# Structured Querying of Web Text

## A Technical Challenge

Michael J. Cafarella, Christopher Ré, Dan Suciu, Oren Etzioni, Michele Banko
University of Washington
Seattle, WA 98195
{mjc, chrisre, suciu, etzioni, banko}@cs.washington.edu

## ABSTRACT

The Web contains a huge amount of text that is currently beyond the reach of structured access tools. This unstructured data often contains a substantial amount of implicit structure, much of which can be captured using information extraction (IE) algorithms. By combining an IE system with an appropriate data model and query language, we could enable structured access to all of the Web's unstructured data. We propose a general-purpose query system called the *extraction database*, or ExDB, which supports SQL-like structured queries over Web text. We also describe the technical challenges involved, motivated in part by our experiences with an early 90M-page prototype.

## 1. INTRODUCTION

The vast quantity of text on the Web is currently only accessible using search engines' *keyword-in, documents-out* queries. We would often prefer to pose queries that take advantage of the structure embedded in much of that text. For example, consider a website of classified postings. Although the classifieds consist of free-form text, many contain easily-recognizable fields such as price, the seller's phone number, the seller's address, etc. No one has published a formal schema, but nonetheless there is clearly structural information that a query-writer should be able to use.

Information extraction (IE) systems can often extract small pieces of structured data from text. For example, Brin used DIPRE to extract author/book pairs, Agichtein, et al. used Snowball to find corporation/headquarters pairs, and Etzioni, et al. have used KnowItAll to discover hypernym ("is-a") relationships, among others [6, 1, 20]. Mansuri and Sarawagi proposed an extraction integration system that works with any existing schema and constraint set [31].

But IE systems to date require an administrator to predefine the schema that will be populated, as well as any database constraints. Manual schema design is not feasible when building a structured query system for the entire Web. The number of possible tables and their attributes is prohibitively large, and moreover, may change as new communities and topics emerge. Other proposed or realized systems integrate IE results, but not in an automatic domain-independent way. The AVATAR system relies on a series of hand-written *annotators* to emit structured tuples, *e.g.*, a tuple that has a person's name or phone number [27]. The CIMple project's *semantic schema* integrates information from a variety of sources, but is specific to one application domain; for example, CIMple's DBLife prototype is meant to manage data relevant to the database community, including sets of researchers, publications, conferences, etc [16].

| a | b | c | probability |
|---|---|---|---|
| Kepler | log books | 1630 | 0.7902 |
| Heisenberg | matrix mechanics | 1976 | 0.7897 |
| Galileo | telescope | 1642 | 0.7395 |
| Newton | calculus | 1727 | 0.7366 |

**Figure 1: The top-ranked results for a query to our** ExDB **prototype. The query here is** `q(?a, ?b, ?c) :- invented(?a, ?b), died-in(?a, <year> ?c).` **This query took 30 seconds to process on a database of 90M Web pages.**

We propose a structured Web query system called the **extraction database**, or ExDB. It uses IE systems to extract data, schema, and constraint information from Web; every string on the Web may become both a data value and a structural component. Since the extractions are inevitably flawed, we model them as tuples that are probabilistically true (as in the MYSTIQ system) [15]. These extractions form a probabilistic database. ExDB also offers a language for describing probabilistic queries over this extracted database.

Figure 1 shows the top-four results of a sample query posed to an early ExDB prototype. The user has asked for a list of 3-column tuples, in which the first and second columns make up an extracted `invented` fact, and the first and third columns form a `born-in` fact. Finally, the third column is constrained to be a `year`. The results are ranked by probability of being true. The user's predicates and types are not published in any official metadata; the user simply enters them as the most appropriate way to describe her desired query.

In Figure 2 we show the ExDB work flow. Having downloaded a corpus of Web text, the ExDB runs a series of un-


$3^{rd}$ *Biennial Conference on Innovative Data Systems Research (CIDR)* January 7-10, 2007, Asilomar, California, USA.

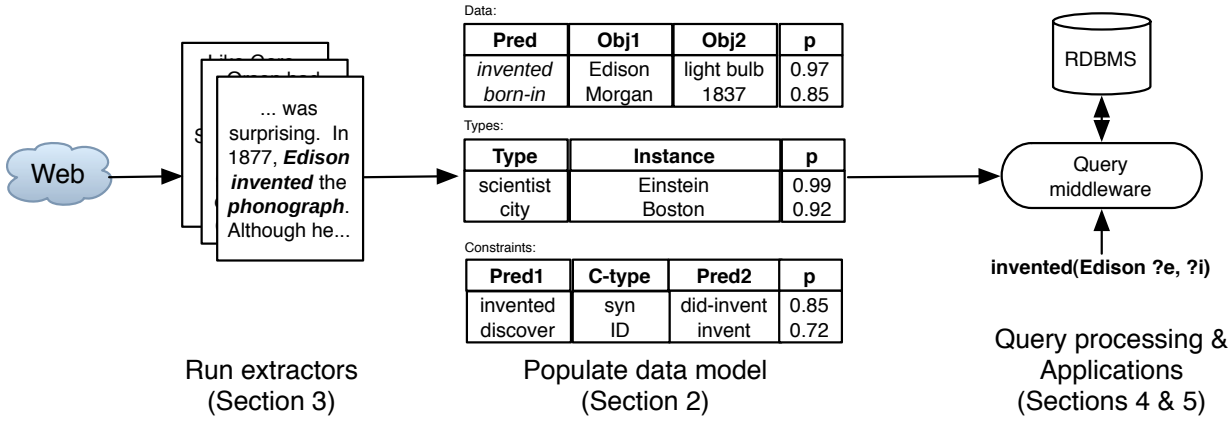


**Figure 2: Constructing the ExDB requires several processing steps. In step 1, we run information extractors over the downloaded web text, as described in Section 3. In step 2, the extracted information is stored in the ExDB data model, described in Section 2. Finally, applications can query the ExDB middleware and probabilistic RDBMS. Sections 4 and 5 describe query processing and possible applications.**

supervised natural-language-driven extractors over the text; this extraction step is described in Section 3. The resulting extractions are used to populate a huge probabilistic database that contains Web data, schemas, and constraints. We call this database the Web Data Model, and describe it in Section 2. Finally, as described in Sections 4 and 5, we use this database for users' queries and many other possible applications.

At each step in constructing the ExDB, there are substantial problems that remain unsolved. This paper is devoted to describing the ExDB framework, and we have even constructed an initial prototype that runs on a corpus of 90M downloaded pages. However, the ExDB raises outstanding technical issues in almost every step of its construction, from information extraction, to reference reconciliation, to probabilistic query processing. The ExDB is a promising idea that motivates many different research questions, serving as a technical challenge to the entire field.

## 2. THE WEB DATA MODEL

This section discusses ExDB's extraction-based data model, its handling of imprecision, and the way ExDB queries make use of both.

### 2.1 Data Model

Through a combination of IE techniques, ExDB should extract several base-level concepts:

**Objects** are the data values in the system. Examples: `Einstein`, `telephone`, `Boston`, `light-bulb`[1]. Our prototype contains 102M unique objects.

**Predicates** are binary tables populated by pairs of objects. Examples include `discovered(Edison, phonograph)`, `born-in(A.-Einstein, Switzerland)`, and `sells(Amazon, PlayStation)`. Our prototype has populated these tables with 338M facts. These tuples make up the data contents of the ExDB.

[1]Multiple-word concepts are presented with hyphens for readability. Our system does not need them for parsing.

**Semantic types** are unary tables populated by objects. Examples are `city(Boston)`, `city(New-York)`, `electronics(dvd-player)`. Our prototype contains 6.6M type instances. In addition, the system should have a fixed set of predefined data types, such as `integer`, `date`, `year`, etc.

This data model can capture many useful real-world facts, but obviously cannot perfectly model an arbitrary domain: for the sake of simplicity, we currently limit it to predicate tuples with arity two. ExDB should allow for higher-arity tuples. However, as will be noted below, extractors there are practical limitations in finding good domain-independent (and thus scalable) extractors for high-arity tuples.

The resulting ExDB database may contain an enormous amount of structural information (in the form of predicates and semantic types), but the user is not expected to know it a priori. Instead, the user formulates queries using her own choices for objects, types, and predicates; ExDB then matches the query to the extracted elements as best it can.

In addition to the base schema elements, the ExDB should extract a series of relationships designed to make queries even easier for the user:

**Synonyms** denote two equivalent objects, predicates, or types. For example, `Einstein` and `A.-Einstein` almost certainly refer to the same real-world object. Similarly, `invented` and `has-invented` refer to the same predicate. ExDB should maintain a set of these synonyms for each object, predicate, and type. By transforming user queries using this synonym information, the ExDB could answer queries using extractions that do not match the actual query text. Object synonymy alone could enable hugely improved equijoin processing.

Our prototype implements only predicate synonymy, and has extracted 17,000 synonym pairs.

**Inclusion dependencies** describe a subset relationship between two predicates. For example, `invented(?x, ?y)` $\subseteq$ `discovered(?x, ?y)` indicates that true instances

| Database element | Description | Linguistic term | Extractor |
|---|---|---|---|
| `invented(Edison, phonograph)` | *Edison* invented the *phonograph* | arity-two fact | TextRunner [4] |
| `scientist(Einstein)` | *Einstein* is a *scientist* | hypernymy | KnowItAll [20] |
| `invented(x, y)` = `has-invented(x, y)` | *invented* is similar to *has invented* | synonymy | DIRT [29] |
| `invented(x, y)` ⊆ `discovered(x, y)` | *invented* specializes *discovered* | troponymy | ? |
| `FD: has-capital[x, y]` → `has-capital[y]` | there is just one capital for any entity | rule | ? |

**Table 1: Database concepts, their text equivalents, and their extraction mechanisms. Recent advances in IE can help find some constraints; others are future work. In addition to algorithmic extractors, linguistic resources like thesauri could help populate an** ExDB**.**

of tuples in `invented` are also in `discovered` [2]. An elaboration of a verb is called a *troponym* and is the closest linguistic analogue to this kind of dependency [22]. As with synonyms, these inclusion dependencies can be used by the query processor to return results for, say, `invented(?x, ?y)` whenever the user asks for `discovered(?x, ?y)`.

Our prototype does not yet extract troponyms from the Web data.

**Functional dependencies** are especially useful for queries with negation or for explaining why an object is *not* an answer to a query. We discuss functional dependencies in more detail in Section 2.2.

It is not yet clear how necessary these synonyms and other constraints will be. Their main advantage is that they allow a probabilistic mapping between the strings from text extractions and the strings in a user's query. (For example, a query on the `for-sale` predicate can be rewritten to include results from the `on-auction` predicate.) It may be that data embedded in Web text is so prevalent and easily-extracted that the IE system naturally derives all reasonable synonyms. It seems likely that the utility of synonyms will depend on the type of data extracted (*e.g.*, `invented(Edison phonograph)` will probably appear under more synonyms than will `is-selling(John-Smith, Toyota-9392)`.)

Table 1 shows all the database concepts the ExDB incorporates, their language analogues, and an appropriate extraction mechanism, if available. For some useful concepts, we are unaware of a matching extractor. We discuss these mechanisms in more detail in Section 3.

## 2.2 Managing Imprecision

We use probabilities to represent imprecision arising from genuine uncertainty about the world and from errors by inevitably-flawed text extractors. Several recent research projects have used probabilities to describe uncertainty at the data level [36, 5, 35, 3, 15, 14]. However, ExDB also needs to represent uncertainties at the schema level, and in the constraints.

**Probabilistic Data and Schema** Tuples in the predicate tables and in the semantic type tables are probabilistic: each tuple in these relations has a value $0 \leq p \leq 1$ describing the probability that the tuple is *in* or *out*. This is similar to existing systems [36, 14].

**Synonyms** Predicate, type, and object synonyms are also probabilistic; there is always a chance that `Einstein` and `A.-Einstein` refer to different entities. The predicates `invented` and `created` are only partly equivalent. Items in each synonym set are present with some value $0 \leq p \leq 1$.

**Probabilistic Constraints** Probabilistic inclusion dependencies are similar to probabilistic predicate synonyms that are applied only "one-way." Functional dependencies are somewhat more interesting; they describe rules about what fact sets should be simultaneously true. ExDB will use them to process negated queries, or to explain to a user why a certain object is not the answer set. For example, consider the following probabilistic FD, indicating that (with high probability) a person can only be born in one country.

`born-in(?x, <country> ?y):  ?x` → `?y p=0.95`

Suppose the user asks for all scientists born in `Germany` that taught at `Princeton`, and, surprised by the answer, asks the system: "why is `Einstein` not an answer?". Using the functional dependency above, the system could answer that because `Einstein` was born in `Switzerland`, and because the FD tells us a person can be born in only one country, his probability of being born in `Germany` is exceedingly low. (Also, we note that `Switzerland` and `Germany` have very low probability of being synonyms.)

Probabilistic FDs allow us to make statements about the data with less than total confidence, which is important given that the managed data is not curated; the extraction set may include obscure or confusing cases that violate a constraint that is generally true.

Our prototype has implemented support for extracted data, types, and synonyms, but not yet for inclusion dependencies or functional dependencies.

## 2.3 ExDB Queries

We propose that users ask SPJU queries over the Web Data Model, using a Datalog-like notation. For example, the query `q(?i) :- invented(Edison, ?i)` returns all inventions by `Edison`. Query answers are ranked by their probabilities.

Data items in the head are returned as query results. Variable names start with `?` to distinguish them from data values. We can constrain variables to types or to specific values:

```
q(?x, ?y) :- died-in(<scientist> ?x, 1955 ?y)
```

This query returns a two-column table (three, if including the probability) in which the first column is a an instance of `scientist`, the second column has value `1955`, and they were extracted as part of the `died-in` predicate (or perhaps one of its synonyms). When querying a set of classifieds, we might search for locally-available electronics with the following query:

```
q(?x, ?y) :- for-sale-in(<electronics> ?x, Seattle ?y)
```

[2]This relationship may not be technically accurate, but it is inarguably present in real-world usage. We would expect to find it in both Web text and in the ExDB query workload.

We can also add additional clauses, as in:

```
q(?x, ?y, ?z) :- invented(<scientist> ?x, ?y),
                 died-in(?x, <year> ?z),
                 (?z < 1900)
```

which returns a three-column table, where the third column is the year in which the scientist died. We will only return tuples where we know the scientist's year of death, and where the scientist died prior to 1900. Similarly, we can limit our shopping search for inexpensive items with:

```
q(?x, ?y, ?z) :- for-sale-in(<electronics> ?x,
                                 Seattle ?y),
                 costs(?x, ?z),
                 (?z < 25)
```

Finally, we can perform projections by modifying variables in the head. The following query simply finds scientists who have invented something:

```
q(?s) :- invented(<scientist> ?s, ?i)
```

Efficient probabilistic query processing is an active research area. For additional details on query processing, see Section 4.

It is important to note again the difference between the database query paradigm deployed by ExDB and the keyword-in, documents-out paradigm of search engines. In ExDB the granularity of the data is a concept (a word or short phrase), not a document. Oftentimes a query's answer is obtained by joining multiple facts that have been extracted from multiple, unrelated Web pages. Search engines make no attempt to integrate information across the page boundary.

ExDB can explain the answers it retrieves at two levels: the query lineage [41], which is essentially a proof tree for each item in the query's answer, and the extraction lineage [13, 7], which is an explanation of how and from which pages the basic facts used to derive the query were extracted. Our prototype implements support for extraction lineage, but not yet query lineage.

# 3. EXTRACTION MECHANISMS

The ExDB is feasible because its data, schema, and constraints have analogues with extractable linguistic phenomena. In this section we describe how ExDB converts information embedded in Web text into ExDB elements. Although the fact that such extraction is possible is more important than the actual technique, the approaches described here give a flavor of what is possible [3].

## 3.1 Fact Extractions

The most basic form of extracted fact for the ExDB is a simple predicate of arity two. Every query returns one or more of these basic facts.

Many recent information extraction projects have attempted to find fact triples with little or no supervision [40, 37]. Our prototype uses an unsupervised system called TextRunner [4]. It runs once over an entire corpus of text, generating a large set of probabilistically-assessed extractions. Unlike many extractors, TextRunner does not require an input list of target predicates or object pairs, so it can be run over the entire Web without concern for the domain.

The TextRunner extractor starts by using a heavyweight deep linguistic parser to identify a small number of high-quality extraction triples. These triples consist of two entity strings within a single sentence and a descriptive relationship string that links them. It then computes a number of lightweight language-driven statistics on this small set of good examples. These statistics are used to compute an "extraction-classifier" that is lightweight enough to apply to a Web-scale corpus.

Finally, the extractor applies the classifier to the entire corpus, and counts extraction repetitions. Repeated independent extractions make a fact extraction more probable, whereas rare extractions are considered errors and receive a low probability. (The probability model is similar to that described by Downey, et al. [19].)

## 3.2 Type Extractions

Semantic types allow the user to refer to large collections of objects. By placing many objects in, say, the *scientist* type relation, the user can naturally query many facts at once.

Type hierarchies are natural in the real-world (*e.g.*, a `chemist` is a `scientist`) but for simplicity's sake the ExDB type hierarchy has a single level. Where necessary, we flatten the hierarchy (*e.g.*, so that both `chemist` and `scientist` contain `Lavoisier`).

Semantic types are very similar to natural language "hypernyms" or "is-a" relationships. The ExDB prototype extracts these types from Web text using the KnowItAll system [20]. KnowItAll searches the text corpus for phrases that strongly indicate the "is-a" relationship (*e.g.*, "I like **cities such as** Seattle and Boston."), noting the pair of words embedded in each phrase. Each extractions is given a probability based on its frequency, just as with fact extractions.

KnowItAll is limited to facts that appear in well-phrased sentences, so it will fail to extract facts that are only implied by web page layout or typographic practices. For example, it will fail to extract the fact `is-PC-chair(Weikum, CIDR)` from the `CIDR` web page because there is no declarative sentence that explicitly states that it[4].

Because the ExDB is intended to handle Web data in a domain-independent way, we avoid using any specific reference databases (*e.g.*, DBLP) which may allow us to significantly improve extraction quality, as in [9].

## 3.3 Synonymy Extractions

Our prototype uses a large-scale version of the DIRT algorithm to deduce predicate synonyms automatically from a corpus of text [29]. DIRT works by comparing the degree to which the arguments of two predicates coincide. For example, a corpus' worth of argument-pairs to the `invented` and `has-invented` predicates will overlap to a substantial degree. We expect them to share pairs such as `Edison/light-bulb` and `Einstein/theory-of-relativity`, but these pairs are fairly rare for the average predicate. Predicates that strongly share argument pairs are likely synonyms.

[3]It is sometimes possible to use hand-annotated linguistic resources such as WordNet[33] or thesauri. They contain high-quality information, but are often limited in coverage, limited to a certain kind of phrase (*e.g.*, only single-word phrases), and are unavailable in many target languages. Thus in general, algorithmic approaches are indispensable.

[4]The rendered text is, "Program Committee Chair: Gerhard Weikum, MPI Saarbruecken."

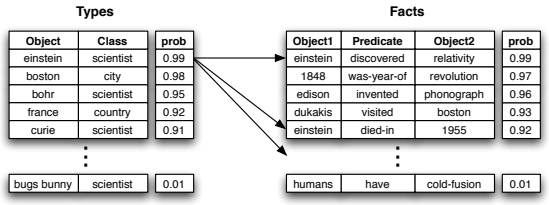


**Figure 3: The physical database schema, performing a join between the `Types` and `Facts` tables.**

We do not yet have object synonyms. There are existing techniques for object synonyms, such as q-grams [23] or Dong, et al.'s solution for reference reconciliation [18].

## 3.4 Inclusion and Functional Dependencies

Inclusion dependencies among predicate tables can be modeled in text by verb specializations, or "troponyms." Our prototype does not yet extract these, nor have we seen an extraction mechanism which does. But it seems possible that an enhanced DIRT algorithm may be able to test for near-subset relationships between textual predicates. It seems that a high-quality troponym extractor should be possible.

Extracting functional relationships from text is probably best considered a subject for data mining research. Rule learning is an active research topic in data mining, and advances in this area would directly benefit the quality of ExDB constraints. However, some initial work suggests that good results are possible by examining how often a given predicate preserves an FD over large numbers of contained object-pairs.

# 4. QUERY PROCESSING

Our Web Data Model is simply a probabilistic database, whose mathematical properties have been discussed elsewhere. Instead, this section covers how the current prototype optimizes query execution, whether or not projections are present.

## 4.1 Non-projecting queries

Queries that contain no projections are fairly straightforward An ExDB query involves a series of joins against tables in the Web Data Model. Following the probabilistic semantics from the MYSTIQ system [15], the probability of a joined tuple is the product of the local tuples' probabilities. (See the simple two-table join in Figure 3.4.) We are usually satisfied with just the top-k tuples (as ranked by probability), so we use top-k queries to try to obtain results as quickly as possible.

Our prototype builds heavily on the work of Natsev, et al., who introduced an algorithm for performing top-k join queries over ordered data sets [34]. For each dimension's score in their system, we employ a tuple's probability; our aggregation function is simply product.

We also improve non-projection queries using the work of Theobald, et al. [39] who observed that a user of top-k ranking systems will typically accept an approximation of the true top-k results. They were able to achieve much lower running times than the standard Threshold Algorithm [21] at the cost of reduced top-k recall.

Similarly, in our prototype we attempt to compute the probability that a partially-processed output tuple (with a partially-computed score) will ever have a high enough score to make it into the top-k. We apply some statistically-derived probability bounds to tell us when to bother with additional processing for a tuple.

## 4.2 Projections

Consider the query `q(?s) :- invented(<scientist> ?s, ?i)`, which should rank scientists by the probability the scientist invented something. (The actual inventions are irrelevant.) A scientist, say `Tesla`, appears in the output of $q$ whenever the tuple `invented(Tesla,` $I_0$`)` is in the database. There may be many inventions, $I_1, \ldots, I_m$, such that `invented(Tesla,` $I_i$`)`. Any of these are sufficient to return Tesla as an answer for $q$.

In the ExDB, these tuples are present only probabilistically. The probability of the query should be the probability that *any* tuple `invented(Tesla,` $I_i$`)` is truly in the database. Unfortunately, this means computing a disjunction of $m$ probabilistic events, and at Web scale $m$ could be extremely large. Computing the correct answer will doom any hope of real-time performance.

Projections also pose a question of semantics that is slightly unsettling. Many of these purported inventions (which we are projecting out) will be the result of very low-probability extractions. These extractions are likely to be inaccurate and perhaps reflect idiosyncratic language use more than any other factor. But since we use disjunction to compute the final tuple probability, a large number of very-low-probability extractions can unexpectedly result in a quite large probability.

What can we do to make projections practical?

**A Panel of Experts** Our current attempt to solve the problems associated with large numbers of contributing terms is the abstraction of a *panel of experts*. An *expert* is a tuple with a score (e.g. `invented(Tesla, Fluorescent-Lighting), 0.95`) that tells the probability a tuple appearing in the output of $q$. We then select a small panel of experts (*e.g.*, 5) that best supports the output tuple. Let $S(\vec{t})$ denote the set of tuples which contributes to the presence of the output tuple $\vec{t}$, then our returned score for a tuple $\omega(\vec{t})$ is given by

$$\omega(\vec{t}) = \max_{L \subseteq S \ \wedge \ |L|=5} \Pr[\bigvee_{l \in L} l]$$

**Effect of the Panel** The small size of the panel eliminates a slew of low quality tuples boosting a poor answer. Computationally, it reduces the calculation to a small number of facts. Two additional benefits of this semantic are that it circumvents known lower bounds [15] about computing output probabilities in the general case, and allows us to place more aggressive bounds on the contributions of future tuples.

The panel semantics offers some advantages and is our current best solution to the problem of computing probabilistic projections at large scale. Although promising, this semantic choice requires much more study to determine its impact on result quality and performance.

**Implementation Issues** We compute join statistics between tables to predict the panel probability, given a prejoin tuple probability. We can use these statistics to determine when the query processor has likely discovered the top-k results.

| | died-in | invented | published | taught |
|---|---|---|---|---|
| Aristotle | 322 BC | logic | records | Alexander |
| Galileo | 1642 | telescope | Dialogue | mathematics |
| Newton | 1727 | calculus | Principia | |
| Johannes Kepler | Regensberg | log books | Rudolphine Tables | |

**Table 2: A possible *synthetic table* about scientists, generated by merging answers from `died-in(⟨scientist⟩ ?x, ?y)`, `born-in(⟨scientist⟩ ?x, ?y)`, `invented(⟨scientist⟩ ?x, ?y)`, `published(⟨scientist⟩ ?x, ?y)`, and `taught(⟨scientist⟩ ?x, ?y)`. All tuples are derived from queries on the working ExDB prototype, though the table was composed by hand.**

# 5. APPLICATIONS

ExDB suggests a number of interesting new applications that make use of structured Web data.

## 5.1 Synthetic Tables

When human designers create schemas, they are designing structures that are useful for database software as well as for human observers trying to make sense of the domain. ExDB's extracted structural elements are not meant to be examined directly, but perhaps an application can use them to build topic-specific tables that a human would appreciate.

In some cases, a simple ExDB star join with appropriately-chosen predicates will create a table relevant to the topic. If the database is sparse, the synthetic-table application may instead choose to merge multiple ExDB queries and accept empty table cells when ExDB lacks a high-probability answer. An interesting research question is how to balance competing notions of what makes a good schema table. For example, a high-quality table should contain relatively few NULL values, but a NULL value might be preferable to including an extremely low-probability tuple.

Table 2 shows a small sample synthetic table, composed with several ExDB queries on our prototype system.

## 5.2 Unstructured Access to Structured Data

Unstructured access to structured data is a well-known database problem, considered by keyword systems such as DISCOVER and DBXplorer [26, 2]. As a search engine does with documents, these systems return structured database tuples that are relevant to the user's query.

We have seen how the ExDB can generate a structured database from text extractions and a user's structured query. If it were possible to generate an ExDB query automatically from keywords, we could build a slightly circuitous but very powerful query system. Its inputs would be Web text and keyword queries, just as with a standard search engine. However, its output would be a structured table of relevant data, not simply a ranked list of documents.

## 5.3 Web Data Warehousing

An ExDB contains a large amount of read-only structured data, so it is natural to consider data warehousing operations such as exploration and visualization. We would like to enable a form of OLAP-like functionality over the ExDB, creating the "Web data cube." The result could be a novel tool for exploring large aggregates of web data, which today can only be explored a single document at a time via keyword-driven search engines.

Doing so will require that we add at least GROUPBY functionality to our query language, and also consider probabilistic query processing for aggregates in more depth.

# 6. PRELIMINARY FINDINGS

Even though our prototype lacks many of our desired ExDB features, we have some promising initial evidence about result quality and execution times.

First, for result quality, examine Table 3. The left-hand side shows the results of the following projection query:

```
q(?s) :- invented(<scientist> ?s, x)
```

It lists scientist names but projects out the inventions themselves. The justification for the projection here is to find practically-minded scientists, not to find the inventions themselves.

On the right side is a listing of results from a Google search query for *scientist invented*. The search query will obviously return just unstructured documents, but we expected the documents to contain some easily-extracted structure.

ExDB did very well. All of the listed scientists in Table 3 had practical inventions as well as scientific discoveries. In contrast, the bottom of the result (not pictured) has a much higher concentration of incorrect answers, including the theoretician Wolfgang Pauli and the biochemist/novelist Asimov. Clearly, these results would have been impossible without knowledge of the `scientist` semantic type and the extracted `invented` predicate.

The Google search for *scientist invented* returns just three pages in the top-ten that contain factual information about multiple scientists, and only item 1 comes close to answering the query. After this document's fourth paragraph, there is an inline bulleted-list of 30 scientific fields and the "father" of each field. Information in this list arguably satisfies the query, but it would still need to be discovered and extracted. Also, the text contains few structural "hints" about the data. Without an extremely sophisticated extractor, it is hard to see how this document can be useful[5]

The query as shown took 100 seconds to execute. With more aggressive use of join statistics in query processing, we can currently achieve a runtime of 22 seconds at a cost of about 1.5 errors in the top-10 result set. (The author Tolkien and the physicist Heisenberg, who was primarily a theoretician, make it into the bottom two positions.)

# 7. ALTERNATIVE MODELS

This paper focuses on one particular model for structured queries over unstructured data. However, it is not the only

[5]Of course, it is probably possible to formulate a search query that is much more elaborate than "scientist invented," and which would yield better results. However, it is not clear how to formulate such a search query, and the problems of structure-extraction still stand. Our point here is only to show that the results for a reasonably-chosen keyword query are very far from meeting the system goals.

| scientist | p |
|---|---|
| Tesla | 0.9987 |
| Galileo | 0.9945 |
| Benjamin Franklin | 0.9935 |
| Newton | 0.9905 |
| Ben Franklin | 0.9905 |
| Gauss | 0.9869 |
| Henry | 0.9817 |
| Isaac Newton | 0.9817 |
| Farnsworth | 0.9817 |
| Leonardo | 0.9756 |

| Rank | Summary |
|---|---|
| 1 | Article on history of science with a religious focus |
| 2 | Many short scientist biographies |
| 3 | Inventors and inventions from 1800s |
| 4 | Article on the inventor of plastic |
| 5 | Page containing a single trivia question |
| 6 | Same page, different format |
| 7 | Discussion board on ancient science |
| 8 | News article about the early German telephone |
| 9 | Two sentences on the thermometer |
| 10 | Summary of an article about cat allergies |

**Table 3: Top-10 prototype-**ExDB **results for** `q(?s) :- invented(⟨scientist⟩ ?s, ?x)`**, and Google search results for *scientist invented*. The goal is to retrieve a list of practical-minded scientists. Only one document returned by Google arguably contains an answer to the query; it is still embedded in unstructured text. Note that some of the** ExDB **entries are duplicates and should be merged; object synonyms will make this possible.**

plausible one. In this section we describe two alternative models which take somewhat more extreme places in the design space. In the **Schema Extraction Model**, we propose to compute a single best schema for the entire set of input extractions; after doing so, the text has been transformed into a traditional relational database. In the **Text Query Model**, the query system does not perform any information extraction at all, but instead offers users a query language that blends extractive and structural elements.

## 7.1 Schema Extraction Model

The Schema Extraction Model attempts to derive a single "best" schema for an input set of extractions. It then populates the schema with the extractions to generate a relational database that can be queried using standard SQL. The work flow for this model is shown in Figure 4. As in the ExDB system, we start with a set of Web text and run IE over them. Unlike the ExDB, we do not preserve any ambiguity about the extracted data or structure.

We assume a slightly different extraction model from that of ExDB. Rather than derive a series of semantic relationships (*i.e.*, those listed in Table 1), the IE system simply emits a series of objects with associated values. These emitted items are similar to the **facts** table in the Web Data Model. For example, the object `Edison` might have values `phonograph`, `Menlo-Park`, etc. Each value has at least one attribute label (*e.g.*, `invention` for `phonograph`). We call these items **Unstructured Tuples**. We apply a score threshold to tuples emitted by the extractor, removing unlikely ones; after thresholding, we do not treat the remainder probabilistically.

Although most objects in a domain will contain similar information, the difficulties of human authors and unreliable IE systems mean that there will inevitably be missing values and off-topic extraneous ones. Assembling a good schema from this set of messy extractions will be very difficult, even for a human designer. We expect that in most cases, the best schema will be "noisy"; for example, the schema might be very easy to understand but might lack a few extracted attributes.

Noisiness opens several competing design criteria for relational schemas, and an instance of the Schema Extraction Model should be able to make compromises among them. It may decide to drop some extracted values rather than complicate the schema with additional columns. It may also decide to fill some cells with NULL rather than split a table into two separate smaller ones. We believe that three good criteria are: simplicity (the output has few tables), completeness (all extractions from text appear in the output) and fullness (the output database has no NULLs).

If we define the costs of violating these criteria (*e.g.*, we give a score to the "badness" of complicating the schema with an additional table) then creating a schema for all of the Web is simply a matter of cost minimization.

Note that this problem of choosing one schema over another also appears in the "synthetic table" application from Section 5.1.

## 7.2 Text Query Model

Figure 5 shows the simple Text Query work flow. It simply indexes Web text for later querying. Unlike the Schema Extraction Model, we do not attempt any extraction at all. The user's query provides all the needed information. The role of the query system is simply to provide an expressive language that generates answers quickly.

As an example, consider a user who would like to check whether a favorite band is playing nearby. In one query, she can: a) extract the city/date tuples from the band's website, b) indicate the city where she lives, c) compute the dates when the band's city and her own city are within 100 miles of each other:

```
SELECT bandCity, bandDate
FROM ("http://thebandilike.com/**",
      ["to appear in <string> on <date>",
                    bandCity, bandDate])
WHERE
bandDate > 2006 AND
geographicdist(bandCity, "Seattle") =< 100
```

The `FROM` clause here indicates a relevant set of web pages, a regular expression that can be applied to the text to generate a table with two columns plus labels for those columns. The `WHERE` clause tests the date to see if it is valid, declares the user's current city, and uses a built-in function to measure the distance between two cities.

Others have proposed Web text query languages, such as Squeal, WebSQL, and W3QL [38, 32, 28]. These languages are quite elaborate and some can express a query similar to the one above. However, these systems were not designed for good query-time performance. For example, in all three, executing a query may entail actually fetching remote pages or finding URLs via remote search engines (as opposed to simply using prefetched and preindexed pages).

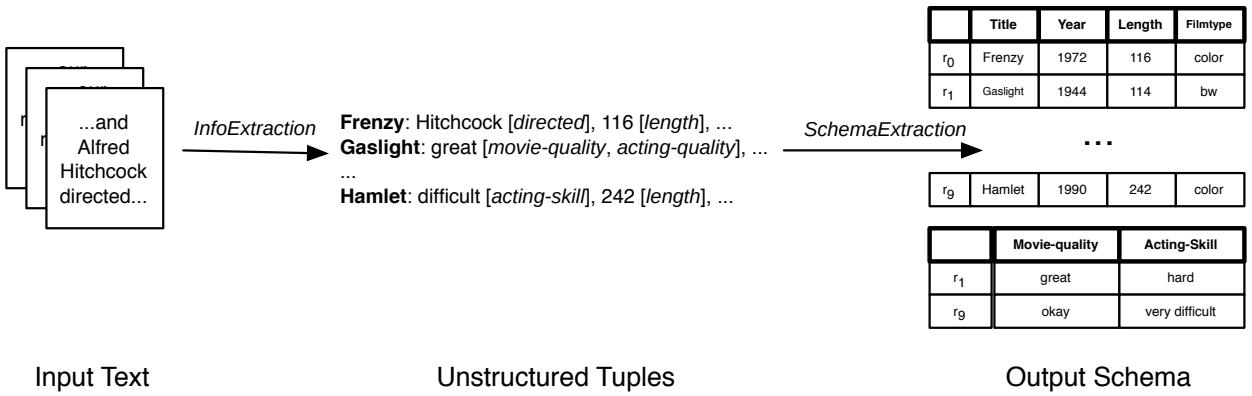


**Figure 4: Construction pipeline for the Schema Extraction Model. We still run an IE system over downloaded text, but use the resulting extractions to compute a single traditional relational database.**

In contrast, we would like to construct a system that can scale to a large number of users while providing search engine-like response times. These queries are more complicated than just keywords, and so we expect that users would want to interactively edit queries and see immediate results.

There are tantalizing opportunities for query optimization when we consider both the extractive and structural query components. If there are no values of `bandDate` greater than `2006`, then ideally we would avoid half the extraction work and all of the work of a call to `geographicdist`.

Unfortunately, neither a search engine's inverted index nor a standard relational index can solve this problem efficiently. With an inverted index, we cannot retrieve any extracted values without incurring an I/O to fetch the entire original text; even if the `bandDate` test fails for every item in the database, we do not avoid any significant extraction work. With a standard relational database, any index based on the extracted tuples would need to be recomputed (over the entire Web corpus) whenever a user enters a new extraction clause.

We believe that text indexing techniques such as the *neighbor index* and the *multigram index* could be very helpful to an implementation of the Text Query Model [8, 11].

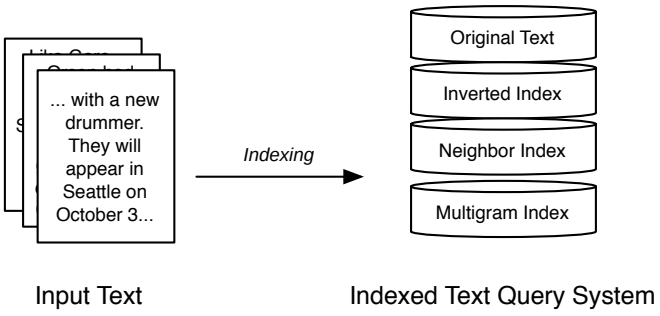


**Figure 5: The simple construction pipeline for the Text Query Model.**

## 8. RELATED WORK

The Information Extraction literature is clearly critical to a useful ExDB; we have cited a substantial amount of work above. Good surveys on research in IE are available [12, 17].

Halevy, et al. proposed that structural elements are critical when managing non-traditional data, largely in the context of schema matching [25]. Many concepts from schema matching could be used in our system, to refine the structure after extraction.

Liu, et al. recently suggested using *triple queries*, very similar to our extracted predicate facts, to perform structured queries on unstructured data [30]. Their "query graph" may be similar to a join-graph performed across our extracted structures, though there is no probabilistic interpretation for the query graph.

Gubanov and Bernstein described a TDBMS, or Text Database Management System [24]. The TDBMS allows users to perform textual versions of selection and join queries. A set of sentences is the basic handled object: the result of a selection query is a sentence-set relevant to a given topic, and two joined sentence-sets cover the same topic. The TDBMS parser finds sentences that are similar to the objects and predicates in ExDB, and TDBMS makes use of synonyms. However, data items are always sets of sentences rather than data values extracted from the text, limiting the usefulness of the query output.

Chang, et al.'s MetaQuerier for deep-web querying uses an extractor that assumes a strong underlying schema; that schema helps in the extraction process [10]. While MetaQuerier is designed to be scalable to the entire web, the schema matching component is only used in certain domain subject areas. It is intriguing to consider a similar scheme with ExDB, in which structural information is fed to the IE components to improve their accuracy.

## 9. CONCLUSIONS

Despite the implicit structural information that abounds in Web text, modern search engines do not offer any kind of structured query service. We have suggested a system that uses IE to make the structure in text available to query writers.

We believe an ExDB-like system would be an important tool for querying the Web. Our work so far suggests a number of interesting research directions across information extraction, data modeling, and probabilistic query processing.

By extracting data, schemas, and constraints from text, we believe the database community can someday offer powerful new tools for applying structured queries to unstructured data.

**Acknowledgments** This work was supported in part by Suciu's NSF CAREER grant IIS-00992955 and NSF grants IIS-0428168, IIS-0513877, IIS-0535284, and IIS-0312988. It was also supported by DARPA contract NBCHD030010, ONR grant N00014-02-1-0324, the University of Washington's Turing Center, as well as gifts from Google.

## 10. REFERENCES

[1] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the 2001 ACM SIGMOD International Conference on Digital Libraries*, 2000.

[2] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *ICDE*, 2002.

[3] P. Andritsos, A. Fuxman, and R. Miller. Clean answers over dirty databases: A probabilistic approach. In *ICDE*, 2006.

[4] M. Banko, M. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *IJCAI*, 2007.

[5] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *VLDB*, 2006.

[6] S. Brin. Extracting patterns and relations from the world wide web. In *WebDB Workshop at EDBT '98*, 1998.

[7] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.

[8] M. Cafarella and O. Etzioni. A search engine for natural language applications. In *WWW*, 2005.

[9] A. Chandel, P. C. Nagesh, and S. Sarawagi. Efficient batch top-k search for dictionary-based entity recognition. In *ICDE*, 2006.

[10] K. C.-C. Chang, B. He, and Z. Zhang. Toward large scale integration: Building a metaquerier over databases on the web. In *CIDR*, pages 44–55, 2005.

[11] J. Cho and S. Rajagopalan. A fast regular expression indexing engine. In *ICDE*, 2002.

[12] W. Cohen. Information extraction and integration: An overview, 2004.

[13] Y. Cui and J. Widom. Practical lineage tracing in data warehouses. In *ICDE*, pages 367–378, 2000.

[14] N. Dalvi, C. Ré, and D. Suciu. Query evaluation on probabilistic databases. In *IEEE Data Engineering Bulletin*, 2006.

[15] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.

[16] A. Doan, R. Ramakrishnan, F. Chen, P. DeRose, Y. Lee, R. McCann, M. Sayyadian, and W. Shen. Community information management. In *IEEE Data Engineering Bulletin*, 2006.

[17] A. Doan, R. Ramakrishnan, and S. Vaithyanathan. Managing information extraction, 2006.

[18] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD*, 2005.

[19] D. Downey, S. Soderland, and O. Etzioni. A probabilistic model of redundancy in information extraction. In *IJCAI*, 2005.

[20] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1):91–134, 2005.

[21] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[22] C. Fellbaum. English verbs as a semantic net. *International Journal of Lexicography*, 3(4):278–301, 1990.

[23] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, 2001.

[24] M. Gubanov and P. A. Bernstein. Structural text search and comparison using automatically extracted schema. In *WebDB*, 2006.

[25] A. Y. Halevy, O. Etzioni, A. Doan, Z. G. Ives, J. Madhavan, L. McDowell, and I. Tatarinov. Crossing the structure chasm. In *CIDR*, 2003.

[26] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, 2002.

[27] T. Jayram, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar information extraction system. In *IEEE Data Engineering Bulletin*, 2006.

[28] D. Konopnicki and O. Shmueli. W3QS - A System for WWW Querying. In *13th International Conference on Data Engineering (ICDE'97)*, 1997.

[29] D. Lin and P. Pantel. Discovery of inference rules from text. In *KDD*, pages 323–328, 2001.

[30] J. Liu, X. Dong, and A. Halevy. Answering structured queries on unstructured data. In *WebDB*, 2006.

[31] I. R. Mansuri and S. Sarawagi. Integrating unstructured data into relational databases. In *ICDE*, 2006.

[32] A. O. Mendelzon, G. A. Mihalia, and T. Milo. Querying the World Wide Web. *International Journal on Digital Libraries*, 1996.

[33] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. Introduction to wordnet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–312, 1990.

[34] A. Natsev, Y.-C. Chang, J. R. Smith, C.-S. Li, and J. S. Vitter. Supporting incremental join queries on ranked inputs. In *VLDB*, pages 281–290, 2001.

[35] D. V. K. Reynold Cheng and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, pages 551–562, 2003.

[36] A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, April 2006.

[37] Y. Shinyama and S. Sekine. Preemptive information extraction using unrestricted relation discovery. In *Proceedings of the Human Language Technology and*

*North American Chapter of Association of Computational Linguistics Conference (HLT/NAACL-06)*, 2006.

[38] E. Spertus and L. A. Stein. Squeal: A Structured Query Language for the Web. In *WWW*, pages 95–103, 2000.

[39] M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In *VLDB*, pages 648–659, 2004.

[40] P. D. Turney. Expressing implicit semantic relations without supervision. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL-2006)*, 2006.

[41] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.