python-sqlparse Documentation

Release 0.2.4.dev0

Andi Albrecht

Apr 30, 2017

Contents

sqlparse is a non-validating SQL parser for Python. It provides support for parsing, splitting and formatting SQL statements.

The module is compatible with Python 2.7 and Python 3 (>= 3.3) and released under the terms of the New BSD license.

Visit the project page at https://github.com/andialbrecht/sqlparse for further information about this project.

CHAPTER 1

tl;dr

```
$ pip install sqlparse
$ python
>>> import sqlparse
>>> print(sqlparse.format('select * from foo', reindent=True))
select *
from foo
>>> parsed = sqlparse.parse('select * from foo')[0]
>>> parsed.tokens
[<DML 'select' at 0x7f22c5e15368>, <Whitespace ' ' at 0x7f22c5e153b0>, <Wildcard '*'␣
↪... ]
>>>
```

CHAPTER 2

Contents

Introduction

Download & Installation

The latest released version can be obtained from the Python Package Index (PyPI). To extract the install the module system-wide run

```
$ tar cvfz python-sqlparse-VERSION.tar.gz
$ cd python-sqlparse/
$ sudo python setup.py install
```

Alternatively you can install *sqlparse* using **pip**:

```
$ pip install sqlparse
```

Getting Started

The *sqlparse* module provides three simple functions on module level to achieve some common tasks when working with SQL statements. This section shows some simple usage examples of these functions.

Let's get started with splitting a string containing one or more SQL statements into a list of single statements using *split()*:

```
>>> import sqlparse
>>> sql = 'select * from foo; select * from bar;'
>>> sqlparse.split(sql)
[u'select * from foo; ', u'select * from bar;']
```

The end of a statement is identified by the occurrence of a semicolon. Semicolons within certain SQL constructs like `BEGIN ... END` blocks are handled correctly by the splitting mechanism.

SQL statements can be beautified by using the `format()` function.

```
>>> sql = 'select * from foo where id in (select id from bar);'
>>> print sqlparse.format(sql, reindent=True, keyword_case='upper')
SELECT *
FROM foo
WHERE id IN
  (SELECT id
   FROM bar);
```

In this case all keywords in the given SQL are uppercased and the indentation is changed to make it more readable. Read *Formatting of SQL Statements* for a full reference of supported options given as keyword arguments to that function.

Before proceeding with a closer look at the internal representation of SQL statements, you should be aware that this SQL parser is intentionally non-validating. It assumes that the given input is at least some kind of SQL and then it tries to analyze as much as possible without making too much assumptions about the concrete dialect or the actual statement. At least it's up to the user of this API to interpret the results right.

When using the `parse()` function a tuple of `Statement` instances is returned:

```
>>> sql = 'select * from "someschema"."mytable" where id = 1'
>>> parsed = sqlparse.parse(sql)
>>> parsed
(<Statement 'select...' at 0x9ad08ec>,)
```

Each item of the tuple is a single statement as identified by the above mentioned `split()` function. So let's grab the only element from that list and have a look at the `tokens` attribute. Sub-tokens are stored in this attribute.

```
>>> stmt = parsed[0]  # grab the Statement object
>>> stmt.tokens
(<DML 'select' at 0x9b63c34>,
 <Whitespace ' ' at 0x9b63e8c>,
 <Operator '*' at 0x9b63e64>,
 <Whitespace ' ' at 0x9b63c5c>,
 <Keyword 'from' at 0x9b63c84>,
 <Whitespace ' ' at 0x9b63cd4>,
 <Identifier '"somes...' at 0x9b5c62c>,
 <Whitespace ' ' at 0x9b63f04>,
 <Where 'where ...' at 0x9b5caac>)
```

Each object can be converted back to a string at any time:

```
>>> str(stmt)  # str(stmt) for Python 3
'select * from "someschema"."mytable" where id = 1'
>>> str(stmt.tokens[-1])  # or just the WHERE part
'where id = 1'
```

Details of the returned objects are described in *Analyzing the Parsed Statement*.

Development & Contributing

To check out the latest sources of this module run

```
$ git clone git://github.com/andialbrecht/sqlparse.git
```

to check out the latest sources from the repository.

sqlparse is currently tested under Python 2.5, 2.6, 2.7, 3.2 and pypy. Tests are automatically run on each commit and for each pull request on Travis: https://travis-ci.org/andialbrecht/sqlparse

Make sure to run the test suite before sending a pull request by running

```
$ tox
```

It's ok, if **tox** doesn't find all interpreters listed above. Ideally a Python 2 and a Python 3 version should be tested locally.

Please file bug reports and feature requests on the project site at https://github.com/andialbrecht/sqlparse/issues/new or if you have code to contribute upload it to http://codereview.appspot.com and add albrecht.andi@googlemail.com as reviewer.

For more information about the review tool and how to use it visit it's project page: http://code.google.com/p/rietveld.

sqlparse – Parse SQL statements

The *sqlparse* module provides the following functions on module-level.

sqlparse.**split**(*sql*, *encoding=None*)
Split *sql* into single statements.

Parameters

- **sql** – A string containing one or more SQL statements.
- **encoding** – The encoding of the statement (optional).

Returns A list of strings.

sqlparse.**format**(*sql*, *encoding=None*, ***options*)
Format *sql* according to *options*.

Available options are documented in *Formatting of SQL Statements*.

In addition to the formatting options this function accepts the keyword "encoding" which determines the encoding of the statement.

Returns The formatted SQL statement as string.

sqlparse.**parse**(*sql*, *encoding=None*)
Parse sql and return a list of statements.

Parameters

- **sql** – A string containing one or more SQL statements.
- **encoding** – The encoding of the statement (optional).

Returns A tuple of *Statement* instances.

In most cases there's no need to set the *encoding* parameter. If *encoding* is not set, sqlparse assumes that the given SQL statement is encoded either in utf-8 or latin-1.

Formatting of SQL Statements

The *format()* function accepts the following keyword arguments.

keyword_case Changes how keywords are formatted. Allowed values are "upper", "lower" and "capitalize".

identifier_case Changes how identifiers are formatted. Allowed values are "upper", "lower", and "capitalize".

strip_comments If `True` comments are removed from the statements.

truncate_strings If `truncate_strings` is a positive integer, string literals longer than the given value will be truncated.

truncate_char (default: "[...]") If long string literals are truncated (see above) this value will be append to the truncated string.

reindent If `True` the indentations of the statements are changed.

indent_tabs If `True` tabs instead of spaces are used for indentation.

indent_width The width of the indentation, defaults to 2.

wrap_after The column limit for wrapping comma-separated lists. If unspecified, it puts every item in the list on its own line.

output_format If given the output is additionally formatted to be used as a variable in a programming language. Allowed values are "python" and "php".

Analyzing the Parsed Statement

When the *parse()* function is called the returned value is a tree-ish representation of the analyzed statements. The returned objects can be used by applications to retrieve further information about the parsed SQL.

Base Classes

All returned objects inherit from these base classes. The *Token* class represents a single token and *TokenList* class is a group of tokens. The latter provides methods for inspecting its child tokens.

class sqlparse.sql.**Token**(*ttype*, *value*)
Base class for all other classes in this module.

It represents a single token and has two instance attributes: `value` is the unchange value of the token and `ttype` is the type of the token.

flatten()
Resolve subgroups.

has_ancestor(*other*)
Returns `True` if *other* is in this tokens ancestry.

is_child_of(*other*)
Returns `True` if this token is a direct child of *other*.

match(*ttype*, *values*, *regex=False*)
Checks whether the token matches the given arguments.

ttype is a token type. If this token doesn't match the given token type. *values* is a list of possible values for this token. The values are OR'ed together so if only one of the values matches `True` is returned. Except for keyword tokens the comparison is case-sensitive. For convenience it's ok to pass in a single string. If *regex* is `True` (default is `False`) the given values are treated as regular expressions.

within(*group_cls*)
Returns `True` if this token is within *group_cls*.

Use this method for example to check if an identifier is within a function: `t.within(sql.Function)`.

class sqlparse.sql.**TokenList**(*tokens=None*)
A group of tokens.

It has an additional instance attribute `tokens` which holds a list of child-tokens.

flatten()
Generator yielding ungrouped tokens.

This method is recursively called for all child tokens.

get_alias()
Returns the alias for this identifier or `None`.

get_name()
Returns the name of this identifier.

This is either it's alias or it's real name. The returned valued can be considered as the name under which the object corresponding to this identifier is known within the current statement.

get_parent_name()
Return name of the parent object if any.

A parent object is identified by the first occuring dot.

get_real_name()
Returns the real name (object name) of this identifier.

get_token_at_offset(*offset*)
Returns the token that is on position offset.

group_tokens(*grp_cls*, *start*, *end*, *include_end=True*, *extend=False*)
Replace tokens by an instance of *grp_cls*.

has_alias()
Returns `True` if an alias is present.

insert_after(*where*, *token*, *skip_ws=True*)
Inserts *token* after *where*.

insert_before(*where*, *token*)
Inserts *token* before *where*.

token_first(*skip_ws=True*, *skip_cm=False*)
Returns the first child token.

If *skip_ws* is `True` (the default), whitespace tokens are ignored.

if *skip_cm* is `True` (default: `False`), comments are ignored too.

token_index(*token*, *start=0*)
Return list index of token.

token_next(*idx*, *skip_ws=True*, *skip_cm=False*, *_reverse=False*)
Returns the next token relative to *idx*.

If *skip_ws* is `True` (the default) whitespace tokens are ignored. If *skip_cm* is `True` comments are ignored. `None` is returned if there's no next token.

token_prev(*idx*, *skip_ws=True*, *skip_cm=False*)
Returns the previous token relative to *idx*.

If *skip_ws* is `True` (the default) whitespace tokens are ignored. If *skip_cm* is `True` comments are ignored. `None` is returned if there's no previous token.

SQL Representing Classes

The following classes represent distinct parts of a SQL statement.

class sqlparse.sql.**Statement**(*tokens=None*)
Represents a SQL statement.

get_type()
Returns the type of a statement.

The returned value is a string holding an upper-cased reprint of the first DML or DDL keyword. If the first token in this group isn't a DML or DDL keyword "UNKNOWN" is returned.

Whitespaces and comments at the beginning of the statement are ignored.

class sqlparse.sql.**Comment**(*tokens=None*)
A comment.

class sqlparse.sql.**Identifier**(*tokens=None*)
Represents an identifier.

Identifiers may have aliases or typecasts.

get_array_indices()
Returns an iterator of index token lists

get_ordering()
Returns the ordering or `None` as uppercase string.

get_typecast()
Returns the typecast or `None` of this object as a string.

is_wildcard()
Return `True` if this identifier contains a wildcard.

class sqlparse.sql.**IdentifierList**(*tokens=None*)
A list of *Identifier*'s.

get_identifiers()
Returns the identifiers.

Whitespaces and punctuations are not included in this generator.

class sqlparse.sql.**Where**(*tokens=None*)
A WHERE clause.

class sqlparse.sql.**Case**(*tokens=None*)
A CASE statement with one or more WHEN and possibly an ELSE part.

get_cases(*skip_ws=False*)
Returns a list of 2-tuples (condition, value).

If an ELSE exists condition is None.

class sqlparse.sql.**Parenthesis**(*tokens=None*)
Tokens between parenthesis.

class sqlparse.sql.**If**(*tokens=None*)
An 'if' clause with possible 'else if' or 'else' parts.

class sqlparse.sql.**For**(*tokens=None*)
A 'FOR' loop.

class sqlparse.sql.**Assignment**(*tokens=None*)
An assignment like 'var := val;'

class sqlparse.sql.**Comparison**(*tokens=None*)
A comparison used for example in WHERE clauses.

User Interfaces

sqlformat The sqlformat command line script ist distributed with the module. Run **sqlformat --help** to list available options and for usage hints.

sqlformat.appspot.com An example Google App Engine application that exposes the formatting features using a web front-end. See http://sqlformat.appspot.com for details. The source for this application is available from a source code check out of the *sqlparse* module (see extras/appengine).

Changes in python-sqlparse

Upcoming Deprecations

- sqlparse.SQLParseError is deprecated (version 0.1.5), use sqlparse.exceptions.SQLParseError instead.

Changelog

Development Version

Bug Fixes

- Fix detection of identifiers using comparisons (issue327).

Release 0.2.3 (Mar 02, 2017)

Enhancements

- New command line option "–encoding" (by twang2218, pr317).
- Support CONCURRENTLY keyword (issue322, by rowanseymour).

Bug Fixes

- Fix some edge-cases when parsing invalid SQL statements.
- Fix indentation of LIMIT (by romainr, issue320).
- Fix parsing of INTO keyword (issue324).

Internal Changes

- Several improvements regarding encodings.

Release 0.2.2 (Oct 22, 2016)

Enhancements

- Add comma_first option: When splitting list "comma first" notation is used (issue141).

Bug Fixes

- Fix parsing of incomplete AS (issue284, by vmuriart).
- Fix parsing of Oracle names containing dollars (issue291).
- Fix parsing of UNION ALL (issue294).
- Fix grouping of identifiers containing typecasts (issue297).
- Add Changelog to sdist again (issue302).

Internal Changes

- *is_whitespace* and *is_group* changed into properties

Release 0.2.1 (Aug 13, 2016)

Notable Changes

- PostgreSQL: Function bodys are parsed as literal string. Previously sqlparse assumed that all function bodys are parsable psql strings (see issue277).

Bug Fixes

- Fix a regression to parse streams again (issue273, reported and test case by gmccreight).
- Improve Python 2/3 compatibility when using parsestream (isseu190, by phdru).
- Improve splitting of PostgreSQL functions (issue277).

Release 0.2.0 (Jul 20, 2016)

IMPORTANT: The supported Python versions have changed with this release. sqlparse 0.2.x supports Python 2.7 and Python >= 3.3.

Thanks to the many contributors for writing bug reports and working on pull requests who made this version possible!

Internal Changes

- sqlparse.SQLParseError was removed from top-level module and moved to sqlparse.exceptions.
- sqlparse.sql.Token.to_unicode was removed.
- The signature of a filter's process method has changed from process(stack, stream) -> to process(stream). Stack was never used at all.
- Lots of code cleanups and modernization (thanks esp. to vmuriart!).
- Improved grouping performance. (sjoerdjob)

Enhancements

- Support WHILE loops (issue215, by shenlongxing).
- Better support for CTEs (issue217, by Andrew Tipton).
- Recognize USING as a keyword more consistently (issue236, by koljonen).
- Improve alignment of columns (issue207, issue235, by vmuriat).
- Add wrap_after option for better alignment when formatting lists (issue248, by Dennis Taylor).
- Add reindent-aligned option for alternate formatting (Adam Greenhall)
- Improved grouping of operations (issue211, by vmuriat).

Bug Fixes

- Leading whitespaces are now removed when format() is called with strip_whitespace=True (issue213, by shenlongxing).
- Fix typo in keywords list (issue229, by cbeloni).
- Fix parsing of functions in comparisons (issue230, by saaj).
- Fix grouping of identifiers (issue233).
- Fix parsing of CREATE TABLE statements (issue242, by Tenghuan).
- Minor bug fixes (issue101).
- Improve formatting of CASE WHEN constructs (issue164, by vmuriat).

Release 0.1.19 (Mar 07, 2016)

Bug Fixes

- Fix IndexError when statement contains WITH clauses (issue205).

Release 0.1.18 (Oct 25, 2015)

Bug Fixes

- Remove universal wheel support, added in 0.1.17 by mistake.

Release 0.1.17 (Oct 24, 2015)

Enhancements

- Speed up parsing of large SQL statements (pull request: issue201, fixes the following issues: issue199, issue135, issue62, issue41, by Ryan Wooden).

Bug Fixes

- Fix another splitter bug regarding DECLARE (issue194).

Misc

- Packages on PyPI are signed from now on.

Release 0.1.16 (Jul 26, 2015)

Bug Fixes

- Fix a regression in get_alias() introduced in 0.1.15 (issue185).
- Fix a bug in the splitter regarding DECLARE (issue193).
- sqlformat command line tool doesn't duplicat newlines anymore (issue191).
- Don't mix up MySQL comments starting with hash and MSSQL temp tables (issue192).
- Statement.get_type() now ignores comments at the beginning of a statement (issue186).

Release 0.1.15 (Apr 15, 2015)

Bug Fixes

- Fix a regression for identifiers with square bracktes notation (issue153, by darikg).
- Add missing SQL types (issue154, issue155, issue156, by jukebox).
- Fix parsing of multi-line comments (issue172, by JacekPliszka).
- Fix parsing of escaped backslashes (issue174, by caseyching).
- Fix parsing of identifiers starting with underscore (issue175).
- Fix misinterpretation of IN keyword (issue183).

Enhancements

- Improve formatting of HAVING statements.
- Improve parsing of inline comments (issue163).
- Group comments to parent object (issue128, issue160).
- Add double precision builtin (issue169, by darikg).
- Add support for square bracket array indexing (issue170, issue176, issue177 by darikg).
- Improve grouping of aliased elements (issue167, by darikg).
- Support comments starting with '#' character (issue178).

Release 0.1.14 (Nov 30, 2014)

Bug Fixes

- Floats in UPDATE statements are now handled correctly (issue145).
- Properly handle string literals in comparisons (issue148, change proposed by aadis).
- Fix indentation when using tabs (issue146).

Enhancements

- Improved formatting in list when newlines precede commas (issue140).

Release 0.1.13 (Oct 09, 2014)

Bug Fixes

- Fix a regression in handling of NULL keywords introduced in 0.1.12.

Release 0.1.12 (Sep 20, 2014)

Bug Fixes

- Fix handling of NULL keywords in aliased identifiers.
- Fix SerializerUnicode to split unquoted newlines (issue131, by Michael Schuller).
- Fix handling of modulo operators without spaces (by gavinwahl).

Enhancements

- Improve parsing of identifier lists containing placeholders.
- Speed up query parsing of unquoted lines (by Michael Schuller).

Release 0.1.11 (Feb 07, 2014)

Bug Fixes

- Fix incorrect parsing of string literals containing line breaks (issue118).
- Fix typo in keywords, add MERGE, COLLECT keywords (issue122/124, by Cristian Orellana).
- Improve parsing of string literals in columns.
- Fix parsing and formatting of statements containing EXCEPT keyword.
- Fix Function.get_parameters() (issue126/127, by spigwitmer).

Enhancements

- Classify DML keywords (issue116, by Victor Hahn).
- Add missing FOREACH keyword.
- Grouping of BEGIN/END blocks.

Other

- Python 2.5 isn't automatically tested anymore, neither Travis nor Tox still support it out of the box.

Release 0.1.10 (Nov 02, 2013)

Bug Fixes

- Removed buffered reading again, it obviously causes wrong parsing in some rare cases (issue114).
- Fix regression in setup.py introduced 10 months ago (issue115).

Enhancements

- Improved support for JOINs, by Alexander Beedie.

Release 0.1.9 (Sep 28, 2013)

Bug Fixes

- Fix an regression introduced in 0.1.5 where sqlparse didn't properly distinguished between single and double quoted strings when tagging identifier (issue111).

Enhancements

- New option to truncate long string literals when formatting.
- Scientific numbers are pares correctly (issue107).
- Support for arithmetic expressions (issue109, issue106; by prudhvi).

Release 0.1.8 (Jun 29, 2013)

Bug Fixes

- Whitespaces within certain keywords are now allowed (issue97, patch proposed by xcombelle).

Enhancements

- Improve parsing of assignments in UPDATE statements (issue90).
- Add STRAIGHT_JOIN statement (by Yago Riveiro).
- Function.get_parameters() now returns the parameter if only one parameter is given (issue94, by wayne.wuw).
- sqlparse.split() now removes leading and trailing whitespaces from splitted statements.
- Add USE as keyword token (by mulos).
- Improve parsing of PEP249-style placeholders (issue103).

Release 0.1.7 (Apr 06, 2013)

Bug Fixes

- Fix Python 3 compatibility of sqlformat script (by Piet Delport).
- Fix parsing of SQL statements that contain binary data (by Alexey Malyshev).
- Fix a bug where keywords were identified as aliased identifiers in invalid SQL statements.
- Fix parsing of identifier lists where identifiers are keywords too (issue10).

Enhancements

- Top-level API functions now accept encoding keyword to parse statements in certain encodings more reliable (issue20).
- Improve parsing speed when SQL contains CLOBs or BLOBs (issue86).
- Improve formatting of ORDER BY clauses (issue89).
- Formatter now tries to detect runaway indentations caused by parsing errors or invalid SQL statements. When re-indenting such statements the formatter flips back to column 0 before going crazy.

Other

- Documentation updates.

Release 0.1.6 (Jan 01, 2013)

sqlparse is now compatible with Python 3 without any patches. The Python 3 version is generated during install by 2to3. You'll need distribute to install sqlparse for Python 3.

Bug Fixes

- Fix parsing error with dollar-quoted procedure bodies (issue83).

Other

- Documentation updates.
- Test suite now uses tox and py.test.
- py3k fixes (by vthriller).

- py3k fixes in setup.py (by Florian Bauer).
- setup.py now requires distribute (by Florian Bauer).

Release 0.1.5 (Nov 13, 2012)

Bug Fixes

- Improve handling of quoted identifiers (issue78).
- Improve grouping and formatting of identifiers with operators (issue53).
- Improve grouping and formatting of concatenated strings (issue53).
- Improve handling of varchar() (by Mike Amy).
- Clean up handling of various SQL elements.
- Switch to py.test and clean up tests.
- Several minor fixes.

Other

- Deprecate sqlparse.SQLParseError. Please use sqlparse.exceptions.SQLParseError instead.
- Add caching to speed up processing.
- Add experimental filters for token processing.
- Add sqlformat.parsestream (by quest).

Release 0.1.4 (Apr 20, 2012)

Bug Fixes

- Avoid "stair case" effects when identifiers, functions, placeholders or keywords are mixed in identifier lists (issue45, issue49, issue52) and when asterisks are used as operators (issue58).
- Make keyword detection more restrict (issue47).
- Improve handling of CASE statements (issue46).
- Fix statement splitting when parsing recursive statements (issue57, thanks to piranna).
- Fix for negative numbers (issue56, thanks to kevinjqiu).
- Pretty format comments in identifier lists (issue59).
- Several minor bug fixes and improvements.

Release 0.1.3 (Jul 29, 2011)

Bug Fixes

- Improve parsing of floats (thanks to Kris).
- When formatting a statement a space before LIMIT was removed (issue35).
- Fix strip_comments flag (issue38, reported by ooberm...@gmail.com).
- Avoid parsing names as keywords (issue39, reported by djo...@taket.org).
- Make sure identifier lists in subselects are grouped (issue40, reported by djo...@taket.org).

- Split statements with IF as functions correctly (issue33 and issue29, reported by charles....@unige.ch).
- Relax detection of keywords, esp. when used as function names (issue36, nyuhu...@gmail.com).
- Don't treat single characters as keywords (issue32).
- Improve parsing of stand-alone comments (issue26).
- Detection of placeholders in paramterized queries (issue22, reported by Glyph Lefkowitz).
- Add parsing of MS Access column names with braces (issue27, reported by frankz...@gmail.com).

Other

- Replace Django by Flask in App Engine frontend (issue11).

Release 0.1.2 (Nov 23, 2010)

Bug Fixes

- Fixed incorrect detection of keyword fragments embed in names (issue7, reported and initial patch by andyboyko).
- Stricter detection of identfier aliases (issue8, reported by estama).
- WHERE grouping consumed closing parenthesis (issue9, reported by estama).
- Fixed an issue with trailing whitespaces (reported by Kris).
- Better detection of escaped single quotes (issue13, reported by Martin Brochhaus, patch by bluemaro with test case by Dan Carley).
- Ignore identifier in double-quotes when changing cases (issue 21).
- Lots of minor fixes targeting encoding, indentation, statement parsing and more (issues 12, 14, 15, 16, 18, 19).
- Code cleanup with a pinch of refactoring.

Release 0.1.1 (May 6, 2009)

Bug Fixes

- Lexers preserves original line breaks (issue1).
- Improved identifier parsing: backtick quotes, wildcards, T-SQL variables prefixed with @.
- Improved parsing of identifier lists (issue2).
- Recursive recognition of AS (issue4) and CASE.
- Improved support for UPDATE statements.

Other

- Code cleanup and better test coverage.

Release 0.1.0 (Apr 8, 2009)

Initial release.

Indices and tables

- genindex
- modindex
- search

CHAPTER 3

Resources

Project page https://github.com/andialbrecht/sqlparse

Bug tracker https://github.com/andialbrecht/sqlparse/issues

Documentation https://sqlparse.readthedocs.io/

Python Module Index

S

Index