JOINT – Join Optimization and Inference via Network Traversal

Szu-Yun Ko
b12705066@ntu.edu.tw
Department of Information Management
National Taiwan University

Bo-Cian Chang
b09508009@ntu.edu.tw
Department of Economics
National Taiwan University

Alan Shu-Luen Chang
b11501010@ntu.edu.tw
Department of Civil Engineering
National Taiwan University

Ethan Chen
r13921091@ntu.edu.tw
Department of Electrical Engineering
National Taiwan University

Abstract

Traditional relational databases require users to manually specify join keys and assume exact matches between column names and values. In practice, this limits joinability across fragmented or inconsistently named tables. We propose a fuzzy join framework that automatically identifies joinable column pairs and traverses indirect join paths across multiple databases. Our method combines column name similarity with row-level fuzzy value overlap, computes edge weights using negative log-transformed Jaccard scores, and performs join path discovery via graph traversal. Experiments on synthetic healthcare-style databases demonstrate the system's ability to recover valid joins despite fuzzified column names and partial value mismatches. This research has direct applications in data integration.

Keywords

Intermediate Join, Multi-hop Join, Entity Matching, Fuzzy Matching

ACM Reference Format:
Szu-Yun Ko, Bo-Cian Chang, Alan Shu-Luen Chang, and Ethan Chen. 2025. JOINT – Join Optimization and Inference via Network Traversal. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

In traditional relational database systems, join operations require users to explicitly specify matching columns, typically relying on exact name and value equivalence. Even `NATURAL JOIN`, which infers join keys automatically, is limited to columns with identical names and compatible data types, and still assumes exact value matches. These rigid constraints often fall short in real-world scenarios, where datasets may exhibit inconsistent naming conventions, abbreviations, or contain typos and semantic variations in both schema and content.

To address these limitations, we aim to enable a more flexible and realistic join discovery framework that can handle fuzzy column names and record values. In addition, we intend to enable the system to discover indirect join paths through multiple tables.

This capability is particularly valuable in applications such as data integration, exploratory data analysis, and automated schema mapping, where datasets from different sources often lack standardized structures.

By relaxing the reliance on exact matches, we move toward a more robust interpretation of table relationships in relational systems.

2 Related Works

2.1 Fuzzy Row-Level Matching

Fuzzy joins, or similarity joins, have been widely studied in the context of matching tuples with non-identical values. Traditional approaches typically require users to manually specify a join function and a similarity threshold, which can be complex and error-prone. Auto-FuzzyJoin [2] addresses this by automatically generating high-quality fuzzy join programs without labeled training data. It learns appropriate distance functions and thresholds for a given pair of columns, achieving high precision and recall. However, Auto-FuzzyJoin assumes that the matching columns are already known and focuses only on single-step joins between two tables. Also, Auto-FuzzyJoin heavily depends on the assumption that a reference table with nearly no duplicate values exist in the DBMS, which may not be realistic for all DBMS designs.

2.2 Entity Matching with Language Models

Recent work has explored using large language models (LLMs) for entity matching across datasets. Peeters et al. [3] evaluate the effectiveness of generative LLMs, such as GPT-4, Gemini and LLaMA, in both zero-shot and few-shot scenarios. They demonstrate that LLMs can perform competitively with traditional pre-trained models like RoBERTa, even without task-specific training data. However, these models operate at the tuple level and require prompts to be manually constructed. The system is also highly sensitive to prompt wording, and different prompts are required for different context and datasets, there is currently no general prompt that may suit all scenarios. Also, this system does not generalize to multi-hop joins or automatically infer which columns or paths to consider.

2.3 Our Contribution

As we identified some gaps in the current literature, we attempt to bridge these gaps with our JOINT system. Our system integrates fuzzy column name matching with row-level similarity comparisons to automatically identify joinable column pairs. Furthermore, we extend this capability to discover multi-table join paths using a weighted graph traversal approach. Our method relaxes the need for exact matches at both the column and row levels, supporting more robust join discovery in real-world, messy datasets.

3 Research Objectives

(1) **Fuzzily matching column names** to identify semantically similar attributes across tables, even when their names differ (e.g., `'doctor_name'` vs. `'assigned_doctor_name'`).
(2) **Identify fuzzy record matches** across databases despite discrepancies in notation (e.g., typos, abbreviations, different word ordering...etc.).
(3) **Discovering indirect join paths** by traversing through intermediate tables based on fuzzy match results, allowing us to infer joinability even when no direct foreign key relationship exists.

4 Methods

To achieve the research objectives, we propose a multi-stage methodology that integrates metadata reflection, column-level and row-level fuzzy similarity scoring, and graph-based join path construction. The overall process is composed of five main stages:

4.1 Metadata Extraction and Identification of Foreign Key Edges

Purpose: Establish the baseline join graph using foreign key (FK) constraints between tables within individual DBMS.
Implementation: We use `SQLAlchemy` to extract FK constraints for tables in the same DBMS. Foreign key relations between two tables will be converted into FK edges between the respective tables in the join graph.

4.2 Fuzzy Column Matching Across DBMS

Purpose: Detect column pairs with high semantic similarity despite inconsistent naming across databases (e.g., `'drug_name'` vs `'medication_name'`).
Implementation: Candidate column pairs are scored using a weighted combination of three metrics:

(1) Name similarity using `SequenceMatcher`.
(2) Token overlap ratio, which is order-insensitive.
(3) Semantic similarity from transformer embeddings (`all-MiniLM-L6-v2`) using cosine distance [4].

Each score is normalized to [0, 1] and aggregated using:

$$\text{total\_score} = \alpha \cdot \text{name\_sim} + \beta \cdot \text{semantic\_sim} + \gamma \cdot \text{token\_overlap}$$

where $\alpha + \beta + \gamma = 1$. Only pairs exceeding a threshold (e.g., 0.6) are retained. All parameters and threshold can be defined by the user.

4.3 Row-Level Fuzzy Validation

Purpose: Confirm column pair joinability by measuring actual row-level value similarity.
Implementation: We compute fuzzy similarity scores between value lists using `rapidfuzz`'s `token_sort_ratio` [1]. The row-wise similarity matrix is reduced by taking the maximum score per row and then averaged:

$$\text{value\_score} = \frac{1}{n}\sum_{i=1}^{n} \max_j \text{sim}(x_i, y_j)$$

Pairs with average scores above a second threshold (e.g., 0.5) are appended to the join graph as fuzzy edges.

4.4 Weight Assignment to FK and Fuzzy Edges in Join Graph

Column overlap s_i denotes the record overlap between the i^{th} and $(i+1)^{th}$ table in the join sequence, s_i is computed as:

$$s_i = \text{Jaccard}(C_i, C_{i+1}) = \frac{|C_i \cap C_{i+1}|}{|C_i \cup C_{i+1}|}$$

We aim to maximize cumulative overlap across multiple tables, since overlap is a percentage measure, cumulative overlap is computed as a product of each pair-wise overlap in the join sequence:

$$\max \prod_{i=1}^{n} s_i$$

therefore we define the edge weight as:

$$\text{weight}_i = -\log(s_i + \epsilon)$$

The pathfinding algorithm will aim to minimize cumulative edge weights (shortest path), which is equivalent to maximizing cumulative overlap across multiple tables:

$$\min \sum_{i=1}^{n} w_i = \min -\log(\prod_{i=1}^{n}(s_i + \epsilon))$$

A higher overlap implies stronger joinability and thus a lower traversal weight. Edges are added to a `networkx` graph with associated columns and scores. This edge weight definition ensures additive cost under Dijkstra traversal.

4.5 Join Graph Construction and Path Discovery

Purpose: Form a heterogeneous join graph and identify optimal multi-hop join paths for analytics.
Implementation: Nodes in the graph represent tables across all DBMSs. Edges include FK joins and validated fuzzy joins. Each edge carries metadata: join type, columns, similarity, and traversal weight. We apply Dijkstra's algorithm to compute shortest paths between any two target tables. The effective retained data can be approximated as:

$$\text{retained\_percentage} \approx 2^{-\text{total\_weight}}$$

This favors paths with stronger join overlaps and higher semantic integrity.

5 Experimentation

5.1 Synthetic Database Design

We construct a synthetic multi-database environment consisting of the following four datasets:

- **hospital_db** – Contain five tables: 'Patients', 'Clinics', 'Prescriptions', 'Doctors' and 'Appointments'.
- **insurance_db** – Contain three tables: 'Insurance_Providers', 'Insured_Patients' and 'Claims'.
- **pharmacy_db** – Contain three tables: 'Pharmacies', 'Drugs', and 'Pharmacy_Orders'.
- **public_info_db** – Contain three tables: 'Citizen_Registry', 'Hospital_Survey', and 'Drug_Watchlist'.

The generator functions, detailed tables and records of hospital_db, insurance_db and pharmacy_db can be found in the attached code '`healthcare_data.ipynb`'. Details of public_info_db can be found in '`add_fuzzy_db.ipynb`'.

Each database simulates independently maintained systems with partial schema overlap. We currently assume that all appropriate foreign key constraints are properly configured.

5.2 Fuzzification of Records

To simulate real-world noise, we apply controlled fuzzification to a subset of the records. The transformations include:

1. **Character removal:** Randomly delete 1–2 characters from a name string to simulate typos.
2. **Name reordering:** Reverse name parts, e.g., "John Smith" becomes "Smith John".
3. **Synonym injection:** Replace drug names using a predefined mapping (e.g., "Amoxicillin" $\longrightarrow$ "Amoksillin").
4. **Label variation:** Append suffixes like "Clinic" or "Hospital" to simulate entity naming inconsistency.

These utilities are implemented as randomized functions (see '`add_fuzzy_db.ipynb`') and applied selectively during data generation.

5.3 Join Graph Construction

We build a join graph where:

- Each node represents a table.
- Edges represent joinable column pairs, scored by fuzzy matching (e.g., token overlap, embedding similarity).
- Edge weights are computed as mentioned in section 4.4.

The join graph built upon our synthetic data warehouse consisting of four DBMS is as shown in Figure 1:

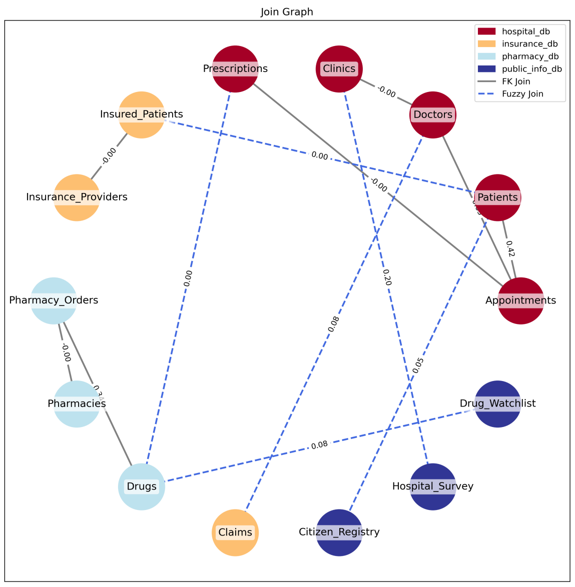

Figure 1: Join graph constructed across all databases. Nodes represent tables, colored by database. Edges represent joinable column pairs, with solid gray lines for foreign key joins and dashed blue lines for fuzzy joins.

Example Scenario

In this scenario, the user wants to determine the government evaluation score (from `Hospital_Survey` in `public_info_db`) of the clinic that each doctor (from `Doctors` in `hospital_db`) is affiliated with.

Join Path Selection. It first performs a foreign key join between the `Doctors` and `Clinics` tables using the `clinic_id` field. Then, it performs a fuzzy join between the `clinic_name` and `hospital_name` columns, with an overlap score of 0.81. The output includes each doctor's name, their affiliated clinic, the clinic's satisfaction score, and the fuzzy match score for the final join step.

Join Output. Since current RDBMS doesn't support fuzzy record level join directly in databases, we output the joined results as a `Pandas Dataframe`. This is a function we are still working to refine. We aim to make the dataframe manipulation more stable for joins with many steps, and also to reduce the memory consumption during row manipulation.

Clinic ID	**Doctor ID**	**Doctor Name**	**Clinic Name**	**Satisfaction Score**	**Fuzzy Score**
e2f9...	dd06...	Valerie Williams	Rodriguez–Johnson	4.96	1.000
2ea5...	3a60...	Thomas Fleming	Fox–Medina	4.48	1.000
8762...	1b8e...	Patrick Sheppard	Floyd–Hunt	4.12	1.000
a45f...	47d7...	Alan Mercer	Reed, Blair and Allen	4.66	0.810
a45f...	e8e1...	Stephanie Jones	Reed, Blair and Allen	4.66	0.810

Table 1: Join result showing each doctor's clinic and the corresponding government satisfaction score, along with the fuzzy match score. Column names are reformatted for improved readability.

6 Expected Contributions

6.1 Future Work: Many-to-Many Intermediate Joins on a Fuzzy Basis

While our current system effectively identifies indirect join paths through intermediate tables, it is currently limited to one-to-one mappings. In future work, we plan to extend the system to handle many-to-many intermediate joins, which are common in real-world data scenarios such as mapping a set of columns (e.g., `ZIP`, `City`, `Address`) to another column (e.g., `Full_Address`). This will require aggregations across multiple branches.

6.2 Contributions

This research contributes toward making relational databases more flexible and realistic in the face of schema inconsistencies and real-world data noise. By enabling automatic fuzzy joins across multiple tables—even without explicit foreign key constraints—we help lower the barrier for:

- **Cross-domain data integration** in fragmented systems (e.g., healthcare, finance, government).
- **Exploratory data analysis** without prior knowledge of schema structure.

Our approach complements existing entity matching systems by extending join reasoning to the multi-table level, enhancing automation in join path discovery.

7 Conclusions

We present a fuzzy join discovery framework that combines column-level schema matching with row-level similarity scoring to infer joinable relationships across multiple tables and databases. By modeling the problem as a weighted join graph traversal, we allow for the discovery of indirect join paths through intermediate tables.

Our experiments on synthetic yet realistic databases demonstrate the system's ability to recover meaningful joins and trace useful relationships in noisy environments. In the future, we aim to experiment the system on larger real-world databases, and expand the system to support many-to-many joins.

References

[1] Max Bachmann. 2023. RapidFuzz: Fast fuzzy string matching in Python. https://github.com/rapidfuzz/RapidFuzz. Version 3.4.0, Accessed: 2025-06-11.

[2] Peng Li, Xiang Cheng, Xu Chu, Yeye He, and Surajit Chaudhuri. 2021. Auto-FuzzyJoin: Auto-Program Fuzzy Similarity Joins Without Labeled Examples. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD).*

[3] Ralph Peeters, Aaron Steiner, and Christian Bizer. 2025. Entity Matching using Large Language Models. In *Proceedings of the 2025 International Conference on Extending Database Technology (EDBT).*

[4] Nils Reimers and Iryna Gurevych. 2021. Sentence-BERT: all-MiniLM-L6-v2. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2. Accessed: 2025-06-11.