GRIT: Guided Relational Integration for Efficient Multi-Table Understanding

Yujin Kang[1] **Seong Woo Park**[2] **Yoon-Sik Cho**[1]*
[1]Department of Artificial Intelligence, Chung-Ang University, Republic of Korea
[2]Intellivix, Republic of Korea
{zinzin32, yoonsik}@cau.ac.kr, starspder@intellivix.ai

Abstract

Recent advances in large language models (LLMs) have opened new possibilities for table-based tasks. However, most existing methods remain confined to single-table settings, limiting their applicability to real-world databases composed of multiple interrelated tables. In multi-table scenarios, LLMs face two key challenges: reasoning over relational structures beyond sequential text, and handling the input length limitations imposed by large-scale table concatenation. To address these issues, we propose Guided Relational Integration for multiple Tables (**GRIT**), a lightweight method that converts relational schemas into LLM-friendly textual representations. GRIT employs hashing-based techniques to efficiently infer primary–foreign key relationships and constructs prompts that explicitly encode relevant join paths and question-relevant columns. GRIT consistently improves table-column retrieval performance across diverse multi-table benchmarks while significantly reducing memory and computational overhead.

1 Introduction

Tables convey complex information through structured row-column relationships, which contributes to high data density and representational efficiency. To effectively understand such structured data, a number of pioneering works (Yin et al., 2020; Herzig et al., 2020; Wang et al., 2021; Deng et al., 2022) have proposed various methods for explicitly encoding table structures. With the rise of large language models (LLMs), recent studies have explored extending their semantic generalization capabilities to table-based tasks (Sui et al., 2024a; Ma et al., 2024; Nahid and Rafiei, 2024; Patnaik et al., 2024).

Specifically, end users are increasingly turning to LLMs with natural language queries in hopes of end-to-end reasoning. These LLM-based table tasks commonly involve a retrieval step, where relevant tables and columns are first identified before generating a response. However, Nararatwong et al. (2024) show that LLMs frequently fail at this retrieval step, and their downstream reasoning performance improves significantly when provided with the correct tabular data. This highlights that retrieving the relevant table-column pairs is critical to the quality and reliability of user-facing outputs in practical applications. While various retrieval methods have been proposed to filter relevant content and enhance table understanding (Ye et al., 2023; Wang et al., 2024; Chen et al., 2024b), most focus on *single-table* scenarios, making them insufficient for practical cases when queries span multiple tables.

In practice, most databases consist of multiple interrelated tables, necessitating research on multi-table reasoning and requiring LLMs to handle this

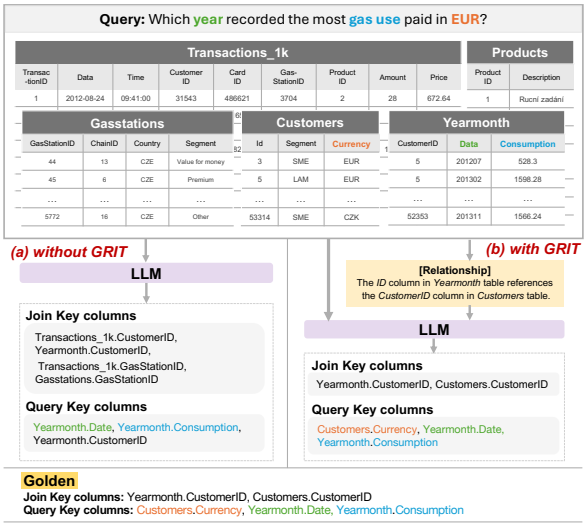

Figure 1: Overview of table-column retrieval without and with our method, GRIT. Given a query and the full database schema, the model selects relevant table-column pairs. Most table rows are omitted for clarity.

*Corresponding author.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 21996–22009
November 4-9, 2025

structural complexity. However, as queries may require integrating data across several tables via joins or foreign key relationships, multi-table structures make information retrieval inherently challenging. Specifically, LLMs face several key challenges when dealing with multi-table scenarios:

First, multi-table reasoning requires understanding inter-table schema relationships, which becomes particularly difficult for LLMs pretrained on unstructured text (Yin et al., 2020; Li et al., 2024). Unlike the unidirectional token prediction in text, tables require a two-dimensional structural understanding—spanning both rows and columns—making direct value comparison and schema mapping substantially more challenging for LLMs. Second, the use of multiple tables often causes the input size to exceed the context limits of LLMs. Indeed, this problem is not limited to multi-table setting, which has been discussed earlier. Prior studies have consistently shown that LLM performance degrades as input length increases (Ye et al., 2023; Levy et al., 2024; Zhang et al., 2024a; Patnaik et al., 2024; Nahid and Rafiei, 2024). Furthermore, Chen et al. (2024b) point out the limitation of existing approaches for table reasoning which require the entire table as input.

As shown in Figure 1 (a), the LLM fails to effectively handle multi-table inputs. Limited context prevents the model from capturing essential inter-table relationships, causing it to retrieve incorrect table-column pairs based on surface-level name similarity. Recent studies have made pioneering attempts to address multi-table tasks. However, these approaches still struggle with large input sizes inherent in multi-table scenarios. An LM-based approach (Pal et al., 2023) suffers from input truncation due to limited context length. Following work (Chen et al., 2024a) tries to optimize join performance but remains limited by substantial computational costs.

In this paper, we shed light on the underexplored challenges of multi-table reasoning by proposing Guided Relational Integration for multiple Tables (**GRIT**). GRIT introduces a lightweight hashing-based algorithm that efficiently estimates column uniqueness and identifies referential links between tables. This approach enables schema-level relationship extraction while avoiding the computational cost of exhaustive pairwise comparisons. Based on the inferred key mappings, we generate natural language prompts that explicitly encode table and column names along with their referential links. Rather than feeding the full table contents, GRIT provides LLMs with compact schema metadata that preserves essential relational context while significantly reducing the input size.

We evaluate GRIT on four closed-source and two open-source LLMs using Spider (Yu et al., 2018) and Bird (Li et al., 2023), two multi-table datasets. GRIT consistently improves table-column retrieval performance across all models. Furthermore, our method demonstrates substantial improvements in both inference time and memory efficiency, particularly in large-scale multi-table scenarios. Our contribution is three-fold.

- We propose a lightweight hashing-based method, *GRIT*, for scalable extraction of table-level structural dependencies and transforming relational schemas into LLM-interpretable prompts.
- The proposed method not only maintains high accuracy in identifying table relationships but also achieves efficiency in time and memory usage.
- We integrate GRIT into multiple proprietary and open-source LLMs and demonstrate that it consistently improves table-column retrieval performance across all baselines.

2 Related works

Early works in table understanding (Yin et al., 2020; Herzig et al., 2020; Wang et al., 2021; Deng et al., 2022) focus on explicitly encoding table structures for various downstream tasks. Building upon these foundations, the emergence of LLMs has accelerated new approaches to extend their capabilities to table-based tasks, either through specialized fine-tuning methods (Li et al., 2024; Zhang et al., 2024b) or by integrating advanced reasoning techniques such as chain-of-thought prompting (Wang et al., 2024). Researchers have further expanded into multimodal approaches that combine textual and visual information processing (Alonso et al., 2024; Zheng et al., 2024). However, despite these advancements, the literature has predominantly focused on *single-table settings*, leaving multi-table scenarios relatively underexplored.

In real-world applications, data is often organized into *multi-table* relational databases. These settings introduce complex reasoning challenges, as queries frequently require integrating information through joins and foreign key relationships. Pal et al. (2023) introduced a pretrained model

designed specifically for multi-table question answering (QA) and table generation. While this represents significant progress in multi-table reasoning, the task remains fundamentally challenging due to the difficulty of accurately retrieving and integrating relevant information from multiple tables. Recognizing the importance of schema-guided retrieval in such settings, Chen et al. (2024a) propose a join-aware retrieval framework that identifies relevant tables by analyzing query intent and schema relationships. However, without refining retrieval at the column level, LLMs are still burdened with processing unnecessary content. Building upon these prior efforts, our work introduces a fine-grained table-column retrieval approach that addresses these limitations.

On the other hand, several studies have questioned the efficiency of LM-based methods for table tasks, as table tasks typically require the entire table as input. Row-column retrieval studies (Lin et al., 2023; Sui et al., 2024b) attempt to encode essential information for retrieval purposes. However, these methods suffer from degraded encoding quality as table length increases and incur substantial computational costs for embedding entire tables. Recently, TableRAG (Chen et al., 2024b) propose an approach combining schema retrieval with selective cell value retrieval and frequency-aware truncation. While this method efficiently delivers table information to LLMs regardless of table size, it remains applicable only to single-table scenarios and cannot address larger multi-table scenarios that require understanding relationships between tables. In this work, we leverage hashing-based algorithms to efficiently identify multi-table schemas, providing both temporal and spatial efficiency even for large-scale tables within complex relational database environments.

3 Methodology

3.1 Problem Definition

A standard multi-table retrieval task aims to return a set of relevant tables from a table corpus given a query Q. In this paper, we extend this task to multi-table column retrieval, which seeks to identify the relevant tables and the specific columns required to answer the query precisely. Given a query Q and a database $\mathcal{D} = \{T_1, T_2, \ldots, T_J\}$ consisting of multiple tables, the task is to retrieve all table-column pairs (T, C) that are necessary to construct the correct answer. In multi-table scenarios, it is essential to retrieve both the query-relevant columns needed to construct the answer and the join-supporting columns that act as intermediate keys for navigating across related tables. Without these relational links, the reasoning path cannot be constructed, rendering multi-hop inference over the database schema infeasible. Therefore, the retrieval objective is to extract both *join-key* and *query-key* table-column pairs.

3.2 Efficient Relation Discovery for Multi-Table Understanding

User queries over tabular data frequently necessitate integrating information across multiple tables. As illustrated in Figure 2, answering such queries typically involves simultaneous value matching across distinct tables. LLMs are required to infer relationships among disjoint records, yet they are inherently optimized for unidirectional text sequences and lack inductive bias for modeling two-dimensional tabular structures (Li et al., 2024). Moreover, while multi-table reasoning often relies on value-level comparisons, feeding the full content of multiple tables into LLMs is computationally prohibitive. For example, in a financial database, a *contact* column from the *user* table and a similarly named *contact* column from the *bank* table may appear semantically similar, yet refer to entirely different entities. Without examining the underlying values, such cases can lead to erroneous matches. Previous research has attempted to address this issue through exhaustive pairwise value comparisons, such as computing Jaccard similarity across all column pairs (Chen et al., 2024a). These approaches, however, raise concerns regarding computational cost and scalability.

Our method, GRIT, pre-analyzes table structures and encodes them into a more compact, semantically aligned format, which is then provided to the LLM. This design alleviates the need for structural reasoning at inference time and allows the model to focus on query-relevant operations. Furthermore, by avoiding exhaustive value-level comparisons, our method remains scalable and efficient even when applied to large-scale tables with million plus rows—making it well-suited for multi-table environments.

3.2.1 Primary Key Detection

Primary keys (PKs) designate a unique identifier for each row in a table, which is essential for enabling other tables to reference specific records un-

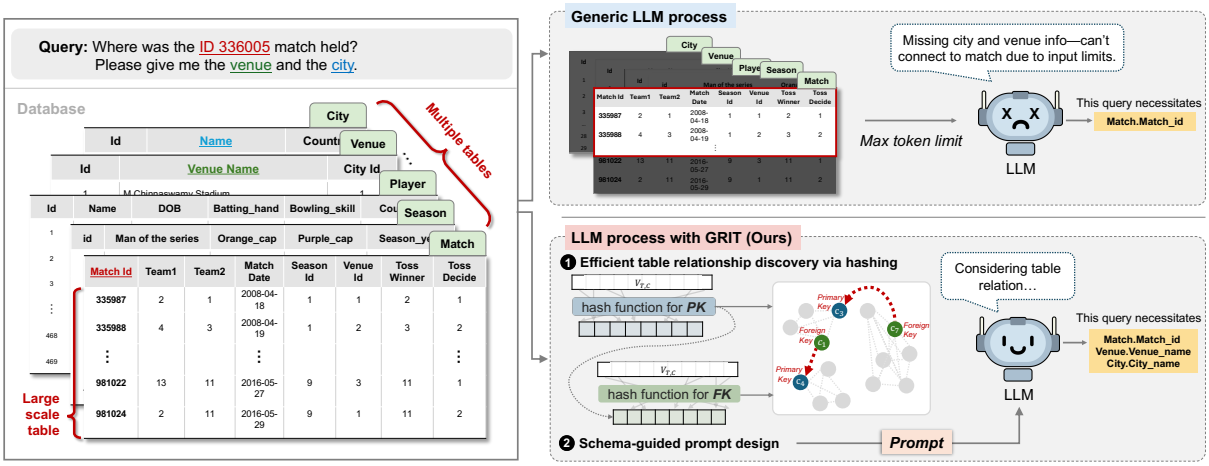

Figure 2: Framework of GRIT. The left block illustrates the challenges in multi-table settings, where conventional LLMs struggle to identify relevant table-column pairs. GRIT mitigates this by pre-analyzing table relationships, and enabling efficient multi-table processing.

ambiguously. Foreign keys (FKs) reference these identifiers across tables, thereby enabling the explicit modeling of join paths. Our approach focuses on identifying primary and foreign key relationships as the foundation for table structure understanding. The process of detecting PK candidates begins with analyzing each table T and its columns. For a given column C, PK detection aims to identify columns whose values are unique across all rows. This is equivalent to evaluating the cardinality of the value set *i.e.*, the number of distinct values contained in the column.

We employ the HyperLogLog (HLL) algorithm (Flajolet et al., 2007). HLL leverages stochastic principles to efficiently approximate the count of unique elements in large-scale data. Each input value is hashed into a binary string using a hash function [1], and the hash space is divided into multiple buckets based on a subset of the hash bits. Each bucket records only the maximum number of leading zeros observed among the hashed values assigned to it. The final cardinality estimate is computed by aggregating these maximum leading-zero counts across buckets using a harmonic mean. This probabilistic cardinality estimation technique offers near-linear scalability in data volume while requiring only logarithmic space. The following equation defines the identification of primary key candidates for each column across all tables in the database.

$$\mathcal{K}_{\text{PK}} = \{(T, C) \mid T \in \mathcal{D},\ C \in \mathcal{C}_T,\ \frac{\text{HLL}(V_{(T,C)})}{|V_{(T,C)}|} \geq \tau_{\text{PK}}\} \quad (1)$$

, where $\mathcal{D}$ and $\mathcal{C}_T$ denote the database consisting of multiple tables and the set of columns in table T, respectively. $V_{(T,C)}$ is the set of all values in column C of Table T. τ_{PK} is a hyperparameter that controls the threshold for uniqueness of PK. HLL eliminates the need for materializing all values in memory, making it highly memory-efficient, with fixed-size storage regardless of the number of rows. Thus, by estimating the distinct ratio of each column using HLL, we efficiently detect primary key candidates.

3.2.2 Foreign Key Detection

After identifying candidate primary keys, the next stage focuses on detecting foreign key (FK) relationships between tables. Specifically, for every column C in any table $T \in \mathcal{D}$ such that $T \neq T_{\text{pk}}$, we verify whether its values reference a primary key candidate $(T_{\text{pk}}, C_{\text{pk}}) \in \mathcal{K}_{\text{PK}}$. A foreign key relationship implies that the value set of C should be a subset of that in C_{pk}: $V_{T,C} \subseteq V_{T_{\text{pk}},C_{\text{pk}}}$.

Enforcing this condition exactly requires materializing and comparing large sets, which is computationally expensive and memory-intensive, especially when candidate columns contain substantial row counts. To mitigate this challenge, we leverage Bloom Filter (Bloom, 1970) to approximate membership in the candidate PK set. A Bloom filter is a probabilistic data structure that enables fast and

[1]We adopt SHA-1 as the hash function, following the default provided in Datasketch's HyperLogLog library.

memory-efficient membership testing. Each element is hashed using i independent hash functions, and the resulting i positions in a bit array are set to 1. To check whether an element is in the set, it is hashed again using the same functions; if all corresponding bits are 1, the element is possibly present; if any bit is 0, we consider the element is definitely not in the set. The containment score can be approximated by:

$$\phi_{\text{con}}(C, C_{\text{pk}}) = \frac{1}{|V_{T,C}|} \sum_{v \in V_{T,C}} \mathbf{1}\{BF(v) = 1\} \quad (2)$$

, where $\mathbf{1}\{BF(v) = 1\}$ is an indicator function that returns 1 if the value v is present in the Bloom filter constructed from $V_{T_{\text{pk}},C_{\text{pk}}}$, and 0 otherwise.

However, only relying on containment scores can be misleading, particularly in numeric columns, where unrelated fields may share overlapping values. For example, consider the following two tables:

country

origin	country
1	USA
2	Europe
3	Japan

production

id	model_year	country
1	1970	1
2	1970	1
...	...	...
391	1982	3
392	1982	1

The *origin* column in the *country* table may be incorrectly identified as a foreign key referencing the *id* column in the *production* table, simply because the values in *origin* (e.g., 1, 2, 3) fall within the numeric range of *id* from *production*. Under containment-based matching, columns may be mistakenly identified as forming valid PK-FK relationships due to value overlap, even in the absence of any semantic relationship. Thus, containment serves as a precision-oriented measure of relational similarity but does not ensure structural validity. To address this, we incorporate cardinality score and name similarity score as complementary features.

For structural validation, we define the cardinality score as:

$$\phi_{\text{card}}(C, C_{\text{pk}}) = \frac{\text{HLL}(V_{T,C})}{\text{HLL}(V_{T_{\text{pk}},C_{\text{pk}}})}. \quad (3)$$

Here, $\phi_{\text{card}}(C, C_{\text{pk}})$ denotes the cardinality ratio between a foreign key candidate column C and a primary key candidate C_{pk}, estimated using HyperLogLog. It provides a normalized signal of value distribution characteristics by measuring the ratio of unique values in the candidate FK to those in the candidate PK.

In addition to cardinality, we also consider name similarity. Let $\mathcal{T}_C$ denote the set of tokens extracted from the name of column C. We define $f(\mathcal{T}_C, \mathcal{T}_{C_{\text{pk}}})$ as the number of matched token pairs between $\mathcal{T}_C$ and $\mathcal{T}_{C_{\text{pk}}}$, where two tokens are considered matched if one is a substring of the other. The name similarity score is then computed as:

$$\phi_{\text{name}}(C, C_{\text{pk}}) = \frac{2 \cdot f(\mathcal{T}_C, \mathcal{T}_{C_{\text{pk}}})}{|\mathcal{T}_C| + |\mathcal{T}_{C_{\text{pk}}}|}. \quad (4)$$

While more sophisticated methods exist for computing name similarity such as embedding-based approaches using language models, we opt for a token-overlap-based strategy to ensure inference efficiency.

We integrate three features into a unified scoring function that estimates the likelihood of a column C serving as a valid foreign key for a given primary key candidate C_{pk}:

$$S_{C,C_{\text{pk}}} = w_{\text{con}} \cdot \phi_{\text{con}} + w_{\text{card}} \cdot \phi_{\text{card}} + w_{\text{name}} \cdot \phi_{\text{name}} \quad (5)$$

, where w_{con}, w_{card}, and w_{name} are non-negative weights representing the relative importance of containment, cardinality ratio, and name similarity, respectively.

For each candidate PK, we compute a final score indicating the likelihood that any given column from any table serves as its FK. Since each FK can reference only one PK, we assign the FK to the PK with the highest score, provided that the score exceeds a predefined threshold τ_{FK}. This procedure ensures that only the most plausible PK–FK pairs are retained. Finally, we obtain all valid PK–FK pairs across all tables in the given databases. The entire procedure for extracting primary–foreign key relationships is presented in Algorithm 1.

3.3 Schema-Aware Prompt Design for Table-column Retrieval

In multi-table settings, effective table-column retrieval with LLMs requires a clear understanding of inter-table relationships. While LLMs excel at handling natural language, they often fall short when it comes to understanding relational database structures. Here, we transform the table structures and relationships identified in the previous section into an LLM-friendly textual representation. In our approach, we do not provide the full table contents to the LLM. Instead, we supply only the headers which allows the model to grasp the structure of each table. Additionally, we explicitly specify the

Algorithm 1 GRIT: Structure Extraction via PK–FK Inference.

Input:
Database $\mathcal{D} = \{T_1, T_2, ..., T_J\}$
Output: Detected PK–FK relationships $\mathcal{K}_{\text{PK}}, \mathcal{K}_{\text{FK}}$ over tables in $\mathcal{D}$

$\mathcal{K}_{\text{PK}} = \{\}$
$\mathcal{K}_{\text{FK}} = \{\}$
▷ `Step 1: Detect Primary Key`
for $T \in \mathcal{D}$ **do**
  for $C \in \mathcal{C}_T$ **do**
    cardinality $\gamma \leftarrow \text{HyperLogLog}(V_{(T,C)})$
    if $\gamma / |V_{(T,C)}| \geq \tau_{\text{PK}}$ **then**
      $\mathcal{K}_{\text{PK}} \leftarrow \mathcal{K}_{\text{PK}} \cup \{(T, C)\}$
    end if
  end for
end for
▷ `Step 2: Detect PK-FK Relationship`
for $T \in \mathcal{D}$ **do**
  for $C \in \mathcal{C}_T$ **do**
    $S_{\text{best}} \leftarrow 0$, $C_{\text{best}} \leftarrow \text{None}$
    for $(T_{\text{pk}}, C_{\text{pk}}) \in \mathcal{K}_{\text{PK}}$ **where** $T_{\text{pk}} \neq T_s$ **do**
      $\phi_{\text{con}}(C, C_{\text{pk}}) = \frac{\sum_{v \in V_{T,C}} \mathbf{1}\{BF(v)=1\}}{|V_{T,C}|}$
      $\phi_{\text{card}}(C, C_{\text{pk}}) = \frac{\text{HLL}(V_{T,C})}{\text{HLL}(V_{T_{\text{pk}}, C_{\text{pk}}})}$
      $\phi_{\text{name}}(C, C_{\text{pk}}) = \frac{2 \cdot f(\mathcal{T}, \mathcal{T}_{\text{pk}})}{|\mathcal{T}| + |\mathcal{T}_{\text{pk}}|}$
      $S \leftarrow w_{\text{con}} \cdot \phi_{\text{con}} + w_{\text{card}} \cdot \phi_{\text{card}} + w_{\text{name}} \cdot \phi_{\text{name}}$
      if $S > S_{\text{best}}$ **then**
        $S_{\text{best}} \leftarrow S$, $C_{\text{best}} \leftarrow (T_{\text{pk}}, C_{\text{pk}})$
      end if
    end for
    if $S_{\text{best}} \geq \tau_{\text{FK}}$ **then**
      $\mathcal{K}_{\text{FK}} \leftarrow \mathcal{K}_{\text{FK}} \cup \{(T, C) \rightarrow C_{\text{best}}\}$
    end if
  end for
end for
return $\mathcal{K}_{\text{PK}}, \mathcal{K}_{\text{FK}}$

referential relationships between columns across different tables, based on the primary-foreign key constraints identified in the previous section. By presenting this structural metadata in advance, the model is relieved of the burden of processing raw data values and can instead focus on reasoning over inter-table connections and the roles of columns. See Appendix B for detailed prompts.

This input design offers two key advantages in table-column retrieval with LLMs. First, by using only table headers, the model requires minimal input and is able to infer the organization of the database schema without being distracted by row-level content. Second, the explicit specification of inter-table relationships eliminates the need for the model to infer join paths, allowing it to focus more directly on identifying which columns are relevant to answering the question. As a result, the model allocates more attention to information selection and column relevance, rather than structural inference, ultimately improving both the accuracy and efficiency of table-column retrieval.

4 Experimental setting

Dataset. We follow prior table-retrieval studies (Chen et al., 2024a) using large-scale multi-table datasets and adopt the text-to-SQL datasets Spider (Yu et al., 2018) and Bird (Li et al., 2023). These datasets consist of databases organized by topic, with queries specific to each topic that require reasoning over the corresponding tables. Based on the verified gold SQL expressions from each dataset, we extract the table-column pairs involved in each query using GPT-4. The prompt used for this extraction is provided in the Appendix A.2. Although the datasets provide explicit primary and foreign key, we do not use them, as such relationships are often unavailable in real-world scenarios. To reflect the multi-table setting, we restrict our evaluation to queries that require joining at least two tables. Further details and statistics of datasets can be found in the Appendix A.

Evaluation metric. We evaluate table-column retrieval performance from two perspectives. Answering a query requires two types of columns: (1) columns necessary for joining tables, and (2) columns directly contributing to the final answer. Accordingly, we separate the evaluation into *join key* table-column pairs and *Query* table-column pairs. We evaluate performance using precision, recall, and F1 score, which are suited for table-column retrieval tasks where each query involves multiple relevant columns. Precision measures the proportion of correctly predicted columns among all predictions, indicating how well the model avoids irrelevant outputs, while recall reflects the model's ability to recover all necessary columns required to answer the query.

Backbone. We evaluate four API-based LLMs—GPT-3.5 Turbo, Claude Haiku, Claude Sonnet (Anthropic, 2024), and Grok 3—and two open-source models, LLaMA 3 (Grattafiori et al., 2024) and Mistral (Jiang et al., 2023). These

Model	Input	Bird		Spider	
		Join-key	Query-key	Join-key	Query-key
Closed-source LLM					
GPT-3.5 Turbo	Header	64.05	59.39	69.81	71.28
	+ GRIT	67.98 (+ 6.14%)	60.74 (+ 2.27%)	73.32 (+ 5.03%)	71.86 (+ 0.82%)
Claude Haiku	Header	64.68	73.12	81.43	79.41
	+ GRIT	73.08 (+ 12.98%)	74.13 (+ 1.38%)	86.22 (+ 5.88%)	82.32 (+ 3.67%)
Claude Sonnet	Header	80.63	74.86	87.06	80.48
	+ GRIT	82.16 (+ 1.90%)	75.33 (+ 0.62%)	89.73 (+ 3.06%)	82.26 (+ 2.21%)
Grok3	Header	78.76	74.38	84.33	80.74
	+ GRIT	79.88 (+ 1.43%)	75.33 (+ 1.28%)	86.55 (+ 2.64%)	81.38 (+ 0.80%)
Open-source LLM					
LLaMA3	Header	24.00	33.77	26.86	52.75
	+ GRIT	34.02 (+ 41.76%)	36.48 (+ 8.03%)	44.95 (+ 67.32%)	55.51 (+ 5.23%)
Mistral	Header	4.28	44.45	19.33	55.62
	+ GRIT	8.63 (+ 101.74%)	48.88 (+ 9.98%)	37.30 (+ 92.95%)	56.86 (+ 2.22%)

Table 1: Performance comparison of LLMs with and without GRIT-enhanced schema in table-column retrieval under multi-table settings. The Header setting provides only table and column names as input, whereas the GRIT setting additionally includes schema relationships inferred through our method. Join-key refers to the performance of identifying the table-column pairs required for join operations, while Query-key evaluates the performance of retrieving the table-columns necessary to answer the question.

Model	Input	
	Header	+ GRIT
GPT-3.5 Turbo	34.68	53.71 (+ 54.89%)
Claude Haiku	39.11	55.14 (+ 41.0%)

Table 2: Performance comparison of LLMs with and without GRIT-enhanced schema in text-to-SQL task.

LLMs, selected for their strong reasoning capabilities, allow us to evaluate both table understanding performance and the effectiveness of providing table relationships in an LLM-interpretable format at inference time.

Implementation details. For effective PK-FK relationship construction in GRIT, the uniqueness threshold τ_{PK} is set to 0.9. The weights for the three FK scoring components—containment, cardinality, and name similarity—are configured as 0.7, 0.2, and 0.1, respectively. The threshold for final FK selection τ_{FK} is set to 0.7. For all closed-source LLMs, the temperature is fixed at 0 to ensure deterministic outputs. We test the open-source LLM on a single Nvidia RTX A6000.

5 Results and Analysis

5.1 Performance Comparison on Table-Column Retrieval

Our approach operates as a preprocessing step prior to feeding data into the LLM. If all tables in the database are provided as input along with their actual cell values, the input length exceeds the LLM's capacity. Thus, the model cannot identify the relevant tables and columns required to answer a given query. To validate the effectiveness of GRIT, we compare two settings: one where only the table headers are supplied, and another where our GRIT-derived schema is additionally provided.

Table 1 demonstrates the effectiveness of our approach, showing consistent improvements across all baseline models, particularly in join tasks that require identifying relevant tables and columns. Notably, models with relatively limited reasoning capabilities, such as Llama3 and Mistral, show substantial performance gains with our method. This improvement can be attributed to these models' inherent difficulties in inferring inter-table relationships, which GRIT mitigates by explicitly pre-processing relational structures into an LLM-compatible format.

5.2 Performance Comparison on Text-to-SQL

We evaluate whether GRIT's structural guidance enhances end-to-end performance beyond table-column retrieval. The text-to-SQL task involves translating a natural language question into an executable SQL query, conditioned on both the question and the schema of a relational database. To accomplish this, the model must identify relevant tables and columns, infer appropriate join paths,

Model	Queries requiring 2 tables		Queries requiring ≥ 3 tables	
	Header	+ GRIT	Header	+ GRIT
ChatGPT 3.5	71.03	72.45 (+ 2.00%)	64.25	76.91 (+ 19.71%)
Claude Haiku	80.76	85.39 (+ 5.74%)	83.49	89.34 (+ 7.01%)
Claude Sonnet	86.39	88.50 (+ 2.44%)	88.95	93.16 (+ 4.73%)
Grok3	84.31	85.58 (+ 1.51%)	84.69	88.70 (+ 4.74%)
LLaMA3	28.09	42.22 (+ 50.31%)	24.79	52.66 (+ 112.43%)
Mistral	18.22	34.72 (+ 90.51%)	24.19	45.93 (+ 89.92%)

Table 3: Comparison of F1 scores in join key table-column retrieval, evaluated across varying levels of query complexity in the Spider dataset.

	Precision	Recall	F1
Primary Key	72.23	**99.17**	82.19
Foreign Key ($\phi_{\text{cont}} + \phi_{\text{name}} + \phi_{\text{card}}$)	95.13	**98.85**	96.81
$-\phi_{\text{name}}$	83.85	83.51	82.86
$-\phi_{\text{card}}$	74.36	93.95	80.51
$-\phi_{\text{name}} - \phi_{\text{card}}$	71.59	87.70	76.71

Table 4: Detection performance of primary and foreign keys by GRIT on the BIRD dataset.

and apply correct filtering conditions. We compare two input configurations: (1) providing only the table schema (i.e., table and column names), and (2) providing the schema along with structural cues inferred by GRIT. We assess SQL execution accuracy on the BIRD dataset using two large language models: GPT-3.5 Turbo and Claude Haiku.

The results, shown in Table 2, demonstrate that GRIT not only improves schema-level understanding but also significantly enhances the models' ability to generate structurally valid and executable SQL queries. These findings support our hypothesis that incorporating relational schema information is crucial for improving LLM performance on complex multi-table reasoning tasks.

5.3 Performance by Number of Join Tables

Table 3 shows how model performance in join column detection varies with the number of tables required to answer a query in the Spider dataset. This number directly reflects the number of join operations involved and serves as a key indicator of query complexity. As the number of required tables increases, the task becomes increasingly challenging due to the need for multi-hop relational reasoning. GRIT mitigates this challenge by explicitly identifying inter-table relationships and providing corresponding join paths, resulting in up to 19.71% performance improvement in the most join-intensive queries.

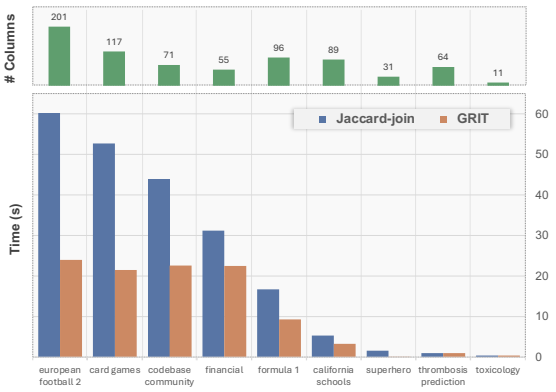

Figure 3: Comparison of PK-FK relationship construction time across databases. The figure consists of two bar charts: the lower chart compares the processing time of Jaccard-join and GRIT, and the upper chart displays the total number of columns aggregated from all tables in each database.

5.4 Accuracy and Efficiency Analysis of GRIT

Key Detection Accuracy

In this section, we analyze the trade-off between GRIT's accuracy and efficiency on the Bird dataset. As a hashing-based method, GRIT is inherently approximate. As shown in Table 4, primary key (PK) detection yields lower precision due to approximate uniqueness estimation via HyperLogLog, yet achieves high recall, indicating minimal omission of true PKs. Since Foreign key (FK) detection builds on these PK candidates, high PK recall is crucial for preserving correctness. FK detection benefits from stricter matching constraints and achieves higher overall scores. Notably, our method maintains high recall for both PKs and FKs, demonstrating its ability to recover most key relationships. To analyze the contribution of each component to FK detection, we perform ablation study by independently removing ϕ_{card} and ϕ_{name}. Removing each component leads to a performance drop. In particular, excluding ϕ_{card} results in a substantial decline in FK precision, highlighting the risk of relying solely on containment. Without cardinality filtering, overlapping values across unrelated columns can lead to over-predicted FK relationships.

Efficient Key Discovery at Scale

We then turn to examining the efficiency of our method. In prior works (Yang et al., 2023; Chen et al., 2024a), primary key candidates in multi-table settings are typically identified by evaluating column uniqueness through exact set operations, while PK–FK relationships are established by computing

Database	Profile			Efficiency (MB)	
	# Rows	# Columns	# Tables	Jaccard-join	GRIT
financial	1,079,680	55	8	3486.36	26.63 (-99.24%)
card games	803,451	117	6	1881.34	13.63 (-99.28%)
codebase community	740,646	71	8	2107.29	9.80 (-99.53%)
formula 1	514,297	96	13	768.31	11.17 (-98.55%)
debit card specializing	423,051	23	5	1260.92	10.00 (-99.21%)
european football 2	222,803	201	7	2613.49	5.75 (-99.78%)
toxicology	49,813	11	4	43.21	0.78 (-98.19%)
california schools	29,941	89	3	232.67	0.76 (-99.67%)
thrombosis prediction	15,952	64	3	101.66	0.52 (-99.49%)
superhero	10,614	31	10	9.03	0.23 (-97.45%)

Table 5: Memory consumption comparison between Jaccard-join and GRIT across databases in the BIRD dataset.

Jaccard similarity between candidate column value sets. We refer to this combination of exact set-based uniqueness evaluation and Jaccard similarity-based relationship construction as the Jaccard-join. Accordingly, we compare the efficiency of GRIT against Jaccard-join.

Time efficiency. Figure 3 illustrates that GRIT achieves significantly higher time efficiency, particularly in complex database environments with a large number of columns. The Jaccard-based approach, which exhaustively visits all possible table-column pairs, exhibits a steep increase in processing time as the number of columns grows, leading to significant inefficiency. In contrast, our method maintains a near-linear relationship between processing time and the number of columns, demonstrating its scalability for large-scale table collections.

Memory efficiency. To assess memory usage, we track the peak memory consumption during PK–FK relationship construction using the tracemalloc function [2]. The results are summarized in Table 5. GRIT consistently achieves over 97% memory savings across diverse database environments. The overall memory complexity of Jaccard-join grows approximately as $O(N^2 \cdot M_{avg})$, where N is the total number of columns and M_{avg} is the average number of rows per column. In contrast, GRIT maintains robust memory efficiency, with usage growing nearly linearly with the total number of columns and sustaining an approximate $O(N + M)$ efficiency, where M denotes the total number of rows across all tables. These results demonstrate that GRIT enables reliable PK–FK relationship discovery, even in memory-constrained environments.

[2]https://docs.python.org/3/library/tracemalloc.html

6 Conclusion

In this paper, we examine the challenges of multi-table reasoning with LLMs, which requires processing interrelated tables under large-scale input and structural constraints. To address these issues, we propose *GRIT*, a lightweight, hashing-based algorithm that efficiently infers schema-level relationships across all tables in a database. GRIT enables the construction of natural language prompts that explicitly encode table structures and inter-table relationships, allowing LLMs to better understand and reason over tabular data. Through extensive experiments, we demonstrate that GRIT consistently improves the performance of multiple LLMs across multiple benchmarks, while significantly reducing both time and memory consumption.

Limitations

Our work has two main limitations.

First, as GRIT relies on hashing-based algorithms—HyperLogLog and Bloom Filter—it inevitably introduces some approximation errors. Although these probabilistic data structures provide significant efficiency gains, they cannot guarantee exact correctness, particularly in corner cases with highly skewed or ambiguous data distributions. To mitigate this limitation, we combine multiple components, including cardinality, containment, and name similarity, which helps reduce error bounds and enhance robustness. Nevertheless, GRIT cannot fully eliminate the inherent risk of approximation errors. Despite this trade-off, GRIT delivers substantial efficiency gains that make it a practical solution for handling large-scale multi-table inputs. Accordingly, an appropriate balance between efficiency and accuracy is necessary.

Second, in multi-table settings, it is common to encounter composite keys, where multiple columns

jointly serve as a primary or foreign key. Although GRIT currently focuses solely on single-column key detection and thus cannot capture relationships involving composite keys, we believe that exploring efficient methods for composite key handling represents a promising direction for future work.

Acknowledgments

This work was partly supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) [RS-2021-II211341, Artificial Intelligence Graduate School Program (Chung-Ang University)] and partly supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00419201). We also would like to thank Chung-Chi Chen for valuable comments.

References

Iñigo Alonso, Eneko Agirre, and Mirella Lapata. 2024. Pixt3: Pixel-based table-to-text generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6721–6736.

Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku.

Burton H Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426.

Peter Baile Chen, Yi Zhang, and Dan Roth. 2024a. Is table retrieval a solved problem? exploring join-aware multi-table retrieval. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2687–2699.

Si-An Chen, Lesly Miculicich, Julian Eisenschlos, Zifeng Wang, Zilong Wang, Yanfei Chen, Yasuhisa Fujii, Hsuan-Tien Lin, Chen-Yu Lee, and Tomas Pfister. 2024b. Tablerag: Million-token table understanding with language models. *Advances in Neural Information Processing Systems*, 37:74899–74921.

Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. Turl: Table understanding through representation learning. *ACM SIGMOD Record*, 51(1):33–40.

Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete mathematics & theoretical computer science*, (Proceedings).

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Mueller, Francesco Piccinno, and Julian Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Mosh Levy, Alon Jacoby, and Yoav Goldberg. 2024. Same task, more tokens: the impact of input length on the reasoning performance of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15339–15353.

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. *Advances in Neural Information Processing Systems*, 36:42330–42357.

Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2024. Table-gpt: Table fine-tuned gpt for diverse table tasks. *Proceedings of the ACM on Management of Data*, 2(3):1–28.

Weizhe Lin, Rexhina Blloshmi, Bill Byrne, Adrià de Gispert, and Gonzalo Iglesias. 2023. An inner table retriever for robust table question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9909–9926.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. 2024. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Md Nahid and Davood Rafiei. 2024. Tabsqlify: Enhancing reasoning capabilities of llms through table decomposition. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5725–5737.

Rungsiman Nararatwong, Chung-Chi Chen, Natthawut Kertkeidkachorn, Hiroya Takamura, and Ryutaro Ichise. 2024. DBQR-QA: A question answering dataset on a hybrid of database querying and reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 15169–15182, Bangkok, Thailand. Association for Computational Linguistics.

Vaishali Pal, Andrew Yates, Evangelos Kanoulas, and Maarten de Rijke. 2023. Multitabqa: Generating tabular answers for multi-table question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6322–6334.

Sohan Patnaik, Heril Changwal, Milan Aggarwal, Sumit Bhatia, Yaman Kumar, and Balaji Krishnamurthy. 2024. Cabinet: Content relevance-based noise reduction for table question answering. In *The Twelfth International Conference on Learning Representations*.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024a. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 645–654.

Yuan Sui, Jiaru Zou, Mengyu Zhou, Xinyi He, Lun Du, Shi Han, and Dongmei Zhang. 2024b. Tap4llm: Table provider on sampling, augmenting, and packing semi-structured data for large language model reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10306–10323.

Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. 2021. Tuta: Tree-based transformers for generally structured table pre-training. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1780–1790.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. 2024. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In *The Twelfth International Conference on Learning Representations*.

John Yang, Akshara Prabhakar, Karthik R Narasimhan, and Shunyu Yao. 2023. Intercode: Standardizing and benchmarking interactive coding with execution feedback. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning. In *Proceedings of the 46th international ACM SIGIR conference on research and development in information retrieval*, pages 174–184.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Siyue Zhang, Luu Anh Tuan, and Chen Zhao. 2024a. Syntqa: Synergistic table-based question answering via mixture of text-to-sql and e2e tqa. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2352–2364.

Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. 2024b. Tablellama: Towards open large generalist models for tables. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6024–6044.

Mingyu Zheng, Xinwei Feng, Qingyi Si, Qiaoqiao She, Zheng Lin, Wenbin Jiang, and Weiping Wang. 2024. Multimodal table understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9102–9124.

A Dataset

A.1 Statistics from Datasets

Table 6 summarizes row, column, and table statistics for the Bird (Li et al., 2023) and Spider (Yu et al., 2018) datasets. Average, maximum, and minimum values are computed over individual databases within each dataset. Notably, Bird and Spider contain large-scale databases with up to 1M and 531K rows, respectively. When issuing a query to a database, users often do not know in advance which tables and columns are required to answer the query. As a result, the entire database is typically provided as input, leading to the inclusion of all tables in the input sequence. Figure 4 presents the total input size of each database when fully tokenized by an LLM under this scenario. We observe that databases far exceed even the largest context windows. This not only leads to context truncation and critical information loss but also exacerbates

	Bird			Spider		
	Row	Column	Table	Row	Column	Table
Avg	357,524	73	7	26,993	22	4
Max	1,079,680	201	14	531,377	56	11
Min	10,614	11	3	12	7	2

Table 6: Statistics from the Bird and Spider. Avg, Max, and Min are computed across individual databases within each dataset.

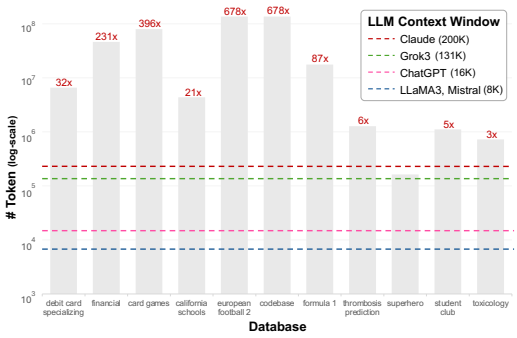

Figure 4: Token length comparison between multi-table databases in Bird dataset and LLM context limits. Due to the large scale of databases, we apply a log scale to ensure clearer comparison. Text above each bar shows how much the token length exceeds Claude context window.

the challenge for LLMs to understand schema relationships.

A.2 Prompt for Extracting Table-Column Pairs from SQL Queries

We design the prompt to obtain the golden table-column pairs required to execute a given SQL query. As shown in Figure 5, the prompt guides the model to identify all necessary table-column pairs from the query and categorize them into two distinct groups: columns used exclusively for table joins, and columns used for answering question.

B Prompt for Schema-Aware Table-Column Retrieval

We design the prompt to identify the necessary tables and columns required to answer a given natural language question using a provided database schema. The overall structure consists of two key steps. First, it determines the minimal join path by selecting only the essential tables and join key columns needed to connect them. Next, it extracts the question-relevant columns, which are directly involved in filtering the data or formulating the answer. Figure 6 illustrates the overall prompt.

C Algorithm for GRIT

Algorithm 1 illustrates the GRIT process for efficiently extracting structural relationships between tables to provide LLMs with explicit schema-level information. The method consists of two main steps for automatically detecting Primary Key (PK) and Foreign Key (FK) relationships within a database. The first step (lines 3–10) involves identifying candidate primary keys by estimating the cardinality γ of each column using the HyperLogLog (Flajolet et al., 2007) algorithm. The second step (lines 11-27) identifies foreign key candidates based on the extracted primary keys $\mathcal{K}_{\text{PK}}$. For each potential FK–PK pair, a matching score S is computed based on three components: containment, cardinality ratio, and name similarity. This algorithm enables explicit extraction of structural dependencies, which are later transformed into prompt formats that are more interpretable to LLMs for downstream tasks.

Instruction

```
You are given a SQL query.
Your task is to identify all `table.column` pairs necessary to execute the query.

You must:
1. Analyze step by step which tables and columns are involved.
2. Identify all columns used *only* for joining tables across the minimum set of tables required.
3. Identify all columns that are used to filter, compute, or return results.

Return exactly two comma-separated lists with the following labels:
- The first list must include columns used *only* for joins. If none, write `None`.
- The second list must include all other columns used in the query, including for filtering, computation,
aggregation, or output.

Use this format:
### Join: table.column, table.column, ...
### Question: table.column, table.column, ...

Do not include explanations or reasoning.
Let's go through some examples together.
```

Example 1

```
Query:
SELECT CAST(SUM(IIF(Currency = 'EUR', 1, 0)) AS FLOAT) / SUM(IIF(Currency = 'CZK', 1, 0)) AS ratio FROM
customers
Answer:
### Join: None
### Question: customers.Currency
```

Example 2

```
Query:
SELECT T1.CustomerID FROM customers AS T1 INNER JOIN yearmonth AS T2 ON T1.CustomerID = T2.CustomerID WHERE
T1.Segment = 'LAM' AND SUBSTR(T2.Date, 1, 4) = '2012' GROUP BY T1.CustomerID ORDER BY SUM(T2.Consumption) ASC
LIMIT 1
Answer:
### Join: customers.CustomerID, yearmonth.CustomerID
### Question: customers.Segment, yearmonth.Date, yearmonth.Consumption
```

Figure 5: Prompt design for extracting and categorizing relevant table-column pairs from SQL queries.

Instruction

```
You are given a database schema and a natural language question.
Your task is to identify all table.column pairs necessary to answer the question.

You must:
1.   Think step by step about which tables and columns are relevant to the question.
2.   Identify all columns used to perform joins only across the minimum set of tables required to answer the question.
3.   Identify all columns that are directly used to filter or answer the question.
4.   Match each column to the correct table name exactly as shown in the schema.
5.   When multiple tables contain similar values, prioritize values from tables with a primary key.

In your final output, return two comma-separated lists with clear labels:
-   The first list should contain only the columns used for joining tables.
-   The second list should contain only the columns used to filter or answer the question.

Format:

Join key columns: table.column, table.column, ...
Question-relevant columns: table.column, table.column, ...

Only output these two lines. Do not explain your reasoning.
```

Example 1

```
---
SCHEMA:
[RELATIONSHIPS]
students.school_id = schools.school_id

TABLE students {
student_id # Unique
name
school_id # schools.school_id
grade_level
}
TABLE schools {
school_id # Unique
school_name
district_code
city
}

QUESTION: What is the district code of the school where the student named Emily is enrolled?

ANSWER:
Join key columns: students.school_id, schools.school_id
Question-relevant columns: students.name, schools.district_code
```

Example 2

```
SCHEMA:
[RELATIONSHIPS]
appointments.patient_id = patients.patient_id,
appointments.doctor_id = doctors.doctor_id,
patients.doctor_id = doctors.doctor_id

TABLE patients {
patient_id # Unique
name
doctor_id # doctors.doctor_id
}
TABLE doctors {
doctor_id # Unique
name
department
}
TABLE appointments {
appointment_id # Unique
patient_id # patients.patient_id
doctor_id # doctors.doctor_id
appointment_date
purpose
}

QUESTION: What is the department of the doctor who had an appointment with patient ID 101 on 2024-04-02?

ANSWER:
Join key columns: appointments.patient_id, patients.patient_id, appointments.doctor_id, doctors.doctor_id
Question-relevant columns: appointments.patient_id, appointments.appointment_date, doctors.department
```

Example 3

```
SCHEMA:
[RELATIONSHIPS]
students.major_id = majors.major_id,
students.advisor_id = professors.professor_id

TABLE students {
student_id # Unique
name
major_id # majors.major_id
advisor_id # professors.professor_id
}
TABLE majors {
major_id # Unique
major_name
department
}
TABLE professors {
professor_id # Unique
name
office_location
}

QUESTION: What is the office location of Professor Kim?

ANSWER:
Join key columns: None
Question-relevant columns: professors.name, professors.office_location
```

Figure 6: Prompt design for table and column retrieval from questions. The prompt first identifies the minimal set of tables and join keys required to connect them, then extracts the columns necessary for answering the question.