SQL Params Documentation

Release 5.0.0

Caleb P. Burns

Aug 31, 2022

Contents

Contents:

CHAPTER 1

SQL Params

sqlparams is a utility package for converting between various SQL parameter styles. This can simplify the use of SQL parameters in queries by allowing the use of named parameters where only ordinal are supported. Some Python DB API 2.0 compliant modules only support the ordinal *qmark* or *format* style parameters (e.g., pyodbc only supports *qmark*). This package provides a helper class, *SQLParams*, that is used to convert from any parameter style (*qmark*, *numeric*, *named*, *format*, *pyformat*; and the non-standard *numeric_dollar* and *named_dollar*), and have them safely converted to the desired parameter style.

1.1 Tutorial

You first create an *SQLParams* instance specifying the named parameter style you're converting from, and what ordinal style you're converting to. Let's convert from *named* to *qmark* style:

```
>>> import sqlparams
>>> query = sqlparams.SQLParams('named', 'qmark')
```

Now, lets to convert a simple SQL SELECT query using the *SQLParams.format()* method which accepts an SQL query, and a dict of parameters:

```
>>> sql, params = query.format('SELECT * FROM users WHERE name = :name;', {'name':
↪"Thorin"})
```

This returns the new SQL query using ordinal *qmark* parameters with the corresponding list of ordinal parameters, which can be passed to the .execute() method on a database cursor:

```
>>> print sql
SELECT * FROM users WHERE name = ?;
>>> print params
['Thorin']
```

tuples are also supported which allows for safe use of the SQL IN operator:

```
>>> sql, params = query.format("SELECT * FROM users WHERE name IN :names;", {'names':␣
↪("Dori", "Nori", "Ori")})
>>> print sql
SELECT * FROM users WHERE name in (?,?,?);
>>> print params
['Dori', 'Nori', 'Ori']
```

You can also format multiple parameters for a single, shared query useful with the .executemany() method of a database cursor:

```
>>> sql, manyparams = query.formatmany("UPDATE users SET age = :age WHERE name =␣
↪:name;", [{'name': "Dwalin", 'age': 169}, {'name': "Balin", 'age': 178}])
>>> print sql
UPDATE users SET age = ? WHERE name = ?;
>>> print manyparams
[[169, 'Dwalin'], [178, 'Balin']]
```

Please note that if an expanded `tuple` is used in *`SQLParams.formatmany()`*, the tuple must be the same size in each of the parameter lists. Otherwise, you might well use *`SQLParams.format()`* in a for-loop.

1.2 Source

The source code for *`sqlparams`* is available from the GitHub repo cpburnz/python-sql-parameters.

1.3 Installation

`sqlparams` can be installed from source with:

```
python setup.py install
```

`sqlparams` is also available for install through PyPI:

```
pip install sqlparams
```

1.4 Documentation

Documentation for *`sqlparams`* is available on Read the Docs.

1.5 API

`sqlparams` is a utility package for converting between various SQL parameter styles.

class sqlparams.**SQLParams**

The *`SQLParams`* class is used to support named parameters in SQL queries where they are not otherwise supported (e.g., pyodbc). This is done by converting from one parameter style query to another parameter style query.

By default, when converting to a numeric or ordinal style any `tuple` parameter will be expanded into "(?,?,...)" to support the widely used "IN {tuple}" SQL expression without leaking any unescaped values.

__init__(*in_style: str, out_style: str, escape_char: Union[str, bool, None] = None, expand_tuples: Optional[bool] = None, strip_comments: Union[Sequence[Union[str, Tuple[str, str]]], bool, None] = None*) → None

Instantiates the *SQLParams* instance.

in_style (`str`) is the parameter style that will be used in an SQL query before being parsed and converted to *SQLParams.out_style*.

out_style (`str`) is the parameter style that the SQL query will be converted to.

escape_char (`str`, `bool`, or `None`) is the escape character used to prevent matching an in-style parameter. If `True`, use the default escape character (repeat the initial character to escape it; e.g., "%%"). If `False`, do not use an escape character. Default is `None` for `False`.

expand_tuples (`bool` or `None`) is whether to expand tuples into a sequence of parameters. Default is `None` to let it be determined by *out_style* (to maintain backward compatibility). If *out_style* is a numeric or ordinal style, expand tuples by default (`True`). If *out_style* is a named style, do not expand tuples by default (`False`).

Note: Empty tuples will be safely expanded to `(NULL)` to prevent SQL syntax errors,

---

strip_comments (`Sequence`, `bool`, or `None`) whether to strip out comments and what style of comments to remove. If a `Sequence`, this defines the comment styles. A single line comment is defined using a `str` (e.g., `"--"` or `"#"`). A multiline comment is defined using a `tuple` of `str` (e.g., `("/*", "*/")`). In order for a comment to be matched, it must be the first string of non-whitespace characters on the line. Trailing comments are not supported and will be ignored. A multiline comment will consume characters until the ending string is matched. If `True`, `DEFAULT_COMMENTS` will be used (`"--"` and `("/*", "*/")` styles). Default is `None` to not remove comments.

The following parameter styles are supported by both *in_style* and *out_style*:

- For all named styles the parameter keys must be valid Python identifiers. They cannot start with a digit. This is to help prevent incorrectly matching common strings such as datetimes.

  Named styles:

  - "named" indicates parameters will use the named style:

    ```
    ... WHERE name = :name
    ```

  - "named_dollar" indicates parameters will use the named dollar sign style:

    ```
    ... WHERE name = $name
    ```

    Note: This is not defined by PEP 249.

  - "pyformat" indicates parameters will use the named Python extended format style:

    ```
    ... WHERE name = %(name)s
    ```

    Note: Strictly speaking, PEP 249 only specifies "%(name)s" for the "pyformat" parameter style so only that form (without any other conversions or flags) is supported.

- All numeric styles start at `1`. When using a `Sequence` for the parameters, the 1st parameter (e.g., ":1") will correspond to the 1st element of the sequence (i.e., index `0`). When using a `Mapping` for the parameters, the 1st parameter (e.g., ":1") will correspond to the matching key (i.e., `1` or `"1"`).

  Numeric styles:

  - "numeric" indicates parameters will use the numeric style:

    ```
    ... WHERE name = :1
    ```

  - "numeric_dollar" indicates parameters will use the numeric dollar sign style (starts at `1`):

    ```
    ... WHERE name = $1
    ```

    Note: This is not defined by PEP 249.

- Ordinal styles:

  - "format" indicates parameters will use the ordinal Python format style:

    ```
    ... WHERE name = %s
    ```

    Note: Strictly speaking, PEP 249 only specifies "%s" for the "format" parameter styles so only that form (without any other conversions or flags) is supported.

  - "qmark" indicates parameters will use the ordinal question mark style:

    ```
    ... WHERE name = ?
    ```

escape_char
escape_char (`str` or `None`) is the escape character used to prevent matching an in-style parameter.

expand_tuples
expand_tuples (`bool`) is whether to convert tuples into a sequence of parameters.

format(*sql: AnyStr, params: Union[Dict[Union[str, int], Any], Sequence[Any]]*) → Tuple[AnyStr, Union[Dict[Union[str, int], Any], Sequence[Any]]]
Convert the SQL query to use the out-style parameters instead of the in-style parameters.

sql (`str` or `bytes`) is the SQL query.

params (`Mapping` or `Sequence`) contains the set of in-style parameters. It maps each parameter (`str` or `int`) to value. If *`SQLParams.in_style`* is a named parameter style. then *params* must be a `Mapping`. If *`SQLParams.in_style`* is an ordinal parameter style, then *params* must be a `Sequence`.

Returns a `tuple` containing:

- The formatted SQL query (`str` or `bytes`).
- The set of converted out-style parameters (`dict` or `list`).

formatmany(*sql: AnyStr, many_params: Union[Iterable[Dict[Union[str, int], Any]], Iterable[Sequence[Any]]]*) → Tuple[AnyStr, Union[List[Dict[Union[str, int], Any]], List[Sequence[Any]]]]
Convert the SQL query to use the out-style parameters instead of the in-style parameters.

sql (`str` or `bytes`) is the SQL query.

many_params (`Iterable`) contains each set of in-style parameters (*params*).

- *params* (`Mapping` or `Sequence`) contains the set of in-style parameters. It maps each parameter (`str` or `int`) to value. If *`SQLParams.in_style`* is a named parameter style. then *params* must be a `Mapping`. If *`SQLParams.in_style`* is an ordinal parameter style. then *params* must be a `Sequence`.

Returns a `tuple` containing:

- The formatted SQL query (`str` or `bytes`).
- A `list` containing each set of converted out-style parameters (`dict` or `list`).

in_style
in_style (`str`) is the parameter style to expect in an SQL query when being parsed.

out_style
out_style (`str`) is the parameter style that the SQL query will be converted to.

strip_comments
strip_comments (`Sequence` or `None`) contains the comment styles to remove.

CHAPTER 2

Change History

2.1 5.0.0 (2022-08-11)

- Dropped support of EOL Python 3.6.
- Support Python 3.11.
- Changed build system to pyproject.toml and build backend to setuptools.build_meta which may have unforeseen consequences.
- Safely expand empty tuples. Fixes Issue #8.
- Add support for stripping comments. This helps prevent expansion of unexpected variables in comments. Fixes Issue #9.
- Rename GitHub project from python-sql-parameters to python-sqlparams.

2.2 4.0.0 (2022-06-06)

- Drop support for EOL Python 3.5.
- Issue #10: When converting to 'format'/'pyformat' types, escape existing '%' characters.
- When converting from 'format'/'pyformat' types, set *escape_char=True* to unescape double '%' characters.

2.3 3.0.0 (2020-04-04)

- Major changes to internal implementation.
- Support converting any parameter style to any parameter style (all named, numeric, and ordinal styles).
- Renamed attribute *named* to *in_style* on *sqlparams.SQLParams*.
- Renamed attribute *ordinal* to *out_style* on *sqlparams.SQLParams*.

- Removed attributes *match* and *replace* from *sqlparams.SQLParams* which should have been private.
- Named parameters must now be valid identifiers (can no longer start with a digit to help prevent incorrectly matching common strings such as datetimes). Fixes Issue #4.
- Issue #7: Support dollar sign style for numeric and named parameters.

2.4 2.0.0 (2020-02-26)

- Drop support for EOL Python 2.7, 3.2, 3.3, 3.4.

2.5 1.2.0 (2020-02-26)

- Require setuptools.
- Support up to Python 3.8.

2.6 1.1.2 (2018-05-04)

- Improved support for byte strings.

2.7 1.1.1 (2017-09-07)

- Fixed support for byte strings.

2.8 1.1.0 (2017-08-30)

- Support Python 3.2+.

2.9 1.0.3 (2012-12-28)

- Fixed documentation for issue 1.

2.10 1.0.2 (2012-12-22)

- Added sphinx documentation.

2.11 1.0.1 (2012-12-20)

- Fixed running test as a script.

2.12 1.0.0 (2012-12-20)

- Initial release.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

Symbols

E

F

I

O

S