C3: Zero-shot Text-to-SQL with ChatGPT

Xuemei Dong[1], **Chao Zhang**[1], **Yuhang Ge**[1], **Yuren Mao**[1], **Yunjun Gao**[1],
Lu Chen[1], **Jinshu Lin**[2], **Dongfang Lou**[2]
[1]Zhejiang University, Zhejiang, China
[2]Hundsun Technologies INC., Zhejiang, China
{xm.dong,zjuzhangchao,yuhangge,yuren.mao,gaoyj,luchen}@zju.edu.cn
loudongfang2022@gmail.com,linjs13607@hundsun.com

Abstract

This paper proposes a ChatGPT-based zero-shot Text-to-SQL method, dubbed C3, which achieves 82.3% in terms of execution accuracy on the holdout test set of Spider and becomes the state-of-the-art zero-shot Text-to-SQL method on the Spider Challenge. C3 consists of three key components: Clear Prompting (CP), Calibration with Hints (CH), and Consistent Output (CO), which are corresponding to the model input, model bias and model output respectively. It provides a systematic treatment for zero-shot Text-to-SQL. Extensive experiments have been conducted to verify the effectiveness and efficiency of our proposed method. Our code is available at https://github.com/bigbigwatermalon/C3SQL

1 Introduction

Text-to-SQL, which aims to convert natural language questions into executable SQL queries, can be used to provide a user-friendly interface to relational databases. It can benefit various aspects of data management, such as accessibility to databases, flexibility of website design and so on.

Traditional Text-to-SQL methods (Li et al., 2023a) typically fine-tune a decoder-encoder model with an amount of training data to achieve proper Text-to-SQL performance. These fine-tuning-based methods require a training set that consists of amounts of text-SQL pairs. However, in practice, obtaining the text-SQL pairs is extremely expensive. Furthermore, this fine-tuning paradigm can cause overfitting of the training set and degenerate the transferability of the model.

To address these problems, few-shot or zero-shot Text-to-SQL methods are natural choices. However, the decoder-encoder model-based methods cannot achieve proper performance (Gu et al., 2023). Recently, the emergent abilities of Large Language Models (LLMs) make few-shot (Pourreza and Rafiei, 2023) or zero-shot (Liu et al., 2023) Text-to-SQL possible. The few-shot Text-to-SQL method (Pourreza and Rafiei, 2023) can outperform the fine-tuning-based methods by means of few-shot in-context learning; however, it requires more than 10,000 tokens for each query, which is expensive and infeasible in practical Text-to-SQL tasks. By contrast, the zero-shot Text-to-SQL method (Liu et al., 2023) can save tokens; however, its performance is inferior to the fine-tuning-based methods.

To address these issues, this paper proposes a novel ChatGPT-based zero-shot Text-to-SQL method, dubbed C3, which utilizes only approximately 1,000 tokens per query and achieves a better performance than fine-tuning-based methods. C3 consists of three key components: Clear Prompting (CP), Calibration with Hints (CH) and Consistent Output (CO), which are corresponding to the model input, model bias and model output respectively. Specifically, CP is a novel prompting paradigm for zero-shot Text-to-SQL, which improves the zero-shot Text-to-SQL performance via adopting proper input. Furthermore, CH is proposed to mitigate the biases of ChatGPT in generating SQL queries, which improves the zero-shot Text-to-SQL performance via calibrating the model biases. Besides, CO is designed to keep the consistency of the generated SQL queries, which improves the zero-shot Text-to-SQL performance via overcoming the inherent randomness and uncertainty in the output of large language models.

We evaluate our proposed C3 method on the widely used Spider dataset, where C3 achieves 82.3% in terms of execution accuracy on the holdout test set of Spider and becomes the state-of-the-art zero-shot Text-to-SQL method. It outperforms state-of-the-art fine-tuning-based approaches by 2.4% execution accuracy on the holdout test set; besides, it only uses approximately 1,000 tokens per query. Furthermore, our extensive experimental analysis provides several noteworthy discoveries,

which are potential to forge a new trend in GPT-based Text-to-SQL research.

2 Related Work

Existing Text-to-SQL methods can be divided into several categories: *rule-based methods*, *fine-tuning methods* and *ICL-based methods*. The *rule-based Text-to-SQL methods* (Zelle and Mooney, 1996; Saha et al., 2016) use well designed templates to generate SQL queries, which have achieved good performance in specific cases. However, these methods rely heavily on manual rule design, making them difficult to apply to other domains and limiting their scalability and generalizability.

To address these limitations, researchers explore Seq2Seq models based on bidirectional LSTMs (Guo et al., 2019) and CNNs (Choi et al., 2021). These models offer improved flexibility and effectiveness but face challenges in integrating database structural information seamlessly. To tackle these constraints, researchers utilize graph neural networks, treating the database schema as a graph, to better integrate structural information (Wang et al., 2020; Cao et al., 2021). Besides, fine-tuning on pre-trained language models like T5 has demonstrated better performance (Raffel et al., 2020; Scholak et al., 2021; Li et al., 2023a). However, fine-tuning methods typically require a substantial volume of labeled training data specific to the target task and they often suffer from overfitting to the training data.

The emergence of large language models (LLM) has left a deep impression and points towards new directions for the Text-to-SQL task. LLMs like GPT models outperform fine-tuning models on many NLP down-stream tasks in few-shot or zero-shot setting due to the ability of in-context learning (ICL) (Brown et al., 2020). The design of prompts in the ICL profoundly influences the quality of outputs (Min et al., 2022; Liu et al., 2022; Wei et al., 2022) in LLM. Some work (Rajkumar et al., 2022; Liu et al., 2023) conduct the evaluation of LLMs' ICL performance on Text-to-SQL task with different prompts. However, none of them outperform the current fine-tuning methods.

All though the standard prompting only provides a lower bound on the capabilities of LLMs in principle (Wei et al., 2022), LLMs with well designed prompts have the potential to achieve better performance. Recently, a few-shot paradigm using GPT-4 has achieved SOTA performance on Text-to-SQL task (Pourreza and Rafiei, 2023). But this method needs some handcraft demonstrations and numerous tokens, which are time-consuming and resource-intensive.

In our work, we demonstrate that ChatGPT can effectively generate SQL queries without the need for any demonstrations when provided with proper instructions. We show its capability as a skilled zero-shot SQL writer.

3 Preliminaries

3.1 Problem Definition of Text-to-SQL

Given natural language question Q and database schema $\mathcal{S}$, the database schema $\mathcal{S} = \{\mathcal{T}, \mathcal{C}, \mathcal{R}\}$ includes multiple tables $\mathcal{T}$, columns $\mathcal{C}$ and foreign key relations $\mathcal{R}$. The problem of Text-to-SQL parsing aims to generate the SQL query $\mathcal{Y}$ corresponding to the question.

3.2 Large Language Model for Text-to-SQL

The task of Text-to-SQL parsing has been formulated as a generation task in recent works (Sun et al., 2023; Liu et al., 2023), utilizing appropriate prompts $\mathcal{P}$ to guide a large language model $\mathcal{M}$. This model estimates a probability distribution over SQL queries $\mathcal{Y}$ and allows us to generate queries token by token. The generation process for the SQL query $\mathcal{Y}$ can be formulated as follows:

$$P_{\mathcal{M}}(\mathcal{Y}|\mathcal{P}, \mathcal{S}, Q) = \prod_{i=1}^{|\mathcal{Y}|} P_{\mathcal{M}}(\mathcal{Y}_i|\mathcal{P}, \mathcal{S}, Q, \mathcal{Y}_{<i}) \tag{1}$$

Here, $\mathcal{Y}_{<i}$ is the prefix of the SQL query $\mathcal{Y}$ and $P_{\mathcal{M}}(\mathcal{Y}_i|\cdot)$ is the conditional probability of the i-th token in the SQL query $\mathcal{Y}$ given the prefix $\mathcal{Y}_{<i}$, the prompt $\mathcal{P}$, the schema $\mathcal{S}$ and question $\mathcal{Q}$.

Recent studies have demonstrated that large language models (LLMs) can learn from a few examples within a given context, known as in-context learning (Sun et al., 2023; Min et al., 2022; Pourreza and Rafiei, 2023). These works have shown that in-context learning effectively enables LLMs to perform a range of complex tasks. However, including additional examples tends to increase the manual cost and token cost associated with using the OpenAI API. Therefore, in this study, we specifically focus on the zero-shot prompt setting.

4 Proposed Approach

In this section, we propose the C3 method which can achieve proper zero-shot Text-to-SQL perfor-

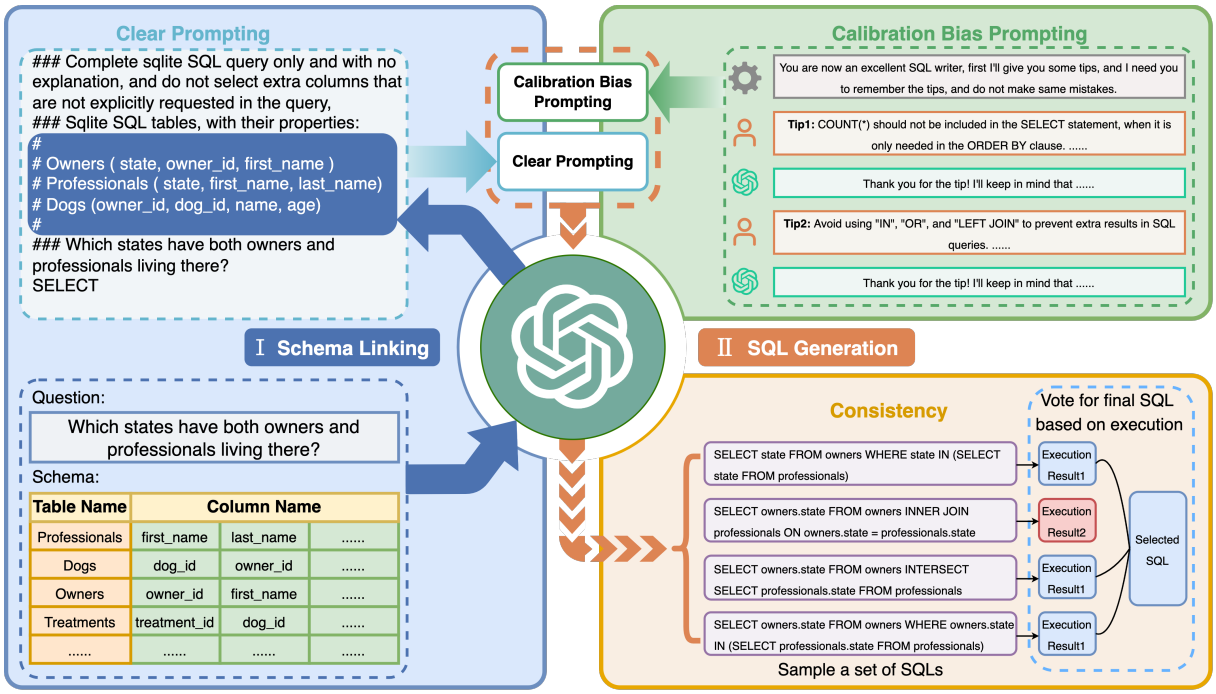


Figure 1: The framework of C3.

mance based on ChatGPT. The framework of the C3 method is demonstrated in Figure 1. C3 consists of three key components: Clear Prompting (CP), Calibration with Hints (CH), and Consistency Output (CO), which are corresponding to the model input, model bias and model output respectively. The details of each component are introduced as follows.

4.1 Clear Prompting

The Clear Prompting (CP) component aims to provide effective prompts for Text-to-SQL parsing. It consists of two parts: clear layout and clear context.

Clear Layout In Text-to-SQL, the widely used prompt layout styles can be divided into two types. We denote them as complicated layout and clear layout respectively. The details of each type are introduced as follows.

- **Type 1: Complicated Layout**: This type of prompt layout directly concatenates the instruction, question and context (database schema) together, which looks a mess. An example of this prompt layout is illustrated in Figure 2a.

- **Type 2: Clear Layout**: This type of layout divides the instruction, context (database schema) and questions by adopting sharp symbols, which look clear. An example of this prompt layout is illustrated in Figure 2b.

Intuitively, the clear layout will be easily understood by ChatGPT and achieve better performance. The experiments illustrate that the clear layout outperforms the complicated layout by 7.0% in term of execution accuracy, which verify this intuition. The experimental setting can be found in Section 5.3). Therefore, this paper adopts a clear layout to construct our prompt template. We also provide an interesting material that asks ChatGPT itself whether clear layout is important (please refer to Appendix A). However, our experimental results show that directly using the instruction provided in Figure 2b causes redundant columns in the generated SQL. To solve this problem, we improve the instruction part by adding a clause *and do not select extra columns that are not explicitly requested in the query*.

Moreover, the context provided in Figure 2b requires to refine. We refine it in the following section.

Clear Context Including the entire database schema in the context part of a prompt causes two issues. Firstly, introducing too many irrelevant items in the prompt increases the likelihood of ChatGPT generating irrelevant schema items in the output SQL. Secondly, using the complete database

Complete sqlite SQL query only and with no explanation.
Which States have both owners and professionals living there? Sqlite SQL tables, with their properties: owners : owners.state , owners.owner_id , owners.first_name | professionals : professional.state , professionals.first_name , professionals.last_name | dogs : dogs.owner_id , dogs.dog_id, dogs.name, dogs.age)

(a) Traditional PLM prompt.

Instruction
Complete sqlite SQL query only and with no explanation.

Context (Schema)
Sqlite SQL tables, with their properties:
#
Owners (state, owner_id, first_name)
Professionals (state, first_name, last_name)
Dogs (owner_id, dog_id, name, age)
#

Question
Which States have both owners and professionals living there?
SELECT

(b) OpenAI prompt.

Instruction
Complete sqlite SQL query only and with no explanation, and do not select extra columns that are not explicitly requested in the query.

Context (Schema)
Sqlite SQL tables, with their properties:
#
Owners (state, owner_id, first_name)
Professionals (state, first_name, last_name)
Dogs (owner_id, dog_id, name, age)
#
Clear Content (Schema Linking)

Question
Which States have both owners and professionals living there?
SELECT

(c) Our proposed C3 prompt.

Figure 2: The comparison of Traditional PLM prompt, OpenAI prompt and C3 prompt.

schema results in excessive text length, leading to unnecessary API costs. To overcome these two issues, we propose that it is necessary to conduct schema linking which recalls relevant tables and columns. With the schema linking results, we can only put the linked tables and columns into the context part. This paper presents a ChatGPT-based zero-shot schema linking approach, which can be divided into the following two steps:

- **Table Recall** We design a zero-shot prompt to instruct ChatGPT to recall tables using three steps. Firstly, the tables should be ranked based on their relevance to the question. Secondly, the model should check if all tables have been considered. Finally, the output format is specified as a list. To ensure the stability of table recall, we employ a self-consistency method. Specifically, the model generates ten sets of retrieval results, each set containing the top four tables. The final result is determined by selecting the set that appears most frequently among the ten sets. The complete prompt is shown in Appendix B.1.

- **Column Recall** Based on the table recall results, we further retrieve the columns within the candidate tables. We also design a zero-shot prompt and instructed ChatGPT to recall columns in two steps. Firstly, all columns within each candidate table are ranked based on their relevance to the question. Then, the output format is specified as a dictionary. In the prompt, we also emphasize that columns matching more with the question words or the foreign key should be placed ahead to assist in more accurate recall results. Similarly, we employ the self-consistency method. Specifically, for each table, we generate ten sets of recalled columns at once. Then we choose five columns that appear most frequently among each set as the final result. The complete prompt is shown in Appendix B.2.

Besides the recalled tables and columns, we also add foreign key information of the recalled tables into the context part to specify the columns required for *JOIN* operations.

Combining the Clear Layout and Clear Context, we propose Clear Prompt, as demonstrated in Figure 2c.

4.2 Calibration of Model Bias

Through analyzing the errors that occurred in the generated SQL queries, we find that some errors are caused by certain biases inherent in ChatGPT. As demonstrated in Figure 3, ChatGPT prefers to provide extra columns and extra execution results. This paper concludes them as the following two kinds of bias.

- **Bias 1**: ChatGPT tends to be conservative in its output and often selects columns that are relevant to the question but not necessarily required. Furthermore, we find that this tendency is particularly pronounced when it comes to issues involving quantities. For example, for the first question in Figure 3 (left), ChatGPT chooses *Year* and *COUNT(*)* in the *SELECT* clause. However, the gold SQL in Spider dataset only selects *Year* as *COUNT(*)* is only needed for ordering purposes.

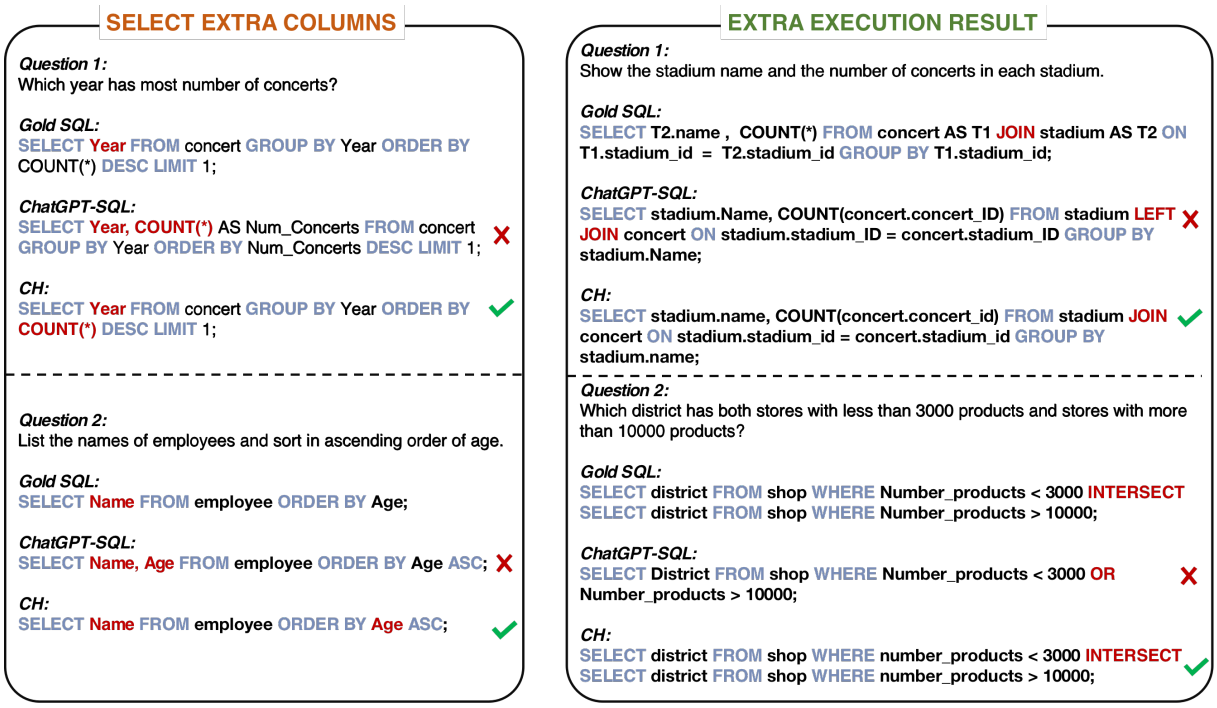


Figure 3: The examples of mistakes caused by the model's biases.

- **Bias** 2: ChatGPT tends to use *LEFT JOIN*, *OR* and *IN* when writing SQL queries, but often fails to use them correctly. This bias often leads to extra values in execution results. Some examples of this bias can be found in Figure 3 (right).

To calibrate these two biases, we propose a plug-in calibration strategy, dubbed *Calibration with Hints (CH)*. CH incorporates prior knowledge into ChatGPT by using contextual prompts which include historical conversations. In the historical conversation, we initially regard ChatGPT as an excellent SQL writer and guide it to follow our proposed debias hints.

- **Hint 1**: For the first bias, we design a tip to guide ChatGPT in selecting only the necessary column. This tip is illustrated in the upper right part of Figure 1. It emphasizes that items like *COUNT(*)* should not be included in *SELECT* clause when it is only needed for ordering purposes.

- **Hint 2**: For the second bias, we design a tip to prevent ChatGPT from misusing SQL keywords. As shown in the upper right part of Figure 1, we straightforwardly ask ChatGPT to avoid using *LEFT JOIN*, *IN* and *OR*, and use *JOIN* and *INTERSECT* instead. We also instruct ChatGPT to use *DISTINCT* or *LIMIT* when appropriate to avoid repetitive execution results.

By incorporating these two hints, ChatGPT can generate SQL queries that align more closely with the desired output. As shown in Figure 3, using our CH prompt can effectively calibrate the model biases. More examples of CH prompt can be found in Appendix C.3.

4.3 Consistency Output

Using the CP and CH methods, ChatGPT is able to generate higher-quality SQL. However, the output of ChatGPT is unstable due to the inherent randomness of large language models (Lin et al., 2022). To find out the influence of the uncertainty output of ChatGPT, we analyze the distribution of correctness counts on the dev set across thirty independent experiments under different prompts, as shown in Figure 4. In this figure, ChatGPT-SQL is the method proposed in literature (Liu et al., 2023); besides, CP and CP + CH denote our proposed Clear Prompt and the combination of Clear Prompt and Clear Hint method respectively. Regardless of the method used, only less than 65% of SQL statements can consistently be written correctly. This means that by enhancing the consistency of the output, the model has a great potential to correctly write the majority of the queries.

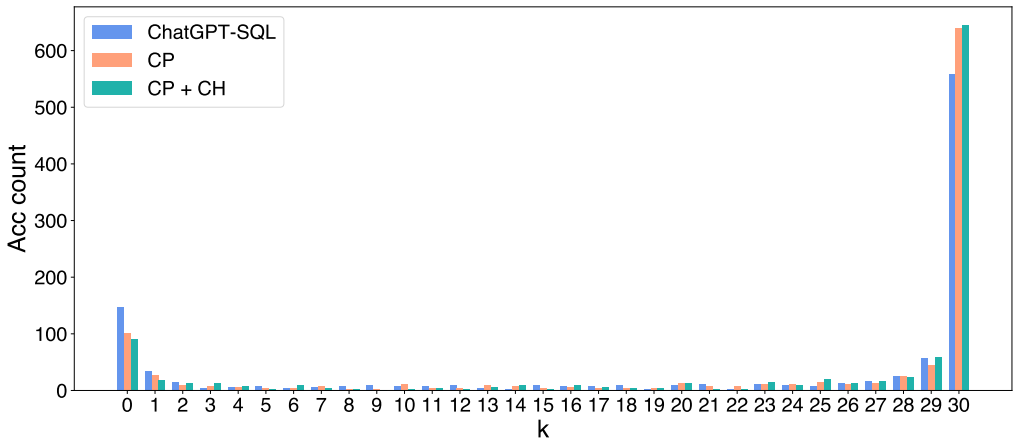


Figure 4: The distribution of correctness counts on the dev set across thirty independent experiments under different prompts. The horizontal axis represents the number of correctness, and the vertical axis represents the number of samples achieving k correct results out of 30 trials.

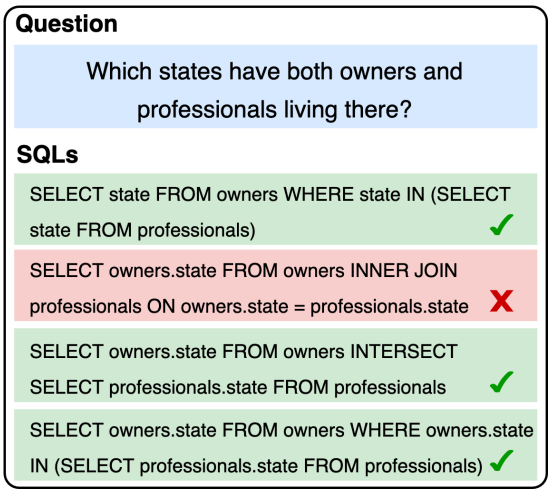


Figure 5: A example of multiple different SQL queries for the same question.

To enhance the consistency, we draw inspiration from the concept of *self-consistency* (Wang et al., 2023) proposed in previous work. *Self-consistency* method is motivated by the fact that in complex reasoning problems, there are multiple different reasoning paths to the unique right answer. It first samples multiple different reasoning paths and then selects the most consistent answer to improve the quality of the output remarkably. Text-to-SQL problem is similar to reasoning problems, where there are multiple ways to write SQL queries to represent the same meaning as shown in the figure 5. Therefore, we implement execution-based *self-consistency* to Text-to-SQL.

Specifically, we first sample multiple reasoning paths to generate diverse SQL answers. Then, execute these SQL queries on the database and collect the execution outcomes. After removing the errors from all the outcomes, we identify the most consistent SQL as the final SQL by applying a voting mechanism to these execution results. For instance, in figure 5, we classify the SQL queries based on the execution outcomes and represent them with different colors. Then we compare the categories to determine which one contains more SQL queries, and select one SQL from that category as the final SQL. This approach allows us to harness the collective knowledge derived from these multiple paths, resulting in more reliable and resilient results in the generation of SQL queries.

5 Experiments

5.1 Experimental Setup

Datasets We conduct experiments on the Spider dataset. Spider (Yu et al., 2018) is a large-scale complex and cross-domain Text-to-SQL dataset. It consists of 10,181 questions and 5,693 unique complex SQL queries on 200 databases with multiple tables covering 138 different domains. It is divided into 8659 training samples across 146 databases, 1034 evaluation samples across 20 databases and 2147 test samples across 34 databases. There is no overlap between the databases used in each of these sets. Meanwhile, according to their difficulty levels, these samples are classified into four categories: easy, medium, hard, and extra.

Evaluation Metrics The most commonly used

Table 1: Comparison of previous methods and our method in terms of execution accuracy on Spider dataset.

Model	Zero-Shot	Few-Shot	Fine-tuning	Dev	Test
ChatGPT-SQL + ChatGPT (Liu et al., 2023)	✓			72.3	-
RATSQL + GAP + NatSQL (Wang et al., 2020)			✓	75.0	73.3
T5-3B + PICARD (Scholak et al., 2021)			✓	79.3	75.1
Graphix-3B + PICARD (Li et al., 2023b)			✓	81.0	77.6
SC-Prompt + T5-3B (Gu et al., 2023)		✓	✓	81.1	-
RESDSQL-3B + NatSQL (Li et al., 2023a)			✓	84.1	79.9
DIN-SQL + GPT-4 (Pourreza and Rafiei, 2023)		✓		82.8	85.3
C3 + ChatGPT (Ours)	✓			81.8	82.3

evaluation metrics are exact match (EM) and execution accuracy (EX). For EM, it considers each part of the SQL query predicted by the model after removing the values, and it is considered a correct prediction only if all parts match the corresponding parts in the standard SQL query. For EX, it evaluates the correctness of the predicted SQL query by comparing the execution results of the predicted SQL and the standard SQL in the same database. In our experiment, we only use EX as the evaluation metrics(Zhong et al., 2020), because we think that the SQL can be written in multiple ways to generate the same result. Moreover, ChatGPT has its own style of writing SQL queries. Thus it is more meaningful to make a direct comparison of the results in our experiment.

Implementation We use OpenAI ChatGPT API: GPT-3.5-Turbo-0301 as our experiment model. We recall tables and columns in two steps with ChatGPT on their relevance according to the given question. Then we construct input prompt using our CP and CH methods. Lastly, we utilize ChatGPT to generate 20 SQL queries at once and select the most consistent SQL as the final result using our CO method.

Baselines We conduct experiments on the Spider dataset and compare our method with the following baselines:

- **ChatGPT-SQL** (Liu et al., 2023) introduces a simple zero-shot method to evaluate the capability of ChatGPT on the Text2SQL task.
- **RATSQL** (Wang et al., 2020) proposes a unified framework utilizing the relation-aware self-attention to encode the relational structure of a database schema and the query.
- **PICARD** (Scholak et al., 2021) introduces an incremental parsing method with constrained decoding, which checks and filters out inadmissible tokens at each generation step.
- **Graphix** (Li et al., 2023b) designs a Graphix layer to encode a combination of semantic and structural information based on the pre-trained T5 model.
- **SC-Promp** (Gu et al., 2023) decomposes the Text2SQL task into two subtasks, predicting structure and content separately, achieving high performance with fewer training samples.
- **RESDSQL** (Li et al., 2023a) proposes a ranking-enhanced encoding and skeleton-aware decoding framework to decouple the schema linking and the skeleton parsing. It is the best approach based on fine-tuning.
- **DIN-SQL** (Pourreza and Rafiei, 2023) decomposes the text-to-SQL task into smaller subtasks and designs different prompts for each subtask to instruct GPT-4 to complete each subtask and obtain the final SQL. It is the SOTA at the time of writing.

5.2 Overall Performance

In Table 1, we report the performance of our method and baseline methods on the Spider dataset. It can be observed that our method outperforms all traditional fine-tuning-based methods in terms of execution accuracy on the test set. Our method also outperforms the zero-shot setting of ChatGPT-SQL by 9.5% on the dev set. At the time of writing, our model achieved 2rd on the Spider Leaderboard. Compared with the few-shot setting of top-1 method, DIN-SQL, our method focuses on the zero-shot setting. Our approach only uses approximately 10% of the token numbers of DIN-SQL;

Table 2: Comparison with different prompt layout

Technique	Dev Acc
Clear Layout	72.3
Complicated Layout	65.3 (↓ 7.0)

Table 3: Ablation Study

Technique	Dev Acc
C3 + ChatGPT	81.8
w/o Recall of Tables and Columns	79.5
w/o Foreign Keys	79.2
w/o Calibration	80.3
w/o Self-Consistency	80.5

furthermore, the cost of GPT-3.5 engine is also significantly lower than GPT-4 which is used in DIN-SQL. Therefore, our method is more budget-friendly.

5.3 Effect of Clear Prompt

In this section, we experimentally investigate the impact of the prompt layout and the content on the performance of Text-to-SQL respectively. The results are the average execution accuracy obtained on the dev set over five runs.

Clear Layout We first explore the effect of clear prompt layout by comparing the performance of the clear layout (refer to Type 2 in Section 4.1) and the complicated layout (refer to Type 1 in Section 4.1). The results are recorded in Table 2. From this table, we can see that clear layout outperforms complicated layout by 7.0%. It demonstrates that the clear prompt layout is important for chatGPT-based zero-shot Text-to-SQL. We suppose that this is because a clear layout allows ChatGPT to better understand the prompt, while the complicated one may blur the information. So a clear layout can help ChatGPT better follow our instructions and make the most of the information provided in the prompt.

Clear Content To investigate the effect of our proposed clear prompt content, we conduct ablation study on (1) recall of tables and columns and (2) specify foreign keys. The results are recorded in Table 3. From this table, we can see that the performance of C3 decreases by 2.3% when removing the operation of recalling of tables and columns; besides, the performance of C3 decreases by 2.6% when removing the operation of specifying foreign keys. It demonstrates that the operations of recalling tables and columns and specifying foreign keys, which provides clear prompt content, are necessary for C3. We further use Area Under ROC Curve (AUC) to evaluate the performance of table and column recall. As illustrated in Section 4.1, we only keep top-4 tables in the database and top-5 columns for each remained table to form a ranked schema sequence. The results are recorded in Table 4, which shows that the AUC of table recalling and column recalling are 0.9588 and 0.9833 respectively.

Table 4: AUC of Table and Column Recall

Table AUC	Column AUC
0.9588	0.9833

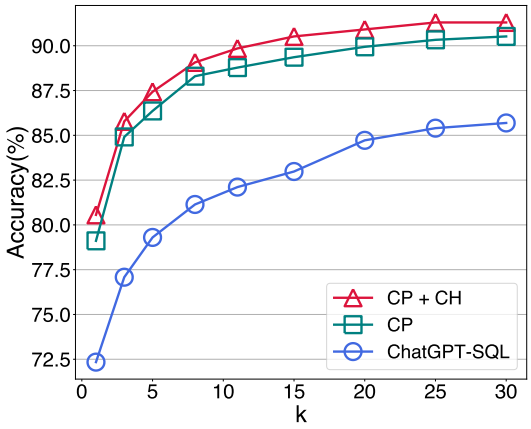


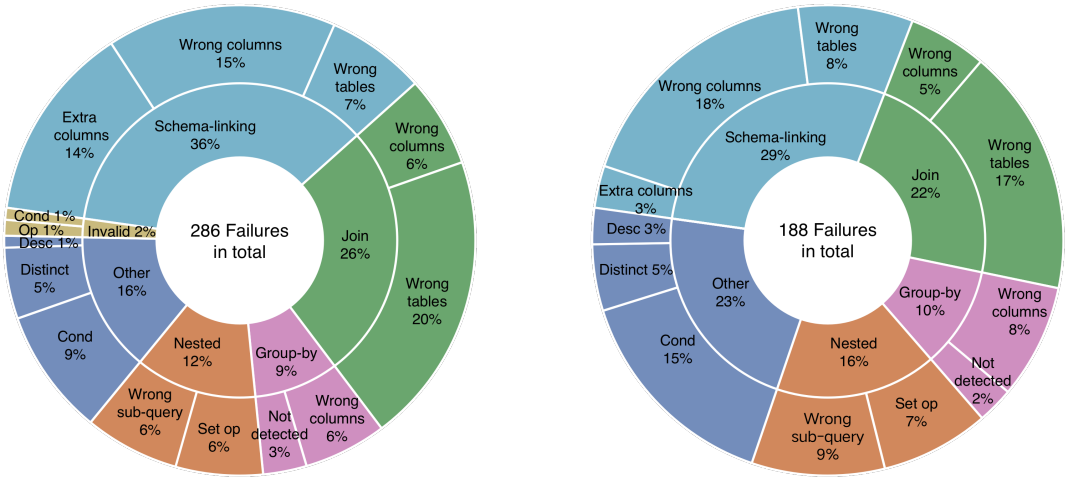
Figure 6: The accuracy of k experiments under different prompts. For each sample in the dev set, if it gets the correct answer at least once in k experiments, it is counted as correct. We gradually increase k until it reaches 30.

5.4 Effect of Calibration of Model Bias

To explore the effect of our proposed calibration method, Calibration with Hints (CH), we perform an ablation study on it. The experimental results are illustrated in Table 3. From this table, we can see that the execution accuracy decreases by 1.5% when calibration is removed. It verifies that calibration of model bias is necessary for chatGPT-based Text-to-SQL.

5.5 Effect of Self-Consistency

In this section, we discuss about the effect of self-consistency. We first conduct an experiment to show ChatGPT's high potential on Text-to-SQL



(a) ChatGPT-SQL + ChatGPT

(b) C3 + ChatGPT (Ours)

Figure 7: Error Distributions of ChatGPT-SQL and C3 on dev set. There are a total of 286 failures in ChatGPT-SQL, and 188 failures in C3. If a sample contains errors of multiple categories, it will be counted in all these categories.

task. We perform k independent experiments and record the changes of accuracy. The results are shown in Figure 6. For each question, if ChatGPT generates one or more correct SQL in the k experiments, we think ChatGPT can generate a accurate SQL for the question. It can be observed that when using ChatGPT-SQL, the accuracy can reach approximately 85.7% when $k = 30$. However, the accuracy is only 72.3% when $k = 1$, resulting in a 13.4% gap. This gap shows the enormous potential of ChatGPT. When using our two prompts, though the gaps are relatively smaller and the curves are flatter, the gap is still noticeable. However, We can find that a slight increase of k can lead to a large increase of accuracy. This is due to the unstability of ChatGPT, which may not generate the correct answer every time. Therefore, as illustrated in 4.3, using self-consistency can utilize multiple answers to get the most consistent one and mitigate this unstability. In the experiment, we set the parameter n to 20 in GPT-3.5 API call function in order to generate twenty SQL queries in a single run. Among these SQL queries, we cluster them based on their execution results and select one SQL query from the largest cluster as the final SQL. The ablation study of self-consistency is recorded in Table 3, we can see that self-consistency improves performance by 1.3% and is therefore necessary to maintain the stable performance.

5.6 Error Analysis

To clearly demonstrate the effectiveness of our proposed method, we conduct an error analysis of ChatGPT-SQL and our C3 method on the dev set. We adopt the error classification method proposed in literature (Pourreza and Rafiei, 2023). Specifically, we manually examined the errors occurred in the generated SQL answers and classified them into six categories, as shown in Figure 7.

Firstly, we can see that, by using our C3 method, the total number of errors decreases by approximately 34% compared to ChatGPT-SQL. The most noteworthy reduction happens in the *Schema-linking* category, which contains three subcategories. Among these subcategories, the subcategory *Extra-columns* is mostly caused by Bias 1(refer to Section 4.2). We can see that the errors belonging to this subcategory has a proportional decrease of 11%. It demonstrates the significant effectiveness of CH in calibrating the model bias.

Secondly, the subcategory *Set op* in category *Nested* demonstrates the misuse of operations like *INTERSECT* and *EXCEPT*, and the subcategory *Distinct* in category *Other* indicates repetitive execution results. Errors belonging to these subcategories are mostly caused by Bias 2 (refer to Section 4.2). Though the proportions of these categories change a little, the actual number of errors in these categories decreases. This indicates that CH can effectively calibrate Bias 2.

And finally, in the category *Schema-linking* and

Join, we can see that the errors of choosing wrong tables and columns significantly decrease, which can be credited to the operation of table and column recall as well as specifying foreign keys in the proposed CP. As for the other categories, the number of errors is also reduced, which can be credited to the CO method for it improves the stability and maintains high performance in each experiment. The overall reduction of errors is attributed to the combined effect of the C3 method.

6 Conclusion

This paper proposes a novel zero-shot Text-to-SQL method based on ChatGPT, called C3, which achieves the state-of-the-art zero-shot Text-to-SQL performance. C3 provides a systematic treatment for GPT-based Text-to-SQL from the perspective of model input, model bias, and model output. It is potential to forge a new trend in the GPT-based Text-to-SQL research.

7 Acknowledgments

We thank Yu Mi and Yilin Li for their valuable assistance with the experiments.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *NeurIPS*.

Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. LGESQL: line graph enhanced text-to-sql model with mixed local and non-local relations. In *ACL*.

DongHyun Choi, Myeongcheol Shin, EungGyun Kim, and Dong Ryeol Shin. 2021. RYANSQL: recursively applying sketch-based slot fillings for complex text-to-sql in cross-domain databases. In *COLING*.

Zihui Gu, Ju Fan, Nan Tang, Lei Cao, Bowen Jia, Sam Madden, and Xiaoyong Du. 2023. Few-shot text-to-sql translation using structure and content prompt learning. In *SIGMOD*.

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. In *ACL*.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. Resdsql: Decoupling the skeleton parsing and schema linking for text-to-sql. In *AAAI*.

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023b. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *AAAI*.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. Teaching models to express their uncertainty in words. *arXiv preprint arXiv:2205.14334*.

Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S Yu. 2023. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. *arXiv preprint arXiv:2303.13547*.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2022. What makes good in-context examples for gpt-3? In *DeeLIO*.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the role of demonstrations: What makes in-context learning work? In *EMNLP*.

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

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal Of Machine Learning Research*, 21:140:1–140:67.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. *arXiv preprint arXiv:2204.00498*.

Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R. Mittal, and Fatma Özcan. 2016. ATHENA: an ontology-driven system for natural language querying over relational data stores. In *VLDB*.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: parsing incrementally for constrained auto-regressive decoding from language models. In *EMNLP*.

Ruoxi Sun, Sercan O Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, and Tomas Pfister. 2023. Sql-palm: Improved large language modeladaptation for text-to-sql. *arXiv preprint arXiv:2306.00739*.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: relation-aware schema encoding and linking for text-to-sql parsers. In *ACL*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *ICLR*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *EMNLP*.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *AAAI*.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-sql with distilled test suite. In *EMNLP*.

A Clear Prompting

Large language models (LLMs) are highly sensitive to the input prompt. A clear prompt can have a substantial positive impact on the performance of downstream tasks. This claim is empirically evidenced by our C3 prompt and corresponding experimental results in the Text-to-SQL task. Furthermore, we present the following question to ChatGPT for suggestions:

Human: **Does using a clear prompt layout benefit ChatGPT in performing Text-to-SQL tasks?**

ChatGPT: Yes, using a structured prompt style can greatly benefit ChatGPT in performing Text-to-SQL tasks. Text-to-SQL tasks involve converting natural language queries into SQL queries that can be executed on a database. These tasks often require a clear understanding of the query intent and the underlying database schema.

Structured prompts provide a specific format or template that guides the model to generate SQL queries in a structured and predictable manner. By using a structured prompt style, you can specify the desired structure and components of the SQL query, such as the SELECT, WHERE, and JOIN clauses, as well as the table and column names. This helps to guide the model's generation process and improve the accuracy and consistency of the generated SQL queries.

Structured prompts can also help overcome some of the challenges of language models, such as ambiguity and verbosity. By providing explicit instructions and constraints in the prompt, you can guide the model to generate more precise and concise SQL queries. This reduces the chance of the model producing queries that are syntactically correct but semantically incorrect or incomplete.

However, it's important to note that while structured prompts can improve the performance of ChatGPT in Text-to-SQL tasks, they are not a complete solution. These tasks can be complex, and the model may still require additional training, fine-tuning, or post-processing techniques to achieve optimal performance. Additionally, structured prompts may limit the model's flexibility in handling novel or out-of-domain queries, so a balance between structure and flexibility should be considered when designing prompts for Text-to-SQL tasks.

B Schema Linking

B.1 The example of table recall prompt

Given the database schema and question, perform the following actions:
1 - Rank all the tables based on the possibility of being used in the SQL according to the question from the most relevant to the least relevant, Table or its column that matches more with the question words is highly relevant and must be placed ahead.
2 - Check whether you consider all the tables.
3 - Output a list object in the order of step 2, Your output should contain all the tables. The format should be like:
[
"table_1", "table_2", ...
]

Schema:
continents (contid, continent)
countries (countryid, countryname, continent)
car_makers (id, maker, fullname, country)
model_list (moddeli, maker, model)
car_names (makeid, model, make)

cars_data (id, mpg, cylinders, edispl, horsepower, weight, accelerate, year)

Question:
What is the name of the different car makers who produced a car in 1970?

B.2 The example of column recall prompt

Given the database tables and question, perform the following actions:
1 - Rank the columns in each table based on the possibility of being used in the SQL, Column that matches more with the question words or the foreign key is highly relevant and must be placed ahead. You should output them in the order of the most relevant to the least relevant.
Explain why you choose each column.
2 - Output a JSON object that contains all the columns in each table according to your explanation. The format should be like:
{
"table_1": ["column_1", "column_2", ...],
"table_2": ["column_1", "column_2", ...],
"table_3": ["column_1", "column_2", ...],
.....
}

Schema:
car_makers (id, maker, fullname, country)
model_list (modelid, maker, model)
car_names (makeid, model, make)
cars_data (id, mpg, cylinders, edispl, horsepower, weight, accelerate, year)
Foreign keys:
model_list.maker = car_makers.id
car_names.model = model_list.model
cars_data.id = car_names.makeid

Question:
What is the name of the different car makers who produced a car in 1970?

C The examples of prompts

C.1 Clear Layout

Complete sqlite SQL query only and with no explanation
Sqlite SQL tables, with their properties:
#
stadium (stadium_id, location, name, capacity, highest, lowest, average);
singer (singer_id, name, country, song_name, song_release_year, age, is_male);
concert (concert_id, concert_name, theme, stadium_id, year);
singer_in_concert (concert_id, singer_id).
#
How many singers do we have?
SELECT

C.2 Complicated Layout

Complete sqlite SQL query only and with no explanation. How many singers do we have? Sqlite SQL tables, with their properties: singer : singer.singer_id , singer.name , singer.country , singer.age ,

singer.song_name , singer.song_release_year , singer.is_male | stadium : stadium.location , stadium.name , stadium.capacity , stadium.highest , stadium.lowest , stadium.average , stadium.stadium_id | concert : concert.theme , concert.year , concert.concert_id , concert.concert_name , concert.stadium_id | singer_in_concert : singer_in_concert.concert_id , singer_in_concert.singer_id
SELECT

C.3 C3 Prompt

Role: SYSTEM
Content:
You are now an excellent SQL writer, first I'll give you some tips and examples, and I need you to remember the tips, and do not make same mistakes.

Role: USER
Content:
Tips 1:
Question: Which A has most number of B?
Gold SQL: select A from B group by A order by count (*) desc limit 1;
Notice that the Gold SQL doesn't select COUNT(*) because the question only wants to know the A and the number should be only used in ORDER BY clause, there are many questions asks in this way, and I need you to remember this in the the following questions.

Role: ASSISTANT
Content:
Thank you for the tip! I'll keep in mind that when the question only asks for a certain field, I should not include the COUNT(*) in the SELECT statement, but instead use it in the ORDER BY clause to sort the results based on the count of that field.

Role: USER
Content:
Tips 2:
Don't use "IN", "OR", "LEFT JOIN" as it might cause extra results, use "INTERSECT" or "EXCEPT" instead, and remember to use "DISTINCT" or "LIMIT" when necessary.
For example,
Question: Who are the A who have been nominated for both B award and C award?
Gold SQL should be: select A from X where award = 'B' intersect select A from X where award = 'C';

Role: ASSISTANT
Content:
Thank you for the tip! I'll remember to use "INTERSECT" or "EXCEPT" instead of "IN", "NOT IN", or "LEFT JOIN" when I want to find records that match or don't match across two tables. Additionally, I'll make sure to use "DISTINCT" or "LIMIT" when necessary to avoid repetitive results or limit the number of results returned.

Role: USER
Content:
Complete sqlite SQL query only and with no explanation, and do not select extra columns that are not explicitly requested in the query.
Sqlite SQL tables, with their properties:
#
singer (singer_id, name, country, age)
stadium (capacity, highest, lowest, average)

concert (theme, year, concert_id, concert_name)
singer_in_concert (concert_id, singer_id)
concert.stadium_id = stadium.stadium_id
singer_in_concert.singer_id = singer.singer_id
singer_in_concert.concert_id = concert.concert_id
#
How many singers do we have?
SELECT