Proceedings of Machine Learning Research **318** (2026)
The 39th Canadian Conference on Artificial Intelligence

Struct-SQL: Distilling Structured Reasoning for Small Text-to-SQL Models

Khushboo Thaker[†,*], Yony Bresler[†]
[†]Crater Labs, Toronto, Canada

Abstract

Deploying accurate *Text-to-SQL* systems at the enterprise level faces a difficult trilemma involving cost, security and performance. Current solutions force enterprises to choose between expensive proprietary Large Language Models (LLMs) and low-performing Small Language Models (SLMs). Efforts to improve SLMs often rely on distilling reasoning from large LLMs using unstructured Chain-of-Thought (CoT) traces, a process that remains inherently ambiguous. Instead, we hypothesize that a formal, structured reasoning representation provides a clearer, more reliable teaching signal, as the *Text-to-SQL* task requires explicit and precise logical steps. To evaluate this hypothesis, we propose *Struct-SQL*, a novel Knowledge Distillation (*KD*) framework that trains an SLM to emulate a powerful large LLM. To implement this approach, we adopt the query execution plan as a formal blueprint to derive structured reasoning. Our SLM, distilled with a structured CoT, achieves an absolute improvement of 8.1% over an unstructured CoT distillation baseline. A detailed error analysis reveals that a key factor in this gain is a marked reduction in syntactic errors. This demonstrates that teaching a model to reason using a structured logical blueprint is beneficial for reliable SQL generation in SLMs.

Keywords: *Text-to-SQL*, Knowledge Distillation, In-Context learning

1. Introduction

Text-to-SQL systems have seen substantial advancements driven by Large Language Models (LLMs) [1]. However, widespread enterprise adoption is hindered by an Adoption Trilemma balancing cost, security, and performance. Enterprises are often forced to choose between expensive, proprietary APIs and private, open-source Small Language Models (SLMs) that lack sufficient reasoning capabilities. While open-source SLMs offer a secure and cost-effective alternative, they typically lack the zero-shot reasoning capabilities required for complex real-world queries [2].

Much of the recent progress in Text-to-SQL is attributed to reasoning-driven prompting and In-Context Learning (ICL). ICL-based methods, particularly those that use decomposition and multi-step reasoning, have demonstrated substantial gains in performance on the *Text-to-SQL* task [1]. A prominent ICL technique for this is Chain-of-Thought (CoT), which encourages models to think step-by-step [3, 4]. More advanced methods like *DAIL-SQL* [5], *DIN-SQL* [6], and *DC-CoT* [7] improve accuracy by breaking questions into intermediate sub-queries. Building on this logical foundation, the Query Plan CoT (*QP-CoT*) guides the model through a structured database execution plan, ensuring that the generation process mirrors the inherent logic of the database engine itself [7]. Although these reasoning techniques achieve considerable success, their effectiveness is observed almost exclusively in the large LLMs. This reliance on large LLMs constitutes a significant limitation: these methods depend on the very models that exacerbate the cost and security challenges of the Adoption Trilemma.

The performance gap for private SLMs is significant. On the BIRD mini-dev benchmark, LLMs such as *GPT-4o* and *Claude 3.7 Sonnet* ($\geq$150B parameters) achieve execution accuracies between 30% and 45% using standard prompting, whereas widely used SLMs such

[*] corresponding_khushboo@craterlabs.io

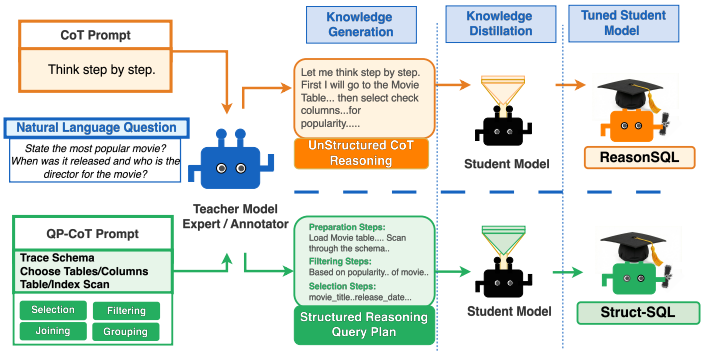


Figure 1. Unstructured vs. Structured Reasoning Distillation. The figure contrasts the two methods of reasoning distillation: (Top) Unstructured Distillation (*ReasonSQL*), which relies on a free-form CoT prompt, with (Bottom) the proposed Structured Distillation (*Struct-SQL*), which uses a *QP-CoT* prompt to generate a structured logical blueprint. The Teacher Model's output serves as the supervisory signal for *KD*.

as *Mistral*, *Mixtral* and *Qwen2.5 Coder* (∼7B parameters) achieve only 4% to 12% with the standard prompt. Our experiments reveal that this degradation persists even with advanced prompting; SLMs fail to internalize logical decompositions and frequently hallucinate schema elements [8]. This motivates investigating whether the *structure of reasoning* itself can be transferred.

To bridge this performance gap, Knowledge Distillation (*KD*) [9], which transfers reasoning ability from a capable teacher model, is a key strategy [1]. *KD* is a model compression technique in which a smaller "student" model is trained to mimic the behavior of a larger, pretrained "teacher" model. Beyond compression, *KD* enables the transfer of complex reasoning [10] and logical skills [11]. Through *KD*, our aim is to build SLMs for *Text-to-SQL* that deliver accuracy comparable to larger LLMs while meeting enterprise cost and security requirements through private deployment. While prior work like *ReasonSQL* [12] distills unstructured natural language traces, we hypothesize that the structure of the reasoning signal is critical.

To evaluate this hypothesis, we introduce *Struct-SQL*, a framework for distilling structured reasoning. Figure 1 shows the *KD* workflow, contrasting the proposed structured reasoning approach against the standard unstructured CoT method. Within this framework, a state-of-the-art (SOTA) Teacher Model is used to generate a *QP-CoT* trace, which formally decomposes the query into a logical execution plan [7]. This structured plan, together with the generated SQL query, constitutes the supervisory signal. A student model is then trained to replicate the entire structured output sequence (query plan, SQL). The formal query plan serves as a clear, hierarchical blueprint that guides the student model in learning the precise logical steps of query construction, from schema linking and join-path selection to aggregation and filtering. During inference, this structured reasoning is retained: the student model is given the *QP-CoT* prompt to autonomously generate the query plan before synthesizing the final SQL query.

To validate this approach, we perform an extensive comparative analysis on the BIRD mini-dev benchmark [2], evaluating *Struct-SQL* against key baselines. Our contributions are threefold: (1) First Systematic Evaluation: We present the first systematic study on distilling structured reasoning signals for *Text-to-SQL*, demonstrating that *Struct-SQL* outperforms unstructured baselines. (2) Mechanism Analysis & Generalization: Through comprehensive error analysis, we show that our structured approach significantly reduces syntactic hallucinations (e.g., schema errors) and generalizes effectively across different SLM architectures. (3) Open Artifacts: We release the code (https://github.com/craterlabs/

struct-sql-distillation) and, the model and the dataset (https://huggingface.co/collections/craterlabs/struct-sql) to facilitate reproducible research.

2. Methodology

2.1. Problem Formulation

The standard *Text-to-SQL* task involves mapping a natural language question Q and a database schema S to an executable SQL query Y_{Gold}. To transfer capabilities from a high-performance Teacher M_T to a smaller Student M_S via KD, we formulate the task as learning the teacher's intermediate reasoning steps R_T alongside the final output Y_T. Let $Z_T = R_T \oplus Y_T$ represent the complete output sequence. We train the student model parameters θ on a dataset $\mathcal{D}_{\text{DISTILL}}$ by minimizing the standard sequence completion loss on Z_T:

$$\mathcal{L}_{KD} = - \sum_{(Q,S,Z_T)\in\mathcal{D}_{\text{DISTILL}}} \log P_{M_S(\theta)}(Z_T|Q,S) \tag{2.1}$$

We instantiate R_T either as an unstructured CoT or a structured *QP-CoT* to test the hypothesis that a structured reasoning trace provides a more effective supervisory signal.

2.2. Structured CoT via Query Execution Plan

Our structured CoT strategy is inspired by a database engine's query execution plan. A query execution plan defines the precise sequence of steps that a database follows to access and manipulate data, often generated using an *EXPLAIN* command. Adapting this *QP-CoT* strategy [7], we prompt the Teacher Model to generate a query plan along with the final SQL query (refer to Appendix C for an example). This plan decomposes the query into a sequential execution flow that explicitly performs selections, filters, joins, and groupings through step-by-step table scanning and data manipulation, providing a systematic signal for the student model. Although the query plan has been explored as a prompting technique for large LLMs [7], the key contribution here is to demonstrate the effectiveness of this structured reasoning as the primary teaching signal within a *KD* framework.

2.3. Experimental Setup

The experiments use the SQLite-based *BIRD benchmark* for the data [2]. The BIRD training dataset is used for model training, while the BIRD mini-dev dataset serves for evaluation. The Teacher Model (M_T), GPT-4o (OpenAI), is selected due to its high performance on the BIRD mini-dev dataset and serves as an oracle to generate high-quality query plans. The *Student Model* (M_S) is *Qwen3-4B-Instruct-2507* (Alibaba Cloud). It was chosen for its strong performance-to-size ratio, which enables low-latency, private deployment and thus addresses the Adoption Trilemma [13]. All models utilize a single-pass inference mechanism and operate without multi-agent collaboration, self-consistency checks or external correction loops. For a fair comparison with baselines, no external reasoning signals were provided at test time. During inference, the student model autonomously generates the Query Execution Plan based on the input prompt before generating the final SQL.

2.3.1. Model Configurations

We compare model performance across two distinct categories to isolate the impact of supervision signals. First, pretrained models establish intrinsic ICL bounds without parameter updates; we use *GPT-4o* (M_T) as the teacher upper bound and *Qwen3-4B* (M_S) as the student lower bound. Second, we evaluate tuned student models. We finetune *FN-Gold* using the Gold SQL query (Y_{Gold}) on the BIRD training dataset, with a basic system instruction that directly maps natural language to SQL [6] and *ReasonSQL* on unstructured

CoT traces to serve as standard baselines. Finally, for Struct-SQL (Structured KD), we fine-tune on the formal query plan sequence Z_T to test our core hypothesis. During inference, all models employ the *QP-CoT* prompt, except for *ReasonSQL*, which uses the unstructured CoT prompt.

2.3.2. Distillation Dataset Construction

The *KD* datasets were constructed using active generation and filtering to maximize data quality and query complexity diversity. Databases in the corpus are partitioned into 75% in-domain (ID) and 25% out-of-domain (OOD) pools. We applied stratified, success-based sampling across SQL complexity categories listed in Appendix A, Table 2. A sample is admitted only if GPT-4o generates syntactically valid and execution-correct SQL, yielding 1,000 training samples, 150 ID validation samples, and 150 OOD validation samples. To ensure a controlled comparison, we utilized the same data generation pipeline for both *Struct-SQL* and *ReasonSQL*, differing only in the format of the supervisory signal: unstructured CoT traces for *ReasonSQL* versus structured *QP-CoT* for *Struct-SQL*.

2.4. Implementation and Evaluation Details

2.4.1. Post-Training Details

All models were finetuned using QLORA on a single NVIDIA H200 GPU. Detailed training configurations, including hyperparameters, batch sizes, and adapter settings, are available in the extended preprint [14]. We employed an early-stopping strategy based on aggregated validation loss to ensure robust generalization across both in-domain and out-of-domain datasets.

2.4.2. Evaluation Metrics

The primary metric is Execution Accuracy (EX), which measures the percentage of generated SQL queries that execute without error and return the same result set as the ground-truth SQL query on the BIRD mini-dev set. A detailed analysis of model failure modes reveals why and where certain models perform best. To facilitate analysis, we classified failures into three distinct types. This classification establishes a severity hierarchy: starting with the most severe Generation Failure (GEN), where the model produces no recognizable SQL output; followed by Syntactic Failure (SYN), indicating an unexecutable query due to grammar or schema errors; and finally, Logical Failure (SEM), representing a query that is syntactically correct but semantically inaccurate. Recognizing this progression is beneficial for discerning the specific challenges and improvements associated with each model. Detailed categories and subcategories are listed in Table 3 in Appendix B.

3. Results

3.1. Overall Performance

The execution accuracy of *Struct-SQL*, compared to all baselines on the BIRD mini-dev dataset, provides strong support for the central hypothesis. As summarized in Table 1, distilling structured reasoning results in significantly better performance compared to unstructured distillation, *ReasonSQL* and traditional finetuning *FN-Gold*. The native Student Model's performance (17.0%) illustrates the Performance Trade-Off in the Adoption Trilemma. *Struct-SQL* achieved an 8.1-point absolute improvement from 36.90% to 45.00% over the *ReasonSQL* baseline. To disentangle the effect of the inference prompt from the training signal, we also evaluate *ReasonSQL* with the QP-CoT prompt, denoted as *ReasonSQL* (mismatched) in Table 1. Its performance drops from 36.9% to 29.2%, revealing a prompt-training mismatch. This confirms that *Struct-SQL*'s gains are attributable to the structured supervisory signal during training, not merely to the *QP-CoT* prompt format at

Base Model	Method	Prompt	Overall EX (%)	Simple	Mod.	Chall.
Main Experiment: Qwen3-4B-Instruct-2507 (Student) vs. GPT-4o (Teacher)						
-	Teacher (GPT-4o)	*QP-CoT*	53.60	68.24	52.00	36.27
Qwen3-4B	Base Student	*QP-CoT*	17.00	34.45	9.60	9.80
Qwen3-4B	FN-Gold	*QP-CoT*	34.30	45.94	32.80	20.58
Qwen3-4B	*ReasonSQL*	CoT	36.90	49.32	33.20	27.45
Qwen3-4B	*ReasonSQL* (mismatched)[1]	*QP-CoT*	29.20	46.62	24.80	14.71
Qwen3-4B	***Struct-SQL* (Ours)**	*QP-CoT*	**45.00**	**65.54**	**40.40**	25.49
Qwen3-4B	*Struct-SQL* (Official BIRD Test)[2]	*QP-CoT*	**60.42**	69.02	54.41	43.51
Generalization Study: Mistral-7B						
Mistral-7B	Base Student	*QP-CoT*	7.22	11.49	6.00	3.92
Mistral-7B	*ReasonSQL*	CoT	25.10	40.54	20.08	12.75
Mistral-7B	***Struct-SQL* (Ours)**	*QP-CoT*	**29.31**	**49.32**	**23.20**	**14.71**

Table 1. Execution Accuracy (EX) results. The upper section compares our primary student model (Qwen3-4B-Instruct-2705) against baselines on BIRD mini-dev, including the official leaderboard result on the non public test set. The lower section demonstrates the generalization of *Struct-SQL* to a different architecture (Mistral-7B).

inference time. To validate the transferability of our framework to other architectures, we replicated our experiments on *Mistral-7B-Instruct-v3.0*. This model was selected for its low zero-shot accuracy of 7.2% with *QP-CoT*. *Struct-SQL* shows generalizability, achieving a performance advantage (29.3% EX) over the Student Model (7.2% EX) and the *ReasonSQL* baseline (25.1% EX), confirming that the structured query plan offers a robust supervision signal independent of the base model.

Our model obtained 60.42% EX on the non-public BIRD benchmark [2], securing the top position among ≤4B models (as of January 30, 2026) in single model inference. This result is achieved using strict single-model inference with greedy decoding and no self-consistency.

3.2. Failure Analysis: The Syntactic Bottleneck

As visually summarized in Figure 2, a key dichotomy emerges between the Teacher and the tuned student models. The Teacher Model (M_T), shown in (a), achieved the highest success rate, with failures predominantly stemming from semantic errors (∼30%) rather than from syntax. In contrast, the student models face a severe syntactic bottleneck.

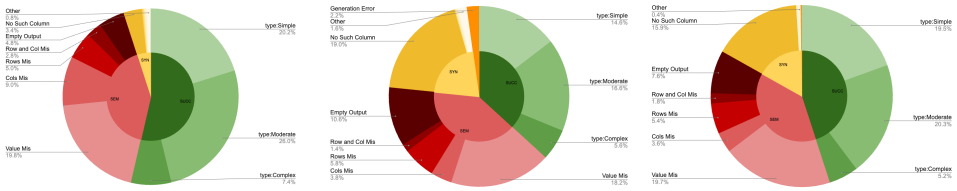
The unstructured distillation baseline (*ReasonSQL*), shown in (b), improves stability over naive models but fails to strictly enforce schema adherence. While it achieves an execution accuracy of 36.9%, it retains a high syntactic error rate of 21.2%. The model struggles with basic schema linking, as evidenced by the persistence of 'No Such Column' errors.

In contrast, *Struct-SQL* (c) directly addresses this bottleneck. There is a marked reduction in syntactic errors from 21.2% for *ReasonSQL* to 16.8% for *Struct-SQL* (a 4.4-point reduction), specifically minimizing schema hallucinations from 19.0% to 15.8% (a 3.2-point reduction) and eliminating 'Keyword Issues'. Although this structural rigidity leads to a minor trade-off in value mismatches, the substantial reduction in syntactic and generation errors drives the overall performance gain. Failure distributions for the Base Student and *FN-Gold* are provided in Appendix B, Figure 3. Additional analysis and details are available in the extended preprint [14].

4. Conclusion

This work presents *Struct-SQL*, a *KD* framework that transfers structured reasoning (*QP-CoT*) from a Teacher LLM to a smaller student model for the *Text-to-SQL* task. The experiments demonstrate that distilling structured reasoning is a better teaching method.

[1]Distilled using CoT, evaluated with QP-CoT. [2]https://bird-bench.github.io/



(a) Teacher using *QP-CoT* *(b) ReasonSQL* using CoT *(c) Struct-SQL* using *QP-CoT*

Figure 2. Failure Analysis: Compared to the Teacher (a), the baseline *ReasonSQL* using CoT (b) exhibits higher syntactic errors (red). *Struct-SQL* (c) successfully mitigates this, reducing syntactic errors from 21.2% to 16.8%.

Detailed error analysis indicates that these improvements are primarily due to a significant reduction in syntactic errors. While structured generation introduces a minor inference overhead, the substantial gains in reliability establish *Struct-SQL* as a robust blueprint for efficient, private reasoning.

Acknowledgments

We thank Khalid Eidoo, Abdul Hamid Dabboussi, María Rodríguez-Liñán and Ting Fung Lam for their insightful technical discussions and valuable feedback.

References

[1] L. Shi, Z. Tang, N. Zhang, X. Zhang, and Z. Yang. "A Survey on Employing Large Language Models for Text-to-SQL Tasks". In: *ACM Comput. Surv.* 58.2 (2025).

[2] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo, et al. "Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls". In: *Advances in Neural Information Processing Systems* 36 (2024).

[3] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. "Chain-of-thought prompting elicits reasoning in large language models". In: *NeurIPS* (2022).

[4] H. Liu, H. Li, X. Zhang, R. Chen, H. Xu, T. Tian, Q. Qi, and J. Zhang. "Uncovering the impact of chain-of-thought reasoning for direct preference optimization: Lessons from text-to-sql". In: *arXiv:2502.11656* (2025).

[5] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou. "Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation". In: *Proc. VLDB Endow.* (2024).

[6] M. Pourreza and D. Rafiei. "DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction". In: *Neural Information Processing Systems* (2023).

[7] M. Pourreza, H. Li, R. Sun, Y. Chung, S. Talaei, G. T. Kakkar, Y. Gan, A. Saberi, F. Ozcan, and S. O. Arik. "CHASE-SQL: Multi-Path Reasoning and Preference Optimized Candidate Selection in Text-to-SQL". In: *Intl. Conf. on Learning Representations* (2024).

[8] K. Maamari, F. Abubaker, D. Jaroslawicz, and A. Mhedhbi. "The Death of Schema Linking? Text-to-SQL in the Age of Well-Reasoned Language Models". In: *NeurIPS RL Wo.* 2024.

[9] G Hinton. "Distilling the Knowledge in a Neural Network". In: *DLRL Workshop, NIPS.* 2014.

[10] S. Li, J. Chen, Z. Chen, X. Zhang, Z. Li, H. Wang, J. Qian, B. Peng, Y. Mao, W. Chen, et al. "Explanations from Large Language Models Make Small Reasoners Better". In: *Sustainable AI.* 2024.

[11] C.-Y. Hsieh, C.-L. Li, C.-K. Yeh, H. Nakhost, Y. Fujii, A. Ratner, R. Krishna, C.-Y. Lee, and T. Pfister. "Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes". In: *Findings ACL.* 2023.

[12] G. Rossiello, N. H. Pham, M. Glass, J. Lee, and D. Subramanian. "Rationalization Models for Text-to-SQL". In: *Workshop on Reasoning and Planning for LLM.* 2025.

[13] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, et al. "Qwen3 technical report". In: *arXiv:2505.09388* (2025).

[14] K. Thaker and Y. Bresler. *Knowledge Distillation with Structured Chain-of-Thought for Text-to-SQL.* 2025. arXiv: 2512.17053 [cs.CL].

Appendix A. Distillation Dataset Details

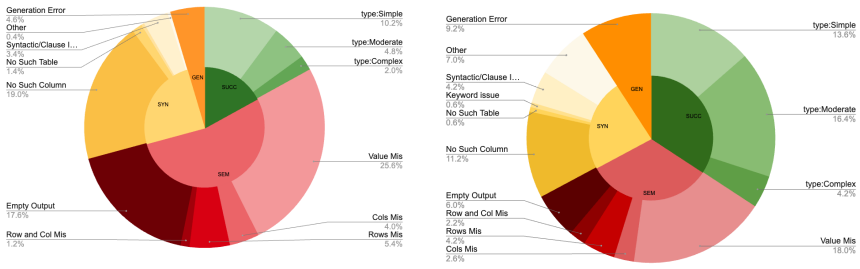
Complexity Category	Training Count	ID Val Count	OOD Val Count
Single Table Queries	295 (29.50%)	37 (24.67%)	46 (30.67%)
Subquery (no join or set operations)	229 (22.90%)	39 (26.00%)	31 (20.67%)
With JOINs / Set Ops (no Subquery)	398 (39.80%)	57 (38.00%)	60 (40.00%)
JOINs / Set Ops and Subquery	78 (7.80%)	17 (11.33%)	13 (8.67%)
Total Successful Samples	**1000**	**150**	**150**

Table 2. Distribution of Query Complexity Across Distillation Datasets. The data is partitioned by unique database identifiers to evaluate generalization. ID Val = In-Domain Validation; OOD Val = Out-of-Domain Validation (databases unseen in training).

Appendix B. Error Taxonomy and Additional Failure Analysis

Error Type	Subcategory	Description
Generation (GEN)	—	No SQL output
Syntactic (SYN)	No Such Column	Non-existent column reference
	No Such Table	Non-existent table reference
	Keyword Issue	Incorrect or misplaced SQL keyword
	Syntax/Clause Order	Incorrect clause order, missing parentheses
	Other	Unclassified syntactic errors
Semantic (SEM)	Column Mismatch	Incorrect number of columns
	Row Mismatch	Incorrect number of row
	Row & Column Mismatch	Incorrect number of columns and rows
	Value Mismatch	Incorrect data values
	Empty Output	Empty result set

Table 3. The proposed hierarchical error taxonomy. This table categorizes *Text-to-SQL* errors in order of decreasing severity and is used for subsequent failure analysis.



(a) Student Model using *QP-CoT* *(b) FN-Gold* using *QP-CoT*

Figure 3. The Student Model (a) and the *FN-Gold* (b) exhibit substantially lower performance than Teacher Model, primarily due to high syntactic errors.

Appendix C. *Struct-SQL* Output Example

This section provides an example of the *Struct-SQL* model's output. The entry includes the Question ID and Database ID for traceability.

Struct-SQL Output (Difficulty: Challenging)

Question ID: 36
Database ID: `california_schools`
Question: Under whose administration is the school with the highest number of students scoring 1500 or more on the SAT? Indicate their full names.

— Query Plan —

```
## Preparation Steps:
1. Initialize the process: Start preparing to execute the query.
2. Prepare storage: Set up storage space (registers) to hold temporary results,
   initializing them to NULL.
3. Open the satscores table: Open the satscores table so we can read from it.
4. Open the schools table: Open the schools table so we can read from it.
** Finding the School with the Highest Number of Students Scoring 1500 or More:**
1. Start reading the satscores table: Move to the first row in the satscores
   table.
2. Check if the score is 1500 or more: Look at the `NumGE1500` column of the
   current row in `satscores`. If it's not greater than or equal to 1500,
   skip this row.
3. Identify the matching row: Store the `cds` of this score entry. This `cds` is
   for the school with the highest number of students scoring 1500 or more.
** Retrieving Administrator Full Names:**
1. Prepare to retrieve administrator names: If a `cds` is found from the previous
   step, prepare to retrieve administrator names.
2. Start reading the schools table: Move to the first row in the `schools` table.
3. Check if the school code matches: Look at the `CDSCode` column of the current
   row in `schools`. If it doesn't match the `cds` found in the "Finding the
   School with the Highest Number of Students Scoring 1500 or More" step,
   skip this row.
4. Retrieve administrator names: For each matching row, retrieve the `AdmFName1`,
   `AdmLName1`, `AdmFName2`, `AdmLName2`, `AdmFName3`, and `AdmLName3` columns,
   which represent the full
   names of the administrators.
5. Move to the next row in schools: Go back to the `schools` table and move to
   the next row, repeating the process until all rows are checked.
** Delivering the Result:**
1. Output the administrator full names: Output the full names of the
   administrators for the school with the highest number of students scoring
   1500 or more.
2. End the process: Stop the query execution process.
```

SQL Query:

```
SELECT T1.AdmFName1, T1.AdmLName1, T1.AdmFName2, T1.AdmLName2,
    T1.AdmFName3, T1.AdmLName3 FROM schools AS T1 INNER JOIN
    satscores AS T2 ON T1.CDSCode = T2.cds WHERE T2.NumGE1500 = (
     SELECT MAX(NumGE1500) FROM satscores )
```