Postgres Extension: Table Union Search

Pratik Pokharel

Department of Computer Science and Engineering
University at Buffalo
pratikpo@buffalo.edu

1 Abstract

The exponential growth of data repositories, has increased the necessity for efficient data discovery methods to assist analysts in locating related datasets. This challenge is worsened by two primary factors: (i) users often lack knowledge of the existing datasets within extensive data repositories, and (ii) there is often no cohesive data model to represent the interconnections between heterogeneous datasets from various sources. An important way for addressing dataset discovery needs is: Table Union Search. Existing works in table union search are typically not available as a native feature of database systems, nor as extensions, due to their reliance on language models for evaluating semantic similarity. The dependence on language models hinders integration into extensible database systems like PostgreSQL, as these state-of-the-art models are often implemented in Python, which is not considered a trusted language for writing extensions in PostgreSQL. We propose to combine fuzzy similarity of attribute names with instance-based similarity to develop an extension for PostgreSQL that can identify unionable tables within a collection. This project aims to explore the data discovery process on a smaller scale within an SQL setting- with SQL operators and functions- and assess its feasibility and scalability.

2 Introduction

Data integration is the process of combining data from multiple sources to provide a unified view, enabling better analysis, decision-making, and operational efficiency. As organizations increasingly rely on diverse and distributed data sources, effective data integration becomes crucial for deriving meaningful insights. With large collections of data, analyst often struggle to discover all the data that is relevant to their question. For them to do their work, analysts need to know (i) which datasets could be about their subject and (ii) what the fields in those datasets might mean. Within the realm of data integration, **Schema Matching** attempts to carry out these two tasks in an automated or semi-automated way. More precisely, it involves identifying and establishing correspondences between the schemas of different data sources. Schema matching helps ensure that the data is accurately aligned, allowing for coherent integration. For example, when merging datasets from different databases, schema matching is essential to map columns that semantically represent the same information, such as "Customer_ID" in one database and "ID_Customer" in another.

Another important aspect of data integration is **Table Union Search (TUS)**. This operation aims to find related tables, that can be unioned with a query table based on their column compatibility, even when they exist within vast data repositories. Two tables are considered unionable if their column values are drawn from the same domains. While table union search is similar to schema matching and join discovery, it presents a unique complexity that distinguishes it from them. Schema matching involves taking two schemas as input and producing a mapping between columns that semantically correspond to each other. In this context, table union search can be viewed as an extended version of schema matching. Instead of working with two schemas, table union search operates with a single schema and must find a partially matching schema within a vast dataset, such as data lakes. This adds an extra layer of complexity: beyond the matching itself, table union search must also identify potential matches among a multitude of non-matching options, which is a considerably more challenging task.

Another problem that relates to table union search is join discovery, which focuses on finding tables that can be joined within dataset collections. Although both are search problems, table union search requires the identification of correspondences between groups of columns in two tables, while join discovery focuses on identifying pairs of join keys between two tables. This fundamental difference places a greater emphasis on the quality of embeddings used in table union search,

as a moderate similarity between all column pairs complicates the task of matching unionable pairs.

Table Union Search particularly, is a special case of a wider domain of problem called **"Set Similarity Search"**. In set similarity search, given a query set Q and a collection $\mathcal{L} = \{C_1, C_2, \ldots, C_N\}$ of candidate sets, the goal is to find similar sets, often the top-k, to Q in $\mathcal{L}$. Usually these Sets Q and C_k are a collection of words or texts. For finding similar sets, typically the similarity between each of the elements in these sets are evaluated. Various techniques can be employed for this purpose:

- **Vanilla Overlap**: This method evaluates exact matches between set elements, where the similarity $\text{sim}(q_i, c_j)$ is defined as:

$$\text{sim}(q_i, c_j) = \begin{cases} 1 & \text{if } q_i = c_j \\ 0 & \text{if } q_i \neq c_j \end{cases}$$

- **Fuzzy Overlap**: This technique allows for syntactically similar matches, where similarity scores are in a continuous range:

$$\text{sim}(q_i, c_j) \in [0, 1]$$

  Some common techniques are- cosine/jaccard similarity of text n-grams, edit/Levenshtein distance, etc. For example, pairs like (New York, NewYork) can yield a non-binary similarity score.

- **Semantic Overlap**: This is a generalization that combines both Vanilla and Fuzzy Overlap by allowing semantically similar elements. The similarity function can be expressed as:

$$\text{sim}(q_i, c_j) \in [0, 1]$$

  An example includes pairs like (Big, Large), which are semantically related. Common techniques include Sentence Embeddings, which is similar to word embeddings but for entire sentences or paragraphs. Language Models like Universal Sentence Encoder or Sentence-BERT can be used to generate embeddings that capture semantic meaning.

In **Table Union Search**, given a query set Q and a collection of tables $\mathcal{L}$, where k is less than or equal to the number of tables $\mathcal{L}$ with a positive similarity score α with Q, we need to find a sub-collection $\omega \subseteq \mathcal{L}$ of k distinct tables satisfying the following conditions:

1. $\text{SO}(Q, C) > 0 \quad \forall C \in \omega$, and
2. $\min\{\text{SO}(Q, C) \mid C \in \omega\} \geq \text{SO}(Q, X) \quad \forall X \in \mathcal{L} \setminus \omega$

Where:

$$\text{SO}(Q, C) = \max_{M} \sum_{q_i \in Q} \text{sim}_\alpha(q_i, M(q_i))$$

Here, $\alpha > 0$ (where α is the threshold value for similarity scores between two elements), and $M : Q \rightarrow C$ is a one-to-one matching that matches each $q_i \in Q$ to at most one $M(q_i) \in C$ or none. The function sim() is a similarity function, defined such that sim() $\in [0, 1]$:

$$\text{sim}_\alpha(x, y) = \begin{cases} \text{sim}(x, y) & \text{if sim}(x, y) \geq \alpha \\ 0 & \text{otherwise} \end{cases}$$

3 Project Goal

The objectives of this project are:

1. To generate an efficient embedding for each table attributes, which can assist in getting refined similarity scores
2. To implement an efficient search algorithm with cheap to implement filters

The overarching goal of this project is to explore the extensibility mechanism of a specific database system and how extensions can be developed within that framework. In this context, the feasibility of implementing a table union search as an extension is analyzed. A crucial question is whether this task can be accomplished using SQL, given that such operations are often analyzed using Python libraries. If it is indeed possible to implement table union search in SQL, it is important to consider why such systems are not commonly available. Conversely, if it is not feasible, the limitations that hinder the implementation of this functionality directly within a database system is investigated.

Extensibility is one of PostgreSQL's most powerful and distinctive features. Even the managed postgres database services offered by cloud service providers support extensions written in trusted languages. While table union search may be more applicable in a data-lake environment, where extending systems like Apache Spark might seem more appropriate, several factors influenced the decision to focus on PostgreSQL for this project. Given the time constraints of a semester-long project, my familiarity with PostgreSQL, its robust documentation and community support, SDKs available, along with my limited experience with Spark extensions led me to prioritize PostgreSQL as the safer and more practical choice.

4 Methodology

4.1 Generating Embeddings and Similarity Measures

To find unionable tables, the primary task is to evaluate similarity between individual attributes among the table. When it comes to similarity measures, they are broadly classified into two types- (i) name-based similarity, (ii) isntance-based similarity.

Name-based measures evaluate similarity based on attribute names. Attribute names involve natural languages in most of the cases because schema definition is mostly a manual task, and users often define schema with relevant names. If not, then the description of each attributes are recorded as metadata. For this work, we assume that the attribute names are relevant words or text describing the entity. To evaluate if two attribute names are similar we used one of the methods mentioned in Section 2- Cosine Similarity based on tri-grams.

For example, consider two attribute names: `customer_id` and `id_customer`. If we use **3-grams** (substrings of length 3), the n-grams for each attribute name would be:

- **3-grams for `customer_id`**: {`cus`, `ust`, `sto`, `tom`, `ome`, `mer`, `er_`,`r_i`,`_id`}
- **3-grams for `id_customer`**: {`id_`, `d_c`, `_cu`, `cus`, `ust`, `sto`, `tom`, `ome`, `mer`}

Now we combine the unique trigrams from both strings into a set:

{`cus`, `ust`, `sto`, `tom`, `ome`, `mer`, `er_`,`r_i`,`_id`}, `id_`, `d_c`, `_cu`}

Then we create Vectors by assigning frequency counts of trigrams for each string:

Trigram	`customer_id`	`id_customer`
"cus"	1	1
"ust"	1	1
"sto"	1	1
"tom"	1	1
"ome"	1	1
"mer"	1	1
"er_"	1	0
"r_i"	1	0
"_id"	1	0
"id_"	0	1
"d_c"	0	1
"_cu"	0	1

Vectors:
`customer_id`: [1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0]
`id_customer`: [1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1]

Now we calculate Dot Product

$$\text{Dot Product} = 6$$

Magnitudes

$$\text{Magnitude of } \texttt{customer\_id} = \sqrt{9}$$

$$\text{Magnitude of } \texttt{id\_customer} = \sqrt{9}$$

$$\text{Cosine Similarity} = \frac{\text{Dot Product}}{\text{Magnitude of Vector A} \times \text{Magnitude of Vector B}}$$

$$= \frac{6}{\sqrt{9} \cdot \sqrt{9}}$$

$$= \frac{6}{9}$$

$$\approx 0.667$$

The cosine similarity between `customer_id` and `id_customer` based on their trigrams is approximately 0.667.

Although such fuzzy methods for measuring text similarity can be useful in certain contexts, but they also come with limitations that can impact their effectiveness. Fuzzy algorithms typically do not consider the broader context in which words or phrases appear. For example, "bank" could refer to a financial institution or the side of a river, and fuzzy methods may fail to differentiate between these meanings. Another example is ignoring semantic meanings. "New York" and "Big Apple" despite referring to the same entity would be evaluated as a pair with no similarity as far as fuzzy techniques are concerned. To match attributes based on semantic meanings, data instances can be of much help. Consider the two tables below. Table 1 and Table 2 have exactly same attribute names defined. However by looking at the tuple values we can see that these two tables are describing completely different entities. Attribute "Height" in Table 1 is in the order of a couple of metres, however, in Table 2, it in the order of thousands of metres. A vector representation of the summary statistics of these two "Heights" would create two completely different embeddings and hence a very low similarity scores, meaning that these attributes are not unionable. Such a similarity measure based on vector representation of column values is known as **instance-based similarity measure**.

We encoded each columns statistics into vectors of dimension 9. For numeric columns- count, mean, stddev, min, percentile_25, median, percentile_75, max, range. And for string type columns - count, min, stddev, min, max and range (of lengths); and average_numerical_chars_ratio, average_whitespace_ratio. The intuition behind the latter two is that the ratio of white space to total characters is less for names whereas it is higher for addresses. Similarly, the ratio of number of numeric characters to total number of characters is 0 for names but higher for SSNs or IDs.

Table 1: Table: Q

Name	Country	Height (metres)
Tom Cruise	USA	1.70
Raj Magar	NPL	1.54
Aarif Sheikh	NPL	1.78

Table 2: Table C

Name	Country	Height (metres)
Everest	NPL	8848
Annapurna	NPL	8091
K2 Godwin	PAK	8611

The scope of this Work is to use weighted average of similarity of attribute names based on n-grams and similarity of vector representation of data distribution of column values.

4.2 Efficient Similarity Search

Let Q be a Queryset and $q_i \in Q$ denote each item in the queryset. Let $D = \bigcup_{C_j \in \mathcal{L}, c_i \in C_j} c_i$, be the vocabulary of $\mathcal{L}$. With brute force approach, to find a set with highest similarity value for a given queryset, we need to create a stored procedure in sql that loops over columns of each table and computes the sum of similarities of pairs across n tables. This approach is of polynomial time complexity.

To avoid computing all pairwise similarities between vocabulary and query elements, we used the Faiss Index [3]. Its algorithmic enhancements that vastly narrow down the search space for a vector's t-nearest neighbours allow it to have a much faster similarity search between vectors. This technique is called Approximate Nearest Neighbours (ANN) search and sacrifices some precision to obtain the vast speedups. For each $q_i \in Q$ faiss gives I_e: a sequence of top-t tuples $(c_j, \text{sim}(q_i, c_j))$ in descending order of $\text{sim}(q_i, c_j) > \alpha$, where $c_j \in D$. We can consider max(t)= cardinality of set $\mathcal{L}$ because ideally we'd want one matching attribute in each candidate table. With these sequences of similarity values, we aggregate the similarity overlap for each tables, and find the top-k.

4.3 Greedy Matching for Maximum Weight Bipartite Graph

For each $C_i \in \mathcal{L}$ we have obtained a set of similarity values for matches with Q. We now have a maximum weight bipartite graph matching problem in hand.

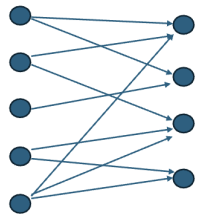

Figure 1: Bipartite graph representation for Query Q and Candidate C

If we assume N attributes in query table Q, N tables in total in the candidate set $\mathcal{L}$ and N atttribute each in all the tables in $\mathcal{L}$, then the time complexity in generating a maximum weight bipartite graph match for a pair using the optimal algorithm- Ford Fulkerson- is $O(N^3)$ and the space complexity is $O(N^2)$. However, the problem with optimal algorithm is that in the objective of maximizing the aggregate match score, it often disregards the highest valued matchings which needs to be incorporated in the final match. For that reason, we resort to greedy matching that has a time complexity of $O(N^2 LogN)$ and space complexity of $O(N^2)$.

5 Interface, Experiment and Output

The implementation is a user defined function (UDF) that takes a queryset Q, and a k-value as input arguments, and gives a set of top-k similar tables ω to the queryset Q as output.

Syntax:
SELECT unionableFindTopK('*table_name*', value_*k*);

```
postgres=# select unionableFindTopK('employee',2);
INFO:  ______________RANKED TABLES__________________
INFO:  ( workers, 5.952976 )
INFO:  ( workers_1, 5.926013 )
```

Figure 2: Sample Output

We loaded the following 15 open tables in a postgres database: **country, city, countrylanguage, data_src, datsrcln, deriv_cd, employee, fd_group, food_des, footnote, nut_data, nutr_def, src_cd, weight, workers.** On experimentation, the top-3 similar tables found for 'food_des' is 'nut_data', 'nutr_def' and 'fd_group'. All of these four tables are semantically related and refer to concepts about and/or related to food. Similarly, for the table 'city', the top 3 similar tables found were- 'country', 'countrylanguage' and 'nutr_def'- semantically similar being concepts related to geographic locations. The reason 'nutr_def' came in 3rd place was because it has columns that refer to geographical locations.

6 Discussion

The function unionableFindTopK is designed to efficiently compute the top-k similar tables in the collection to the query table. It supports merging results from multiple tables, leveraging PostgreSQL's capabilities. PostgreSQL's Server Programming Interface (SPI) allowed us to operate at a low level, providing direct access to the database's internals for maximum efficiency. It also provided an efficient way to extend the capabilities of the database with custom logic. Native SQL functions might require complex and verbose queries for similar tasks. Custom UDFs facilitates a dedicated, reusable, and efficient implementation. It can outperform generic SQL queries by reducing overhead. It can be packaged and shared and version-controlled, making this function available to the wider community or team.
However, there were some pain points experienced which might have been the reason why finding unionable tables are not available as native features for traditional database systems. For instance,

- For a queryset (that is a result of joins of tables or a subset of a table), we need to store it as a table (or temp table), before unionableFindTopK().
- If user wants some tables not to appear in candidates list, it needs to be explicitly written; which is boring.

7 Related Works

While there has been a ton of works on table union search [1, 5, 6, 7, 8, 9] none of them have been packaged as an extension or a native feature to common database systems. Some common PostgreSQL extensions available for similarity search tasks are:

- **pg_trgm**: text similarity measurement and index searching based on trigrams.
- **fuzzystrmatch**: determines similarities and distance between strings (also supports soundex).
- **pgstattuple**: gives tuple-level statistics.
- **dict_xsyn**: text search dictionary template for extended synonym processing.
- **pgvector**: vector similarity search.

References

[1] Tianji Cong, Fatemeh Nargesian, and H. V. Jagadish. Pylon: Semantic table union search in data lakes, 2023. URL: https://arxiv.org/abs/2301.04901, https://arxiv.org/abs/2301.04901 arXiv:2301.04901.

[2] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of Data Integration.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012.

[3] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. 2024. https://arxiv.org/abs/2401.08281 arXiv:2401.08281.

[4] The PostgreSQL Global Development Group. *Documentation PostgreSQL 10.3*, 2018.

[5] Xuming Hu, Shen Wang, Xiao Qin, Chuan Lei, Zhengyuan Shen, Christos Faloutsos, Asterios Katsifodimos, George Karypis, Lijie Wen, and Philip S. Yu. Automatic table union search with tabular representation learning. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 3786–3800, Toronto, Canada, July 2023. Association for Computational Linguistics. URL: https://aclanthology.org/2023.findings-acl.233, https://doi.org/10.18653/v1/2023.findings-acl.233 doi:10.18653/v1/2023.findings-acl.233.

[6] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. Santos: Relationship-based semantic table union search, 2022. URL: https://arxiv.org/abs/2209.13589, https://arxiv.org/abs/2209.13589 arXiv:2209.13589.

[7] Hamed Mirzaei and Davood Rafiei. Table union search with preferences. In Rajesh Bordawekar, Cinzia Cappiello, Vasilis Efthymiou, Lisa Ehrlinger, Vijay Gadepally, Sainyam Galhotra, Sandra Geisler, Sven Groppe, Le Gruenwald, Alon Y. Halevy, Hazar Harmouch, Oktie Hassanzadeh, Ihab F. Ilyas, Ernesto Jiménez-Ruiz, Sanjay Krishnan, Tirthankar Lahiri, Guoliang Li, Jiaheng Lu, Wolfgang Mauerer, Umar Farooq Minhas, Felix Naumann, M. Tamer Özsu, El Kindi Rezig, Kavitha Srinivas, Michael Stonebraker, Satyanarayana R. Valluri, Maria-Esther Vidal, Haixun Wang, Jiannan Wang, Yingjun Wu, Xun Xue, Mohamed Zaït, and Kai Zeng, editors, *Joint Proceedings of Workshops at the 49th International Conference on Very Large Data Bases (VLDB 2023), Vancouver, Canada, August 28 - September 1, 2023*, volume 3462 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3462/TADA2.pdf.

[8] Pranay Mundra, Jianhao Zhang, Fatemeh Nargesian, and Nikolaus Augsten. Koios: Top-k semantic overlap set search, 2023. URL: https://arxiv.org/abs/2304.10572, https://arxiv.org/abs/2304.10572 arXiv:2304.10572.

[9] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. Table union search on open data. *Proc. VLDB Endow.*, 11(7):813–825, March 2018. https://doi.org/10.14778/3192965.3192973 doi:10.14778/3192965.3192973.

[10] Pratik Pokharel, Juseung Lee, Oliver Kennedy, Marianthi Markatou, Andrew Talal, Jeff Good, and Raktim Mukhopadhyay. Drag, drop, merge: A tool for streamlining integration of longitudinal survey instruments. In *Proceedings of the 2024 Workshop on Human-In-the-Loop Data Analytics*, HILDA 24, page 1–7, New York, NY, USA, 2024. Association for Computing Machinery. https://doi.org/10.1145/3665939.3665965 doi:10.1145/3665939.3665965.