Search String Parser

Release 0.2.3

Sep 27, 2017

Contents

Contents:

CHAPTER 1

Overview

Search String Parser

docs	
tests	
package	

Parse a more general search syntax to conform with a particular SQL dialect.

Currently, this is implemented using ply with a general lexer and a parser for generating PostgreSQL-specific search queries.

- Free software: BSD license

Installation

```
pip install searchstringparser
```

Documentation

https://searchstringparser.readthedocs.org/

Development

To run the all tests run:

```
tox
```

CHAPTER 2

Installation

At the command line:

```
pip install searchstringparser
```

CHAPTER 3

Usage

To use Search String Parser in a project:

```
import searchstringparser
```

or directly import one of the *lexers* or *parsers*, e.g.,

```
>>> from searchstringparser import GeneralSearchStringLexer
>>> from searchstringparser import PostgreSQLTextSearchParser
```

You can then use an instance of the lexer for your own parser or parse a search query string.

```
>>> parser = PostgreSQLTextSearchParser()
>>> parser.parse("find my term")
'find & my & term'
```

CHAPTER 4

Reference

Top-level package that exposes lexer and parser classes.

searchstringparser.lexer

class searchstringparser.lexer.general.**GeneralSearchStringLexer**(*illegal='ignore'*, ***kw_args*)

Bases: object

__init__(*illegal='ignore'*, ***kw_args*)

A composite class of a ply.lex.lex.

This is the setup step necessary before you can iterate over the tokens.

Parameters

- **illegal** (*{'record', 'ignore', 'error'} (optional)*) – Action to be taken when illegal characters are encountered. The default is to record them but continue without regarding them.
- **kw_args** – Keyword arguments are passed to the ply.lex.lex call.

get_illegal()

Return encountered illegal characters.

Returns

- *None* – If no illegal characters occurred.
- *Tuple* – A pair of lists that contain the illegal characters and the positions where they occurred.

input(*data*)

Add a new string to the lexer.

This is the setup step necessary before you can iterate over the tokens.

Parameters **data** (*str*) – Any string.

print_tokens(*data*)
Print all tokens in a string.

First iterates through all tokens found and prints them to sys.stdout. Then prints illegal characters if any occurred.

Parameters **data** (*str*) – Any string.

Exposes the following classes:

- *GeneralSearchStringLexer*

searchstringparser.parser

class searchstringparser.parser.postgresql.**PostgreSQLTextSearchParser**(*lexer=None*, ***kw_args*)

Bases: object

This parser implements the following rules using the tokens generated by an appropriate lexer. The goal is to generate a string for PostgreSQL full text search that conforms with the syntax understood by the function tsquery or to_tsquery.

The following rules are implemented which generate the correct query string.

```
expression : expression expression
           | expression AND expression
           | expression OR expression
           | NOT expression
           | LPAREN expression RPAREN
           | QUOTE term QUOTE
           | WORD WILDCARD
           | WORD

term : term SPACE term
       | term term
       | LITERAL_QUOTE
       | SYMBOL
```

__init__(*lexer=None*, ***kw_args*)
Parser instantiation.

Parameters

- **lexer** (*ply.lex (optional)*) – Any ply.lex lexer instance and generates the tokens listed in the rules. The default uses a GeneralSearchStringLexer instance.
- **kw_args** – Keyword arguments are passed to the ply.yacc.yacc call.

get_illegal()
Inspect encountered illegal characters.

Returns

- *None* – If no illegal characters occurred.
- *Tuple* – A pair of lists that contain the illegal characters and the positions where they occurred.

parse(*query*, ***kw_args*)
Parse any string input according to the rules.

Parameters

- **query** (*str*) – A string expected to conform with the search query syntax.
- **kw_args** – Keyword arguments are passed on to the ply.yacc.yacc.parse call.

Returns

A string that can directly be passed to the PostgreSQL functions tsquery or to_tsquery.

Return type str

Exposes the following classes:

- *PostgreSQLTextSearchParser*

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Documentation improvements

Search String Parser could always use more documentation, whether as part of the official Search String Parser docs, in docstrings, or even on the web in blog posts, articles, and such.

Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/AGHerwig/searchstringparser/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Development

To set up *searchstringparser* for local development:

1. Fork searchstringparser on GitHub.
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/searchstringparser.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

Tips

To run a subset of tests:

```
tox -e envname -- py.test -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 6

Authors

- Moritz Emanuel Beber - https://github.com/AGHerwig/searchstringparser

CHAPTER 7

Changelog

0.2.2 (2015-09-29)

- Complete documentation

0.2.1 (2015-09-28)

- Add more helpful error messages

0.2.0 (2015-09-28)

- Add complete integration with Travis, Appveyor, Coveralls, and Read the Docs
- Increase test coverage to 100%
- Make source compatible with Python 2.6 - 3.4

0.1.1 (2015-09-21)

- Fix GeneralSearchStringLexer regex
- Fix PostgreSQLTextSearchParser rules
- Add first set of tests

0.1.0 (2015-09-16)

- Initial class layout

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

Symbols

G

I

P

S