BareTQL: An Interactive System for Searching and Extraction of Open Data Tables

Davood Rafiei
University of Alberta
Edmonton, Canada
drafiei@ualberta.ca

Harrison Fah
University of Alberta
Edmonton, Canada
fah@ualberta.ca

Thomas Lafrance
University of Alberta
Edmonton, Canada
tlafranc@ualberta.ca

Arash Dargahi Nobari
University of Alberta
Edmonton, Canada
dargahi@ualberta.ca

ABSTRACT

There has been a plethora of research and commercial activities around extracting structured data from documents (e.g. web pages and scientific articles) and making them available to other applications. Many organizations and government bodies have been also making their data available to public. Despite the progress in many different aspects of table extraction and publishing, querying incomplete data in tables with little or no schema has been a challenge. This paper presents BareTQL, an interactive system for querying open data tables in the presence of the aforementioned challenges.

ACM Reference Format:
Davood Rafiei, Harrison Fah, Thomas Lafrance, and Arash Dargahi Nobari. 2021. BareTQL: An Interactive System for Searching and Extraction of Open Data Tables . In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Querying data that do not conform to a predefined or known formatting has been a long-standing research challenge [1, 4]. Much of tabular data collected from web pages and online resources have little or no schema information. Also, data exchanged between different organizations or shared online in the form of open data often are in tabular form with either little schema information or a schema that is not known to the users searching them. This reduces querying over such collections to simple keyword searches. We propose to demonstrate Bare Table Query Language, or in short *BareTQL* (pronounced as bear tickle), an interactive framework for querying large collections of tables. Compared to table search approaches in the literature (e.g. [2]), BareTQL offers three novel and distinctive features: (1) the composability and interoperability of operations with little reliance on the schema information of the tables being queried, (2) ability to transform tables for joinability, and (3) search customization in an interactive manner.

BareTQL moves beyond keyword search and provides a set of algebraic operators over a table collection and ways of combining those operators in a query to achieve a desired task. Supporting algebraic operations is a challenge when little is known or can be assumed about the underlying table schemes. BareTQL achieves this by taking an exploratory approach to search with a focus on what is known already and building on top.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 OVERVIEW OF BARETQL

To account for variations in the number of rows and columns, BareTQL stores table content at the cell level. Column types (e.g., numeric, text, etc.) provide hints on which columns can match queries, hence they are maintained, and so are titles and captions (when present), which help with keyword searches. Three inverted indexes are constructed on the cell values to support (1) exact matches on cell values, (2) keyword matches, and (3) ngram matches. A keyword index maintains the terms after the cell values are split based on space and punctuation, and an ngram index maintains all ngrams of size n for $n \in [minNG, maxNG]$.

As shown in Figure 1, five classes of operations are supported: (1) keyword search, (2) table search, (3) detecting joinability, (4) join, and (5) table expansion. Details of these operations are discussed next.

2.1 Keyword search

The exploration may start with a keyword search when little information is known about the tables being queried and their structures. Each table can be treated as a bag of words, and the standard IR techniques may be used to find tables that are relevant to a keyword query. BareTQL uses BM25 [6] for ranking the results, and the user has the option to select a table to develop further queries.

2.2 Table search

BareTQL supports table search where the user has a table (either obtained through a keyword search or put together manually) and wants to find other tables that are related. Let's first consider a query table with a single column. The similarity between two columns can be defined in terms of the number of entries they have in common, which gives rise to the Jaccard similarity. Our experiment

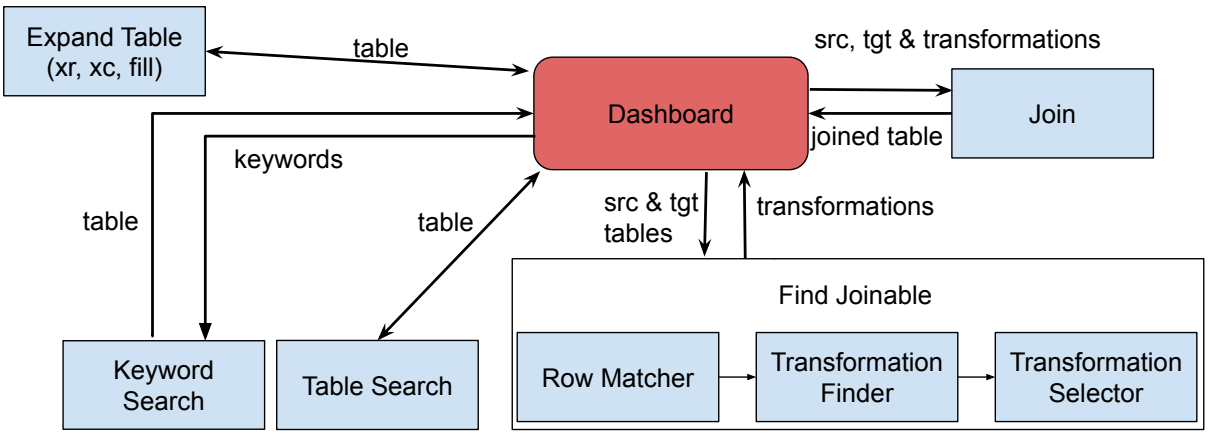

Figure 1: System architecture with more details of the operations described in the overview section

on various datasets shows that this is not a good measure if some values appear more commonly than others. To avoid this, BareTQL also implements a weighted scoring where cell values are inversely weighted based on the number of rows they appear in, following inverse document frequency scoring in IR. In particular, let $irf(v, q)$ be the logarithmically-scaled inverse fraction of the rows in the query table q that contain v, i.e. $log(|q|/n(v, q))$, and $irf(v, d)$ be the logarithmically-scaled inverse fraction of the rows in the data table d, i.e. $log(N/n(v, d)$, where N is the total number of rows in the collection, q is the number of rows in the query table t and $n(v, q)$ and $n(v, d)$ are the number of rows in tables q and d that contain v respectively. The product $irf(v, q).irf(v, d)$, which gives the inverse row frequency of v, and the frequency of v in data table d, $n(v, d)$, are plugged into the BM25 scoring after some smoothing and are aggregated over different cell values in the query table to give a relevance score. Different types of matches between cell values are supported including exact and ngram-based matching.

When query tables have more than one column, the user can tag the columns that can be considered in the match and if some columns must be grouped together (e.g. first name and last name). For a query table q with multiple column groups and a candidate data table d, the product of the relevance scores of column groups gives the relevance score, based on which the candidate table is ranked.

To evaluate our table search, we ran it over two recent benchmarks in the literature: AutoJoin [7] and AutoFuzzyJoin [3]. Both datasets include pairs of tables that are joinable after some transformations or formatting of the rows. Some tables in the dataset participate in multiple joins (e.g., AutoJoin has five table listings of New York governors and six listings of US presidents); and we picked one table pair randomly from each of those sets. Our BM25 adaptation for join (as discussed above) retrieves the ground truth matching table at top position in 88% cases for AutoJoin and in 86% of the cases for AutoFuzzyJoin, whereas the Jaccard similarity retrieves the ground truth matching table at top position in 65% cases for Autojoin and in 94% of the cases for AutoFuzzyJoin.

2.3 Transforming tables for joinability

Tables obtained through a table search may not be joinable with the query table when data is formatted differently or a cell value in one table is spread over multiple cells in the other tables. BareTQL supports transformations which are applied to a source table to produce rows that are joinable with the rows of a target table. Transformations include basic string operations such as split and substring and more complex operations in the form of a sequence of basic operations [5]. Given a query table and a candidate table, BareTQL searches the space of possible transformations and identifies those that transform the largest number of rows or cover the input with the least number of transformations. This is quite useful since instead of manually finding transformations, which is a tedious and time-consuming job, the user may only verify or confirm a few transformations before doing a join. Figure 2 shows an example where the name in one table is mapped to a usedid in another table using a sequence of transformations automatically obtained in BareTQL.

2.4 Expand rows (xr)

Given an example set of tuples, sometimes we want to find more tuples that may belong to the same class or have the same properties. The xr operation takes a query table as input and expands it vertically by adding more tuples that are similar to the given set. The new tuples may be ordered based on their relatedness to the query set.

To find tables that are related to a given set of tuples and may expand it, the table search operator may be invoked. That operator finds, for each matching table that is returned, not only a matching or similarity score but also a mapping of columns that gives rise to the maximum similarity. With the related tables projected on the columns that are mapped to those of the query table, one can extract the co-occurring tuples and sort them based on their co-occurrence frequencies, treating all related tables the same. BareTQL takes into account the similarity scores of the related tables in the ordering. Hence the similarity between a candidate tuple t and query table

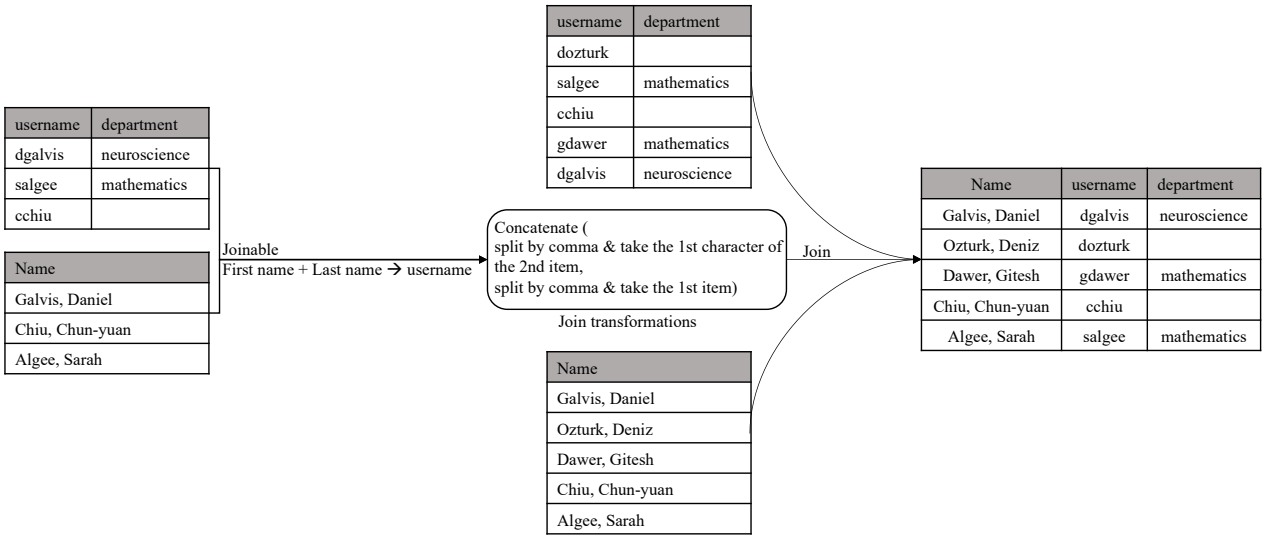

Figure 2: An example of join in BareTQL

T_q is defined over all related tables T that mention tuple t as

$$tupleSim(t, T_q) = \sum_{T \in TC \wedge P \in \pi(T) \wedge UC(P,T_q) \wedge t \in P} tableSim(T_q, T), \quad (1)$$

where T ranges over the tables collection TC that is relevant, $P \in \pi(T)$ is a projection of T, and $UC(P, T_q)$ is an indicator variable which is true when P and T_q are union-compatible.

2.5 Expand Columns (xc)

Given a table, sometimes we want to find more attributes that may describe the given set. For example, given a set of movie titles, we may want to find more information about each movie, such as the production year, the director, the producing studio, the box office revenue, etc. The xc operator expands a given query table horizontally by adding more columns that may describe the given set. The additional columns can vary from one query to next, depending on how the given tuples in the query are mentioned in the matching tables in TC.

To find tables that are related to a given query set and may expand it horizontally, the table search operation can again be invoked. Any column in a related table that does not map directly to a query table column can be seen as a potential extension of the query columns. However, since related tables are ranked, their contributed columns may also be ranked accordingly. Related columns may also be ranked based on the number of their non-empty cell values.

2.6 Fill in the blanks

Sometimes we have partial information in the form of a table, for example, about entities and their properties and wish to fill the gaps. For example, we may have data as shown in Table 1 and want to fill the missing information marked with empty string values, using data in our table collection.

The table search operation can again be used to find tables that may have data to fill the gaps. There can be a disagreement between the relevant tables on how the gaps should be filled though. For example, consider filling the missing information in Table 1 (right). There can be multiple models from Canon and Pentax that may be considered equivalent to Nikon D700. The fill operation may analyze the matches and return either the most likely filler or all possible fillers for each gap.

Table 1: Example query tables for the *fill* operator

Tim Cook	Apple	2011
Sundar Pichai	Google	
	Microsoft	2014

Nikon	D700
Canon	
Pentax	

3 DEMO EXPERIENCE

The followings are some of the use cases for BareTQL.

(1) The user knows a few movie titles and wants to find out more information about those titles (e.g. awards received, director, box office revenue) as well as more similar titles.
(2) The user has a list of companies as a query table and wants to find data tables that provide more information about the entries in the query table.
(3) The user has small samples of two tables, each providing a different type of information about the same class of entities but formatted differently and not directly joinable. BareTQL can find transformations that map the entities in one table to those in the other table. The user can select a transformation from those obtained and apply it to the entire table (including many rows that are not seen) before a join.

Figure 3 shows some examples of search operations.

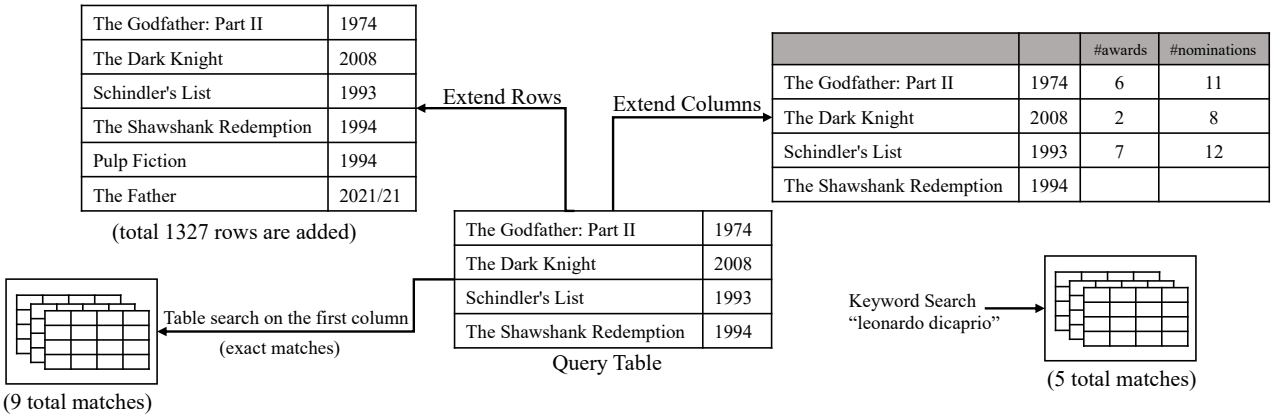

Figure 3: Examples of search operations in BareTQL

4 CONCLUSION

We have presented BareTQL, an operational interface and an interactive system for querying open data tables and web tables. The two design goals of BareTQL have been (1) the composability and interoperability of operations to allow queries to be built on the fly when the schema information is not available for open data tables, and (2) the scalability of operations to large table collections. Despite our progress as reported, challenges still remain. Our future work will explore ways of improving BareTQL in both fronts.

REFERENCES

[1] Hazem Elmeleegy, Jayant Madhavan, and Alon Halevy. 2009. Harvesting relational tables from lists on the web. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1078–1089.

[2] Oliver Lehmberg, Dominique Ritze, Petar Ristoski, Robert Meusel, Heiko Paulheim, and Christian Bizer. 2015. The mannheim search join engine. *Journal of Web Semantics* 35 (2015), 159–166.

[3] 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*. 1064–1076.

[4] Renée J Miller. 2018. Open data integration. *Proceedings of the VLDB Endowment* 11, 12 (2018), 2130–2139.

[5] Arash Dargahi Nobari and Davood Rafiei. 2021. Efficiently Transforming Tables for Joinability. *arXiv preprint arXiv:2111.09912* (2021).

[6] Stephen Robertson and Hugo Zaragoza. 2009. *The probabilistic relevance framework: BM25 and beyond.* Now Publishers Inc.

[7] Erkang Zhu, Yeye He, and Surajit Chaudhuri. 2017. Auto-join: Joining tables by leveraging transformations. *Proceedings of the VLDB Endowment* 10, 10 (2017), 1034–1045.