TACKLE: QUESTION ANSWERING OVER PRACTICAL ENTERPRISE TABLES VIA CONCEPTUAL KNOWLEDGE GRAPHS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tabular data is a primary source of enterprise knowledge, yet it is typically scattered across independently maintained tables. While large language models enable natural-language question answering over tables, existing approaches—such as formal query-based and retrieval-based methods—rely heavily on explicit relational schemas, well-aligned and clear column names, and normalized layouts. These assumptions, however, often do not hold for practical enterprise tables, making enterprise reasoning extremely challenging. To bridge this gap, we propose *TACKLE*, a framework that operates independently of any predefined schema or ontology. TACKLE directly induces a conceptual knowledge graph from practical tables by clustering related columns into enterprise concepts and establishing explicit semantic relations. Building upon this graph-based representation, TACKLE employs a decoupled NL-to-Cypher translation pipeline that separates high-level semantic planning from low-level syntactic rendering to improve the query reliability. We further introduce ERP-QA, a controllable and extensible benchmark generation framework that reproduces these practical table characteristics and provides deterministically verified answers. Experiments on an evaluation set instantiated from ERP-QA and a practical variant of MMQA show that TACKLE consistently outperforms representative formal query-based and retrieval-based baselines across multiple LLM backbones. Ablation studies further support the benefits of the conceptual KG and decoupled translation for practical multi-table QA. Our source code is available at `https://anonymous.4open.science/r/TACKLE-4798/README.md`.

1 INTRODUCTION

Tabular data serves as a primary source of enterprise knowledge, yet it is typically scattered across heterogeneous tables maintained by independent business units Hermans & Murphy-Hill (2015); Halevy et al. (2016). Traditionally, extracting insights from such tables requires data engineers to formulate database queries, a labor-intensive and inaccessible process for non-technical users. Large language models (LLMs) now enable natural-language (NL) question answering (QA) over tables, allowing users to express information needs directly as questions. Consequently, NL QA over multiple tables has become an increasingly important form of enterprise reasoning Yue (2025).

Existing approaches to NL table QA largely follow two directions. (1) **Formal query-based methods**, such as Text2SQL, translate questions into executable queries and derive answers by executing them. This translation depends on two kinds of information — how tables relate to each other (determining joins) and how question terms correspond to specific columns and values (determining selection, filtering etc.). In a well-structured database, both are readily available from the relational schema: foreign keys specify table relations, and column names and types indicate column semantics. A substantial line of work has studied how to leverage such schema [1] effectively (Liu et al., 2025b). Schema linking (Safdarian et al., 2026; Xu et al., 2026), for instance, identifies the relevant tables and columns for a given question. These methods achieve strong results when the schema is explicit and well-aligned. (2) **Retrieval-based methods**, such as table-oriented RAG (Ho et al.,

[1] We use *schema* to refer to the relational schema; *schema-free* means no such schema is explicitly provided.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

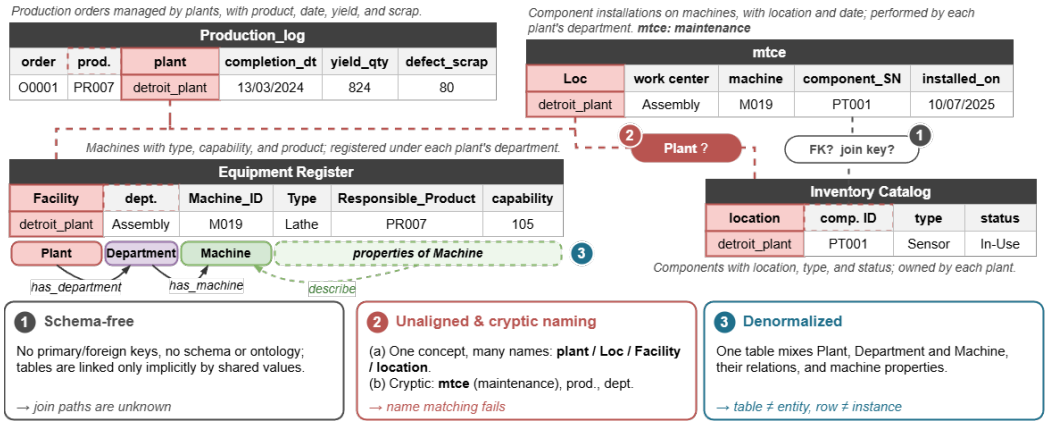


Figure 1: We use *practical enterprise tables* to refer to tables with common characteristics in real enterprise settings: being schema-free, semantically unaligned and cryptic, and denormalized in layout.

2026), instead retrieve relevant table content and provide it to an LLM to generate the answer. They rely on the retrieved content being both semantically relevant to the question and forming a coherent evidence unit (Dantart & Kóvacs-Navarro, 2026; Sonowal & Sadhu, 2025). Graph-based variants further construct a graph from the schema—typically tables as nodes and foreign keys as edges—to enable multi-hop retrieval (Ho et al., 2026; Zhou et al., 2024). Without foreign keys, however, the graph cannot be constructed, and retrieval reduces to surface-level similarity with no explicit mechanism to connect information across tables. These methods achieve strong results when the schema is explicit and table content is organized into well-defined conceptual units.

In practice, however, enterprise tables rarely satisfy these conditions. They are often created and maintained outside relational database systems, typically by domain experts using spreadsheet tools (Halevy et al., 2016; Nargesian et al., 2019), and consequently exhibit several characteristics that pose significant challenges to automatic NL table QA. (Fig. 1). First, they are often *schema-free*, with no explicit primary- and foreign-key specifications (Kandel et al., 2012; Jiang & Naumann, 2020), and are rarely accompanied by a predefined ontology describing their domain concepts and relations (Sequeda et al., 2024). Second, their textual semantics can be *cryptic or unaligned* (Chen et al., 2024a): column names may be inconsistent across tables, table names may be absent or uninformative. Third, their layout is *denormalized* (Papenbrock & Naumann, 2017; Li et al., 2023d): a single table may represent multiple concepts, properties of the same concept may be spread across different tables, and relations with their properties may be implicitly encoded by combinations of columns rather than explicit junction tables. These characteristics jointly undermine the assumptions on which existing methods rely. For formal query-based methods, the absence of a reliable schema means that the join paths and column correspondences required for query generation cannot be determined (Chen et al., 2024a; Lin et al., 2023). For retrieval-based methods, the lack of explicit connections between tables means that multi-hop relationships must be inferred from surface-level content alone (Chen et al., 2024b), while denormalized layouts cause the retrieved content to mix multiple concepts within a single unit. Both directions, therefore, struggle precisely in the settings where enterprise reasoning is most needed.

To address this gap, we propose *TACKLE* (TAble QA via Conceptual KnowLEdge graphs), a two-stage framework that induces and leverages the latent conceptual structure of practical enterprise tables for NL QA, which can operate independently of any predefined schema, ontology, or external knowledge base. Our approach is guided by the assumption that the conceptual content of tables, i.e., which entities are described and what properties they have, and how the entities connect with each other, can support NL table QA without first recovering a relational schema. Rather than attempting to restore the schema that the practical enterprise tables lack, we directly induce the concepts they express and represent them as a conceptual knowledge graph (KG), where nodes correspond to business concepts and edges to semantic relations. Concretely, we analyze the semantics of each table and column, cluster related columns into concepts, merge corresponding concepts across tables, and induce semantic relations between them. This graph-based view offers two structural advantages. First, the relations between concepts are represented as explicit edges, so a multi-hop question can be answered by traversing a defined path rather than by inferring how dispersed content should be connected. Second, it represents conceptual relations explicitly at the schema level, so that seman-

tic connections between tables are established when the graph is built rather than inferred at query time. This representation is more tolerant of noisy or unaligned names, since concept induction only requires identifying which columns plausibly describe the same concept, rather than establishing exact column-level mappings. Recovering such relations is well-suited to LLMs, which prior works have shown can infer column semantics from table context even when names are abbreviated or unaligned (Zhang et al., 2023; Cai et al., 2025). Building on this representation, we further develop a decoupled pipeline for NL-to-Cypher translation. It separates a semantic stage, where an LLM decomposes the question and projects it onto the KG schema, from a syntactic stage that renders the query. This separation allows each stage to focus on a distinct subtask and improves reliability compared to end-to-end generation.

To evaluate our approach in a practical enterprise setting, we introduce ERP-QA, a controllable and extensible benchmark generation framework that produces multi-table QA datasets reflecting the characteristics of practical enterprise tables across diverse reasoning types. On an evaluation set instantiated from ERP-QA, TACKLE consistently outperforms representative formal query-based and retrieval-based approaches. We further evaluate on a practical variant of MMQA Wu et al. (2025), where explicit schema is withheld, and table/column names are made unaligned, while the data and questions remain unchanged. Here, TACKLE also achieves strong improvements over existing baselines. Ablation studies confirm that both conceptual KG and the decoupled NL-to-Cypher pipeline contribute to the overall performance. These results support our central hypothesis — explicitly inducing conceptual KG benefits practical multi-table reasoning, and separating semantic planning from syntactic rendering improves query reliability.

Our contributions are fourfold:

- **A practical problem setting.** We formulate multi-table QA over *practical enterprise tables* characterized by being schema-free, semantically unaligned, cryptic, and denormalized. This setting is ubiquitous in real enterprises but has been largely overlooked as an independent problem.
- **Conceptual knowledge graph induction from practical tables.** We develop an agentic pipeline to construct a conceptual KG from *practical enterprise tables* without a predefined ontology or explicit relational schema, and show that this representation serves as an effective intermediate representation for downstream multi-table reasoning.
- **Decoupled NL-to-Cypher translation.** We develop a decoupled translation pipeline that separates semantic planning from syntactic rendering. Our experiments demonstrate that this decomposition improves the reliability of graph query generation over direct LLM-based translation.
- **A benchmark generation framework for practical enterprise tables.** We introduce ERP-QA, a controllable and extensible framework for generating multi-table QA benchmarks from configurable meta-level specifications grounded in realistic enterprise scenarios. It systematically reproduces the characteristics of practical enterprise tables while providing deterministically verified answers across diverse reasoning types.

2 Related Work

Our work is related to three lines of research: formal query-based methods, retrieval-based methods, and table-to-KG or KGQA approaches. The first two represent the dominant paradigms for NL table QA and serve as our baselines. The third appears related to our use of a KG but differs fundamentally in problem setting and construction. We review each in turn and clarify how our work differs.

2.1 Formal Query-Based Methods

Formal query-based methods translate NL questions into executable programs (mostly SQL, but can also be python TableGPT Team (2024)) over a given database schema (Yu et al., 2018; Li et al., 2023c). Recent LLM-based systems improve this paradigm through decomposition, iterative reasoning, verification, and increasingly sophisticated schema linking (Pourreza & Rafiei, 2023; Gao et al., 2023; Liu et al., 2025a; Wang et al., 2025; Nahid et al., 2026; Safdarian et al., 2026). While these methods achieve impressive performance on academic benchmarks such as Spider and BIRD (Yu et al., 2018; Li et al., 2023b), they fundamentally rely on the availability of explicit relational schemas. This assumption rarely holds in real-world enterprise settings, leading to severe performance degradation. Illustrating this gap, Spider 2.0 reports that a model scoring 91.2% on

standard benchmarks plummets to 21.3% on realistic enterprise schemas, with failures largely stemming from missing join relations and incomplete semantic layers (Lei et al., 2025).

2.2 Retrieval-Based Methods

Retrieval-based methods retrieve relevant table content and let an LLM reason over the retrieved evidence (Lewis et al., 2020). Recent work improves table retrieval through structured serialization and table-aware representations (Chen et al., 2024c; Sui et al., 2024; Si et al., 2025), while downstream reasoners further decompose questions or execute structured operations over the retrieved content (Ye et al., 2023; Wang et al., 2024; Cheng et al., 2023).

Multi-table QA requires retrieving not only directly relevant tables but also intermediate tables needed to connect dispersed evidence (Pal et al., 2023; Wu et al., 2025). This has motivated join-aware retrieval, multi-step retrieval, graph-structured indices, and pre-computed relational chains (Chen et al., 2024b; Boutaleb et al., 2025; Trivedi et al., 2023; Zou et al., 2026; Ho et al., 2026). However, these approaches typically infer cross-table connectivity from signals such as aligned column names, declared keys, or overlapping values. Such signals are unreliable or unavailable in the schema-free and semantically unaligned tables considered in our setting.

2.3 Table-to-KG and KGQA

Mapping tables to KGs has been studied as semantic table interpretation (STI), which annotates cells with entities, columns with types, and column pairs with properties of a reference KG (Liu et al., 2023; Jiménez-Ruiz et al., 2020). Learned column annotators (Hulsebos et al., 2019; Deng et al., 2021; Suhara et al., 2022) and, more recently, LLM-based annotators (Korini & Bizer, 2023) have steadily improved these tasks, but all of them match tables *to an existing vocabulary* such as DBpedia or Wikidata, which is a luxury rarely available in enterprise settings.

Ontology-based data access likewise exposes relational databases as a KG, through mappings that experts write against a given ontology (Poggi et al., 2008; Calvanese et al., 2017). Notably, such a hand-built ontology layer substantially improves LLM accuracy on enterprise SQL questions (Sequeda et al., 2024), underscoring the value of an explicit conceptual layer. While such a layer is usually absent in practical enterprise tables, its demonstrated benefit also suggests that automatically inducing one is a promising direction.

On the querying side, KGQA follows two strategies. Semantic parsing methods translate questions into executable logical forms or graph queries (Yih et al., 2016; Gu et al., 2021; Ye et al., 2022; Gu et al., 2023; Li et al., 2023e; Luo et al., 2024a), while retrieve-and-reason methods retrieve paths or subgraphs and let a model reason over them (Jiang et al., 2023; Luo et al., 2024b; Sun et al., 2024; He et al., 2024; Li et al., 2025; Mavromatis & Karypis, 2025). Text-to-Cypher has recently received dedicated datasets and benchmarks (Ozsoy et al., 2025; Feng et al., 2025), which show that even strong LLMs remain unreliable at generating graph queries. Our decoupled translation adapts to Cypher an observation from Text2SQL, that separating schema grounding from query construction improves reliability (Li et al., 2023a; Pourreza & Rafiei, 2023).

3 Methodology: TACKLE

This section presents the TACKLE framework (Fig. 2), which first induces the conceptual KG from practical enterprise tables (Sec. 3.1), and then translates NL questions into executable graph queries over the resulting KG through a decoupled pipeline (Sec. 3.2).

3.1 Conceptual KG Induction

Let $\mathcal{T} = \{T_1, \ldots, T_m\}$ denote a collection of practical enterprise tables. The first stage of TACKLE aims to transform $\mathcal{T}$ into a conceptual KG $G = (\mathcal{S}, \mathcal{I})$, where $\mathcal{S} = (\mathcal{E}, \mathcal{R})$ denotes the induced schema of enterprise concepts/entities $\mathcal{E}$ and semantic relations $\mathcal{R}$, and $\mathcal{I}$ denotes the corresponding instance-level data. Both entities and relations may carry properties. Each conceptual property is explicitly grounded in one or more source columns, allowing the instance-level graph to be deterministically populated from the original table values.

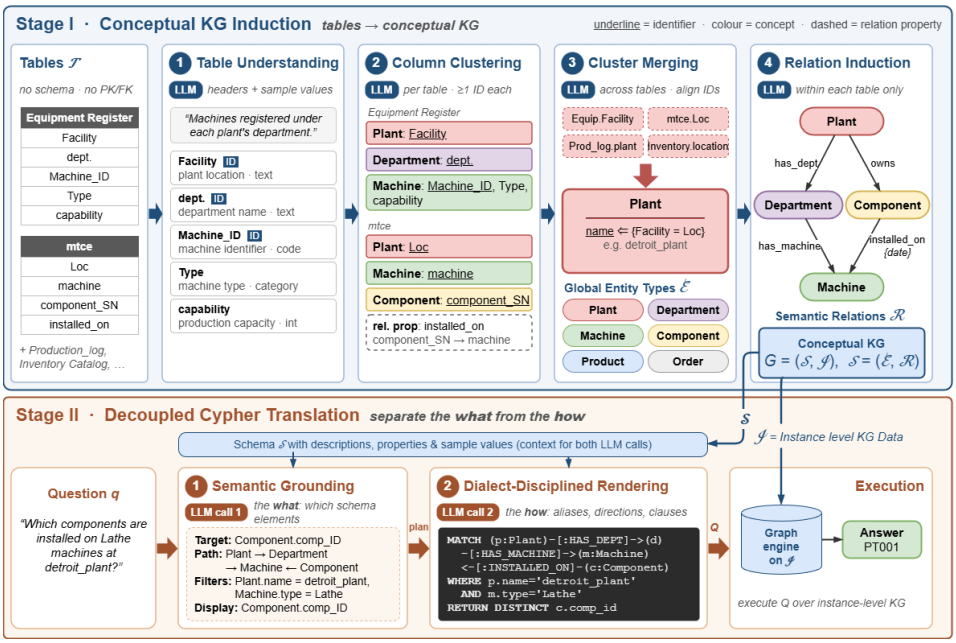


Figure 2: Overview of TACKLE. **Stage I** induces a conceptual KG from practical enterprise tables via *table understanding*, per-table *column clustering*, cross-table *cluster merging*, and intra-table *relation induction*. **Stage II** translates a question into Cypher through two decoupled LLM calls: semantic grounding (*what*) and dialect-disciplined rendering (*how*).

Our induction procedure consists of four LLM-based steps: *Table Understanding*, *Column Clustering*, *Cluster Merging*, and *Relation Induction*. Together, these steps induce the conceptual structure expressed across heterogeneous and denormalized tables and transform it into an explicit KG for downstream question answering. The prompts for all steps are provided in Apx. A.1.

Table Understanding. Given all table names and column headers, together with up to five representative values per column, an LLM generates a semantic description for each table and column, and predicts each column's value format and whether it may serve as an identifier of an entity.

Column Clustering. We independently cluster the columns of each table into table-level concepts using the semantic annotations from *Table Understanding*, rather than relying solely on surface names. Each cluster groups columns describing the same underlying concept and must contain at least one local identifier, possibly composite, for instance tracking and subsequent cross-table alignment; these identifiers are not assumed to be global primary keys.

Columns describing relation properties are excluded from entity clusters. For each such column, the LLM identifies source and target columns that determine the relation endpoints, allowing relation attributes encoded directly in denormalized tables to be represented explicitly.

Cluster Merging. TACKLE merges clusters that represent the same underlying concept across tables into unified entity types. The LLM compares their semantic descriptions, properties, and local identifiers; overlapping properties are consolidated, while complementary ones are retained.

Because corresponding concepts may use different identifiers across tables, TACKLE also aligns local referencing schemes, including numeric IDs, descriptive attributes, or composite identifiers. The resulting merged entities form the conceptual KG schema $\mathcal{S}$.

Relation Induction. TACKLE induces semantic relations $\mathcal{R}$ among the merged entity types by analyzing each table independently. Relations are restricted to entity pairs that co-occur within the same table, avoiding spurious cross-table edges. When relation-property columns were identified during *Column Clustering*, their source and target endpoints are used to determine directed relations, and the corresponding columns are attached as relation properties $\mathcal{P}_r$.

The LLM additionally generates an NL description for each relation, yielding the conceptual schema $\mathcal{S} = (\mathcal{E}, \mathcal{R})$. Given this schema and the established column groundings, the instance-level graph $\mathcal{I}$ is then populated deterministically from the raw tables.

3.2 Decoupled Cypher Translation

Given question q and the recovered conceptual KG schema $\mathcal{S}$, this stage aims to reliably translate q into an executable graph query Q. We adopt Cypher for its native support of graph structures and relation properties, which naturally fit enterprise table associations. Translating natural language into such graph queries presents a dual challenge: the model must accurately resolve complex schema semantics, such as identifying the correct entities, multi-hop join paths, and projection properties (the *what*), while strictly adhering to graph engine syntax, strict projection rules, and dialect constraints (the *how*). Addressing these heterogeneous requirements within a single, monolithic LLM call can easily overload the model, making end-to-end translation sub-optimal.

Grounded in a "What-vs-How" decomposition philosophy, we hypothesize that splitting the task into specialized stages yields higher reliability. We therefore propose a decoupled, two-call LLM-first pipeline that explicitly decouples the *what: semantic grounding* from the *how: syntactic rendering*:

- **Phase 1: Semantic Grounding (The "What").** Given the question and schema, this stage resolves all semantic and structural choices, specifically determining target answer entities, traversal paths, property filters, and display properties (i.e., all "which schema element" decisions). This is framed as an advisory semantic plan using flexible schema-grounded text, granting the LLM the freedom required for complex logical reasoning without syntax-induced constraints.
- **Phase 2: Syntactic Rendering (The "How").** Taking the advisory grounding plan and schema as context, this stage directly renders the executable graph query. It focuses exclusively on mechanical and syntactic execution: variable aliasing, edge directions, proper clause sequencing (`WHERE`/`WITH`/`ORDER BY`/`RETURN`), and engine-specific dialect rules.

By decoupling semantic intent discovery from syntactic code rendering, this architecture effectively mitigates projection drift and dialect violations while preserving the model's core language understanding capabilities.

Execution and Answer Retrieval. Finally, the generated Cypher query Q is executed against the instance-level KG $\mathcal{I}$ to yield the final answer.

4 ERP-QA: A Controllable Benchmark Generation Framework

We introduce ERP-QA, a controllable and extensible framework for generating multi-table QA benchmarks reflecting practical enterprise conditions, rather than a fixed dataset. It aims to reproduce enterprise-table characteristics, cover diverse reasoning types with deterministically verified answers, and vary instance scale without changing the underlying specifications.

The framework consists of three meta-level specifications (Fig. 3): KG schemas, table schemas, and question templates, which jointly govern the generation of concrete benchmark instances. The KG schemas currently cover five realistic enterprise scenarios, including manufacturing production, university administration, and retail operations, comprising 31 entity types, 42 relation types, 135 entity properties, and 15 relation properties. They can be further extended to new scenarios. The table schemas reproduce the characteristics of practical enterprise tables: column names are abbreviated and unaligned with the KG, a single table may mix multiple entity types, and every column maps to exactly one element of the KG schema.

5 Experiments

This section mainly focus on the two hypothesis, corresponding to our two contributions:

- **H1:** Explicitly induced conceptual KGs significantly facilitate NL QA in practical enterprise tables.
- **H2:** The decoupled translation improves query generation reliability over conceptual KGs.

Benchmark Datasets. We evaluate TACKLE on the following two datasets (detailed dataset statistics are in Apx. B):

- **ERP-QA Evaluation Set** is instantiated using the ERP-QA framework described in Sec. 4 and illustrated in Fig. 3. For evaluation, we instantiate two of the five available scenario specifications—manufacturing production (Fig. 1) and university administration—yielding 9 tables and 334 QA pairs. Hereafter, ERP-QA refers to this evaluation set unless otherwise specified.

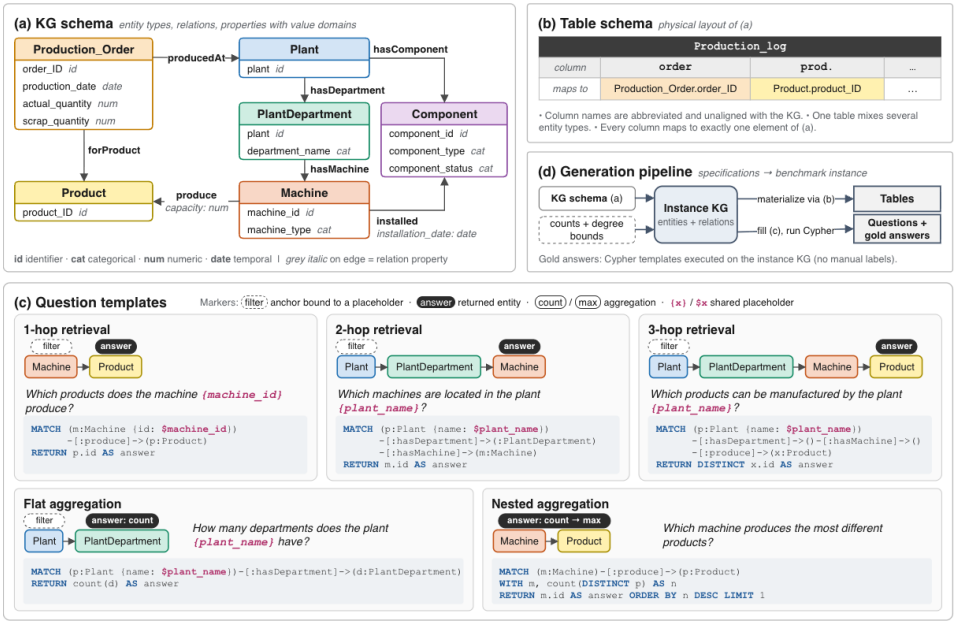


Figure 3: Overview of the ERP-QA benchmark generation framework, illustrated with an example. A KG schema (a) defines the concepts and relations of a scenario; a table schema (b) specifies their physical representation in practical enterprise tables; question templates (c) define NL questions and parameterized Cypher queries across five reasoning types; and the generation pipeline (d) instantiates these specifications into tables and QA pairs with deterministically verified answers.

- **Practical MMQA**: a variant of MMQA (Wu et al., 2025) adapted to the practical enterprise setting. The original MMQA provides explicit relational schema information and uses well-aligned table and column names. To simulate practical enterprise settings, we discard schema information and use an LLM to rewrite table and column names into heterogeneous, abbreviated, or inconsistent formats without cross-table alignment (see prompts in Apx. A.3), while keeping the underlying data and QA pairs intact. This isolates the effect of name unalignment and the absence of schema, penalizing methods that depend on clean naming or explicit schema, and measuring the stability of content-driven structural recovery. The adapted dataset and its generation scripts are publicly available in our anonymous code repository to facilitate reproducibility.

Ablation Study. We evaluate **TACKLE-naive**, an ablated variant that substitutes the decoupled translation with direct Cypher generation.

Baseline Methods. We compare TACKLE and TACKLE-naive with representative baselines from the two dominant paradigms discussed in Sec. 2: formal query-based methods (Text2SQL and TableGPT2-7B) and retrieval-based methods (Row-based RAG and TabRAG).

- **Text2SQL** directly translates each NL question into SQL using table names, column names, and sample values. To isolate the effect of the underlying representation rather than additional agentic machinery, we use a lightweight implementation without advanced components such as schema linking, decomposition, or iterative refinement. These components are orthogonal to our comparison and could likewise be incorporated into TACKLE; our comparison therefore focuses on querying physical tables versus an induced conceptual KG.
- **TableGPT2-7B** (Su et al., 2024) is a table-specialized model whose released agent framework generates executable Python programs (TableGPT Team, 2024). We use the 7B checkpoint with greedy decoding and a 16K context window.
- **Row-based RAG** retrieves the top-50 rows by embedding similarity using `text-embedding-3-small` and provides them to the LLM for answer generation.
- **TabRAG** (Si et al., 2025) uses structured table representations for retrieval and answer generation, rather than retrieving over flat table linearizations.

QA Metrics. We report exact match (EM) between the predicted answer and the ground truth. When the answer is a list, we compare the sets of elements and ignore ordering.

Table 1: Practical-MMQA results (mean ± std. in % over three runs). **Bold**/underline: best/second-best within each backbone. Parentheses show relative gains over the strongest baseline with the same backbone.

LLM-Backbone	Methods	Practical-MMQA2T	Practical-MMQA3T
—	TableGPT2-7B	55.93 ± 0.35	49.33 ± 1.39
GPT-5.4-mini	Text2SQL	53.46 ± 9.93	48.20 ± 9.03
	Row-Based RAG	29.21 ± 7.81	33.26 ± 7.85
	TabRAG	22.54 ± 6.08	20.54 ± 7.63
	TACKLE-naive	67.81 ±5.38 (+26.8%)	68.36 ±4.49 (+41.8%)
	TACKLE	**70.55** ± 5.13 (+31.9%)	**73.17** ± 5.10 (+51.8%)
DeepSeek-v4-flash	Text2SQL	58.36 ± 5.42	60.00 ± 6.18
	Row-Based RAG	45.40 ± 3.61	54.11 ± 10.70
	TabRAG	47.21 ± 6.61	64.29 ± 7.92
	TACKLE-naive	71.32 ±5.19 (+22.2%)	68.92 ± 7.12 (+7.2%)
	TACKLE	**73.75** ± 8.57 (+26.4%)	**75.00** ± 8.28 (+16.7%)
Gemini-3.6-flash	Text2SQL	67.75 ± 7.75	76.85 ± 6.59
	Row-Based RAG	55.89 ± 9.11	65.73 ± 5.73
	TabRAG	54.93 ± 8.18	64.81 ± 8.93
	TACKLE-naive	72.09 ± 3.40 (+6.4%)	87.04 ± 4.43 (+13.3%)
	TACKLE	**74.25** ± 2.95 (+9.6%)	**89.81** ± 5.34 (+16.9%)

5.1 Evaluation on an Established Benchmark (Practical MMQA).

Evaluation Protocol. API-based methods are evaluated on fixed subsets (three 100-QA samples for MMQA2T and three 500-QA samples for MMQA3T); the local TableGPT2-7B is on the full set.

As shown in Tab. 1, TACKLE consistently outperforms all baselines across both Practical-MMQA variants and LLMs, with TACKLE-naive consistently second-best. Retrieval-based methods remain substantially weaker, suggesting that retrieval alone struggles to recover the cross-table conceptual connections required when both explicit schema and aligned names are absent. TableGPT2-7B, though specifically trained for table reasoning, achieves lower results than TACKLE and Text2SQL, consistent with the findings reported in the original MMQA paper Wu et al. (2025).

5.2 Evaluation on Practical Enterprise Tables (ERP-QA).

QA Evaluation. All methods are evaluated on the full ERP-QA set of 334 questions over three runs.

As shown in Tab. 2, TACKLE achieves the best overall accuracy for all three LLM backbones and ranks first in 16 of 18 backbone–reasoning-type settings. The two exceptions occur with gpt-5.4-mini on 1-hop retrieval, where Row-based RAG performs best, and flat aggregation, where TACKLE-naive slightly outperforms TACKLE. As reasoning depth increases, TACKLE remains substantially more robust than the retrieval-based baselines, showing particularly strong performance on 2- and 3-hop questions. TACKLE also consistently achieves the best results on nested aggregation across all three backbones.

Conceptual KG Quality. Beyond end-to-end QA performance, we directly evaluate whether Stage I recovers the underlying conceptual KG from the input tables. ERP-QA retains both the gold *KG Schema* for benchmark generation and the *Table Schema* that maps source columns to KG elements (Fig. 3), enabling strict evaluation without relying on generated concept names. We represent each entity by its set of source columns and count an exact match only when an induced entity contains the identical set. Matching is dependency-aware: *Entities* are matched first; *Entity Properties* and *Relations* are then evaluated with respect to matched entities, followed by *Relation Properties* for matched relations. Specifically, a property matches when the same source column is assigned to the same matched entity, a relation when it connects the same matched endpoint entities, and a relation property when the same source column is attached to the matched relation. We report standard precision, recall, and F1 for each component type averaged on 3 runs. Importantly, these scores measure strict fidelity to ERP-QA's generating schema rather than absolute semantic correctness; alternative but functionally equivalent conceptualizations may therefore be counted as mismatches.

Across all three backbones, the induced KGs show substantial structural agreement with the generating schema under this strict criterion (Tab. 3). This provides direct evidence that Stage I induces meaningful conceptual KG beyond what can be inferred from end-to-end QA performance alone.

Table 2: Results on instantiated ERP-QA. 1–3-hop: 1–3 hop retrieval; Agg.: aggregation. In all cases, we run three times on full sets (334 QAs) with the mean and standard deviation (in a smaller font size) reported. Values are in %; **bold**: best; underline: second-best within each API backbone.

LLM	Methods	Total	1-hop	2-hop	3-hop	Flat Agg.	Nested Agg.
—	TableGPT2-7B	53.9 1.7	85.7 4.8	48.5 2.6	25.5 3.4	50.9 3.0	53.3 2.3
GPT 5.4-mini	Text2SQL	44.6 0.7	59.2 0.7	56.0 1.3	27.5 1.0	52.5 0.2	26.2 0.6
	Row-Based RAG	41.4 1.2	**77.1** 1.7	40.0 0.5	35.4 0.7	26.8 1.4	21.8 1.9
	TabRAG	34.3 2.3	76.2 0.4	31.3 4.5	22.7 6.7	28.0 0.8	12.3 5.0
	TACKLE-naive	61.7 0.6	69.0 1.2	56.0 1.2	62.8 1.5	**77.1** 0.6	41.5 0.6
	TACKLE	**72.8** 0.8	76.1 1.1	**76.0** 0.8	**68.6** 1.0	75.4 1.2	**64.6** 0.9
DeepSeek v4-flash	Text2SQL	54.2 0.5	73.5 0.7	62.5 0.9	45.5 0.7	62.3 0.4	33.3 0.7
	Row-Based RAG	48.0 0.6	75.3 1.7	53.6 0.7	31.2 2.4	58.5 1.1	26.1 0.4
	TabRAG	47.4 1.5	75.8 0.3	41.2 0.5	36.4 0.5	46.7 2.3	36.4 0.6
	TACKLE-naive	74.5 0.6	86.8 0.7	75.0 0.9	70.5 0.8	89.0 0.9	54.6 1.4
	TACKLE	**80.4** 0.6	**89.0** 0.5	**79.5** 1.1	**73.1** 1.3	**90.3** 1.0	**67.2** 0.7
Gemini 3.6-flash	Text2SQL	58.9 0.5	85.7 1.1	63.7 1.0	44.4 0.5	66.7 1.0	48.5 1.2
	Row-Based RAG	52.1 0.6	79.8 0.8	60.8 1.6	42.0 1.4	62.8 1.3	30.3 0.8
	TabRAG	50.0 1.0	76.2 0.3	49.1 3.0	41.2 0.4	50.9 3.0	34.7 4.6
	TACKLE-naive	76.6 0.9	88.1 0.8	83.3 1.1	72.8 0.7	92.2 0.7	54.6 1.0
	TACKLE	**83.0** 0.6	**92.9** 0.8	**84.8** 0.9	**76.5** 0.7	**94.1** 0.8	**65.2** 0.7

Table 3: Strict schema recovery on the ERP-QA (%). Each cell reports Precision / Recall / **F1**.

LLM	Entity	Property	Relation	Relation Property
GPT-5.4-mini	80.2 / 70.5 / **75.0**	84.2 / 88.7 / **86.1**	55.8 / 81.0 / **66.0**	91.7 / 91.7 / **90.4**
DeepSeek-v4-flash	92.3 / 74.6 / **82.3**	78.9 / 76.5 / **77.6**	63.6 / 100.0 / **77.8**	100.0 / 100.0 / **100.0**
Gemini-3.6-flash	90.5 / 94.4 / **92.3**	80.5 / 80.3 / **80.3**	62.7 / 80.1 / **70.5**	72.2 / 100.0 / **82.2**

Recovery scores do not perfectly track QA accuracy, however, since they measure fidelity to one particular conceptualization; alternative but functionally valid structures may be counted as mismatches without preventing correct downstream reasoning.

5.3 Analysis and Discussion

Hypothesis validation. Across both benchmarks and all LLMs, the same ordering holds: TACKLE > TACKLE-naive > Text2SQL, with retrieval-based methods behind. This ordering supports both hypotheses: (1) Since TACKLE-naive and Text2SQL are both naive NL-to-query translators, the gap between them can be attributed to the recovered conceptual KG rather than to the translation mechanism, supporting H1; (2) Since TACKLE and TACKLE-naive share the same recovered KG and differ only in how the query is constructed, the gap between them can be attributed to the decoupled translation pipeline, supporting H2.

Limitations. Most TACKLE errors originate from incorrect concept clustering or cross-table merging. The framework may also fail when the source data are internally inconsistent or when questions require domain knowledge not expressed in the tables. Detailed failure cases are provided in Apx. C.

6 Conclusion

We study multi-table QA over practical enterprise tables that are often schema-free, semantically unaligned, and denormalized in layout. We propose TACKLE, a two-stage framework that induces a conceptual KG from raw tables without predefined schemas or ontologies and translates NL questions into executable Cypher by decoupling semantic planning from syntactic rendering. We further introduce ERP-QA, a controllable and extensible benchmark generation framework that reproduces these practical table characteristics and provides deterministically verified answers across diverse reasoning types. Experiments on an evaluation set instantiated from ERP-QA and a practical variant of MMQA show that TACKLE consistently outperforms representative formal query-based and retrieval-based baselines across multiple LLM backbones, supporting the benefits of conceptual KG induction and decoupled query generation for multi-table QA.

AI USE STATEMENT

Generative AI tools were used to assist with experimental design, code generation and debugging, interpretation of experimental results, and drafting and polishing parts of the manuscript. All AI-assisted methodological decisions, code, experimental results, and written content were reviewed and verified by the authors. The authors take full responsibility for the final content of this work, including all claims, results, code, and text produced with the assistance of generative AI.

REFERENCES

Allaa Boutaleb, Bernd Amann, Rafael Angarita, and Hubert Naacke. Exploring multi-table retrieval through iterative search. In *AI for Tabular Data Workshop, EurIPS*, 2025.

Ting Cai, Stephen Sheen, and AnHai Doan. Columbo: Expanding abbreviated column names for tabular data using large language models. *arXiv preprint arXiv:2508.09403*, 2025.

Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering SPARQL queries over relational databases. *Semantic Web*, 8(3):471–487, 2017.

Peter Baile Chen, Devin Yang, Weiyue Li, Fabian Wenz, Yi Zhang, Nesime Tatbul, Michael Cafarella, Çağatay Demiralp, and Michael Stonebraker. BEAVER: An enterprise benchmark for text-to-sql. *arXiv preprint arXiv:2409.02038*, 2024a.

Peter Baile Chen, Yi Zhang, and Dan Roth. 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 (ACL)*, pp. 2687–2699, 2024b.

Si-An Chen, Lesly Miculicich, Julian Martin Eisenschlos, Zifeng Wang, Zilong Wang, Yanfei Chen, Yasuhisa Fujii, Hsuan-Tien Lin, Chen-Yu Lee, and Tomas Pfister. TableRAG: Million-token table understanding with language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024c.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. Binding language models in symbolic languages. In *International Conference on Learning Representations (ICLR)*, 2023.

Alex Dantart and Marco Kóvacs-Navarro. Topo-rag: Topology-aware retrieval for hybrid text-table documents. *arXiv preprint arXiv:2601.10215*, 2026.

Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. TURL: Table understanding through representation learning. *Proceedings of the VLDB Endowment*, 14(3):307–319, 2021.

Yanlin Feng, Simone Papicchio, and Sajjadur Rahman. CypherBench: Towards precise retrieval over full-scale modern knowledge graphs in the LLM era. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 8934–8958, 2025.

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

Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. Beyond I.I.D.: Three levels of generalization for question answering on knowledge bases. In *Proceedings of the Web Conference 2021 (WWW)*, pp. 3477–3488, 2021.

Yu Gu, Xiang Deng, and Yu Su. Don't generate, discriminate: A proposal for grounding language models to real-world environments. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 4928–4949, 2023.

Alon Halevy, Flip Korn, Natalya F. Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. Goods: Organizing google's datasets. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD)*, pp. 795–806, 2016.

Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V. Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. G-Retriever: Retrieval-augmented generation for textual graph understanding and question answering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Felienne Hermans and Emerson Murphy-Hill. Enron's spreadsheets and related emails: A dataset and analysis. In *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering (ICSE), Volume 2*, pp. 7–16, 2015.

Subeen Ho, Hyeongu Kang, SeongKu Kang, and Susik Yoon. Pearl: Front-loading relational chains for multi-hop table retrieval. *arXiv preprint arXiv:2608.30291*, 2026.

Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çağatay Demiralp, and César Hidalgo. Sherlock: A deep learning approach to semantic data type detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pp. 1500–1508, 2019.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, and Ji-Rong Wen. UniKGQA: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. In *International Conference on Learning Representations (ICLR)*, 2023.

Lan Jiang and Felix Naumann. Holistic primary key and foreign key detection. *Journal of Intelligent Information Systems*, 54(3):439–461, 2020.

Ernesto Jiménez-Ruiz, Oktie Hassanzadeh, Vasilis Efthymiou, Jiaoyan Chen, and Kavitha Srinivas. SemTab 2019: Resources to benchmark tabular data to knowledge graph matching systems. In *The Semantic Web: 17th International Conference, ESWC 2020*, pp. 514–530, 2020.

Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. Enterprise data analysis and visualization: An interview study. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2917–2926, 2012.

Keti Korini and Christian Bizer. Column type annotation using ChatGPT. In *Joint Proceedings of Workshops at the 49th International Conference on Very Large Data Bases (VLDB), TaDA Workshop*, 2023.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *International Conference on Learning Representations (ICLR)*, 2025. Oral.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 9459–9474, 2020.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. RESDSQL: Decoupling schema linking and skeleton parsing for text-to-SQL. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 13067–13075, 2023a.

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. 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 (NeurIPS)*, 2023b.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 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, 2023c.

Mufei Li, Siqi Miao, and Pan Li. Simple is effective: The roles of graphs and large language models in knowledge-graph-based retrieval-augmented generation. In *International Conference on Learning Representations (ICLR)*, 2025.

Peng Li, Yeye He, Cong Yan, Yue Wang, and Surajit Chaudhuri. Auto-tables: Synthesizing multi-step transformations to relationalize tables without using examples. *Proceedings of the VLDB Endowment*, 16(11):3391–3403, 2023d.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. Few-shot in-context learning on knowledge base question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 6966–6980, 2023e.

Yiming Lin, Yeye He, and Surajit Chaudhuri. Auto-BI: Automatically build BI-models leveraging local join prediction and global schema graph. *Proceedings of the VLDB Endowment*, 16(10), 2023.

Jixiong Liu, Yoan Chabot, Raphaël Troncy, Viet-Phi Huynh, Thomas Labbé, and Pierre Monnin. From tabular data to knowledge graphs: A survey of semantic table interpretation tasks and methods. *Journal of Web Semantics*, 76:100761, 2023.

Shu Liu, Alan Zhu, Sumanth Hegde, Shiyi Cao, Shuo Yuan, Samion Suwito, Tyler Griggs, Matei Zaharia, Joseph E Gonzalez, and Ion Stoica. Skyrl-sql: Multi-turn sql data agents via rl. In *First Workshop on Multi-Turn Interactions in Large Language Models*, 2025a.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. A survey of text-to-sql in the era of llms: Where are we, and where are we going? *IEEE Transactions on Knowledge and Data Engineering*, 2025b.

Haoran Luo, Haihong E, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong, Meina Song, Wei Lin, Yifan Zhu, and Luu Anh Tuan. ChatKBQA: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024a.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *International Conference on Learning Representations (ICLR)*, 2024b.

Costas Mavromatis and George Karypis. GNN-RAG: Graph neural retrieval for efficient large language model reasoning on knowledge graphs. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 16682–16699, 2025. doi: 10.18653/v1/2025.findings-acl.856.

Md Mahadi Hasan Nahid, Davood Rafiei, Weiwei Zhang, and Yong Zhang. Rethinking schema linking: A context-aware bidirectional retrieval approach for text-to-sql. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 4516–4546, 2026.

Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. Data lake management: Challenges and opportunities. *Proceedings of the VLDB Endowment*, 12(12):1986–1989, 2019.

Makbule Gulcin Ozsoy, Leila Messallem, Jon Besga, and Gianandrea Minneci. Text2Cypher: Bridging natural language and graph databases. In *Proceedings of the Workshop on Generative AI and Knowledge Graphs (GenAIK) at COLING*, 2025.

Vaishali Pal, Andrew Yates, Evangelos Kanoulas, and Maarten de Rijke. MultiTabQA: Generating tabular answers for multi-table question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 6322–6334, 2023. doi: 10.18653/v1/2023.acl-long.348.

Thorsten Papenbrock and Felix Naumann. Data-driven schema normalization. In *Proceedings of the 20th International Conference on Extending Database Technology (EDBT)*, pp. 342–353, 2017.

Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *Journal on Data Semantics X*, pp. 133–173, 2008.

Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in neural information processing systems*, 36:36339–36348, 2023.

AmirHossein Safdarian, Milad Mohammadi, Ehsan Jahanbakhsh Bashirloo, Mona Shahamat Naderi, and Heshaam Faili. Schemagraphsql: Efficient schema linking with pathfinding graph algorithms for text-to-sql on large-scale databases. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 2585–2599, 2026.

Juan Sequeda, Dean Allemang, and Bryon Jacob. A benchmark to understand the role of knowledge graphs on large language model's accuracy for question answering on enterprise SQL databases. In *Proceedings of the 7th Joint Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, 2024. doi: 10.1145/3661304.3661901.

Jacob Si, Mike Qu, Michelle Lee, Marek Rei, and Yingzhen Li. TabRAG: Improving tabular document question answering for retrieval augmented generation via structured representations. *arXiv preprint arXiv:2511.06582*, 2025. NeurIPS 2025 AI4Tab Workshop.

Himadri Sonowal and Saisab Sadhu. Structure-aware chunking for abstractive summarization of long legal documents. In *Proceedings of the 1st Workshop on NLP for Empowering Justice (JUST-NLP 2025)*, pp. 171–178, 2025.

Aofeng Su, Aowen Wang, Chao Ye, Chen Zhou, Ga Zhang, Gang Chen, Guangcheng Zhu, Haobo Wang, Haokai Xu, Hao Chen, et al. Tablegpt2: A large multimodal model with tabular data integration. *arXiv preprint arXiv:2411.02059*, 2024.

Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. Annotating columns with pre-trained language models. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD)*, pp. 1493–1503, 2022.

Yuan Sui, Jiaru Zou, Mengyu Zhou, Xinyi He, Lun Du, Shi Han, and Dongmei Zhang. 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*, 2024.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M. Ni, Heung-Yeung Shum, and Jian Guo. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *International Conference on Learning Representations (ICLR)*, 2024.

TableGPT Team. Tablegpt agent: A pre-built agent for tablegpt2. `https://github.com/tablegpt/tablegpt-agent`, 2024. Accessed: 2026.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 10014–10037, 2023.

Yihan Wang, Peiyu Liu, and Xin Yang. Linkalign: Scalable schema linking for real-world large-scale multi-database text-to-sql. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 977–991, 2025.

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. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In *International Conference on Learning Representations (ICLR)*, 2024.

Jian Wu, Linyi Yang, Dongyuan Li, Yuliang Ji, Manabu Okumura, and Yue Zhang. Mmqa: Evaluating llms with multi-table multi-hop complex questions. In *International Conference on Learning Representations*, volume 2025, pp. 48626–48643, 2025.

Beiyu Xu, Zhenyu Wu, Jiaoyan Chen, et al. Mdb-link: Hierarchical schema linking for multi-database text-to-sql. *arXiv preprint arXiv:2608.09588*, 2026.

Xi Ye, Semih Yavuz, Kazuma Hashimoto, Yingbo Zhou, and Caiming Xiong. RNG-KBQA: Generation augmented iterative ranking for knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 6032–6043, 2022.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 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*, pp. 174–184, 2023.

Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 201–206, 2016.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 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*, pp. 3911–3921, 2018.

Murong Yue. A survey of large language model agents for question answering. *arXiv preprint arXiv:2503.19213*, 2025.

Jiani Zhang, Zhengyuan Shen, Balasubramaniam Srinivasan, Shen Wang, Huzefa Rangwala, and George Karypis. Nameguess: Column name expansion for tabular data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13276–13290, 2023.

Zhanke Zhou, Yongqi Zhang, Jiangchao Yao, Quanming Yao, and Bo Han. Less is more: One-shot subgraph reasoning on large-scale knowledge graphs. In *International Conference on Learning Representations (ICLR)*, 2024.

Jiaru Zou, Dongqi Fu, Sirui Chen, Xinrui He, Zihao Li, Yada Zhu, Jiawei Han, and Jingrui He. RAG over tables: Hierarchical memory index, multi-stage retrieval, and benchmarking. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026.

A PROMPTS

We present the prompts used in TACKLE here.

A.1 CONCEPTUAL KG CONSTRUCTION

A.1.1 TABLE UNDERSTANDING

```
===== SYSTEM PROMPT =====
You are an expert in database schema understanding. You will be given all tables of one
database together (table names, column headers, sample rows).

For each table, determine:
1. what the table represents
2. what each column means, and its semantic type (numeric | datetime | boolean | string)
3. whether each column is an IDENTIFIER of a concept, if yes try to find another IDENTIFIER
that it corresponds to in other tables

Return JSON:
{"tables": [
    {"table_name": "...",
      "description": "...",
      "columns": [
        {"column_name": "...", "description": "...", "semantic_type": "...",
         "is_identifier": true|false}
      ],
      "identifier_correspondences": [
        {"column": "...", "corresponds_to_table": "...", "corresponds_to_column": "..."}
      ]
}]}
```

```
Rules:
1. Every table and column must appear exactly once.
2. semantic_type reflects the stored value type, not its meaning (e.g. "yes"/"no" is
   string, not boolean).
3. Return ONLY JSON.

===== USER PROMPT TEMPLATE =====
## Tables
### `{table_name}`
Columns: {column_list}
Sample rows: {sampled_rows}
... (repeated per table)

Provide one description per table, and per column a description, semantic type,
identifier role, and identifier correspondences across tables.
```

A.1.2 COLUMN CLUSTERING

```
===== SYSTEM PROMPT =====
You are an expert in database schema understanding. You build conceptual knowledge graphs
from database tables. Your task is to read a single table's columns and group them by
**what real-world thing or concept each column describes**.

Use the table descripion and column descriptions to guide your clustering decisions.

- Columns describing the same thing form one cluster, with an identifying column
  (or set of columns) as its primary_identifier.
- Columns describing a relationship BETWEEN two concepts (not a concept itself) are
  not clustered; they are listed as relation_properties, tied to the two identifying
  columns they depend on.

Return JSON:
{"clusters": [
    {"cluster_id": "...", "label": "...", "description": "...",
     "columns": [...], "primary_identifier": ["col_a", "col_b"] }
  ],
  "relation_properties": [
    {"column_name": "...", "description": "...", "depends_on": ["src_col", "tar_col"]}
]}

Rules:
1. Every column appears exactly once, in a cluster or in relation_properties.
2. Every cluster must have a `primary_identifier`. If a single column uniquely identifies
the concept, use that column. If a combination of columns is required, provide the list of
columns.
3. Every column previously found to be an identifier column must be one cluster's
primary_identifier.
4. A relation_property's depends_on must name exactly two different identifying columns.
5. Return ONLY JSON.

===== USER PROMPT TEMPLATE =====
## Table: `{table_name}`
Table description: {table_description}
Columns: {column_list}
Column understanding:
  - {column_name}: description={column_description}; semantic_type={semantic_type}
Identifier correspondences:
  - {column_name}
Sample data (3 random rows): {sample_rows}

Group these columns by the real-world concepts they describe.
```

A.1.3 CLUSTER MERGING

```
===== SYSTEM PROMPT =====
You are an expert in entity resolution across database tables. You maintain a growing list
of canonical entities. Given the current entity list and one new cluster, decide whether
the cluster should CREATE a new entity or MERGE into exactly one existing entity.

Return JSON:
{"reasoning": "...",
  "action": "create | merge",
  "target_class_id": "... (if merge)",
  "entity": {
    "class_id": "...", "label": "...", "description": "...",
    "identifier": "...", "source_clusters": [...], "all_columns": [...],
```

```
    "properties": [{"name": "...", "description": "...", "source_columns": [...]}]
}}

Rules:
1. If no existing entity is truly the same real-world entity, choose `create`.
2. If one existing entity is clearly the same entity, choose `merge`.
3. When merging, align incoming columns to existing properties where they mean the same
thing.
4. Add new properties when the incoming cluster contributes new attributes.
5. Return ONLY JSON.

===== USER PROMPT TEMPLATE =====
## Current Canonical Entities
- class_id={class_id}, label={label}, identifier={identifier}, source_clusters={...},
all_columns={...}
  description={description}
  properties: - {name}: source_columns={...}; description={...}; sample_values={...}

## Incoming Cluster
cluster_ref={table_name}.{cluster_id}
table_description={table_description}
cluster_label={label}; cluster_description={description}
cluster_columns={columns}; cluster_identifier={primary_identifier}
cluster_column_understanding:
  - {column}: description={...}; semantic_type={...}; sample_values={...}
cluster_identifier_correspondences:
  - {column} -> {other_table}.{other_identifier_column}

Decide whether to create a new entity or merge into one existing entity.
```

A.1.4 RELATION GENERATION

```
===== SYSTEM PROMPT =====
You are an expert in database schema understanding. You are given one table with multiple
clusters, and a list of relation properties. Infer directed semantic relations between the
clusters based on their identifier correspondences, and attach the relevant relation
properties to those relations.

Return JSON:
{
  "reasoning": "...",
  "relations": [
    {"source_entity": "...", "target_entity": "...",
     "source_cluster": "table.cluster_id", "target_cluster": "table.cluster_id",
     "relation": "snake_case_label", "cardinality": "1:1 | 1:N | N:M",
     "description": "...",
     "properties": [{"name": "...", "description": "...", "source_column": "..."}]}
  ]
}

Rules:
1. Only use the listed clusters, entity ids, and relation_properties.
1. For every meaningful connection between two clusters, generate relations in
   BOTH directions, each with its own direction-specific label and description.
2. Generate a distinct relation for every semantically meaningful pair of clusters -
   even when several clusters map to the same entity. Such clusters represent
   distinct ROLES and must each get a role-specific label (never collapsed).
3. Attach each relation property to the relation whose source/target entities match
   the identifying columns it depends on.
4. Reuse a relation label already established for the same directed entity pair in
   another table, IF this table expresses the same real-world relationship.
5. Return ONLY JSON.

===== USER PROMPT TEMPLATE =====
table={table_name}
table_description={table_description}
clusters:
- cluster_ref={table}.{cluster_id}; cluster_label={...}; columns={...};
  column_profiles={...}; identifier={...}; entity_id={...}; entity_label={...}
  ... (one line per cluster)

cluster_pairs: {all pairwise cluster combinations; each must receive >= 1 relation,
or "none" with justification}

existing_relations_for_these_entities:
- {source_class} -> {target_class}: relation="{label}"; cardinality={...};
  description={...}; established_from_table={...}

relation_properties:
```

```
- column_name={...}; description={...}; depends_on={...}; sample_values={...}

Infer relations for this table only.
```

A.2 Decoupled Cypher Translation

A.2.1 Semantic Grounding

```
===== SYSTEM PROMPT =====
You are a schema-grounded query analyser. You map the question's meaning onto the concrete
schema (nodes and relationships with properties), so a later step can render Cypher
mechanically.

Split the question into one or more independent sub-questions (most questions need only
one). For each sub-question, produce a free-text "grounding" that pins down, in schema
terms: the answer entity, the traversal path(s) to every other entity mentioned, the filter
conditions, the columns to project, any aggregation, and any ordering/limit. Use only
schema names, and align filter values with the schema's sample values rather than the
question's literal wording.

This grounding is a HINT for a later Cypher-writing step, not an executed plan.

Return ONLY JSON:
{
  "reasoning": "...",
  "queries": [
    {"id": "q1", "sub_question": "...", "grounding": "..."}
  ]
}
===== USER PROMPT TEMPLATE =====
# GRAPH SCHEMA
{schema_block}

# QUESTION
{question}

Return only the JSON object.
```

A.2.2 Syntactic Rendering

```
===== SYSTEM PROMPT =====
You are an expert Cypher query writer.
Return ONLY the Cypher query text, no explanations, no markdown fences, no JSON.

===== USER PROMPT TEMPLATE =====
# TASK
Translate the sub-question into a single Cypher query using ONLY the node tables,
relationship tables, columns, and relationship directions listed in the schema.

# GROUNDED PLAN (HINT)
A previous analysis step produced this schema-grounded plan. It is a HINT only: if it
conflicts with the sub-question or the schema, follow the sub-question.

{plan}

# RULES
- Use ONLY names from the schema, matching their exact spelling.
- Traverse a relationship table exactly in its schema direction:
`(a:SourceTable)-[r:RelTable]->(b:TargetTable)`. Never flip a relationship direction.
- Return ONE query ending in a RETURN clause. Do not wrap it in code fences.

# GRAPH SCHEMA
{schema_block}

# SUB-QUESTION
{question}

Return only the Cypher query.
```

A.3 Practical MMQA

We present the prompts used to rename the aligned table and column names in Practical MMQA.

Table 4: Statistics of Practical-MMQA, inherited from the original MMQA benchmark (Wu et al., 2025)..

Property	2-table	3-table
Total tables	2591	721
Avg. rows per table	1833.31	1369.01
Avg. columns per table	6.04	4.78
Avg. foreign keys per table	2.81	1.95
Avg. primary keys per table	3.35	2.41
Inter-human agreement	86%	82%
Avg. question length	77.11	85.38
Question Category		
Numerical	889	289
List	214	44
Count	200	42
Select	1289	346

```
===== SYSTEM PROMPT =====
You are an expert at obfuscating relational database schemas for schema-linking
benchmark evaluation. Given one item's tables (names, columns, and sample values),
return a JSON object that assigns each table and column a NEW name.

Rules:
- Preserve each column's meaning while changing its surface form (synonym,
  abbreviation, or different naming convention).
- Every table and column name must differ from its original name, and must stay
  unique within its scope (table names unique within the item, column names
  unique within their table).
- Ignore any primary/foreign-key relationships: rename each column
  independently; matching concepts across tables need not stay aligned.


Return ONLY JSON, tables and columns in the same order as the input:
{
  "tables": [
    {"new": "...",
     "columns": [{"new": "..."}]}
  ]
}

===== USER PROMPT TEMPLATE =====
TABLE 1: {table_name}
Columns (name | type | up to 3 sample values):
  {column_name} | {inferred_type} | {sample_value_1}, {sample_value_2}, {sample_value_3}
  ...

TABLE 2: {table_name}
...
```

B Benchmark Statistics

Tab. 4 and Tab. 5 presents the statistics of the practical-MMQA and the ERP-QA benchmark used in our evaluation.

C Error Analysis and Future Work

Failure Modes. Most errors in TACKLE arise during KG induction, specifically in the clustering and merging steps, where columns describing different concepts are clustered together, or two clusters that should be distinct are merged. These errors are relatively rare and can be addressed by improving better instruction or providing examples in the prompts.

We highlight two fundamental failure modes that are common in real enterprise scenarios and are general rather than specific to our pipeline. They present further challenges for automatic reasoning over such tables in general.

Table 5: Statistics of the self-constructed benchmark across the two domains.

Statistic	Manufacturing	University	Combined
# Entity types	6	6	12
# Entity instances	171	246	417
# Relation types	5	5	10
# Relation instances	258	374	632
# Properties	10	10	20
# Table rows (total)	216	294	510
# Questions (total)	104	230	334
1-hop retrieval	21	46	67
2-hop retrieval	22	46	68
3-hop retrieval	17	46	63
Flat aggregation	19	46	65
Nested aggregation	25	46	71

- **Data instance errors** refer to errors in the data values themselves, rather than in the schema. They can take many forms, such as formatting inconsistencies, out-of-range values, or semantically incorrect entries. One common example is a foreign key column that contains a list of values rather than a single value, as in the `Application_Responsible_Person` table, where `Responsible_Person` should reference `Employee`. Such a list-valued foreign key prevents instance-level construction of the conceptual KG and causes the corresponding reasoning to fail. This example illustrates a broader challenge: the data itself may not conform to the assumptions required for automatic reasoning, independent of how well the schema is recovered.
- **Missing background knowledge.** LLM understanding errors may occur even given the conceptual KG describing the enterprise tables. When NL questions include domain-specific terms, such as a specific `machine_model` name, an LLM may fail to map that name to the `machine_model` without external knowledge.

Future Directions. These limitations point to the direction for future work: integrating lightweight domain ontologies with the recovered conceptual KG. For data instance errors, an ontology can provide domain or default rules that allow data errors to be detected and normalized at the instance level. For missing background knowledge, the same ontology can provide the domain-specific concepts that the tables alone do not express, such as mapping a machine model name to the corresponding concept. Because the conceptual KG already captures the structure of the tables, an ontology can be attached to it without requiring a full relational schema.