INDREX: In-Database Distributional Relation Extraction

Torsten Kilias, Alexander Löser
Technische Universität Berlin
Strasse des 17. Juni
Berlin, Germany
firstname.lastname@tu-berlin.de

Periklis Andritsos
University of Toronto
140 St. George Street
Toronto, Canada
periklis.andritsos@utoronto.ca

ABSTRACT

Relation extraction transforms the textual representation of a relationship into to the relational model of a data warehouse. Early systems, such as SystemT by IBM or the open source system GATE solve this task with handcrafted rule sets that the system executes document-by-document. Thereby the user must execute a highly interactive and iterative process of reading a document, of expressing rules, of testing these rules on the next document and of refining rules. Until now, these systems do neither leverage the full potential of built-in declarative query languages nor the indexing and query optimization techniques of a modern RDBMS that would enable a user interactive rule refinement *across documents* and on the *entire corpus*. We propose the INDREX system that enables a user for the first time to describe corpus-wide extraction tasks in a declarative language and permits the user to run interactive rule refinement queries. For enabling this powerful functionality we extend a standard PostgreSQL with a set of white-box user-defined-functions that enable corpus-wide transformations from sentences into relations. We store the text corpus and rules in the same RDBMS that already holds domain specific structured data. As a result, (1) the user can leverage this data to further adapt rules to the target domain, (2) the user does not need an additional system for rule extraction and (3) the INDREX system can leverage the full power of built-in indexing and query optimization techniques of the underlaying RDBMS. In a preliminary study we report on the feasibility of this disruptive approach and show multiple queries in INDREX on the REUTERS-News'97 corpora.

1. INTRODUCTION

For more than 30 years, relational database management systems (RDBMS), provide the necessary infrastructure for efficient storage of structured information and declarative query processing. With the addition of indexing and optimization techniques, RDBMS' are able to support point as well as range queries over large databases. However, today a large part of the information is produced by humans in textual form. That information might by derived from different sources, such as the Web, corporate intranets or emails. However, there are no solutions that permit declarative queries on top of textual data *and* RDBMS in a unified fashion, which would require an intersection of the capabilities of RDBMS' and systems for executing the task of extracting relational data from large text corpora. As a consequence, it is not possible to combine information extraction queries that operate on both systems at the same time.

Typical queries for the relation extraction task. Consider the document `"Torsten, 25, is a student of TUB"`. Although this is really a simple sentence, we will treat it as a document for our purposes. Having such sentences is a reality if, for instance, we deal with project proposals the professors are writing in a University environment. If we assume more sentences like this, we can build efficient inverted-indexes that store information about the documents in which different keywords appear. It is, however, very difficult to answer distributional queries across the size of a single document or the size of the whole corpus of documents that we have at hand. Such a query would ask `What is the distribution of the keyword "Torsten" in "University" contexts`. This query is binary in that it asks for the frequency of the keyword `Torsten` in `University` contexts. Moreover, it asks for a relationship that cannot be easily induced by NLP tools. Namely, the fact that we are asking about `University`, leads to an inherent assumption that `TUB` is a `University`. This can only be discovered if `TUB` exists in a different source that contains this relationship.

For such queries, the desired situation would be to store a detailed view of the input document in a relation in order to be able to directly query it or join it with other existing relations. This would enable us to combine information either at the value level (text) or higher level (*e.g.* syntactic features) within the same and across different documents. This way a Natural Language specialist, beyond applying specific external tools to produce document annotations, she may also discover what are the most important signals in the document and can use the information extraction in a declarative fashion (*e.g.* by applying SQL queries) to infer rules. For example, a rule that may be extracted and deemed significant from a corpus of proposals that professors write, is that students having a first name `Torsten` work or study exclusively in German universities.

Our contribution. In this work, we extract relations from text and store it in an effective way for declaratively querying it along with other side information. We make contributions towards unifying textual and relational information in one model and present information extraction queries and operations that can be performed on top of them in a unifying fashion. More concretely, we make the following contributions:

1. We propose a new data model for textual data in RDBMS. Our model is able to host plain text as well as several annotations that can be extracted from it. Moreover, it is flexible enough to hold all necessary structures for the relation ex-

traction task, such as document structures, shallow and deep syntactic structures, structures for the task of open information extraction, structures for representing clustered relation synsets and structures for expressing the semantic meaning of relation attributes and relation names.

2. Based on this model we show how we perform in-database information extraction. Our approach has the important advantage that extraction operations, such as scan, index and optimize, can access both kinds of tables within the same system: tables that hold textual data and tables that hold existing relational data (such as tables in a CRM or ERP system). As a result, the domain expert can augment and combine information from different representations into a new table with built-in join operations from the RDBMS. Therefore, the domain expert benefits from more than 30 years of research in declarative query processing. The expert can focus on the task description and the system takes care of the task execution and optimization. Moreover, built-in optimizations of the RDBMS enable a high interactivity at the level of the single document, but also at the level of the entire corpus.

3. For realizing this powerful functionality, we extend a POSTgreSQL system with a set of user defined functions (UDF). We develop these UDFs as 'white-box' functions so the PostgreSQL RDBMS can fully process and optimize them. In our study, we report the query experience of our UDF-based abstractions, show first execution results and discuss potential extensions toward distributed and column-based databases (such as Cloudera's IMPALA) or In-Memory data bases.

The paper is organized as follows: In Section 2 we recapture the domain-independent and domain-dependent elements of the relation extraction stack, describe the iterative relation extraction process and review existing work on document-by-document declarative relation extraction . Next, in Section 3, we describe the novel tasks of (1) corpus-wide declarative relation extraction, and, (2) joining existing relations with text-based information. Then, we propose our novel data model, list our abstraction of user defined functions for executing this task and give an overview on our implementation in PostgreSQL. In Section **??**, we comment on the feasibility of our data model and the UDF-based query abstraction layer and discuss potential extensions. Finally, in Section **??**, we conclude our work.

2. RELATED WORK

We abstract the process of relation extraction as a multi-label multi-class classification task. Given a set of document-specific, surface, syntactic, deep syntactic, corpus-specific and domain-specific features the classifier determines occurrences in text (such as sequences of consecutive and non-consecutive characters and tokens) that likely represent a relation of a particular semantic type. In this Section we present relevant work around algorithms for commuting required features and interactivity for adapting these features to a domain. Finally, we present existing RDBMS techniques for executing this task.

2.1 Understanding Relation Extraction

For identifying these features and their interplay (aka. conditional dependencies) the complex task of relation extraction requires several base extraction functionalities that depend on each other: First, the software needs to recognize document specific structures. We focus here on document structures that include natural language sentences and paragraphs. From these common structures the software will extract shallow syntax, deep syntax [6] and open information extraction [15]. Given these syntactic structures the software can determine un-typed binary [15] and higher order [13, 1] candidate relations and candidate arguments. Next, the system clusters likely synonymous relation candidates with the help of corpus specific distributions into so called synsets [3, 26, 20]. Finally, these synset clusters are further adapted towards the target schema through appropriate human interactions. A system could implement these domain adaption procedures through an active learning procedure [29] or through rule writing environments [11]. In both cases the human requires to overview corpus-wide distributions to learn common signals for the target domain. Furthermore, the human will often pre-select these candidate relations and synsets with the help of additional domain specific data [28], such as existing data from an Enterprise Resource Planning (ERP), Customer Relationships Management (CRM) or Sales Management (SM) system.

2.2 Iterative Domain Adaptation Process

Discovering relations is an iterative task which contains *lookup*, *learn* and *explore* activities. This simple abstraction was recognized and published first by Bloom in 1956 [8]. Later, different disciplines enriched this abstraction model. Authors of [27] refined *lookup* activities into navigational, informational and transactional activities. Furthermore, the work in the context of service search by [25] gives example operators for each activity. For instance, the author considers aggregation, comparison and integration as activities for *learn* and analysis, exclusion and transformation as activities for *explore*. Most recently, authors of [5] apply the original ideas of Bloom to the problem of explorative data and text mining. Given our stack from Figure 1 we abstract this process into a step of an initial sequence of document and language specific transformations and an iterative process of domain specific abstraction transformations.

2.3 Declarative Relation Extraction

Declarative relation extraction enables domain experts to adapt existing and create new extraction rules. This way, similar to SQL and RDBMS technology, the system takes care of optimizing the declarative query. As a result, the domain expert can focus on the task of domain adaptation only. Moreover, some declarative languages have a similar syntax as SQL, therefore these languages often do not require an expensive training phase for most analysts.

Document-by-document extraction. Authors of *UIMA* [17] describe a software architecture for Unstructured Information Management. Similar to the staging area in a data warehouse, they define roles, interfaces and communications of large-grained components essential for transformation steps in natural language processing. Authors of [22] were among the first to recognize the power of declarative languages for NLP domain adoption tasks. They propose an SQL-likish language called AQL that bases on the principle of a span, which is basically an occupance of a string in a document with a single or multiple semantic meaningful labels. Humans can define AQL extractors through rules. An engine called System-T executes these rules. The AQL language provides user defined functions and predicates for comparison and combination of multiple spans. These functions and predicates enable the users to combine multiple basic extractors into a combined and complex extractor. Contrary to *INDREX*, System-T executes AQL queries per document only and does not provide queries that deliver distributions across documents. Hence, AQL and System-T can only rely on document wide distributional information for extraction tasks. As a result, the user still needs to aggregate distributions (such as

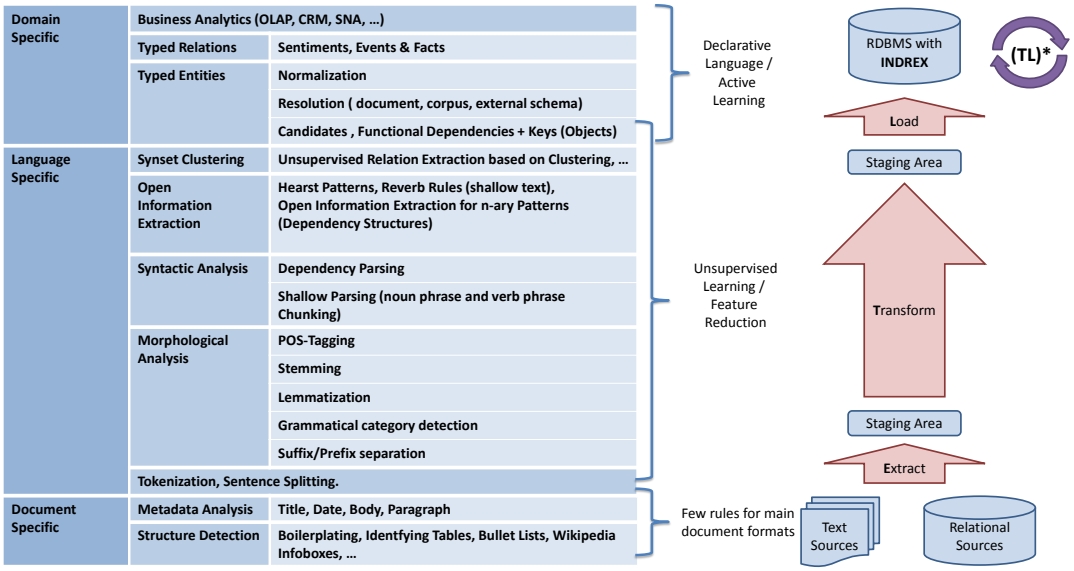

Figure 1: Transformation steps for relation extraction. For the task of domain dependent relation extraction, we abstract document and language adaptation as a linear process and domain adaptation as a interactive and iterative process. This abstraction is true in most scenarios where non-NLP persons bring in domain knowledge on fixed corpus. In scenarios where the goal is to improve NLP techniques, these domain dependent interactions may also trigger learning tasks on the syntactic and document specific layer; for example, authors of [24] consider human click behavior for retrieving only fact-rich documents and authors of [19] consider human click behavior for adapting a part-of-speech tagger.

in an RDBMS) from each document into a single view. *INDREX* overcomes this important limitation by executing the domain specific information extraction task inside the database system while still leveraging the power of a declarative query language.

Optimization strategies for text-joins. The authors of SQOUT [21] assume existing extractors that create views where each view represents the textual content that describes tuples of a single relationship type in the text. The user can integrate these views with select-project-join queries, while the SQOUT system optimizes join processing. We consider the SQOUT system as an orthogonal optimization for a specific join scenario that might further speed up query processing for NLP tasks in a RDBMS. Authors of [14] propose another join optimization and join selection strategy, while authors of [10] discuss optimization strategies for main memory data bases and Hadoop clusters. Finally, authors of GRAFT [7] propose another set of scoring-based optimization strategies in the presence of an index.

All-in-one-system. Most similar to our work is the system proposed by the authors of [30]. Their work is based on the semi-structured data model and propose a parse tree database where they hold dependency tagged sentences and a query language for selecting subtrees that likely indicate a relation. For fast retrieval of relevant subtrees they use an additional key value index called *Lucene*. In contrast, the abstraction in INDREX is based on the relational data model that allows us to leverage the full spectrum of existing RDBMS and data warehouse technology, including main memory and distributed databases with existing adaptors for data integration for a large number of text data sources.

3. DISTRIBUTIONAL EXTRACTION

We focus on the task of transferring textual information into relations. INDREX explores distributions of relations and signals within a single text document, across a corpus of many documents, as well as between documents stored in relations with other existing relational data or distributions. Such joins and aggregations enable NLP-engineers and domain experts to discover new rules and augment existing data with relations from text. In this Section, we describe the data model and operators that enable this functionality.

3.1 Requirements and Tasks

Processing different text abstractions per document. Relation extractors process character, token, syntactic, tree-based and lexical signals from text (see Section 2.1). Interestingly, we observed that commercial vendors of NLP technology often do not agree on a single modernization standard to enforce a vendor lock-in with their customizers. Ideally, our data model supports input and output interfaces of character-, token- and hyperedge-based annotations. In figure 2, we show three example sentences with character- and token-based sentence, syntactic, deep-syntactic and semantic (entities and a ternary relationship type) annotations.

Exploring corpus wide document distributions. Beyond the diversity of text abstractions, the system should permit the domain expert to explore and understand *aggregations and distributions* across these annotations with the goal of unraveling document- and corpus-wide commonalities among sentences. Consider for instance `n` *sentence-annotated*. If the string *holds the position of* is considered a synonym of `is a`, a domain expert can aggregate from the corpus-wide distribution of the two strings *works as (1x)*, and *is a (2x)*. Next, from this distribution a domain expert could derive that these strings are likely synonymous expressions for the relationship type *person-position* for these three documents. Moreover, we may also derive useful clues as to how age is expressed. In all three sentences, the age is syntactically expressed as an apposition of the argument of the semantic type *person* and is expressed lexically by commas, brackets and the keyword *years*.

Augmenting text with signals from other sources. Finally, the domain expert should be able to *join equal or similar strings* from underlying text with information in relational tables. For instance, in figure 2 the domain expert could execute a hash-join or fuzzy-

join over likely synonymous phrases "works as", "is a" with strings over existing synonym dictionaries, like `WordNet`[1]. The unsupervised extraction, domain adaptation and complementation of such synonym dictionaries is an area of active research in computational linguistics [3, 26, 20]. Moreover, the domain expert could complement existing relational data (such as data in an ERP or CRM system) with information from text. Consider again figure 2: Here the relational data consists of a table salary that contains also a column `position`. The domain expert could join the extracted relations from the text over the position attribute with complementary information from the relational database.

3.2 Data Model

Our model uses the relational data model with three data types, namely the *span*, *annotation* and *relations*.

Spans.

We use the term *string* to represent our textual data. We then represent strings as character sequences and assign a unique ID to every sequence. A logical document, such as an email or a web page, consists of one or more strings. We use *spans* to mark parts of a string. They point to an interval of characters. In our model, a span consists of two parts. The first part is the character-based span. A character-based span consists of the string ID, *strID*, and the position of the first character, b, and the last character, e.

Definition 1. *A character-span is the 3-tuple* $csp = (strID, b, e) \in CSP$, *where*

$$CSP = \left\{ (strID, b, e) \;\middle|\; \begin{array}{l} getStr(strID) \in ST, \\ b, e \in \mathbb{N}, \\ 0 \leq b \leq e < length(t) \end{array} \right\}$$

In addition to the definition of the span in other systems, like IBM's SystemT [11, 22], we also store an optional *segment-span* that may hold a complete and non-overlapping segmentation of the string, such as the output of a tokenization or sentence-splitting operation on a character span.

Definition 2. *A segment-span* $ssp = (segID, b, e)) \in CSP$ *is a 3-tuple referring to a segment within a text t with*

$$SSP = \left\{ (segID, b, e) \;\middle|\; \begin{array}{l} segID \in SEG, \\ b, e \in \mathbb{N}, \\ 0 \leq b \leq e \end{array} \right\}$$

Depending on the NLP task, the NLP-operator will require either character- or segment-based spans or, it may require both span types.

Definition 3. *A span is a tuple* $sp = (csp, ssp) \in SP$, *with* $csp \in CSP \wedge ssp \in SSP$.

Consider figure 2. The character-span {26,0,6} represents the fact that in document with ID 26, keyword *Torsten* appears between character positions 0 and 6. At the same time the segmentation-span {26,0,0} represents the fact that the same string is at token position 0.

Annotations.

Until now, systems like IBM's SystemT [11, 22], model a *tuple* as a finite sequence of w spans $s_1, ..., s_w$. We extend this model with the new data type, called *annotation*. Annotations now permit complex corpus-wide annotations to be stored for resolving logical entities, document-wide annotations for storing dependency trees and annotations for storing n-ary relations with non-sequential spans. Consequently, an annotation in INDREX may assign a single meaning to a single span or a single meaning to an array of spans.

[1] http://wordnet.princeton.edu

Meaning.

In addition to annotations, we define meanings, denoted by M, as the possible complex values assigned to a type. For instance, in figure 2 the string *POS* denotes the (syntactic) type and *NNP* denotes the value that a computational linguist would assign to a proper noun. Similarly, *Token* expresses the type and *Torsten* the string value. Typically the extractor component, such as a tokenizer or a part of speech tagger, assigns these meanings. Queries that produce new annotation can also assign meanings.

Definition 4. *An annotation* $an \in AN = SP^+ \times M$ *is an n-tuple of spans with a meaning.* SP^+ *is the transfer of the Kleene Plus*

$$SP^+ = \bigcup_{i=1}^{n} SP_i = SP_1 \cup SP_2 \cup SP_3 \cup \ldots$$

Assigning multiple meanings to a group of spans.

An annotation can consist of multiple spans. This group of spans might have different meanings. In figure 2, the annotations for OIE-relation candidates and the semantic relations share the same spans, but contain different meanings. If we assign different meanings to a group of spans, we create different annotations.

For example, the character string of an entire Web page represents an annotation of the type *Web page*. If we consider figure 2, the character-span {26,0,6} is assigned to the syntactic annotation *POS:NNP*. The character-span {26,0,13} is assigned to another syntactic annotation *Phrases:NP* and to the semantic annotation *Entities:Person*. Dependencies give us a more detailed syntactic information about the function of a word in the sentence and the relationships among words. They permit a more precise extraction of OIE relation candidates [1]. A dependency is an annotation with two spans. For example, the two spans, {26,31,37} and {26,8,13} are the start and end points of the *Dependency:nsubj*, respectively. Open Information Extraction Systems, such as [1], extract untyped n-ary relations. The following spans build in figure 2 a OIE relation candidate: {26,0,13}, {26,16,17}, {26,25,37} are the arguments of the relation and {26,20,23} the predicate.

Relation.

A relation in INDREX is a multi-set of tuples, similar to a standard relational table in an RDBMS. All tuples of a relation share the same schema and its attributes store different data types, such as annotations. This, for example, permits the join between multiple annotations. Moreover, each operator in our algebra takes zero or more relations as input and produces a single relation as output. We use the standard definition of the relational model, first described in [12].

Definition 5. *For the given domains* $S_1, \ldots, S_n$, *a relation R is a multi-set over the subset of* $S_1 \times S_2 \times \cdots \times S_n$. *A tuple is, then, defined as an element r of R.*

We add to this definition the new domain annotations, AN, and spans, SP.

3.3 What is the 'right' abstraction model?

Studying abstraction principles. Finding the right abstraction level for 'declarative programming' on both, text and relational data, is a difficult task. For solving this task we conducted a preliminary user study with fifteen master students of computer science. We selected a random sample of thousand documents from

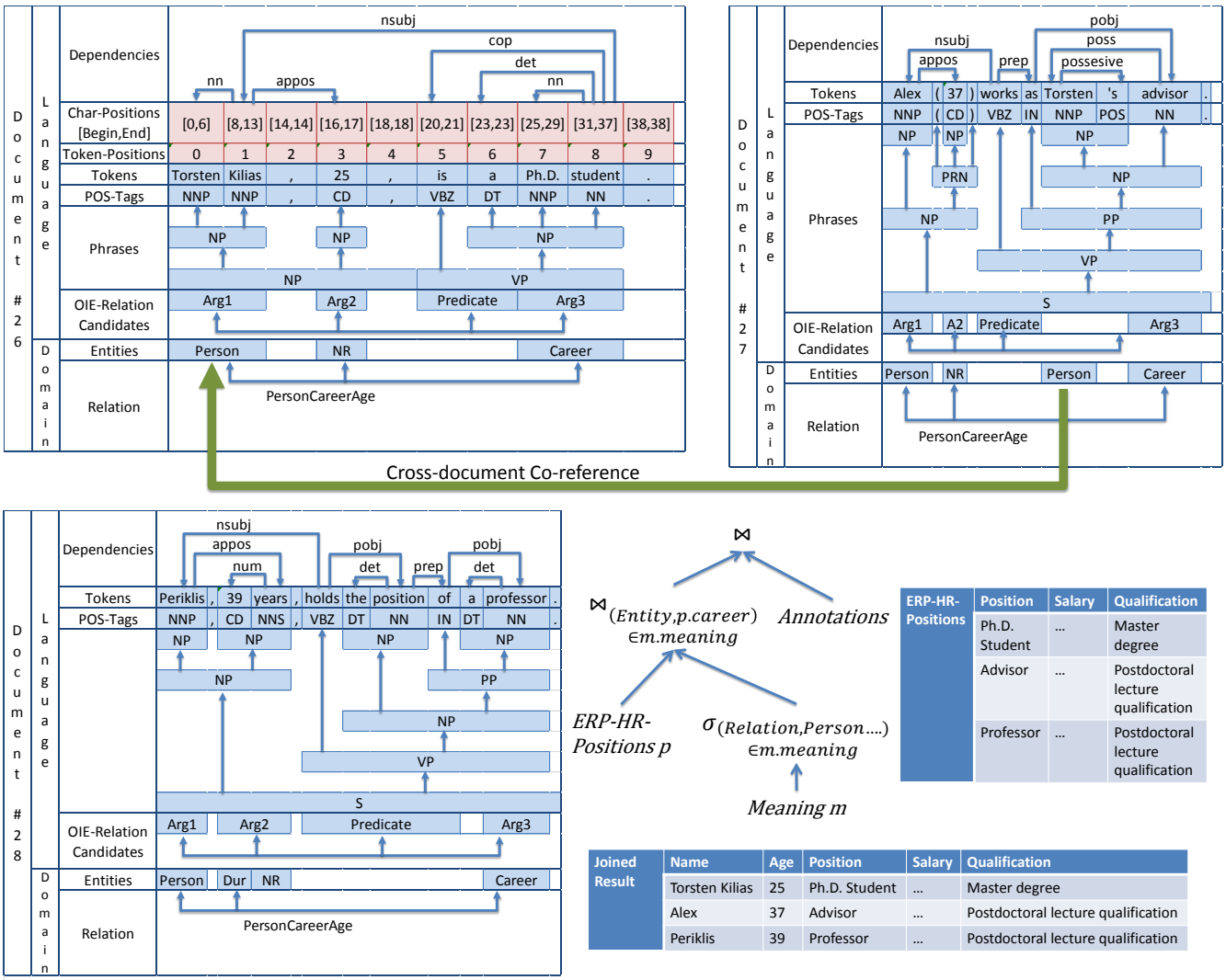

Figure 2: Three documents with the most common language specific annotations, such as tokens, part-of-speech (POS), phrases, dependencies, constituencies (between the phrases), oie-relation candidates, entities and semantic relations. Each horizontal bar represents an annotation with one span and each arrow an annotation with multiple spans. The semantic relation `PersonCareerAge` is an annotation with three spans. The entities "Torsten Kilias" in document 25 and "Torsten" in document 26 are connected by a cross-document co-reference annotation. The query in the right-bottom corner shows a join between a Human Resource relation in a CRM system with the extracted semantic relations from text.

the Reuters NIST new corpus'97, which is a standard evaluation corpus in the information retrieval community. Following the relation extraction stack from Figure 1 we applied language depended taggers from the Stanford CoreNLP pipeline [2] over these documents, including sentence taggers, shallow and deep syntax taggers and the Stanford 7-Class NER tagger. Finally, we applied the open information extraction system REVERB [16] to each sentence to identify relation candidates. Next, we loaded the text data and annotated data in a PostgreSQL Version 9.1 that implemented our data model from the previous section. Then we asked our users to create two relation extractors with that environment:

- Task 'Person-Age-Extractor': Show the distribution of ages assigned to persons in the corpus. Use appositions that link to the entity type *person* via a comma. For executing this task the user needs to identify sentences that contain a person and a apposition that is directly linked in the dependency structure to the person. Moreover the user needs to filter out all appositions that do not contain a number.

- Task 'Company-Acquisition Extractor': First, the user needs to select all sentences that contain two companies that represent arguments in a candidate relation from an open information extraction framework (see also Section 2). Next, the user needs to identify the distribution of predicates for these candidate relations and needs to join this distribution against a pre-computed synset dictionary with words that likely express an acquisition. Computing such dictionaries is a current research task in the computational linguistics community (see also Section 2).

We chose these extraction tasks since they forced our users to master typical hard text mining problems, such as understanding the concept of lexical and deep syntactic analysis, understanding

[2] *nlp.stanford.edu/software/corenlp.shtml*

```
SELECT
        consolidate_phrases(nouns.spans[1]))
FROM (
        SELECT * FROM pos_nnp
        UNION
        SELECT * FROM pos_nnps
        UNION
        SELECT * FROM pos_nn
        UNION
        SELECT * FROM pos_nns
)
```

Listing 1: The task of this query is to seek in the text for sequences of nouns. Our syntax tagger follows the Penn Treebank type system and distinguishes between four classes of nouns. The query forwards the union of potentially overlapping spans for each noun class to the consolidate operator that applies a common heuristic for Latin and Indo German languages and outputs in the case of overlapping spans the left longest span.

the role of pre-computed open information extraction candidates, reusing existing information from synset dictionaries and executing these these tasks with the standard relational algebra.

INDREX features three operator categories only. During this preliminary study we could identify that our users required six categories of operators in INDREX: (1) Operators for executing per-document queries to create and refine initial candidate relations, (2) relational operators for exploring corpus-wide distributional semantics that enable the user refining rules for candidate relations, (3) join operators for augmenting annotations with domain semantics, such as semantics from rule dictionaries, from synset dictionaries or from multi word values in existing relational data. We will introduce these operators and will give examples in the reminder of this section.

3.4 Per-Document Extraction Queries

These operators enable the user important tasks, such as extracting non-overlapping spans that may likely represent candidate arguments for a relation or spans that likely give clues for the name of the relation. We provide operators for executing span, consolidation and regular expression queries.

Span and consolidation queries. We abstract span sequences in text data as intervals and extend standard relational algebra with operations from Allen's interval algebra [4] in order to manage intervals of spans. Rule languages that are based on the interval algebra cannot express complex nested and overlapping structures in a document. To overcome this problem we also implemented consolidation operators [22]. These operators take as an input a set of annotations and output a consolidated single annotation. We implement consolidation operators as *table generating aggregation functions*. Listing 1 illustrates the usage of span and consolidation operators.

Translating regular expression queries into SQL queries. Computational linguists utilize regular expressions for decades to express arguments and relations for lexical and shallow syntactic character sequences (see for example [15, 20]). Consequently, we provide a tool that translates regular expressions on characters, tokens or phrases and the combination of these elements with concatenation, alternation and the Kleene star to span and consolidation Queries. Consider again listing 1, which shows such a translation for the token regular expression (NNP | NNP | NN | NNS)* to a span query with consolidation. For space restrictions we shorten the query by selecting the annotations in the views pos_nn, pos_nns, pos_nnp and pos_nnps.

3.5 Relational Operators for Aggregations

Our data model is a minimal extension to the relational model. Therefore, all of the standard relational operators (`select`, `project`, `group by`, aggregations) apply without any change. Contrary to existing approaches, like SystemT, the user now also executes these operators across the entire corpus. As a result, we enable the user to derive corpus-wide distributions, permit the user to resolve logical entities across documents and understand the distribution of words, spans, annotations and relations across documents.

Solving task 1: Person-Age Extractor. Consider Figure 3, which applies span, consolidation and relational operators for showing a distribution of age information that is linked to a person. The query in the figure applies the pattern $< person >< COMMA >< NUMBER >< COMMA >$ to each sentence in the corpus.

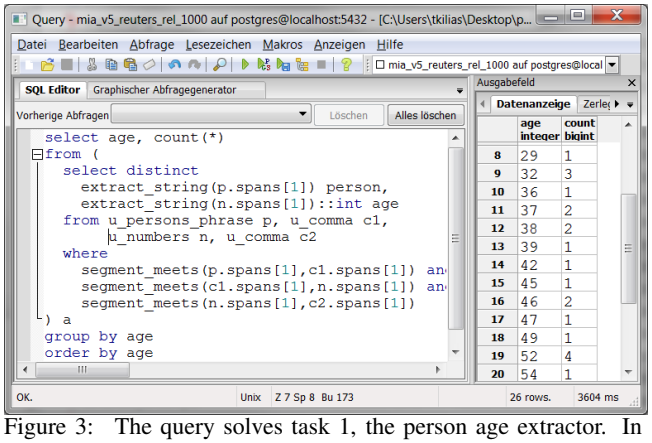

Figure 3: The query solves task 1, the person age extractor. In addition, the query provides a distribution for age information in our corpus. The outer SELECT statement computes the aggregation for each age observation, grouped and ordered by age. The nested select statement projects two strings: the person and the age from conditions in the WHERE clause that implement the heuristic of a syntactic apposition, in this case represented by the patter $< person >< comma >< number >< comma >$.

3.6 Adding Domain Semantics

These operators enable a user further to refine potentially existing annotations with domain specific semantics. In INDREX we propose operators for refining attribute values, for disambiguating relation names and regular expression dictionaries.

Refining relation attributes and entity semantics with multi-word lookups from existing tables. Computational linguists often use so called dictionaries that contain multi-word strings. They use these external resources for refining the semantic type of a sequence of tokens in the text that likely represents an attribute or an entity of a relation. For instance, a user might apply a domain specific dictionary from customer data to a set of company names and might call the resulting set of new annotations *preferred_customers*. Each relational database system may manage tables that already contain such multi-word strings. However, in existing systems, such as SystemT or GATE, these dictionaries must be loaded from the RDBMS into the extraction system before the extraction system

can match strings from the dictionaries to tokens in a text document. INDREX avoids this painful data shipping across system borders. Instead, INDREX provides the user with a user-defined-function (UDF) that permits multi-word lookups from potentially existing tables within the same RDBMS that also holds the text data. In INDREX we implemented multi-word lookups as theta-join with an approximate string matching.

Disambiguating relation names with synset dictionaries. Another common technique of Computational Linguists are pre-computed resources, called synset dictionaries, for resolving synonymous relation names. Latest research, such as [2, 26], shows that these dictionaries can be computed in a unsupervised fashion by clustering approaches from syntactically labeled data. For instance the synset dictionary for the semantic type *acquisition(company, company)* from our work in [2] contains the following (and other) multi-word expressions: *acquired by, acquisition by, acquisition of, acquired, purchased, bought, takeover of, agreed to buy, to acquire, sold, sold to, acquired in, to sell, acquisition in, acquires stake in, purchased by, bid for, bought by, sale of, buys interest in, bought out, completed took over, corporation in, merger with, purchase of, incorporated from, bought from, announced to purchase*. The INDREX user can import these valuable linguistic resources in a standard relational table and might assign each multi-word phrase the semantic type *acquisition*. Next, the user can join this information with syntactic information from sentences, such as predicates or tokens between two company names. Again, we implemented this join as a UDF that executes a theta-join with an approximate string matching.

Regular expression dictionaries for filtering annotations. Another common technique of computational linguists is the reuse of predefined patterns for regular expressions. For instance, the system REVERB uses few regular expressions for identifying strings that likely represent the name of a relation, such as $V = verb particle?adv?$, $W = (noun|adj|adv|pron|det)$ or $P = (prep|particle|inf.marker)$. Often computational linguists even combine these patterns into a single regular expression, such as $V|VP|VW*P$. INDREX enables sharing and reusing such predefined, or even pre-computed pattern, among queries of multiple users. For enabling this functionality, a user in INDREX may store a regular expression and an name in a table. Next, INDREX provides another user-defined-function called *regex_lookup(span,regex_name)* that takes as input the name of a regular expression and a span, executes the regular expression on the span and returns matches as sets of new annotations. This functionality enables users to reuse not only existing regular expressions from the existing literature, but also searching and adapting existing regular expressions to specific domains.

Joining Annotations with existing Tables. Finally, INDREX also permits users joining recognized relations in text data with relations from existing structured data. For instance, the user might know from the document meta data the author, creation date and the author location. From the content of the document the user might extract additional relations, such as the age of persons or acquisition. Using the standard join operations in the RDBMS the user can join the information over the common document identifier.

4. CONCLUSION AND OUTLOOK

We described INDREX, a powerful system for in-database relation extraction. Our system manages both text data and annotated data in a standard relational database management system, together with potentially existing relational data. As a result, the user no longer needs to ship data between the RDBMS and another extraction system, like SystemT or GATE. Moreover, the user can access data with their preferred standard SQL workbench and writes queries in standard SQL. Therefore, the user benefits from built-in query optimization techniques which permit the fast user response times even for corpus-wide operations. Finally, the user may use potentially existing data to refine semantics of relation attributes and names that the system extracts from text data. From a preliminary study we report that this unique feature combination enables non-computational linguists to write even complex relation extractors in standard SQL.

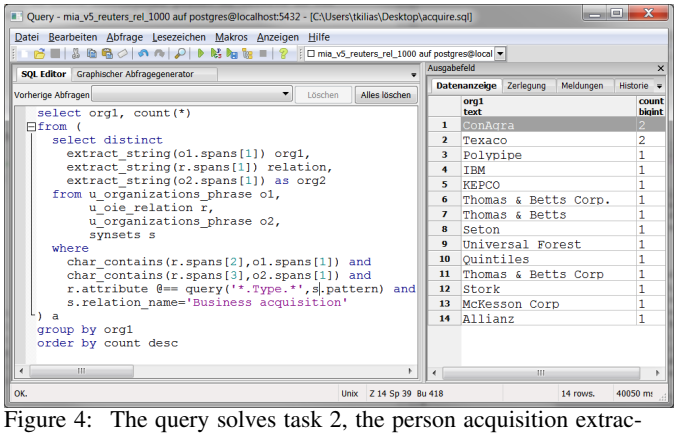

Figure 4: The query solves task 2, the person acquisition extractor. In addition the query provides a distribution for acquisitions grouped by company name, for instance in our corpus we could observe two acquisitions for Conagra and Texaco, while for the other listed companies we could only observe one acquisition. The outer SELECT statement projects organization names and sentence predicates and applies in the WHERE clause the following conditions: (1) The predicate must appear after the first organization. (2) The second organization must follow after the predicate. (3) The syntactic structure of the sentence must follow the spans defined in the table 'oie_relations' (where oie stands for open information extraction) and must follow the structure <any argument> <predicate> <any argument> (expressed in the query by '*<TYPE>*') and (4) the string of the predicate must match with strings in the synset dictionary 'Business Acquisition'.

Our work has only scratched the surface of this exciting new research direction. In our future work we will address the following research questions:

Enable data-driven repair. We believe that the usability of a database is as important as its capability. However, modern database systems are very difficult to use. Our abstraction in INDREX addresses issues in the data model and database design. One interesting direction is further abstracting the relation extraction process by repair operations. Such an operation would inspect candidate argument relation extractors and automatically correct wrong boundaries. Since INDREX holds the raw text data as well as annotations, we prefer a data-driven approach that could learn from existing annotations.

N=ALL: Exploit main-memory and multi-core databases for sub-second response times. In INDREX we used PostgrSQL as underlying RDBMS. However, our PostgreSQL installation can neither leverage pipeline parallelism nor modern multicore hardware. Moreover, most text data sets are not web-scale and will fit in a compressed format in a main memory database [9]. In our future work we will explore the power of such database systems for domain adaption tasks on data sets with the sample size N=all.

N=ALL: Leverage column-based index structures for Web scale data. In some rare cases, the amount of text data is as large

as the web. In that case the INDREX data model and operator design could leverage from existing implementations of broadcast joins, such as in IMPALA or CLYDESDALE, or column-based index structures like the work in [18] or TREVNI.

Incorporate user feedback from OLAP and search applications. Clickstream-based approaches, such as Google's knowledge graph, or query driven part-of-speech tagger models from [19] leverage click stream data for refining rule based and pre-computed extractors. In the case of INDREX, and OLAP or CRM application might trigger a rule refinement. In [23] we defined a preliminary set of such interactions. In our future work, we will explore how these interactions may improve the quality (in terms of precision and recall) of rule-based extractors in INDREX.

5. REFERENCES

[1] A. Akbik and A. Löser. Kraken: N-ary facts in open information extraction. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*, AKBC-WEKEX '12, pages 52–56, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[2] A. Akbik, L. Visengeriyeva, P. Herger, H. Hemsen, Alex, and Löser. Unsupervised discovery of relations and discriminative extraction patterns. pages 17–32, 2012.

[3] A. Akbik, L. Visengeriyeva, P. Herger, H. Hemsen, and A. Löser. Unsupervised discovery of relations and discriminative extraction patterns. In *COLING*, pages 17–32, 2012.

[4] J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, Nov. 1983.

[5] M. Anderson, D. Antenucci, V. Bittorf, M. Burgess, M. J. Cafarella, A. Kumar, F. Niu, Y. Park, C. Ré, and C. Zhang. Brainwash: A data system for feature engineering. In *CIDR*, 2013.

[6] G. Attardi, F. dell'Orletta, M. Simi, A. Chanev, and M. Ciaramita. Multilingual dependency parsing and domain adaptation using desr. In *EMNLP-CoNLL*, pages 1112–1118, 2007.

[7] N. Bales, A. Deutsch, and V. Vassalos. Score-consistent algebraic optimization of full-text search queries with graft. In *SIGMOD Conference*, pages 769–780, 2011.

[8] B. Bloom. *Taxonomy of educational objectives: Handbook I: Cognitive Domain*. New York, Longmans, Green,, 1956.

[9] J.-H. Boese, C. Tosun, C. Mathis, and F. Faerber. Data management with saps in-memory computing engine. In *EDBT*, pages 542–544, 2012.

[10] F. Chen, A. Doan, J. Yang, and R. Ramakrishnan. Efficient information extraction over evolving text data. In *ICDE*, pages 943–952, 2008.

[11] L. Chiticariu, V. Chu, S. Dasgupta, T. W. Goetz, H. Ho, R. Krishnamurthy, A. Lang, Y. Li, B. Liu, S. Raghavan, F. Reiss, S. Vaithyanathan, and H. Zhu. The systemt ide: an integrated development environment for information extraction rules. In *SIGMOD Conference*, pages 1291–1294, 2011.

[12] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4:397–434, 1979.

[13] L. D. Corro and R. Gemulla. Clausie: clause-based open information extraction. In *WWW*, pages 355–366, 2013.

[14] A. El-Helw, M. H. Farid, and I. F. Ilyas. Just-in-time information extraction using extraction views. In *SIGMOD Conference*, pages 613–616, 2012.

[15] O. Etzioni, A. Fader, J. Christensen, S. Soderland, and Mausam. Open information extraction: The second generation. In *IJCAI*, pages 3–10, 2011.

[16] A. Fader, S. Soderland, and O. Etzioni. Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, page 1535–1545, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[17] D. Ferrucci and A. Lally. UIMA: an architectural approach to unstructured information processing in the corporate research environment. *Nat. Lang. Eng.*, 10(3-4):327–348, Sept. 2004.

[18] A. Floratou, J. M. Patel, E. J. Shekita, and S. Tata. Column-oriented storage techniques for mapreduce. *PVLDB*, 4(7):419–429, 2011.

[19] K. Ganchev, K. Hall, R. T. McDonald, and S. Petrov. Using search-logs to improve query tagging. In *ACL (2)*, pages 238–242, 2012.

[20] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING*, pages 539–545, 1992.

[21] A. Jain, A. Doan, and L. Gravano. Optimizing SQL queries over text databases. In *Data Engineering, International Conference on*, volume 0, pages 636–645, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[22] R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, S. Vaithyanathan, and H. Zhu. SystemT: a system for declarative information extraction. *SIGMOD Rec.*, 37(4):7–13, Mar. 2009.

[23] A. Löser, S. Arnold, and T. Fiehn. The goolap fact retrieval framework. In *Business Intelligence*, pages 84–97. Springer, 2012.

[24] A. Löser, C. Nagel, S. Pieper, and C. Boden. Beyond search: Retrieving complete tuples from a text-database. *Information Systems Frontiers*, 15(3):311–329, 2013.

[25] G. Marchionini. Exploratory search: from finding to understanding. *Commun. ACM*, 49(4):41–46, Apr. 2006.

[26] N. Nakashole, G. Weikum, and F. M. Suchanek. Patty: A taxonomy of relational patterns with semantic types. In *EMNLP-CoNLL*, pages 1135–1145, 2012.

[27] D. E. Rose and D. Levinson. Understanding user goals in web search. In *WWW*, pages 13–19, 2004.

[28] F. M. Suchanek, M. Sozio, and G. Weikum. Sofie: a self-organizing framework for information extraction. In *WWW*, pages 631–640, 2009.

[29] A. Sun and R. Grishman. Active learning for relation type extension with local and global data views. In *CIKM*, pages 1105–1112, 2012.

[30] L. Tari, P. H. Tu, J. Hakenberg, Y. Chen, T. C. Son, G. Gonzalez, and C. Baral. Incremental information extraction using relational databases. *IEEE Trans. Knowl. Data Eng.*, 24(1):86–99, 2012.