Is Table Retrieval a Solved Problem? Exploring Join-Aware Multi-Table Retrieval

Peter Baile Chen
MIT
peterbc@mit.edu

Yi Zhang *
AWS AI Labs
imyi@amazon.com

Dan Roth
University of Pennslyvania
danroth@seas.upenn.edu

Abstract

Retrieving relevant tables containing the necessary information to accurately answer a given question over tables is critical to open-domain question-answering (QA) systems. Previous methods assume the answer to such a question can be found either in a single table or multiple tables identified through question decomposition or rewriting. However, neither of these approaches is sufficient, as many questions require retrieving multiple tables and joining them through a *join plan* that cannot be discerned from the user query itself. If the join plan is not considered in the retrieval stage, the subsequent steps of reasoning and answering based on those retrieved tables are likely to be incorrect. To address this problem, we introduce a method that uncovers useful join relations for any query and database during table retrieval. We use a novel re-ranking method formulated as a mixed-integer program that considers not only table-query relevance but also table-table relevance that requires inferring join relationships. Our method outperforms the state-of-the-art approaches for table retrieval by up to 9.3% in F1 score and for end-to-end QA by up to 5.4% in accuracy.

1 Introduction

Structural tables are an important source of knowledge for many real-world applications, such as open domain question answering (Herzig et al., 2021) and open fact-checking (Schlichtkrull et al., 2021). A standard paradigm to leverage structural tables as a knowledge source follows first retrieving relevant tables, then conducting standard question answering or fact verification over those top-ranked tables. This setup has been recently adopted by Retrieval Augmented Generation (RAG) (Lewis et al., 2020; Pan et al., 2022; Gao et al., 2023), trying to address the issue of knowledge update (Yu et al., 2022) and hallucination (Maynez et al., 2020; Zhou et al., 2021) of Large Language Models (LLMs). Once the tables required to answer the user prompt are identified, they will then be fed to LLMs with either zero-shot prompts or in-context learning examples (Brown et al., 2020) to generate the final answer. Since the reasoning and answer generation is conditioned on the retrieved tables, table retrieval is a critical problem. Recent works (Herzig et al., 2021; Wang et al., 2022) have proposed to learn a model that can capture the similarity between table contents and the given query using either a text or table-specific model design. However, these works simplify the table retrieval problem in real-world open-domain settings. The reasons are as follows.

First, to answer a question or verify a fact relevant to structural tables, we may need *multiple* tables. For example, given a database and a query *"Who are the female BOA account holders who own credit cards and also have loans?"*, ideally, we may want to find an account table with a schema including columns such as `account_id`, `gender`, `card_id`, `loan_id`, as shown in the first block in Figure 1, which we can directly use to answer the query by writing a SQL query. However, in real-world databases, it is not very likely that all the information is stored in a single table. In our running example, since a client can have multiple credit cards and loans, to avoid repeating information which leads to a very large table, the single table is typically split into three separate joinable tables, i.e., an `Account` table, a `Card` table, and a `Loan` table. All these tables need to be retrieved to answer the query correctly. Therefore, table retrieval should not only target retrieving a single relevant table but always *multiple* tables.

Secondly, retrieving multiple tables requires understanding the *relationship* among candidate tables, which is independent of how the query is framed. A straightforward approach to solve multi-table retrieval can be training a model that can decompose queries (Schick et al., 2023), so that each

*Work was done before joining AWS.

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 2687–2699

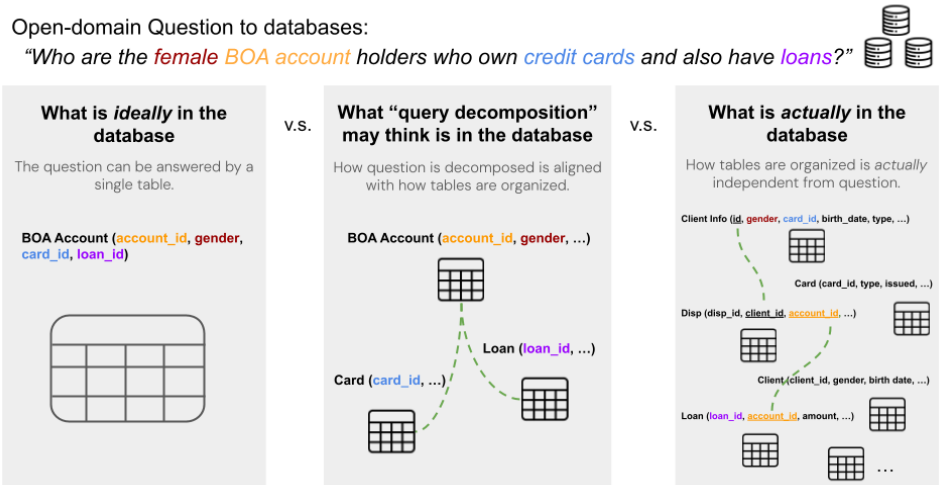

Figure 1: Previous work makes simplifying assumptions for table retrieval. Left block assumes there exists a single table that is sufficient to answer the question. Middle block assumes necessary joins can be recovered from query decomposition. However, in practice, the question may be more complicated and the join plan may not be discerned from the user query itself (right block). Therefore it is important to identify the join relationship while conducting table retrieval.

query is aligned to a table. In our running example, the query mentions `account`, `credit cards`, and `loans`, which may indicate the set of required tables. Although the decomposition aligns with how the query is framed, it might not align with how relevant data are organized and stored. For instance, relevant tables might have the following structures:

Client Info (`id`, `gender`, `card_id`, ...)
Disp (`disp_id`, `client_id`, `account_id` ...)
Loan (`loan_id`, `account_id`, `amount`...)

In this case, we may need to join all of them, as shown in the third block in Figure 1, to answer the given query. However, we may also find another set of tables, for example:

Card (`card_id`, `type`, `issued`, ...)
Client (`client_id`, `gender`, `birth date`, ...)
Loan (`loan_id`, `client_id`, `amount`, ...)

Although this set of tables seems to be highly relevant, it is not the right set of tables to return because they cannot be joined. In particular, the relationship between the cards listed in the `Card` table and the clients in the `Client` table is missing, and therefore they can not produce the answer. Importantly, only if it understands the relationship among tables can the model determine which set of tables to return.

Last but not least, the relationship among tables needs to be *inferred*. Previous works in Text-to-SQLs (Yu et al., 2018a; Li et al., 2023b; Lee et al., 2021) have considered table joins when generating SQL expressions. However, they all assumed that the key-foreign-key constraints (join relationships) had already been provided with the database. In reality, this is not always possible, in particular when we are searching from data lakes (Zhu et al., 2019; Nargesian et al., 2019; Zhang and Ives, 2020; Cong et al., 2022) or simply a big dump of tables, which introduces additional challenges during the retrieval.

To sum up, the problem of table retrieval was simplified in previous works, whereas it can be more challenging in a real-world open-domain setup. To solve the problems we have discussed, in this paper, we formulate the multi-table retrieval problem as a re-ranking problem over a ranked list of tables produced by a standard table retrieval method, and we developed an algorithm to select the best set of tables during test time by considering both relevance and join relationship together. We conduct extensive experiments on Spider (Yu et al., 2018a) and Bird (Li et al., 2023a) datasets, demonstrating that our re-ranking mechanism outperforms baselines in both retrieval and end-to-end performances.

2 Problem Description

A standard table retrieval problem for an open-domain question-answering task over tables can be defined as follows. Given a query Q, a table corpus $C = \{T_i\}_{i=1}^{M}$, retrieve a table T_i from C,

such that the answer of Q resides in T_i. This paper extends the problem definition to consider multiple tables that require table joins.

Specifically, given a query Q, a table corpus C, rather than returning a single table $T_i \in C$, our problem is to return a ranked list of tables with join operators denoted as $E(Q)$, such that the answer of Q resides in the tables in $E(Q)$. Here $E(Q) = \{T_{Q,1} \bowtie_{c_1} T_{Q,2}, ...\}$, where $T_{Q,1}, T_{Q,2}, ... \in C$, and $c_1, ...$ represent the join conditions. For example, if table `Client Info`'s column `id` should join table `Disp` on its column `client_id`, then $c_i = (id, client\_id)$.

3 Join-aware Multi-Table Retrieval

To address the problem described in the section above, i.e., identifying the best table expression over a subset of tables from the given databases, we need to consider two aspects. One is *table-query relevance*, which estimates how likely a candidate table contains the information that can be used to answer the query. The other one is *table-table relevance*, which is to identify the tables that contain complementary information belonging to the same objects. In this case, *table-table relevance* is equivalent to how likely tables can be joined.

We note that these two components should be considered simultaneously to answer the query because only tables that are relevant to the query and compatible with each other can provide sufficient information to answer the query. Therefore, we propose to solve the problem by adjusting the ranking produced by *table-query relevance* based on relationships inferred from *table-table relevance* to find the best retrieval result.

In the following sections, we will focus on this re-ranking problem, developing a computational framework that can consider and infer the join relationship while generating the new table ranking to help answer the query.

3.1 Query-Table Relevance

A standard way to compute query-table relevance is to compute a similarity score through a bi-encoder model architecture (Chen et al., 2020b; Huang et al., 2022; Herzig et al., 2021). Such a similarity score can capture coarse-grained relevance. However, real queries can be complicated (e.g., multi-hop questions). The relevant information to answer the query may be distributed across multiple tables, and each table that should be used to answer the query may be only mapped to a part of the query. In our running example, a `Card` table may only tell us *"who own credit cards"* but not *"who have loans"*.

Therefore, we argue that fine-grained query-table relevance scores are also necessary so that we can find relevant tables (and their columns) for different information that is required to answer the query. To achieve this, we compute the score by first decomposing the query and then computing the semantic similarity between each sub-query and columns from candidate tables.

Specifically, for a query Q, we want to decompose it into sub-queries that can potentially be mapped to tables and columns. To achieve this goal, rather than simply using keywords, we use *concept* and its *attribute* mentioned in Q as a pair to represent a sub-query. For instance, the query *"What is the id of the trip that started from the station with the highest dock count?"* requires information relevant to two main concepts: trip and station. More specifically, the query needs information about the attribute "id" from the trip concept and the attribute "dock count" from the station concept. Therefore, the query should be decomposed into ⟨trip, id⟩ and ⟨station, dock count⟩, which may better indicate that we need to look for a table about trip with id information, and a table about station with dock information respectively.

To perform the decomposition, we use an LLM (GPT-3.5 Turbo for our experiment) directly by feeding the prompt stated in A.1 to it[1]. Then, for each sub-query $q \in Q$, the similarity r_{qik} between q and column c_k of table T_i is calculated as the semantic similarity between q and column c_k using a standard bi-encoder model (Izacard et al., 2021).

3.2 Table Re-ranking with Table-Table Relevance

As discussed above, we prefer tables with high *query-table relevance* and *table-table relevance*.

For query-table relevance, we consider both *coarse-grained* and *fine-grained* relevance introduced in Section 3.1, so that we can find the tables that are most relevant to all the information that is required to answer the query. To ensure the coverage of different information, we further complement the relevance score with a coverage measurement, which will be described in detail in Section 3.2.1.

[1]We used 5 ICL examples.

For *table-table relevance*, we consider the compatibility among the returned tables, i.e., whether selected tables can join with each other. The compatibility (joinability) helps us ensure that the information from the selected tables is compatible — the information is about the same group of entities or objects, so that we can use *all of them* to answer the query correctly. Therefore, during the process of selecting tables that can maximize the compatibility scores, we further encourage the selected tables to be chained through join, which will be discussed in Section 3.2.2.

All in all, we formulate the re-ranking problem as an optimization problem, maximizing the sum of the aforementioned components as a Mixed-integer Linear Program (MIP). To incorporate the coarse-grained relevance scores, we define the binary decision variable b_i that denotes whether table $T_i \in C$ is chosen to be returned and parameter r_i that denotes the coarse-grained relevance score between the input query and table T_i. The maximization term, the total coarse-grained relevance of tables selected, is $\sum_i r_i b_i$. Similarly, to consider the fine-grained relevance scores, we introduce the binary decision variable d_{qik} that denotes whether or not sub-query q is covered by column c_k of table T_i and parameter r_{qik} that denotes the fine-grained relevance score between sub-query q and column c_k of table T_i. The maximization term, the total fine-grained relevance of each sub-query, is $\sum_{q,i,k} r_{qik} d_{qik}$. To take into account the compatibility scores, we define the binary decision variable c_{ij}^{kl} that describes whether column c_k of table T_i is selected to join with column c_l of table T_j and parameter ω_{ij}^{kl} that denotes the compatibility between column c_k of T_i and column c_l of table T_j. The maximization term, the total pair-wise compatibility of columns selected, is $\sum_{i,j,k,l} \omega_{ij}^{kl} c_{ij}^{kl}$.

Objective. Using the above variables, parameters, and three maximization terms, we define our objective function to return the best K tables in the following equation

$$\arg\max \sum_i r_i b_i + \sum_{q,i,k} r_{qik} d_{qik} + \sum_{i,j,k,l} \omega_{ij}^{kl} c_{ij}^{kl}$$

The objective function is subject to the following six constraints:

Constraint 1. The integral constraint ensures the decision variables are binary, as defined above.

$$b_i, c_{ij}^{kl}, d_{qik} \in \{0, 1\} \tag{1}$$

Constraint 2. This constraint sets the maximum number of tables and join relationships that can be selected according to the input K, as mentioned in the objective.

$$\sum_i b_i = K, \sum_{i,j,k,l} c_{ij}^{kl} \leq K - 1 \tag{2}$$

Constraint 3. This constraint enforces that a pair of columns from two tables should only be selected if these two tables are selected to return.

$$2(c_{ij}^{kl} + c_{ji}^{lk}) \leq b_i + b_j, \forall i, j, k, l \tag{3}$$

Constraint 4. For simplicity, we assume that tables are always joined via a single-column key, while we leave joining with compound keys in future work. This constraint requires that if two tables are selected to join, they join via a single column from each table.

$$\sum_{k,l} c_{ij}^{kl} \leq 1 \tag{4}$$

Constraint 5. This constraint requires each sub-query is covered by at most one column of a table.

$$\sum_k d_{qik} \leq 1, \forall q, i \tag{5}$$

Constraint 6. This constraint requires that tables used to cover sub-query must be selected to return. $|Q|$ is the number of sub-queries.

$$\frac{1}{|Q|} \sum_q d_{qik} \leq b_i, \forall i, k \tag{6}$$

3.2.1 Sub-query Coverage

The main purpose of measuring fine-grained table-query relevance is to find different columns (may or may not come from the same table) that can map to different parts of the query, so that all information required to answer the query can be covered. Although we have $\sum_{q,i,k} r_{qik} d_{qik}$ in the objective function to maximize the overall relevance between different parts of the query and columns, it may be biased to aligning a sub-query (weakly) to many columns to maximize the term rather than aligning a sub-query strongly to one column.

Therefore, to encourage strong mappings, we further introduce the following modifications to the MIP formulation. We add an upper limit on the total number of coverage connections between sub-queries and tables that equals to the number of

sub-queries. To prevent some sub-queries from not being covered, we introduce the binary decision variable d_q to denote whether or not sub-query q is covered by at least one column. We then add the sum of d_q terms $\alpha \sum_q d_q$ to the objective function. The α coefficient is a knob that controls which strategy to adopt. Large α increases the importance of covering every sub-query covered, which prefers a single table with strong mappings. A low α, on the other hand, prefers many tables with weak mappings. The detailed constraints are included in Appendix B.1.

3.2.2 Connectedness

We consider a graph where tables are nodes and compatibility relationships between two tables are edges. To ensure selected tables are compatible with each other, as discussed at the beginning of Section 3.2, we formulate the following connectedness problem on this graph: the K selected nodes are connected (i.e., any two nodes can be reached by some paths). This problem can be solved by augmenting the original graph and reducing it to a maximum flow problem where connectedness is ensured if and only if the max total flow between source and sink is K (Even and Tarjan, 1975).

To ensure connectedness among the selected tables $N = \{T_i | T_i \in C, b_i = 1\}$, we create an augmented graph $G' = (N', E')$ based on $G = (N, E)$, with the following modifications: (1) $N' = N \cup \{source, sink\}$ (2) $E' = E \cup \{(source, x)\} \cup \{(n, sink), \forall n \in N\}$ where x is a node in N. Then, we define the capacity $h(m, n)$ for each edge between nodes m and n in N'. All edges have 0 capacity except the following:

$$h(source, x) = K, h(n, sink) = 1, \forall n \in N$$
$$h(m, n) = K, \forall m, n \in N$$

We include the complete set of constraints in Appendix B.2.

3.3 Table-Table Relationship Inference

The compatibility between two tables can be represented as the most *joinable* pair of columns between the two tables. To compute the joinablity between two columns, we consider two aspects: column relevance and the likelihood of satisfying the key foreign-key constraint.

A column consists of its schema (header) and instances (rows). Naturally, to compute column relevance, we consider the similarity between its schema and instances. To measure the similarity between column instances, we compute the exact-match overlap (Jaccard similarity). Given two columns, c_k of T_i and c_l of T_j, it is defined as $\frac{|I(c_k) \cap I(c_l)|}{|I(c_k) \cup I(c_l)|}$ where $I(c_k)$ represents the instances of column c_k. To have a holistic understanding of the column schema, we consider the semantic similarity between two column headers as well as their contextual information, i.e., the corresponding table names and the other columns in the table. Specifically, we encode each segment using a pre-trained embedding model (Izacard et al., 2021), and compute the cosine similarity as the similarity score of each segment. Schema similarity is the weighted sum of similarities of all segments. Column relevance is the sum of instance similarity and schema similarity.

We want to further ensure the correctness of joining two columns by considering key foreign-key constraints. For instance, consider table T_1 with columns: {`student_id`, `city`}, with rows: {⟨1, BOS⟩, ⟨2, BOS⟩} and table T_2 with three columns: {`teacher_id`, `student_id`, `city`} with three rows: {(1, 1, BOS), (2, 1, BOS), (2, 2, BOS)}. Teachers want to see the cities of students their students are from, so they join T_1 and T_2, for example, over `city`. However, this join is incorrect because it creates incorrect teacher-student pairs (e.g., teacher 1 and student 2). A legal join, instead, should correctly connect information from two tables. This occurs when the column c_k with duplicate values (the foreign key) from T_i joins another column c_l (the primary key) from T_j where each of c_k's value is unique. In short, we want to encourage joins between two columns where at least one of them is a primary key.

To address this, we compute the uniqueness u of both columns c_k and c_l and use $\max\{u(c_k), u(c_l)\}$ to approximate the likelihood of these two columns satisfying the key foreign-key constraint. Uniqueness is defined as the ratio between the number of unique values in a column and the number of instances in the table. If a column is a primary key, its uniqueness score equals to 1, whereas if a column is a foreign key, the uniqueness score should generally be smaller than 1 because they are likely to be repeated (e.g., `city` column). Note we take the max of the two uniqueness score because we only need at least one of the two columns to be a primary key.

To summarize, joinablity score ω_{ij}^{kl} comprises

two components: (1) column relevance that includes both instance similarity j and schema similarity e (2) likelihood of satisfying key-foreign key constraint, which is the maximum of the uniqueness u of two columns. Specifically, $\omega_{ij}^{kl} = (e(c_k, c_l) + j(c_k, c_l)) * \max\{u(c_k), u(c_l)\}$.

4 Evaluation

In this section, we evaluate the baselines and our re-ranking method on datasets whose questions require multiple tables to answer. The goal is to answer the following two research questions: (1) To what extent can our re-ranking method, which jointly considers query-table relevance and table-table relevance, improve the existing table retrieval solutions that consider only standard query-table relevance? (2) To what extent can a better retrieval performance due to our re-ranking method enhance the *end-to-end* performance of question answering over tables?

4.1 Experimental Settings

Datasets. To the best of our knowledge, there are no existing large-scale open-domain question-answering datasets that consider multiple tables[2]. Therefore, we use text-to-SQL datasets which naturally consider multiple tables for our evaluation, but under an open-domain question-answering setup. In this case, it is required to first retrieve tables from a table corpus, then answer the query by generating an SQL expression. Specifically, we run evaluation on two popular text-to-SQL datasets: Spider (Yu et al., 2018b) and Bird (Li et al., 2023a). In these two datasets, tables are organized by topic, and each topic has its own database (around 5.4 tables per database) and queries that can be answered. To make these datasets suitable for our setting, we aggregate tables from all databases to construct a centralized table corpus for each dataset. The original datasets contain key foreign-key constraints. However, as described in Section 1, such constraints are not always specified and may need to be inferred during test time. Therefore, we perform evaluations on two setups: with and without the gold key foreign-key constraints provided by the original datasets (if gold constraints are provided, we set the compatibility score of the corresponding pairs of columns to 1). Moreover, since we focus on queries that require multiple tables to answer, we omit those that only use one table to construct the gold SQL expression. We also remove queries with incorrect answers through human inspection. We provide examples to explain the removal process in Appendix C. The above procedure leads to 443 queries and 81 tables for Spider, and 1095 queries and 77 tables for Bird.

Baselines. We set up two baselines for our evaluation. First of all, we consider a strong text retrieval model without explicitly considering the table structure (Izacard et al., 2021). It is inspired by the study from Wang et al. (2022) that table-specific model design may not be necessary for table retrieval. Specifically, we use Contriever-msmarco[3] to compute the embeddings for the given query and the flattened tables. The ranking of tables is based on the cosine similarities between the query and the tables. Then, we consider a retrieval method, which is designed for table retrieval, DTR (Herzig et al., 2021). Specifically, we finetuned TAPAS-large[4] following DTR on the tables in the training set (with a 8:2 train-valid split) of each dataset respectively. Then, the fine-tuned model outputs a score for each question-table pair, which is used for ranking candidate tables.

Environment. We use Python-MIP [5] package and Gurobi as our MIP solver. Fine-tuning of DTR models as well as computing coarse- and fine-grained relevance scores are performed on one Tesla V100 GPU.

Tasks and Metrics. We conduct the following two types of evaluation.

Retrieval Only Evaluation. This evaluation is to directly evaluate to what extent our re-ranking mechanism can improve the baselines with regard to retrieving tables for given queries. We report precision, recall and F1 varying k for our methods and the baselines.

End-to-end evaluation. The end-to-end evaluation is to study whether the improved table retrieval can actually help the downstream task, e.g., question answering. Specifically, the ranked tables from various retrieval methods are used to construct prompts (A.2[6]), which will be fed to an LLM to generate an

[2] KaggleDBQA (Lee et al., 2021) has only 26 queries involving multiple tables, which is not a representative dataset for our task.

[3] https://huggingface.co/facebook/contriever-msmarco

[4] https://huggingface.co/google/tapas-large

[5] https://www.python-mip.com/

[6] We used one ICL example to teach the model how to generate the output.

	Top-2						Top-5						Top-10					
	Spider			Bird			Spider			Bird			Spider			Bird		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Baselines — Standard Retrieval Approach																		
DTR	81.4	77.4	78.9	64.9	58.9	61.3	41.0	95.8	57.1	37.2	82.9	50.9	21.3	99.3	34.9	21.5	95.4	34.8
Contriever	76.2	72.7	74.0	65.0	59.4	61.6	39.8	93.3	55.5	37.0	82.9	50.8	20.6	96.3	33.8	21.1	93.9	34.3
Our methods — Full Re-ranking																		
JAR-F (DTR)	**87.1**	**82.9**	**84.5**	**76.1**	**69.3**	**72.0**	**41.9**	**97.8**	**58.3**	40.2	89.7	55.1	**21.3**	**99.7**	**35.0**	**21.7**	**96.6**	**35.3**
JAR-F (Con.)	82.8	79.1	80.5	73.4	67.0	69.5	39.9	93.7	55.7	**40.3**	**89.9**	**55.2**	20.7	96.6	33.9	21.5	95.4	34.9
*Full Re-ranking with **gold** key foreign-key constraints*																		
JAR-G (DTR)	92.2	87.9	89.6	78.0	71.1	73.8	42.4	99.2	59.1	40.9	91.3	56.0	21.4	99.9	35.1	22.1	97.8	35.8
JAR-G (Con.)	89.2	85.8	87.1	76.7	70.2	72.7	40.3	95.0	56.3	40.4	90.1	55.3	20.6	96.6	33.8	21.8	96.5	35.3
*Ablation study: Full Re-ranking **without** table-table relevance*																		
JAR-D (DTR)	86.4	82.3	83.9	74.7	68.1	70.7	41.4	96.7	57.6	39.5	88.4	54.2	21.3	99.5	35.0	21.6	96.2	35.1
JAR-D (Con.)	81.7	78.1	79.5	72.7	66.3	68.8	39.8	93.3	55.5	39.2	87.5	53.7	20.6	96.4	33.8	21.3	94.5	34.5

Table 1: Our methods outperform baselines in retrieval performances across all datasets. The top half of the table shows that our full re-ranking attains higher retrieval performances than the standard retrieval methods: DTR and Contriever. Boldface numbers indicate the best performances in the top half of the table. The bottom half of the table shows our approach can effectively use constraints if provided to achieve even better performances. It also shows an ablation suggesting that table-table relevance is essential to the retrieval.

SQL expression to answer the query. In our experiments, we use GPT-3.5 Turbo 1106 (temperature = 0) as our backbone LLM. To evaluate the end-to-end performance, generated SQL expressions are executed, and we compare the results with those from gold SQL expressions to report the execution accuracy. Moreover, we report the precision, recall and F1 scores of the tables used in the final SQL expression to demonstrate that better retrieval leads to a better selection of tables by the LLM.

We use the following notations: we use `X` (e.g., DTR) to denote the original baseline approach, `JAR-D (X)` to denote our re-ranking approach based on `X` but only considering the fine-grained query-table relevance without table-table relevance, `JAR-F (X)` to denote our full re-ranking mechanism based on `X`, and `JAR-G (X)` to denote the same approach as `JAR-F (X)` but using the provided gold key foreign-key constraints.

In the following sections, we discuss the results for table retrieval in Section 4.2 and the end-to-end question answering task in Section 4.3. We also include an ablation study as well as a discussion of the results when gold key foreign-key constraints are available and results under different datasets and numbers of tables in Section 4.4.

4.2 Table Retrieval Performances

Results of table retrieval are reported in the upper half of Table 1, with typical choices of k, including 2, 5, and 10. Across different values of k, our re-ranking mechanism (JAR-F) can achieve higher retrieval performances compared to baseline retrieval approaches (DTR and Contriever). Specifically, for the Spider dataset, JAR-F (DTR) outperforms DTR by at most 5.6% in F1 while JAR-F (Contriever) outperforms Contriever by at most 6.5% in F1. For the Bird dataset, JAR-F (DTR) outperforms DTR by up to 10.7%, in F1 while JAR-F (Contriever) outperforms Contriever by at most 7.9%. The extent of performance improvements decreases as k increases because as the baseline performances increase, it is more difficult to further improve upon them.

4.3 End-to-end Performances

In this section, we want to show that a better retrieval performance can translate to a better end-to-end performance on the downstream task, e.g., question answering through text-to-SQL. In table 2, we report the execution accuracy of SQL expressions, and in table 3, we report the "retrieval performance" of tables by the LLM when generating the final SQL expressions.

In table 2, we observe that when feeding the same number of tables (top-5) to the LLM, our full re-ranking solutions can significantly outperform the baselines. Since our re-ranking mechanism is an additional step between the standard table retrieval and LLM response generation, we also want to investigate whether the LLM can do some level of re-ranking. Therefore, we also com-

	DTR			JAR-F (DTR)	JAR-G (DTR)	Contriever			JAR-F (Con.)	JAR-G (Con.)
	Top-5	Top-10	Top-20	Top-5	Top-5	Top-5	Top-10	Top-20	Top-5	Top-5
Spider	46.3	47.4	48.1	**50.6**	52.2	43.5	43.1	44.7	47.4	48.3
Bird	30.4	34.8	31.5	**35.8**	36.9	29.7	30.6	32.9	35.1	36.2

Table 2: Our re-ranking mechanism achieves considerable improvements compared to baseline approaches on both Spider and Bird datasets in terms of end-to-end execution accuracy.

	Spider			Bird		
	P	R	F1	P	R	F1
Top-5						
Full Re-ranking						
JAR-F (DTR)	**77.2**	**77.4**	**77.0**	**65.6**	**66.1**	**65.3**
JAR-F (Con.)	75.4	74.8	74.7	64.7	65.4	64.7
Baselines						
DTR	75.8	75.6	75.2	59.5	59.2	58.8
Contriever	71.9	71.4	71.3	55.9	55.5	55.2
Top-10						
DTR	77.6	76.7	76.7	65.4	65.2	64.7
Contriever	74.1	73.2	73.2	61.8	62.5	61.7
Top-20						
DTR	76.0	74.8	75.1	64.5	64.8	64.2
Contriever	74.4	73.6	73.6	64.7	64.4	64.1

Table 3: Our approach attains significant gains compared to baseline approaches on both Spider and Bird datasets in end-to-end retrieval performances.

pare against the baseline which feeds 20 tables (the same number of tables we feed to our re-ranking method) to the LLM. Table 2 shows, on average across both models, an 2.6% and 3.3% improvement on execution accuracy for Spider and Bird respectively. We then further investigate feeding LLM with top-10 tables, which represents a balance between precision and recall compared to top-5 and 20, and we can observe similar improvements. All these results demonstrate the effectiveness of our re-ranking mechanism.

In table 3, we further investigate the table retrieval performance by the LLM with different input of tables produced either by the baselines or our methods. As shown in the table, our approach with full re-ranking outperforms baselines across all datasets and numbers of input tables for baseline approaches. Compared to the best overall baseline (top-20), our approach still, on average, outperforms it by 1.5% and 0.85% in F1 for Spider and Bird respectively. It demonstrates that our re-ranking mechanism actually improves the end-to-end performance by helping the LLM select the correct tables.

	Queries with 2 tables						Queries with $\geq$ 3 tables					
	Spider			Bird			Spider			Bird		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Table retrieval performances (Top-5)												
DTR												
JAR-D	38.8	97	55.4	36.1	90.3	51.6	58.9	94.7	72.5	50.8	82.7	62.8
JAR-F	39.2	98	56	36.5	91.3	52.2	59.6	95.9	73.4	52	84.6	64.3
Diff.	0.4	1.0	0.6	0.4	1.0	0.6	**0.7**	**1.2**	**0.9**	**1.2**	**1.9**	**1.5**
Contriever												
JAR-D	37.9	94.8	54.2	35.9	89.7	51.2	52.3	83.6	64.2	49.8	80.7	61.5
JAR-F	38	94.9	54.2	36.6	91.6	52.3	53	85.1	65.2	52	84.5	64.3
Diff.	0.1	0.1	0.0.	0.7	1.9	1.1	**0.7**	**1.5**	**1.0**	**2.2**	**3.8**	**2.8**

Table 4: Our approach is the most effective on more challenging and realistic queries involving more tables.

4.4 Discussion

Re-ranking without table-table relevance. As shown in Table 1, performing re-ranking using fine-grained query-table relevance (JAR-D) already outperforms the baseline approaches (DTR and Contriever), demonstrating the effectiveness of fine-grained query-table relevance. Further considering table-table relevance can further enhance retrieval performances as JAR-F generally outperforms JAR-D. This suggests that join compatibility is helpful. We conclude that both fine-grained and table-table relevance are essential to better retrieval performances.

Full re-ranking with gold key foreign-key constraints. Table 1 and Table 2 suggest that complementing full re-ranking with the gold key foreign-key constraints (JAR-G) leads to better performances compared to full re-ranking (JAR-F). This is because gold join relationships can generate higher-quality compatibility scores and our re-ranking mechanism can effectively leverage better scores to generate better outputs.

Retrieval performances under different datasets and numbers of tables. Table 4 shows that table-table relevance contributes (as measured by the difference between the retrieval performances of JAR-F and JAR-D) more to retrieval performances for (1) queries involving more tables and (2) Bird compared to Spider. Since our focus is to improve table-related downstream tasks involving multiple

tables, the fact that table-table relevance provides the most benefits with a higher number of tables should be expected. Moreover, we find that queries in the Bird dataset are more challenging and realistic because they involve more bridging tables (e.g., the *Disp* table as mentioned in Figure 1) that are less directly relevant to the user question. They also contain more confusing tables (tables with similar schema, such as *Client* (`client_id`) and *Customers* (`CustomerID`)). These features resemble what exists in real life and table-table relevance can address these more challenging issues, demonstrating its effectiveness.

5 Related Work

One line of related work is text-to-SQL (Zhong et al., 2017; Yu et al., 2018b). Thanks to recent advances in LLM, SQL queries (including tables) corresponding to a natural language question or query can be generated through prompting engineering (Nan et al., 2023; Liu et al., 2023) or fine-tuning (Gao et al., 2024; Wang et al., 2024) with LLMs. Although, in this paper, we evaluate our method on the aggregated version of text-to-SQL datasets, we target a general multi-table retrieval problem that requires the consideration of multiple (joinable) tables. In other words, we do not limit our work to specific downstream tasks (e.g., text-to-SQL for a given database).

To the best of our knowledge, we are the first work to propose the problem of join-aware multi-table retrieval for complex questions, and our paper encourages considering table relationships for retrieval during inference time. One related direction from the literature is fusion retrieval (Chen et al., 2020a). The basic idea of fusion retrieval is based on an "early fusion" strategy to group relevant heterogeneous data (including text and tables) before retrieval. The early fusion process aims to fuse a table segment and relevant passages into a group. It is implemented by linking entities mentioned in a table segment to the appropriate passages, similar to traditional document expansion based on individual entities. Although this method can also capture the relationship between table and text, it cannot capture key foreign-key relationships among multiple tables. Therefore, when answering aggregation questions, false-positive tables are very likely to be introduced due to the expansion through individual entities. Furthermore, tables do not have to be connected by columns of entities, and the foreign key column can be just as simple as an id column. Last but not least, real questions may require more than 2 tables to answer, and therefore bridging tables may be needed to connect tables. Sometimes, the bridging tables can also cross multiple hops. In this case, tables introduced by an "early fusion" strategy will grow exponentially, which may not help the downstream LLM to answer the question correctly. In this work, we also aim to narrow down the number of correlated tables in a limited ranked list, so that they can best help the LLM to answer the question correctly.

6 Conclusion

Table retrieval is critical for open-domain question answering, fact-checking, and, more generally, retrieval-augmented generation that relies on table information. However, existing work primarily focuses on single-table retrieval without considering real-life situations where tables are typically normalized. Thus, multiple tables need to be joined to produce sufficient information. Works that consider multi-table retrieval only consider query-table relevance. In this paper, we propose a re-ranking method based on mixed-integer programming to select the best set of tables based on joint decisions about relevance and compatibility. Our experiments demonstrated that our approach outperforms baselines in standard retrieval performance and end-to-end evaluation on the question-answering downstream task.

7 Limitations

The primary purpose of this paper is to remind the community that table retrieval is still not a solved problem. For complex queries, retrieving multiple (joinable) tables may be required. The proposed MIP-based reranking mechanism in this paper takes a first step to consider join relationships among tables during the inference time. Though MIP may have scalability issues and can be sensitive to the problem data, it provides a flexible framework to model constraints and objectives.

Furthermore, key-foreign-key relationships may not be the only connection among multiple tables for questions in real-life scenarios. Some questions may ask about values of the same column across multiple tables.

We believe exploring a more scalable solution, as well as considering different types of connections among tables in the corpus, would be interesting

future work.

Acknowledgments

We thank Mike Cafarella and Çağatay Demiralp for their constructive feedback and discussion on this project. We thank the MIT SuperCloud and Lincoln Laboratory Supercomputing Center for providing computing resources that have contributed to the research results reported in this paper. We gratefully acknowledge the support of the DARPA ASKEM project (Award No. HR00112220042), ONR Contract N00014-23-1-2365, and the Croucher Scholarship.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Wang, and William W Cohen. 2020a. Open question answering over tables and text. *arXiv preprint arXiv:2010.10439*.

Zhiyu Chen, Mohamed Trabelsi, Jeff Heflin, Yinan Xu, and Brian D Davison. 2020b. Table search using a deep contextualized language model. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, pages 589–598.

Tianji Cong, James Gale, Jason Frantz, HV Jagadish, and Çağatay Demiralp. 2022. Warpgate: A semantic join discovery system for cloud data warehouse. *arXiv preprint arXiv:2212.14155*.

Shimon Even and R Endre Tarjan. 1975. Network flow and testing graph connectivity. *SIAM journal on computing*, 4(4):507–518.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Jonathan Herzig, Thomas Müller, Syrine Krichene, and Julian Eisenschlos. 2021. Open domain question answering over tables via dense retrieval. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 512–519, Online. Association for Computational Linguistics.

Junjie Huang, Wanjun Zhong, Qian Liu, Ming Gong, Daxin Jiang, and Nan Duan. 2022. Mixed-modality representation learning and pre-training for joint table-and-text retrieval in openqa. *arXiv preprint arXiv:2210.05197*.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Unsupervised dense information retrieval with contrastive learning. *Trans. Mach. Learn. Res.*, 2022.

Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. KaggleDBQA: Realistic evaluation of text-to-SQL parsers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2261–2273, Online. Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *ArXiv*, abs/2005.11401.

Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, 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. 2023a. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls.

Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, et al. 2023b. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *arXiv preprint arXiv:2305.03111*.

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

Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. 2020. On faithfulness and factuality in abstractive summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1906–1919, Online. Association for Computational Linguistics.

Linyong Nan, Yilun Zhao, Weijin Zou, Narutatsu Ri, Jaesung Tae, Ellen Zhang, Arman Cohan, and Dragomir Radev. 2023. Enhancing few-shot text-to-sql capabilities of large language models: A study on prompt design strategies.

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

Feifei Pan, Mustafa Canim, Michael Glass, Alfio Gliozzo, and James Hendler. 2022. End-to-end table question answering via retrieval-augmented generation. *arXiv preprint arXiv:2203.16714*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Michael Sejr Schlichtkrull, Vladimir Karpukhin, Barlas Oguz, Mike Lewis, Wen-tau Yih, and Sebastian Riedel. 2021. Joint verification and reranking for open fact checking over tables. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6787–6799, Online. Association for Computational Linguistics.

Tianshu Wang, Hongyu Lin, Xianpei Han, Le Sun, Xiaoyang Chen, Hao Wang, and Zhenyu Zeng. 2024. Dbcopilot: Scaling natural language querying to massive databases.

Zhiruo Wang, Zhengbao Jiang, Eric Nyberg, and Graham Neubig. 2022. Table retrieval may not necessitate table-specific model design. In *Proceedings of the Workshop on Structured and Unstructured Knowledge Integration (SUKI)*, pages 36–46, Seattle, USA. Association for Computational Linguistics.

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. 2018a. 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.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018b. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.

Wenhao Yu, Chenguang Zhu, Zaitang Li, Zhiting Hu, Qingyun Wang, Heng Ji, and Meng Jiang. 2022. A survey of knowledge-enhanced text generation. *ACM Computing Surveys*, 54(11s):1–38.

Yi Zhang and Zachary G. Ives. 2020. Finding related tables in data lakes for interactive data science. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 1951–1966, New York, NY, USA. Association for Computing Machinery.

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

Chunting Zhou, Graham Neubig, Jiatao Gu, Mona Diab, Francisco Guzmán, Luke Zettlemoyer, and Marjan Ghazvininejad. 2021. Detecting hallucinated content in conditional neural sequence generation. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1393–1404, Online. Association for Computational Linguistics.

Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. Josie: Overlap set similarity search for finding joinable tables in data lakes. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 847–864, New York, NY, USA. Association for Computing Machinery.

A Prompts

A.1 Decomposition

We use the following prompt to ask GPT-3.5 Turbo to decompose a question into (concept, attribute) pairs

```
I'm going to ask you a question. I want you to decompose it into a series of non-composite noun concepts and attributes. If you are uncertain about concepts, only output attributes. You should wrap each concept and attribute in <sub_c></sub_c> tags. Once you have all the concepts you need to cover the question, output <FIN></FIN> tags.
Let's go through some examples together.
Question: For movies with the keyword of "civil war", calculate the average revenue generated by these movies.

Answer:
<sub_c>movies:keyword</sub_c>
<sub_c>movies:revenue</sub_c>
<FIN></FIN>

Question: How many customers have a credit limit of not more than 100,000 and which customer made the highest total payment amount for the year 2004?

Answer:
<sub_c>customers:credit limit</sub_c>
<sub_c>customers:payment amount</sub_c>
<sub_c>year</sub_c>
<FIN></FIN>
```

```
Question: What is the aircraft name for the flight with number 99?

Answer:
<sub_c>aircraft:name</sub_c>
<sub_c>flight:number</sub_c>
<FIN></FIN>

Question: On which day has it neither been foggy nor rained in the zip code of 94107?

Answer:
<sub_c>zip code</sub_c>
<sub_c>weather</sub_c>
<FIN></FIN>

Question: What is the id of the trip that started from the station with the highest dock count?

Answer:
<sub_c>trip:id</sub_c>
<sub_c>station:dock count</sub_c>
<FIN></FIN>

Question: ...

Answer:
```

A.2 SQL generation

We use the following prompt to ask GPT 3.5 Turbo 1106 to generate SQLs:

```
// TASK INSTRUCTION
Generate SQL given the question, tables, and external knowledge to answer the question correctly. First, identify tables with relevant columns. Then, join these tables using only columns in the tables. Finally, decide which columns to return in the SQL to answer the original question. When returning columns, please specify the tables associated with it to prevent ambiguity. Think step by step.

// 1-SHOT PSEUDO EXAMPLE
CREATE TABLE singer(
  singer_id TEXT: <instance 1>, ...,
  nation TEXT: <instance 1>, ...,
  sname TEXT: <instance 1>, ...,
  dname TEXT: <instance 1>, ...,
  cname TEXT: <instance 1>, ...,
  age INTEGER: <instance 1>, ...,
  year INTEGER: <instance 1>, ...,
  birth_year INTEGER: <instance 1>, ...,
  salary REAL: <instance 1>, ...,
  city TEXT: <instance 1>, ...,
  phone_number INTEGER: <instance 1>, ...,
  tax REAL: <instance 1>, ...)

External Knowledge: age = year - birth_year
Question: How many singers in USA who is older than 27?
Answer: SELECT COUNT(*) FROM singer WHERE year - birth_year > 27;

// NEW INPUT
CREATE TABLE <table1>(
  <Column1> <Type1>: <instance 1>, ...,
  <Column2> <Type2>: <instance 1>, ...)
...


External knowledge: ...
Question: ...
Answer:
```

B MIP formulation

B.1 Coverage quality

$$\alpha \sum_{q} d_q \tag{7}$$

$$d_q \in \{0, 1\} \tag{8}$$

(integral constraints)

$$d_q \leq \sum_{i,k} d_{qik}, \forall q \tag{9}$$

(A sub-query is covered if it is covered by at least one column)

$$\sum_{q,i,k} d_{qik} \leq |Q| \tag{10}$$

(total number of question-table connections do not exceed the total number of sub-queries)

B.2 Connectedness

We introduce these additional variables: s_i denotes the amount of flow from source to table i; t_i denotes the amount of flow from table i to sink; e_i denotes

whether or not to give an initial flow of k to table i; f_{ij}^{kl} denotes amount of flow from column k of table i to column l of table j.

$$e_i \in \{0,1\}, 0 \leq t_i \leq 1, s_i \geq 0, f_{ij}^{kl} \geq 0 \quad (11)$$

(variables constraints)

$$s_i + \sum_{j,k,l} f_{ji}^{lk} = \sum_{j,k,l} f_{ij}^{kl} + t_i, \forall i \quad (12)$$

(flow balance constraint)

$$\sum_i t_i = K \quad (13)$$

(the amount of flow into the sink should be K)

$$\sum_i e_i = 1, e_i \leq M b_i, s_i = K e_i, \forall i \quad (14)$$

(select one chosen table to give all K units of flow)

$$\frac{1}{k} f_{ij}^{kl} \leq c_{ij}^{kl} \leq M f_{ij}^{kl} \quad (15)$$

(linking constraint between flow and column selection)

C Dataset processing

We remove questions if their corresponding gold SQL expressions involve table joins that generate incorrect instances. This typically happens when a non-unique foreign key of a table is joined with another non-unique foreign key of another table. We illustrate one example from each dataset:

Spider Consider the following two tables:

AREA_CODE_STATE (area_code, state)

VOTES (vote_id, phone_number, state, ...)

To obtain information about the vote and the area code of each vote, these two tables are joined using "state". However, a state can have multiple areas and thus its value is not unique in *AREA_CODE_STATE*. Therefore, joining over the state column creates incorrect instances where a vote in, for example, Area 1 of a state is connected to Area 2 of the same state simply because they are in the same state.

Bird Consider the following two tables:

Account (account_id, district_id, frequency, date)

Client (client_id, gender, birth date, district_id)

To obtain information about client and their accounts, these two tables are joined using district_id. However, this join creates incorrect instances, similar to the case above in Spider. For example, if two clients live in the same district and opened their accounts in the same district, such a join creates instances involving each client with the other client's account which is incorrect.