explicitly write out the content of the specific value_description and the value in the value_sample.
3. Please check that the value written in the SQL condition exists in the value example, if there may not be a corresponding value in the current column, it is possible that the wrong column is being used, consider whether other columns could complete the ###Input. When performing this step, please refer to the ###Value example and do not rely on the information in the ###Hint.
4. Check the values used in the conditional section of the SQL, compare the values in the SQL with the values in the value_sample displayed, and make sure that the values are case-accurate (this is very important).
5. If you identify any issues with the current SQL after your analysis, please help correct it. While fixing the SQL, ensure that it follows SQLite syntax. If no issues are found, do not make any changes, and provide the original SQL as is.
6. If the SQL contains arithmetic operations, explicitly identify the arithmetic operation parts and force the use of the CAST function to convert those parts to a floating-point type.
7. Provide the final SQL with or without corrections based on your analysis.
8. Please place the final SQL on the last line and write the SQL in a single line following the format below, without adding any line breaks in the SQL and without using any other format:
###SQL: SELECT song_name, song_release_year FROM singer ORDER BY age LIMIT 1; ###END

###Database schema:
financial contains tables such as account, card, client, disp, district, loan, order, trans.
Table account:
  Columns: account_id, district_id, frequency, date
  Primary Key: account_id
  Foreign Keys: district_id -> district.(district_id)
Table card:
  Columns: card_id, disp_id, type, issued
  Primary Key: card_id
  Foreign Keys: disp_id -> disp.(disp_id)
......

###Input:
What is the average amount of loan which are still on running contract with statement issuance after each

transaction?

###Hint:
status = 'C' stands for running contract, OK so far; status = 'D' stands for running contract, client in debt. 'POPLATEK PO OBRATU' stands for issuance after transaction

###Value Examples:
-Table: loan
  -column: date
    -column_description: the date when the loan is approved
    -value_sample: ['1994-01-05', '1996-04-29', '1997-12-08'] (Total records: 682, Unique values: 559)
  -column: loan_id
    -column_description: the id number identifying the loan data
    -value_sample: [4959, 4961, 4962] (Total records: 682, Unique values: 682)
......

###Pre-SQL:
SELECT AVG(l.amount) FROM loan l JOIN trans t ON l.account_id = t.account_id WHERE l.status IN ('C', 'D') AND t.k_symbol = 'POPLATEK PO OBRATU';

A.3 Execution Refinement

The result of the above sql execution is as follows:
[(205065.26074275715,)]

###Input:
What is the average amount of loan which are still on running contract with statement issuance after each transaction?

###Hint:
status = 'C' stands for running contract, OK so far; status = 'D' stands for running contract, client in debt. 'POPLATEK PO OBRATU' stands for issuance after transaction

Please analyze whether the given SQL query meets the following requirements and whether its execution result is reasonable.

Step 1: Requirement Check
- Confirm whether the SQL query aligns with the requirement specified in ###Input.
- Keep an eye on ###Hint for information that is a reference to help you check your SQL, based on the information provided in ###Hint, verify if the SQL query correctly understands and applies the relevant concepts or constraints.
- One situation requires special attention. If you think that the parts related to values in the SQL do not match the ###Hint, please clearly state the relevant value_sample from the ###Value Example. When making corrections to the values, please base them on the value_sample rather than the ###Hint.

Step 2: Result Reasonableness
- Analyze whether the execution result of the SQL query matches the expected outcome and satisfies the requirements in ###Input.
- If the SQL involves arithmetic operations, check that the data types in the arithmetic operations section are correct, and write your analysis in a descriptive manner.
- If the SQL execution result is empty, it indicates an issue with the query, as the database is guaranteed to

contain data that satisfies the ###Input requirements. In such cases, adjust the SQL query to ensure it meets the requirements and returns a valid result.

Guidelines
- If the SQL query already meets the requirements in '###Input' and '###Hint' and produces a reasonable result, no changes are needed.
- If it does not meet the requirements, modify the SQL query to ensure it fulfills all requirements and generates a logical and reasonable result.
- Clearly write out the final corrected SQL in the format below, without using any other format. Format:
###SQL: SELECT song_name, song_release_year FROM singer ORDER BY age LIMIT 1; ###END