

# SEMDISC: An End-to-End Query-by-Example Semantic Join Discovery System

Mir Mahathir Mohammad  
University of Utah  
Salt Lake City, UT, USA  
mahathir.mohammad@utah.edu

El Kindi Rezig  
University of Utah  
Salt Lake City, UT, USA  
elkindi.rezig@utah.edu

## Abstract

Discovering joinable tables in data lakes is essential for constructing datasets, yet tables often represent equivalent concepts using different syntactic forms, and meaningful join paths may traverse intermediate tables with no direct overlap to the user’s query. We demonstrate SEMDISC, an interactive system that discovers join paths in schema-less data lakes through a query-by-example interface. Given a table with example values and optional semantic annotations, SEMDISC retrieves top-K join paths unifying traditional equi-joins and semantic joins. The system leverages locality-sensitive hashing to approximate semantic joinability at scale, employs large language models to infer column semantic types, and introduces an index for fast retrieval of high-quality join paths. The demonstration will allow SIGMOD attendees to upload and preprocess data lakes, explore the join graph and indexed paths, submit query tables, and interactively examine how retrieved paths satisfy their examples across multiple benchmark datasets. A companion video is available at [2].

## CCS Concepts

• Information systems → Information integration; Mediators and data integration; Query representation.

## Keywords

Data Discovery, Join Discovery, Semantic Join, Query-by-Example, Join Path, Data Lake, Semantic Type

## ACM Reference Format:

Mir Mahathir Mohammad and El Kindi Rezig. 2026. SEMDISC: An End-to-End Query-by-Example Semantic Join Discovery System. In *Companion of the International Conference on Management of Data (SIGMOD Companion '26)*, May 31–June 05, 2026, Bengaluru, India. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3788853.3801604>

## 1 Introduction

Tabular data lakes [1, 6] pose significant challenges for data discovery due to the absence of global schemas and Primary Key–Foreign Key relationships, making it difficult for users to determine *which* tables are joinable and *how* to join them to build a dataset of interest. While join discovery systems [8, 10] have emerged to detect joinable tables through overlapping column values or semantic

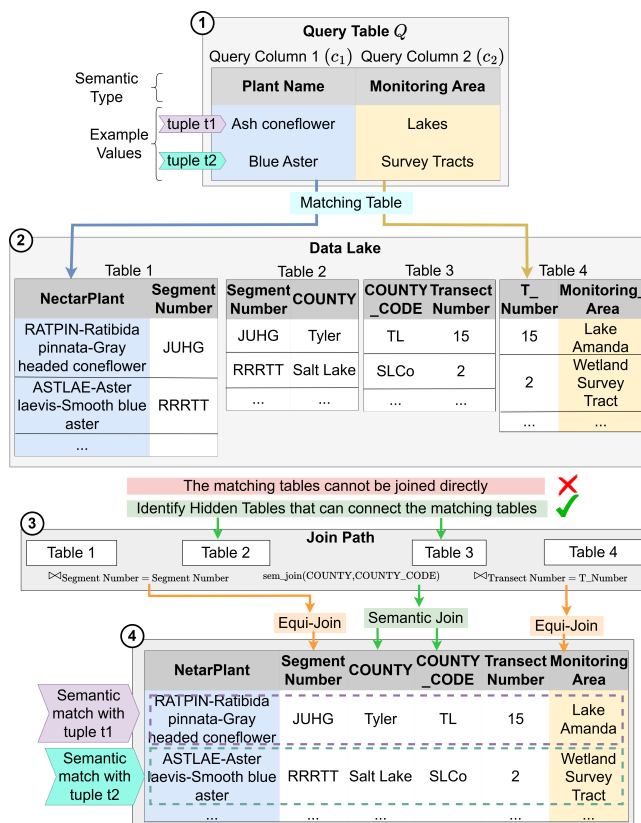


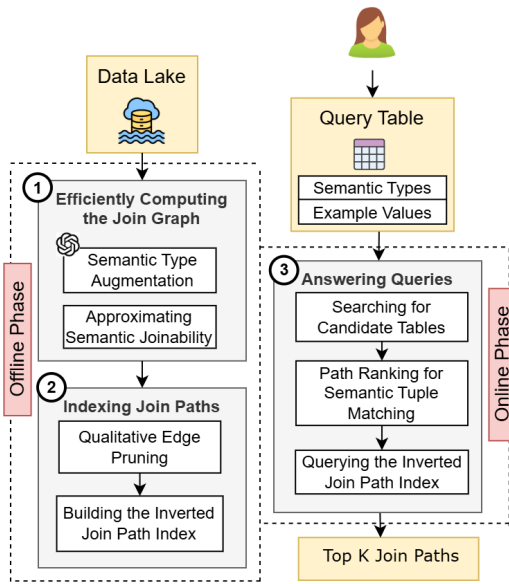
Figure 1: Example of finding join paths from a query table.

similarity [8], they fall short in query-by-example data discovery scenarios, as demonstrated in the following example:

**Example:** Consider a data scientist querying a data lake from data.gov [1] (Figure 1 (2)) with example values and column headers (Figure 1 (1)). Constructing the desired dataset requires addressing three key challenges: (1) identifying hybrid join paths that contain both semantic joins (for columns representing the same concept differently, e.g., “SLCo” vs. “Salt Lake” county) and equi-joins; (2) discovering join paths with hidden tables (Figure 1 (3))- intermediate tables that don’t overlap with the values in the query table but are essential for linking relevant tables; and (3) ensuring semantic tuple matching, where the results include tuples that are semantically similar to the example tuples, in addition to producing other tuples that were not specified in the query table (Figure 1 (4)).

We present a demonstration of our system SEMDISC [14], a query-by-example join discovery system that addresses the above challenges through: (1) scalable hybrid join path discovery via data





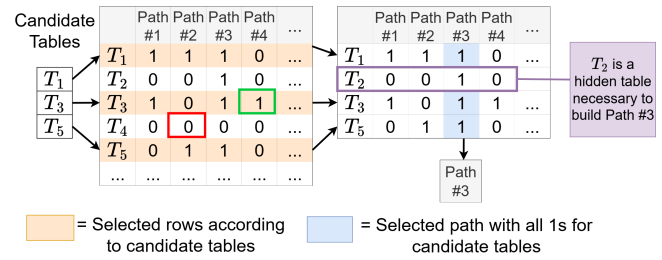
**Figure 2: Overview of SEMDISC’s architecture. The offline phase constructs and indexes a join graph over the data lake, while an online phase efficiently retrieves and ranks the top-K join paths for a given query table.**

sketches to encode semantic and equi-joins in a join graph, (2) an Inverted Join Path Index for efficient retrieval of paths including hidden tables and (3) heuristics to ensure joined results contain tuples semantically matching the user’s examples.

*Related Work.* There have been several data discovery systems that were proposed in the past few years [6]. We focus the discussion on those that support join discovery. DeepJoin [9] and WarpGate [5] focus on semantic column matching using embeddings but do not construct hybrid join paths with tuple matching for an example query table. Ver [11] and DICE [16] are query-by-example systems that discover join paths using MinHash-based Jaccard similarity; however, they are restricted to equi-joins and do not support semantic joins. Snoopy [12] learns embeddings that capture semantic, rather than syntactic, joinability and retrieves the top-k semantically joinable columns. SEMDISC extends this idea by enabling tuple-level semantic matching and hidden-table discovery.

## 2 System Overview

Figure 2 illustrates the overall architecture of SEMDISC. SEMDISC operates in two phases: **Offline phase:** In this phase, SEMDISC (1) computes a join graph (Figure 2 ①) by ingesting a data lake, using an LLM to annotate semantic types of columns, computing semantic joinability between columns and building a join graph with semantic and equi-join edges; (2) indexes join paths (Figure 2 ②) by applying pruning to retain the best join edges, and constructing an Inverted Join Path Index. **Online phase (Figure 2 ③):** In this phase, users submit a query table with example values, SEMDISC matches these against data lake tables using semantic types, applies heuristics to match query tuples to join paths, and retrieves the top-k join paths from the index.



**Figure 3: Retrieving join paths by querying the Join Path Index**

### 2.1 Efficiently Computing the Join Graph

The central data structure in SEMDISC is the *join graph*, where nodes represent tables and edges encode potential join relationships between column pairs. Each edge is weighted by the *strength* of joinability, measured through value overlap.

**2.1.1 Semantic Type Augmentation.** To prevent false joins between columns with overlapping values but different semantics (e.g., ‘student\_id’ vs. ‘part\_id’), SEMDISC annotates all columns with semantic types using an LLM. The system prompts GPT-4o [3] with sample values and co-occurring columns for context, returning semantic types in structured JSON. Each prompt includes task instructions, the expected output format, and input columns identified by their header and up to 20 frequent values of all columns in the table. For prompt details, refer to [14].

**2.1.2 Approximate Semantic Joinability:** Since exhaustive pairwise column comparison is expensive, SEMDISC uses compact data sketches to summarize each column’s value distribution and semantic characteristics, enabling fast pruning of unlikely join candidates:

**1. SimHash Generation:** Column values are embedded using SBERT [15] and hashed via locality-sensitive hashing (SimHash [14]), so semantically similar values (e.g., ‘Salt Lake County’ and ‘SLCo’) map to the same hash, reducing semantic matching to equality comparison.

**2. MinHash Signatures:** Each column’s SimHash values are summarized into fixed-size MinHash signatures [4] for fast semantic joinability computation using set comparison.

**3. Jaccard Similarity:** MinHash signatures are compared using Jaccard similarity to produce edge weights. Both semantic and equi-join edges share this weight metric, yielding a join graph with weighted edges representing column-pair joinability.

### 2.2 Indexing Join Paths

As we show in [14], join graphs can contain millions of edges, many of which are spurious. SEMDISC mitigates this by pruning low-joinability edges and indexing only join paths that consist of high-joinability edges and are estimated—via cardinality estimation—to produce non-empty results.

**2.2.1 Qualitative Edge Pruning.** For each pair of tables, SEMDISC filters edges in two steps:

- Semantic type similarity:** This step prunes edges between columns whose semantic types are not sufficiently similar. This is determined by computing the cosine similarity between their

semantic type embedding vectors and discarding edges that fall below a predefined threshold.

- **Value joinability:** In this step, SEMDISC only keeps edges meeting a user-defined joinability threshold, then retains only the single top-ranked edge having the highest weight per table pair.

**2.2.2 Building the Inverted Join Path Index.** Given a set of candidate tables that have value overlap with the query table’s columns (the retrieval process is described in Section 2.3), SEMDISC needs to find all join paths that contain those tables. Rather than a naive linear search, SEMDISC builds an Inverted Join Path Index— a binary matrix where rows represent tables in the data lake, columns represent join paths and a cell value of 1 indicates the table appears in the path; 0 otherwise.

SEMDISC constructs the matrix with  $|\mathcal{D}| \times |\mathcal{P}|$  zeros where  $|\mathcal{P}|$  denotes the count of paths. For each table in each join path, SEMDISC sets the corresponding cell to 1. Figure 3 shows an example of the Inverted Join Path index. To find paths containing tables  $T_1, T_3, T_5$ — SEMDISC first selects the rows for those tables, identifies columns where all selected rows contain 1s and returns the corresponding join paths. As shown in the figure, a matching path (e.g. Path 3) may include additional hidden tables (e.g.  $T_2$ ) that aren’t candidates but are necessary intermediates to complete the join path.

## 2.3 Answering Queries

This section explains how SEMDISC processes user queries in the online phase— specifically, how it efficiently identifies the most relevant join paths and ranks them for a given query table  $Q$ .

**Searching for Candidate Tables.** For each column in a query table  $Q$ , the system finds the most relevant data lake columns as follows: (1) SEMDISC uses a Hierarchical Navigable Small World (HNSW) [13] index—shown to support highly efficient approximate nearest neighbor (ANN) search—over semantic type embeddings to retrieve the top-matching columns. (2) Then SEMDISC filters the results by checking if the query values’ SimHashes are contained within each candidate column’s SimHashes, ensuring semantic similarity of example values. (3) Finally, SEMDISC combines all candidate column lists to produce candidate join paths.

**Path Ranking for Semantic Tuple Matching.** To avoid exhaustively materializing joins, SEMDISC ranks candidate join paths based on their potential to produce valid semantic tuple matches. For each candidate set of columns, the system groups columns by their source tables and evaluates whether query tuples can be consistently matched across multiple columns within the same table. Candidate sets that fail to support such partial tuple matches are pruned early, while those that do are assigned higher scores reflecting their matching potential. The resulting ranked list of candidate sets defines a compact and high-quality join path search space, which is then used to query the Inverted Join Path Index.

**Querying the Inverted Join Path Index.** Given the final set of candidate tables, SEMDISC queries the Inverted Join Path Index to efficiently retrieve valid join paths. For each candidate set, the system identifies compatible join paths shared across the involved tables. The top- $K$  join paths are then selected and materialized for query execution.

## 3 Demonstration Plan

We demonstrate SEMDISC on four data lakes: (1) **U.S. Fish and Wildlife Service (collected from data.gov [1])**, containing 246 tables about ecological records across the U.S.; (2) **Centers for Disease Control and Prevention (collected from data.gov [1])**, containing 467 tables about public health records on disease, mortality, and drug usage; (3) **SchemaPile [7]**, comprising 34.9K tables; and (4) **LakeBench [6]**, a data discovery benchmark containing ground-truth joins between tables.

**Demonstration Outline.** Our demo places participants in the role of a data scientist seeking to construct a dataset by providing a query table. We demonstrate two scenarios:

### 3.1 Scenario 1: Preprocessing the Data Lake

In this scenario, participants dive into the system’s internals, examining how the data lake is processed and how join paths are indexed offline.

① **Uploading and Preprocessing.** Participants upload a data lake containing CSV files through the SEMDISC web interface. System hyperparameters (Figure 4 ①) such as SimHash size, join threshold, MinHash function count, semantic type similarity, and maximum path length are exposed to users, enabling them to configure the index construction. These hyperparameters control how much pruning is done on the edges of the join graph.

② **Exploring Curated Join Paths.** After preprocessing, participants can browse the indexed join paths (Figure 4 ②) through the interface. For each path, the system displays the estimated cardinality, attributes, hop count, and a visual representation where tables appear as nodes and joinable column pairs appear as edges (Figure 4 ③).

③ **Viewing Annotated Semantic Types.** During preprocessing, the LLM-annotated semantic types are persisted, allowing participants to inspect them via the interface (Figure 4 ④).

④ **Filtering Join Paths via Heatmap.** SEMDISC provides a heatmap (Figure 4 ⑤) that organizes join paths into bins by estimated tuple count (y-axis) and node count (x-axis). Each cell represents a bin of join paths satisfying both criteria, with darker shading indicating higher path counts.

⑤ **Visualizing Hybrid Paths.** The system renders joinable edges within each path (Figure 4 ⑥) using distinct colors: orange for semantic joins and blue for equi-joins.

### 3.2 Scenario 2: Answering Queries

In this scenario, participants load a data lake and submit a query table to retrieve the top- $K$  join paths satisfying their requirements:

Ⓐ **Query Input.** SEMDISC accepts queries structured as a query table (Figure 4 Ⓐ), where participants will be able to specify example values and/or semantic types of columns to generate a dataset of interest.

Ⓑ **Top- $K$  Join Path Retrieval.** The system retrieves the top- $K$  join paths satisfying the query table. Participants can examine each path’s structure (Figure 4 Ⓑ), view a tabular preview of that path, and download the materialized result of any join path as a CSV file (Figure 4 Ⓕ).

