Query syntax design

Context and Problem Statement

All libraries use their own query syntax for advanced search options. To increase usability, users should be able to formulate their (abstract) search queries in a query syntax that can be mapped to the library specific search queries. To achieve this, the query has to be parsed into an AST.

Which query syntax should be used for the abstract queries? Which features should the syntax support?

Considered Options

- Use a simplified syntax that is derived of the lucene query syntax
- Formulate a own query syntax

Decision Outcome

Chosen option: “Use a syntax that is derived of the lucene query syntax”, because only option that is already known, and easy to implement. Furthermore parsers for lucene already exist and are tested. For simplicity, and lack of universal capabilities across fetchers, only basic query features and therefor syntax is supported:

- All terms in the query are whitespace separated and will be ANDed
- Default and certain fielded terms are supported
- Fielded Terms:
  - `author`
  - `title`
  - `journal`
  - `year` (for single year)
  - `year-range` (for range e.g. `year-range:2012-2015`)
- The `journal`, `year`, and `year-range` fields should only be populated once in each query
- The `year` and `year-range` fields are mutually exclusive
- Example:
  - `author:"Igor Steinmacher" author:"Christoph Treude" year:2017` will be converted to
  - `author:"Igor Steinmacher" AND author:"Christoph Treude" AND year:2017`

The supported syntax can be expressed in EBNF as follows:

Query := {Clause}
Clause:= [Field] Term
Field := author: | title: | journal: | year: | year-range: | default:
Term := Word | Phrase \

Word can be derived to any series of non-whitespace characters. Phrases are multiple words wrapped in quotes and may contain white-space characters within the quotes.
Note: Even though this EBNF syntactically allows the creation of queries with year and year-range fields, such a query does not make sense semantically and therefore will not be executed.

Positive Consequences

- Already tested
- Well known
- Easy to implement
- Can use an existing parser

Pros and Cons of the Options

Use a syntax that is derived of the lucene query syntax

- Good, because already exists
- Good, because already well known
- Good, because there already exists a parser for lucene syntax
- Good, because capabilities of query conversion can easily be extended using the flexible lucene framework

Formulate a own query syntax

- Good, because allows for flexibility
- Bad, because needs a new parser (has to be decided whether to use ANTLR, JavaCC, or LogicNG)
- Bad, because has to be tested
- Bad, because syntax is not well known
- Bad, because the design should be easily extensible, requires an appropriate design (high effort)