SQL Queries Over Unstructured Text Databases

Alpa Jain
Columbia University

AnHai Doan
University of Wisconsin-Madison

Luis Gravano
Columbia University

Abstract

Text documents often embed data that is structured in nature. By processing a text database with information extraction systems, we can define a variety of structured "relations," over which we can then issue SQL queries. Processing SQL queries in this text-based scenario presents multiple challenges. One key challenge is efficiency: information extraction is a time-consuming process, so query processing strategies should pick efficient extraction systems whenever possible, and also minimize the number of documents that they process. Another key challenge is result quality: extraction systems might output erroneous information or miss information that they should capture; also, efficiency-related query processing decisions (e.g., to avoid processing large numbers of useless documents) may compromise result completeness. To address these challenges, we characterize SQL query processing strategies in terms of their efficiency and result quality, and discuss the (user-specific) tradeoff between these two properties.

1 Introduction

Text often embeds valuable structured data, such as who recommends selling which stocks or how many people were affected by a disease outbreak. To make use of this intrinsically structured data that is embedded in natural-language text, we can leverage information extraction systems and follow an "extract-then-query" paradigm: we first apply extraction systems to the available documents, in an offline step, to uncover structured data. For example, we can use an extraction system trained for identifying disease outbreaks to extract a tuple ⟨Ebola, Zaire, 1976⟩ from a news article, meaning that an outbreak of Ebola occurred in Zaire in 1976. We can then load such structured data into a DBMS (perhaps after a data cleaning step) so that we can later answer structured queries (e.g., written in SQL) as they arrive.

In some situations, the *fully offline* extraction approach, where all text documents in a database are processed with extraction systems in a lengthy offline step, may be undesirable or simply impossible, because the underlying data is vast or evolves quickly, or because the target information to extract is not known in advance. As an alternative, we explore a *fully online* extraction approach (Section 2): to process a SQL query, we define candidate execution strategies to retrieve text documents at query time and feed them to appropriate extraction systems. The choice of document retrieval strategies and extraction systems for a SQL query affects both the efficiency and the result quality of a query execution, as we will see. So we consider the user-specified desired balance between query execution efficiency and result quality, and choose query execution strategies accordingly, in a principled, cost-based manner (Section 3).

2 SQL Queries over Text Databases

Consider an archive of newspaper articles, together with an information extraction system trained to extract a *Headquarters(Company, Location)* relation, where a tuple $\langle c, \ell \rangle$ indicates that c is a company whose headquarters are located in ℓ, as well as another system trained for an *Executives(Company, CEO)* relation, where a tuple $\langle c,e \rangle$ indicates that person e is the CEO of company c. We can then define a view *CompanyInfo(Company, Location, CEO)* by joining the two "base" relations, and express SQL queries such as:

```
Q1:  SELECT Company, CEO FROM CompanyInfo
     WHERE  Location = 'Redmond'
```

To process the above query, we could sequentially feed each database document to the extraction systems, then join the extracted *Headquarters* and *Executives* relations, and finally return only those tuples that satisfy the query condition. This exhaustive query execution might be unnecessarily inefficient if only a small fraction of the documents in the database contribute to our extraction tasks.

Beyond efficiency considerations, we should also analyze the query *result quality* for the different execution strategies. Unlike in the relational world, where all correct plans for a query produce the same results, different strategies for a query over a text database might indeed produce different results. The exhaustive approach above, which processes all the database documents, is time-consuming but has the advantage of producing "complete" results relative to the extraction systems of choice. Faster alternatives

that process fewer documents may result in the loss of some result tuples, hence hurting result completeness. Another important consideration is the characteristics of the relevant extraction systems, which often differ in their output quality and extraction efficiency. Overall, which plan is best for a query depends on the efficiency and result quality requirements for the query. Sometimes users are after receiving a few result tuples quickly; some other times, users prefer to receive query results that are as complete as possible, even if it takes a relatively long time to produce the results. Our SQL query processing problem is then as follows.

Problem Statement. *Consider a text database D and n "base" relations R_1, ..., R_n defined over D. Each base relation R_i can be extracted from D using one or more information extraction systems. We assume that all base relations R_1, ..., R_n share the same primary key K and no other attributes, and define a view $V = {}_K \bowtie_{i=1}^{n} R_i$ as the natural outerjoin of R_1, ..., R_n over the K attributes. We consider SQL selection-projection queries over V with selection condition conjuncts of the form $A = t$, where A is a textual attribute and t is a constant. Then, given such a SQL query, our goal is to identify an execution strategy that meets the desired efficiency and result quality requirements as closely as possible.*

To evaluate a query Q over a text database D, we need to:

(1) Select an extraction system E_{ij} for each base relation R_i, as well as a document retrieval strategy X_i for E_{ij}.
(2) Use strategy X_i to retrieve from database D the set of text documents P_i that E_{ij} will process.
(3) Process the documents in P_i with extraction system E_{ij} to obtain a relation instance r_i.
(4) Apply data cleaning techniques to the extracted relations, for record linkage, and eliminate data inconsistencies.
(5) Generate a candidate view $v = {}_K \bowtie_{i=1}^{n} r_i'$, where r_i' is a "clean" version of r_i.
(6) Execute Q over v and return the execution results.

The execution strategies for a SQL query differ in the Step (1) choices on how to execute Steps (2) and (3). In some scenarios, we might have more than one extraction system available for a relation; the choice of extraction systems for Step (3) will depend on their efficiency and output quality. For Step (2), we consider the following document retrieval strategies, which lead to SQL query executions with different efficiency and result quality characteristics:

Scan: We can sequentially scan the database and feed each document to an extraction system E. This strategy yields complete query results for E, at the expense of efficiency.

PromD: Alternatively, we could avoid processing all documents by identifying just the "promising" ones via querying, using QXtract [2] to derive keyword queries for each extraction system via machine learning. This query-based document retrieval strategy reduces the SQL query execution time, at the expense of answer completeness.

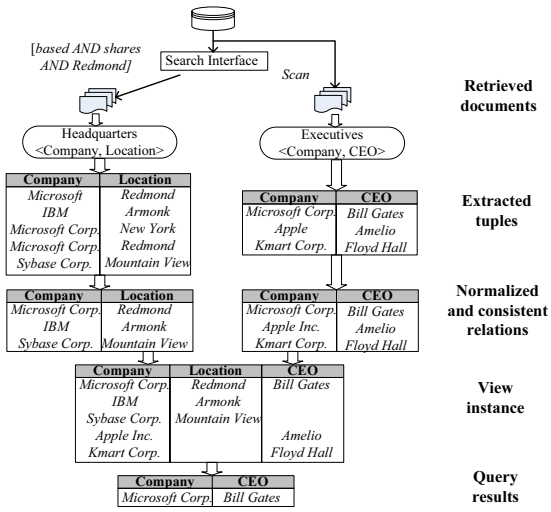

Figure 1. Stages in the execution of query Q1.

Const: As another alternative to reduce the number of documents that we process, we can exploit any SQL query constants to construct keyword queries. For example, the keyword "Redmond" in query Q1 can be used to retrieve only those documents with this word for the extraction of *Headquarters* (documents without the word "Redmond" could not contribute useful tuples for this query). The *selectivity* associated with the constants in the query determines the efficiency of this retrieval strategy.

PromC: We can naturally combine the *PromD* and *Const* strategies by ANDing their queries. For example, for Q1 we combine query [*based AND shares*] from *PromD* with query [*Redmond*] from *Const* to obtain query [*based AND shares AND Redmond*]. Similar to *PromD*, this strategy also has the advantage of reducing the SQL query execution time, but at the expense of answer completeness.

Figure 1 shows a possible execution for query Q1, with two document retrieval strategies, *PromC* for *Headquarters* and *Scan* for *Executives*, chosen in Step (1). *PromC* issues keyword queries such as [*based AND shares AND Redmond*] to the text database to retrieve promising documents (Step (2)). After feeding each of these documents to the extraction system for *Headquarters*, we obtain tuples such as ⟨Microsoft, Redmond⟩ (Step (3)). To extract *Executives*, *Scan* retrieves all documents exhaustively, one at a time (Step (2)), and feeds them to the extraction system for this relation (Step (3)), to extract tuples such as ⟨Microsoft Corp., Bill Gates⟩. After extraction, we use record linkage techniques to conclude that "Microsoft" and "Microsoft Corp." refer to the same company in the two base rela-

tions, and further data cleaning resolves inconsistencies in the extracted relations, to eliminate erroneous tuples such as ⟨Microsoft Corp., New York⟩ (Step (4)). As a final step, we generate the query results (Steps (5) and (6)).

3 Query Execution Properties

To compare alternate execution strategies for a query Q over a database D, we define *efficiency* as follows:

Definition 3.1. (Efficiency) The **efficiency** of a query execution S over a text database D, $E(S, D)$, is the inverse of the execution time of S, in seconds, over D. □

We also compare execution strategies based on their query result quality, for which we need to characterize the "ideal" result for Q over D, $Ideal(Q, D)$. This hypothetical ideal result consists of *all* the *correct* query results for Q that could be derived from database D. Of course, $Ideal(Q, D)$ would be prohibitively expensive to compute for any large database D, since this "computation" would necessarily involve substantial human effort (e.g., no "perfect" extraction systems exist). However, the ideal results are conceptually helpful to characterize the *precision* and *recall* of query execution strategies, as follows:

Definition 3.2. (Precision and Recall) Consider an execution strategy S for a query Q over a text database D, and let R be the results that S produces. We define the **precision** of S over D as $P(S, D) = \frac{|R \cap Ideal(Q,D)|}{|R|}$ and the **recall** of S over D as $R(S, D) = \frac{|R \cap Ideal(Q,D)|}{|Ideal(Q,D)|}$. □

We combine precision and recall into a single metric by computing their geometric mean, as follows:

Definition 3.3. (Quality) Consider an execution strategy S for a query Q over a text database D. We define the **quality** of S over D as $Q(S, D) = (P(S, D) \cdot R(S, D))^{1/2}$. □

As discussed above, query execution strategies over text databases exhibit a tradeoff between efficiency and result quality. Ultimately, the right balance between efficiency and quality is user-specific, and we capture it with a (user-specified) query processing parameter w, ranging in value from 0, to privilege efficiency, to 1, to privilege result quality. In turn, we use parameter w to characterize the overall *goodness* of a query execution, as follows:

Definition 3.4. (Goodness) The **goodness** of a query execution S over a text database D for user-specified parameter w is $G_w(S, D) = Q(S, D)^w \cdot E(S, D)^{(1-w)}$. □

Putting everything together, to process a SQL query we consider the strategies derived from instantiating the general algorithm with the different extraction systems and document retrieval strategies. We estimate the goodness of each instantiation and choose the best option. We omit the details on our (sampling-based) methods to derive these estimates because of space limitations.

4 Related Work

The problem of information extraction from text has received significant attention (see [4, 7] for surveys). Earlier approaches rely on hand-crafted extraction rules, but many recent efforts (e.g., [1, 3, 6]) have developed unsupervised or learning-based extraction techniques. Recent work has started to address efficiency issues, including the QXtract system [2], which we used in this paper.

Closest to this paper is the analysis in [5], which considers (among others) the problem of identifying the document retrieval strategy for a single extraction system S that reaches a pre-specified target recall in minimum time. [5] assumes that S is perfect in that it produces all—and only—correct tuples. Our current work is related to [5], but differs from it in several crucial aspects. First, we consider the more general problem of optimizing the combined execution of multiple information extraction systems *and* multiple document retrieval strategies, all towards the goal of efficiently answering a SQL query. Second, we remove the assumption of perfect extraction systems, and model extraction errors. Finally, we do not optimize for a pre-specified target recall, but consider the goal of balancing recall, precision, and execution time in a flexible manner.

5 Discussion

We performed an extensive evaluation of our query processing approach, which relies on document sampling to estimate the *goodness* of the candidate query execution strategies. We do not report our results because of space constraints. As a summary of our conclusions, our query processing approach produced high *goodness* executions for both selection and projection queries and for all values of user-specified parameter w (see Definition 3.4). As expected, our approach privileges efficiency for low values of w and result quality for high values of w, and dynamically adapts to the user-specified efficiency-quality balance.

References

[1] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *DL*, 2000.

[2] E. Agichtein and L. Gravano. Querying text databases for efficient information extraction. In *ICDE*, 2003.

[3] S. Brin. Extracting patterns and relations from the world wide web. In *WebDB*, 1998.

[4] R. Grishman. Information extraction: Techniques and challenges. In *SCIE*, 1997.

[5] P. Ipeirotis, E. Agichtein, P. Jain, and L. Gravano. To search or to crawl? Towards a query optimizer for text-centric tasks. In *SIGMOD*, 2006.

[6] I. Mansuri and S. Sarawagi. A system for integrating unstructured data into relational databases. In *ICDE*, 2006.

[7] A. McCallum. Information extraction: distilling structured data from unstructured text. *ACM Queue*, 2005.