bearsql Documentation

Release 0.1.0

Shrinivas Vijay Deshmukh

Sep 20, 2023

CONTENTS:

CHAPTER

ONE

BEARSQL

Bearsql adds sql syntax on pandas dataframe. It uses duckdb to speedup the pandas processing and as the sql engine

- Free software: MIT license
- Documentation: https://bearsql.readthedocs.io.

1.1 Basic Usage

To use bearsql in a project:

```
from bearsql import SqlContext
import pandas as pd

sc = SqlContext()
# The above statement will create duckdb instance in memory. Once the session ends, the␣
↪database will be erased and not be persisted
# To persist the database, you can instantiate sqlcontext like:
# sc = SqlContext(database='<YOUR_DATABASE_NAME>.db'

df = pd.DataFrame([{'name': 'John Doe', 'city': 'New York', 'age': 24}, {'name': 'Jane␣
↪Doe', 'city': 'Chicago', 'age': 27}])

# Create table from pandas dataframe
sc.register_table(df, 'testable') # <YOUR_TABLENAME> instead of 'testable'

# Query table and output to pandas dataframe
results = sc.sql('select * from testable', output='df')
output_df = next(results)
print(output_df)

# Query table and output to pyarrow table
results = sc.sql('select * from testable', output='arrow')
output_arrow_table = next(results)
print(output_arrow_table)
```

(continues on next page)

(continued from previous page)

```
# Query table and output raw tuples
results = sc.sql('select * from testable', output='any')
output_rows = next(results)
print(output_rows)
```

Create a relational table from dataframe and apply some operations:

```
rel = sc.relation(df, 'new_relation') # <YOUR_RELATION_NAME> instead of new_relation

print(rel.filter('age > 24'))

# OR convert to df:

rel.filter('age > 24').df()
```

Export the data to filesystem:

```
result = sc.sql('EXPORT DATABASE \'<OUTPUT_FOLDER>\' (FORMAT PARQUET);') # format can␣
↪either be PARQUET or CSV
list(result)
```

For more examples, please visit https://github.com/duckdb/duckdb/blob/master/examples/python/duckdb-python.py

1.2 Features

- TODO

1.3 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install bearsql, run this command in your terminal:

```
$ pip install bearsql
```

This is the preferred method to install bearsql, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for bearsql can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/shrinivdeshmukh/bearsql
```

Or download the tarball:

```
$ curl -OJL https://github.com/shrinivdeshmukh/bearsql/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

THREE

USAGE

To use bearsql in a project:

```
from bearsql import SqlContext
import pandas as pd

sc = SqlContext()
# The above statement will create duckdb instance in memory. Once the session ends, the␣
↪database will be erased and not be persisted
# To persist the database, you can instantiate sqlcontext like:
# sc = SqlContext(database='<YOUR_DATABASE_NAME>.db'

df = pd.DataFrame([{'name': 'John Doe', 'city': 'New York', 'age': 24}, {'name': 'Jane␣
↪Doe', 'city': 'Chicago', 'age': 27}])

# Create table from pandas dataframe
sc.register_table(df, 'testable') # <YOUR_TABLENAME> instead of 'testable'

# Query table and output to pandas dataframe
results = sc.sql('select * from testable', output='df')
output_df = next(results)
print(output_df)

# Query table and output to pyarrow table
results = sc.sql('select * from testable', output='arrow')
output_arrow_table = next(results)
print(output_arrow_table)

# Query table and output raw tuples
results = sc.sql('select * from testable', output='any')
output_rows = next(results)
print(output_rows)
```

Create a relational table from dataframe and apply some operations:

```
rel = sc.relation(df, 'new_relation') # <YOUR_RELATION_NAME> instead of new_relation

print(rel.filter('age > 24'))

# OR convert to df:

rel.filter('age > 24').df()
```

Export the data to filesystem:

```
result = sc.sql('EXPORT DATABASE \'<OUTPUT_FOLDER>\' (FORMAT PARQUET);') # format can␣
↪either be PARQUET or CSV
list(result)
```

For more examples, please visit https://github.com/duckdb/duckdb/blob/master/examples/python/duckdb-python.py

CHAPTER

FOUR

BEARSQL

4.1 bearsql package

4.1.1 Submodules

4.1.2 bearsql.bearsql module

Main module.

class bearsql.bearsql.**SqlContext**(*table: Optional[str] = None*, *view: Optional[str] = None*, *database: Optional[str] = None*)

Bases: `object`

close()
Method to close database connection

register_table(*df: pandas.core.frame.DataFrame*, *table: Optional[str] = None*) → None
This method creates a table in the database with pandas dataframe as the input. To create a table, a view must be created. If there is no view name specified in this class, a new random view name will be generated

Parameters

- **df** (*DataFrame*) – pandas input dataframe
- **table** (*Optional[str]; default None*) – table name; the dataframe will sit in the database and can be referenced using this table name

register_view(*df: pandas.core.frame.DataFrame*, *view: Optional[str] = None*) → None
This method creates a view in the database with pandas dataframe as the input. If there is no view name is passed and not specified in this class, an exception will be thrown

Parameters

- **df** (*DataFrame*) – pandas input dataframe
- **view** (*Optional[str]; default None*) – table name; the dataframe will sit in the database and can be referenced using this table name

relation(*df: pandas.core.frame.DataFrame*, *table: Optional[str] = None*)
Create a relational table on top of pandas dataframe. If tagged with a table name, this name can be used to run sql queries.

Parameters

- **df** (*DataFrame*) – pandas input dataframe
- **table** (*Optional[str]; default None*) – name of the table

returns: duckdb relation :rtype: duckdb

sql(*query: Union[str, list]*, *output: str = 'df'*) → Generator
Method to run sql queries on pandas dataframe.

Parameters

- **query** (*Union[str, list]*) – sql query to execute on pandas dataframe. It can be one single query or a list of multiple queries
- **output** (*str; default df*) – Output format of the query results. This can either be df, arrow or any

returns: Generator object containing all the query results :rtype: Generator

property table
Table property of the class. This will give the name of the table that is currently in use

property view
View property of the class. This will give the name of the view that is currently in use

4.1.3 bearsql.cli module

Console script for bearsql.

bearsql.cli.**main**()
Console script for bearsql.

4.1.4 bearsql.log_source module

class bearsql.log_source.**Logging**(*log_level*)
Bases: object

get_logger()

4.1.5 Module contents

Top-level package for bearsql.

CHAPTER

FIVE

CONTRIBUTING

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

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/shrinivdeshmukh/bearsql/issues.

If you are 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.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

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

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/shrinivdeshmukh/bearsql/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 :)

5.2 Get Started!

Ready to contribute? Here's how to set up *bearsql* for local development.

1. Fork the *bearsql* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/bearsql.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv bearsql
$ cd bearsql/
$ python setup.py develop
```

4. Create a branch for local development:

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

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 bearsql tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. 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
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/shrinivdeshmukh/bearsql/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_bearsql
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

SIX

CREDITS

6.1 Development Lead

- Shrinivas Vijay Deshmukh <shrinivas.deshmukh11@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER

SEVEN

HISTORY

7.1 0.1.0 (2021-06-30)

- First release on PyPI.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

INDEX