Working with Text

Integrating and querying unstructured data

Taught by:

Jaroslav Pullmann
Solutions Architect

Learning Objectives

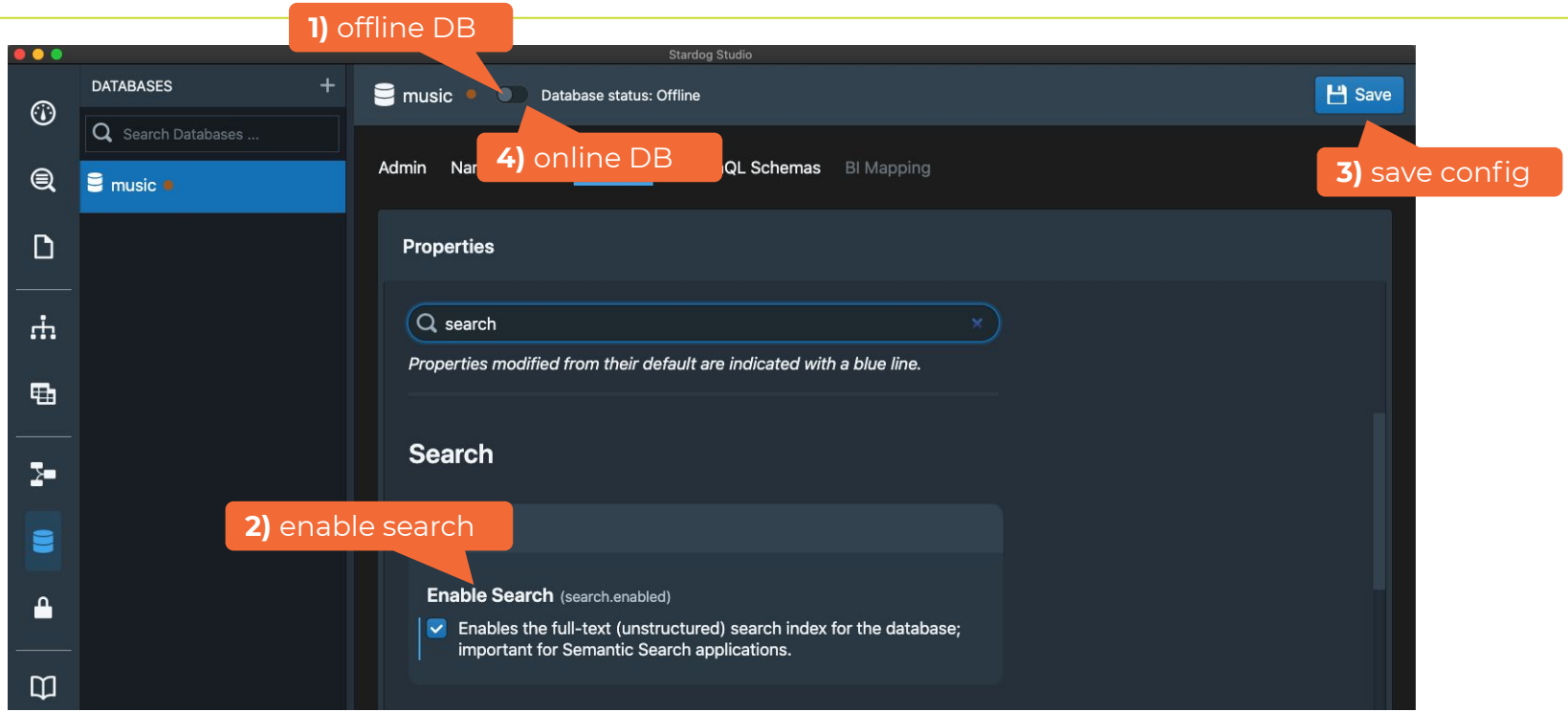
- Configure and use the full text search feature of Stardog
- Understand the options for integrating unstructured data
- Leverage and extend Stardog's text processing feature

Full Text Search

Full Text Search in Stardog

- Full-text search over string literals in RDF graph
- Operates on attribute values (objects) of a particular datatype
  - `xsd:string` and `rdf:langString` by default
- Text indexing and search (query operators) based on Apache Lucene
- Index creation and updates transparently managed by Stardog
  - On database creation, optimization, and transactions
- Seamless integration via SPARQL
- Also available via Java and HTTP interfaces

Enabling Search (Studio)



Enabling Search (CLI)

At database creation

```
$ stardog-admin db create -o search.enabled=true -n music
```

After database creation

```
$ stardog-admin db offline music                          # take the database offline
$ stardog-admin db metadata set -o search.enabled=true music # enable search
$ stardog-admin db online music                           # make the database online
```

Full Text Configuration

- Configuration properties exist to customize the functionality (`search.*`)
- `search.index.datatypes`: IRIs of datatypes designated for indexing
- `search.index.properties.excluded`: IRIs of excluded properties
- `search.index.stopwords`: List of stop words excluded from indexing
  - Redefine stop words when the default set interferes with terms in your domain
- `search.default.query.operator`: Default boolean operator (`OR`)
  - Consider setting to `AND` to enforce searching for all query terms
- `search.wildcard.search.enabled`: Support for leading wildcards (`*/?`)
  - Enable wildcards at the beginning of a term with care (expensive operation)

Querying Text in SPARQL

- SPARQL graph queries work by exact match
- Inexact matches can be done via FILTERs
- FILTERs over literals require inefficient scans

Standard SPARQL

```
SELECT * {
     ?album a :Album ;
     :name ?name.
FILTER(contains(?name, "Let"))
}
```

- Leverage the Lucene full text index
- Search via object of the `textMatch` predicate
- Bind its subject to the constrained variable

Stardog full text search

```
PREFIX sd: <tag:stardog:api:property:>
SELECT * {
  ?album a :Album ;
     :name ?name.
  ?name sd:textMatch "Let" .
}
```

Advanced Search

Retrieve matching score

```
SELECT * {
  # Based on BM25 ranking function
  (?name ?score) sd:textMatch "Let"
}
ORDER BY DESC(?score)
```

Use Lucene modifiers

```
SELECT * {
  ?name sd:textMatch "(Richard OR Ringo)"
}
```

Limit the score

```
SELECT * {
  ?name sd:textMatch ("Let" 0.7).
}
```

Example Lucene modifiers:

- Wildcards: `?` and `*`
- Fuzzy search with similarity weight (e.g. `Let~0.8`)
- Regular expressions: `"/Rich(ard)?/"`
- Proximities (word distance): `"semantic web"~5`
- Term boosting (relevance): `"Yellow submarine^10"`
- Boolean operators: `OR`, `AND(+)`, `NOT(-)`, and groups

Query Service

- Alternative for advanced search. Explicit (named) arguments and extended functionality

Search service

```
PREFIX fts: <tag:stardog:api:search:>
SELECT * WHERE {
  SERVICE fts:textMatch {
      [] fts:query 'Lennon AND Beatles' ;
         fts:threshold 0.6 ;
         fts:offset 0 ;
         fts:limit 10 ;
         fts:score ?score ;
         fts:result ?result ;
         fts:parsedQuery ?luceneQuery ;
  }
}ORDER BY DESC(?score)
```

- `fts:query` may bind to a variable allowing for dynamic search expressions
- Optional `fts:parsedQuery` argument retrieves the effective Lucene query
- Note the differing namespaces `<tag:stardog:api:property:textMatch>` for the predicate versus service: `<tag:stardog:api:search:textMatch>`

Query Service / Example

Search service with variable bindings

```
# Select album artist(s). Each artist's name will be used as
# input to a search query in the full-text index
SELECT * WHERE {
  ?album a :Album ; :artist/:name ?artistName .
  SERVICE fts:textMatch {
      [] fts:query ?artistName ;
         fts:score ?score ;
         fts:result ?otherArtist ;
  }
  ?otherAlbum a :Album ; :artist/:name ?otherArtist .
  FILTER(?artistName != ?otherArtist)
}ORDER BY DESC(?score)
```

1) Find matching literals

2) Look-up related tracks/albums

Custom Lucene Indexing

- By default Stardog uses Lucene's `StandardAnalyzer`
- See here for steps involved in deploying a custom Analyzer

  Extend `Analyzer` > Implement `AnalyzerFactory` > Register it via Java SPI (`META-INF/services`)

Sample custom Analyzer

```
import com.complexible.stardog.search.AnalyzerFactory;
import org.apache.lucene.analysis.Analyzer;
public final class CustomAnalyzerFactory implements AnalyzerFactory {
     public Analyzer get() { return new CustomAnalyzer(); }
}
public class CustomAnalyzer extends Analyzer {
    protected TokenStreamComponents createComponents(String s) { ...
    }
}
```

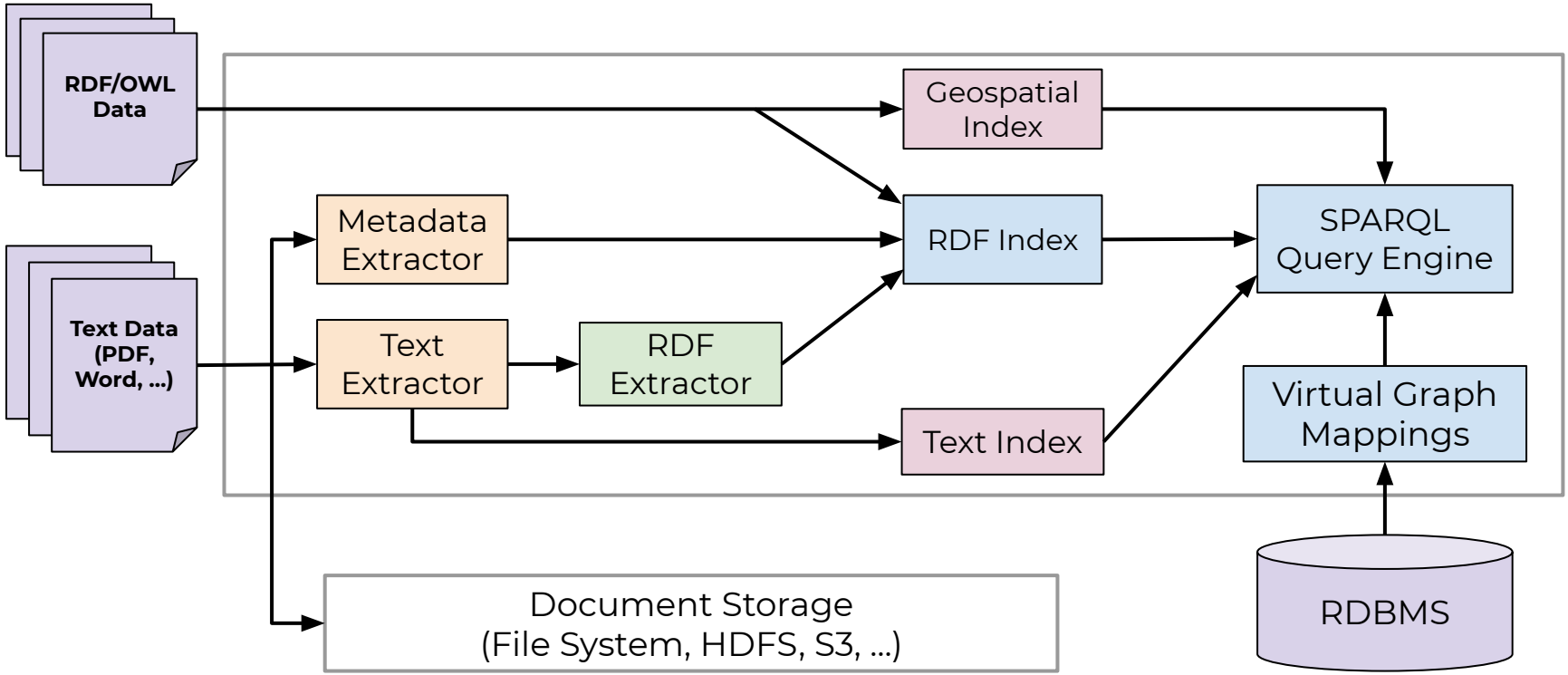
Demo

Text Processing

Text Processing in Stardog

- Stardog supports storage and processing of text documents (BITES)
  - It accepts a wide range of document formats (HTML, MS Word, PDF etc.)
  - It may extract plain text and related facts on document check in
  - Document metadata, named entity occurrences and links are extracted by default
  - Custom extractors may customize or substitute this default processing
- Facts related to a document are held within an own document graph
  - Subsequent updates to a document will replace or augment its document graph
- Stardog may (optionally) persist the documents in file system or a cloud storage
  - Acting as a gatekeeper Stardog prevents any direct document manipulation
  - Various APIs exist to manage the lifecycle of stored documents

Architecture Outline



Configuration

- `docs.*` database options (configurable any time)
- `docs.default.text.extractors`: Names of default text extractor(s) (tika, *mytxt*)
- `docs.default.rdf.extractors`: Names of default RDF extractors (tika, *myrdf*)
- `docs.filesystem.uri`: Storage protocol (`file:///`, `s3:///` or `none` to disable storage)
- `docs.path`: Storage path relative to `$STARDOG_HOME/<db>` (for `file:///`) or S3 URL (bucket)
- `docs.s3.protocol`: Web protocol used with remote storage HTTPS or HTTP
- `docs.opennlp.models.path`: Path relative to `$STARDOG_HOME` containing NLP models
- Enable full text search: `search.enabled=true`

CLI Commands

- `stardog doc` command group for CRUD operations on documents

stardog doc **put**

```
# Check-in (add/modify) a document, replacing or augmenting the document graph
$ stardog doc put mydb acme.pdf
$ stardog doc put --name alias.pdf mydb "ACME Corporation.pdf"
$ stardog doc put --rdf-extractors tika,dictionary mydb acme.pdf
$ stardog doc put --rdf-extractors custom --keep-assertions mydb acme.pdf
```

stardog doc **get**

```
# Retrieve / check-out a specified document
stardog doc get mydb acme.pdf
stardog doc get --file alias.pdf mydb acme.pdf
```

CLI Commands (2)

stardog doc **delete**

```
# Delete specified documents from store and full-text index
stardog doc delete mydb acme.pdf README.txt
stardog doc delete mydb --all
```

stardog doc **count**

```
# Count documents persisted in the document store
stardog doc count mydb
```

stardog doc **reindex**

```
# Reindex all documents (e.g., by applying different extractors)
stardog doc reindex mydb
stardog doc reindex mydb --rdf-extractors tika,entities
```

Full Text Search on Documents

```
# requires full-text search on DB enabled: search.enabled=true
prefix docs: <tag:stardog:api:docs:>
prefix sd: <tag:stardog:api:property:>
select distinct ?doc ?p ?o where {
     ?doc sd:textMatch "Acme Corp*" .
     # document graph containing extracted facts
     graph ?doc {
          ?doc a docs:Document ; ?p ?o
     }
}
```

doc	p	o
stardog:docs:user15:acme_entities.txt	rdf:type	foaf:Document
stardog:docs:user15:acme_entities.txt	rdf:type	stardog:docs:Document
stardog:docs:user15:acme_entities.txt	rdfs:label	"acme_entities.txt"
stardog:docs:user15:acme_entities.txt	stardog:docs:fileSize	416
stardog:docs:user15:acme_entities.txt	dc:format	"text/plain; charset=ISO-8859-1"
stardog:docs:user15:acme_entities.txt	dc:identifier	"acme_entities.txt"
stardog:docs:user15:acme_entities.txt	stardog:docs:hasEntity	stardog:docs:entity:306ccb94bea8181b15c7955c17d63fa0

Named Entity Recognition (NER)

- *Extractors* are extensible components to extract plain text, document metadata and facts
- `tika` extractor (default) based on Apache Tika gathers text and metadata
- Various levels of named entity recognition (NER) and linking via extractors:
  - `entities`, `linker`, and `dictionary`
- `entities` extractor
  - Detects and describes mentions of named entities as RDF statements

```
stardog doc put --rdf-extractors tika,entities mydb acme.txt
```

stardog:docs:entity:426131114e006ce2cd67de2c1784120a	rdf:type	stardog:docs:ner:organization
stardog:docs:entity:426131114e006ce2cd67de2c1784120a	rdfs:label	"Acme Corporation"

NER Configuration

- NER expects Apache OpenNLP models in `docs.opennlp.models.path`
  - Version 1.5 models are (still) compatible with recent OpenNLP version
  - Models are picked up based on naming convention
  - Tokenizer (`<lang>-token.bin`)
  - Sentence detector (`<lang>-sent.bin`)
  - At least one name-finder (NER) model
- Name-finder
  - Trained, statistical (`<lang>-ner-*.bin`)
  - Dictionary-based (`<lang>-ner-*.dict`)

Named Entity Linking (NEL)

- `linker` extractor
  - Augments `entities` extractor by linking entities to existing database resources
  - Matches mention text with the identifier and labels of the existing resources
    - `rdfs:label`, `foaf:name`, `dc:title`, `skos:prefLabel` and `skos:altLabel`
  - Only successfully resolved and linked entities (via `dct:references`) are returned

```
stardog doc put --rdf-extractors linker mydb acme.txt
```

```
{
  <tag:stardog:api:docs:mydb:acme.txt> a docs:Document .
  <tag:stardog:api:docs:entity:426131114e006ce2cd67de2c1784120a> a <tag:stardog:api:docs:ner:organization> ;
    dct:references <http://www.example.com/acme#Acme_Corporation> ;
    rdfs:label "Acme Corporation" .
  <tag:stardog:api:docs:mydb:acme.txt> docs:hasEntity <tag:stardog:api:docs:entity:426131114e006ce2cd67de2c1784120a> .
}
```

Named Entity Linking (2)

- `dictionary` extractor
  - Similar to linker extractor, but leverages a user dictionary of mentions and IRIs
  - Uses any `*.linker` file found in `docs.opennlp.models.path`

```
stardog doc put --rdf-extractors dictionary mydb acme.txt
```

Dictionary Creation

NEL Dictionary

```
// See: https://www.stardog.com/docs/#_dictionary

public class LinkerCreator {
  public static void main(String[] args) throws IOException {
    ImmutableMultimap<String, IRI> aDictionary =
     ImmutableMultimap.<String, IRI>builder()
       .put("Acme", iri("http://dbpedia.org/resource/Acme_Corporation"))
       .put("EMI", iri("https://dbpedia.org/page/EMI"))
       .put("Sony", iri("https://dbpedia.org/page/Sony"))
       .build();
      DictionaryLinker.Linker aLinker = new DictionaryLinker.Linker(aDictionary);
      aLinker.to(new File("companies.linker"));
    }
 }
```

Document-driven Search

```
prefix dct: <http://purl.org/dc/terms/>
prefix docs: <tag:stardog:api:docs:>
select distinct ?doc ?res where {
  graph ?doc {
    ?doc a docs:Document ; docs:hasEntity ?entity .
    ?entity dct:references ?res
    #FILTER( ?res = dbr:Acme_Corporation )
    }
}order by ?doc ?res
```

doc	res
stardog:docs:user15:acme_dictionary.txt	dbpedia:SAP_SE
stardog:docs:user15:acme_dictionary.txt	acme:Acme_Corporation
stardog:docs:user15:acme_dictionary.txt	acme:Person2
stardog:docs:user15:acme_dictionary.txt	acme:Person3
stardog:docs:user15:acme_dictionary.txt	acme:Person4

Data-driven Search / Entity Extractor Service

- `docs:entityExtractor` service
- Applies `entities, linker`, or `dictionary` extractors to any (virtual) graph data

```
select distinct ?res ?mention ?entity ?type {
    # Resolve entities from virtual graph using the dictionary
    graph<virtual://demo>{
        ?res foaf:name ?text    .
    }
    service docs:entityExtractor {
        []  docs:text ?text ;
            docs:mention ?mention ;
            docs:entity ?entity ;
            docs:type ?type ;
            docs:mode docs:Dictionary # use dictionary for entity linking
    }
}
```

Custom TextExtractor

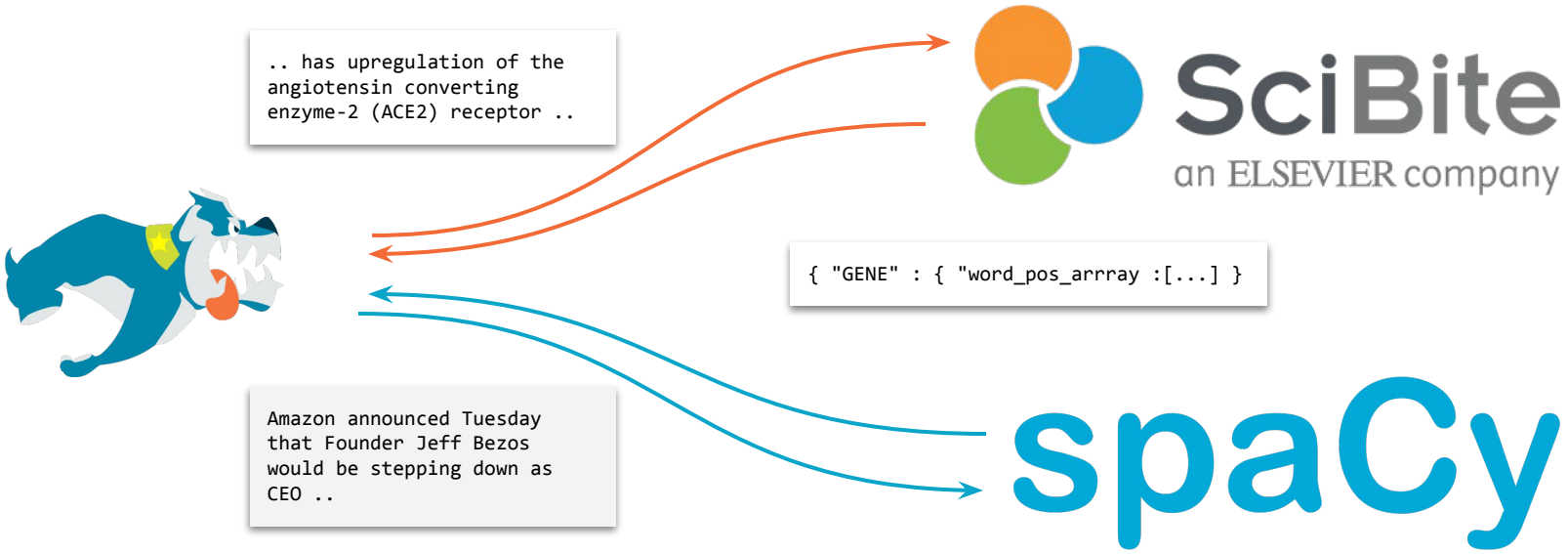
```
 // See: https://docs.stardog.com/javadoc/snarl/com/complexible/stardog/docs/extraction/TextExtractor.html

import  com.complexible.stardog.docs.extraction.TextExtractor
public class LoggingTextExtractor implements TextExtractor { // based on Tika extractor

    private Tika mTika = new Tika();
    private final static Logger LOGGER = LoggerFactory.getLogger(LoggingTextExtractor.class);

    @Override
    public Reader extract(Connection theConnection, IRI theDocIri, Path theDocContents) throws BitesException {
        try {
            Metadata aMeta = new Metadata();
            // assist content type detection by hinting with file name
            aMeta.set(Metadata.RESOURCE_NAME_KEY, theDocContents.getFileName().toString());
            Reader reader = mTika.parse(Files.newInputStream(theDocContents), aMeta);
            // Log or otherwise process the text
            LOGGER.debug(readerToString(bufferedReader));
            return reader;
        }
        catch (IOException e) {
            throw new UncheckedIOException(e);
        }
    }
}
```

Custom RDFExtractor / Outline




```
ID1     org 0 6         Amazon          https://dbpedia.org/resource/Amazon_(company)
ID2     date 17 24      Tuesday
ID3     person 38 48    Jeff Bezos      https://dbpedia.org/resource/Jeff_Bezos
ID4     role 75 78      CEO             https://dbpedia.org/resource/Chief_executive_officer
```

Customization / RDFExtractor

AbstractEntityRDFExtractor

```
public class  SpacyEntityExtractor extends AbstractEntityRDFExtractor
{

    @Override
    protected  StatementSource extractFromText(
        final Connection theConnection,
        final IRI theDocIri,
        final Reader theText) throws Exception {

        BasicMentionExtractor aExtractor = new
BasicMentionExtractor(
                new  SpacyDocumentParser(),     // parser
                new NERMentionExtractor()      // mention extractor
        );

        Set<Statement> aGraph = Sets.newHashSet();

        // add each entity to the model
        for (Span aEntity : aExtractor.extract(theText)) {
            addEntity(aGraph, theDocIri, aEntity, false, true);
        }

        return MemoryStatementSource.of(aGraph);
    }
}
```

DocumentParser

```
public class  SpacyDocumentParser implements DocumentParser {

    @Override
    public Document apply(String string) {
        List<String> list = doSpaCyNER(string)
        Document aDoc = new Document(list.size());
        int i = 0;
        for (String line : list) {
            // Extract columns from  brat standoff format
            Matcher m = p.matcher(line);
            if (m.matches()) {
               // Text, entitiy mention
                Token aToken = new Token(aDoc, i, m.group(5));
               // Entity type
                aToken.put(Token.NER, m.group(2));
                aToken.put(Token.NER_SPAN,
Integer.toString(i));
                Token[] aTokens = {aToken};
                aDoc.set(i, aTokens);
                i++;
            }
        }
        return aDoc;
    }
}
```

Demo

Resources

Resources

- Documentation: Unstructured Content (BITES)
- Blog: Entity Linking in the Knowledge Graph (01/2018)
- Blog: Link All the Entities! (05/2018)
- Blog: Augmenting Search (07/2018)
- Tutorial: Entity Extraction & Linking (05/2018)
- Tutorial: Extending BITES (06/2018)
- Stardog NLP Resources
- Github Custom extractors

Learning Objectives

Learning Objectives

- Configure and use the full text search feature of Stardog
- Understand the options for integrating unstructured data
- Leverage and extend Stardog's text processing feature

Thank you