PandasDatabase Documentation

Release 0.3.3

Oscar Martinez

February 19, 2016

Contents

CHAPTER 1

PandasDatabase

Prototyping database engine for Python

- Free software: MIT License
- Documentation: https://pddb.readthedocs.org.

1.1 Introduction

PandasDatabase is a RESTful database engine application built on top of Pandas. Essentially, it is an abstraction layer that projects the database-table-column model into a very simple set of API's. As a database engine, it has some useful features that make it a good candidate for prototype work:

- Inherits all the performance and robustness of Pandas.
- Very simple and intuitive API set.
- Tables support dynamic schema, so every time columns are changed during development there is no need to alter tables or CREATE statements.
- All data is persisted in plaintext, human-readable CSV format.

Some of those features come at a cost that probably makes PandasDatabase less than ideal for production environments:

- Security. At the server, the security model is based on file permissions. For the API's, production environments should very likely never expose database API's of any form.
- Performance. While low latency production environments might not run into an issue, performance-critical applications will probably run into a bottleneck when writing to disk.
- Data types. Exposing all the table data in CSV format means that complex data types such as date cannot be supported.

1.2 The Problem

A very large number of small projects have fairly simple requirements for data storage. For all those projects, interfacing with a database engine is boilerplate work that adds unnecessary overhead during development. This project provides a very simple yet powerful solution that is great for prototype work to enable those small projects to hit the ground running. Once the critical components of the project are finished and a proof of concept version is running, projects can transition to a more mature database engine that can better suit the needs of a production environment.

1.3 Getting Started

This project is entirely Python based. To be able to use it, first install the dependencies:

```
$ pip install pandas bottle
```

The easiest way to install PandasDatabase is using pip:

```
$ pip install pddb
```

To fire up the database engine, simply run:

```
$ python -m pddb.pddb dbname --permissions w
```

By default, the database is started in read-only mode, which is why we need to pass the `--permissions w` flag. This should start a bottle application with the following endpoints available:

- `/pddb`
- `/pddb/find/<table>`
- `/pddb/insert/<table>`
- `/pddb/upsert/<table>`
- `/pddb/delete/<table>`

The parameters to those endpoints can be passed as a GET query string, or via POST. For example, to insert a new record, the user can simply visit the following URL once the database engine is running:

```
http://127.0.0.1:8080/pddb/insert/table_name?Name=John
```

Likewise, the user can find the inserted record by visiting:

```
http://127.0.0.1:8080/pddb/find/table_name
```

Matching conditions can also be added:

```
http://127.0.0.1:8080/pddb/find/table_name?Name=John
```

Performing an update is a little more complicated. Rather than exposing multiple API's, a single API is used and the parameters are parsed to understand the user's desired operation. So, instead of using `/pddb/upsert/table_name?column_name=column_value`, the user must use `/pddb/upsert/table_name?record__column_name=column_value&where__condition_name=condition_va`
Essentially, prepend `record__` or `where__` to let the database engine know which pair of key-value corresponds to what parameter. For example, to change the name `John` to `Jane` in our record, we can simply visit:

```
http://127.0.0.1:8080/pddb/upsert/table_name?record__Name=Jane&where__Name=John
```

Note that this also applies to the rest of the API's, even though the parameters being parsed default to the most obvious choice. For example, `/pddb/find` assumes that the parameters in the query string correspond to the equivalent of a `WHERE` in `SQL`. However, the find query can also be written as:

```
http://127.0.0.1:8080/pddb/find/table_name?where__Name=John
```

The usefulness of this does not appear evident until more complex queries are used, such as `WHERE-NOT`:

```
http://127.0.0.1:8080/pddb/find/table_name?where_not__Name=John
```

While admittedly a bit quirky, these very simple API's allow for any application in any language to interface with the database engine by performing very simple GET requests. The user does not need to worry about exposing an API or

interfacing with a database in another process or server, which gives more time to developing the critical parts of the project first.

1.4 License

CHAPTER 2

Installation

At the command line:

```
$ pip install pddb
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv pddb
$ pip install pddb
```

CHAPTER 3

Usage

To fire up the database engine, simply run:

```
$ python -m pddb.pddb dbname --permissions w
```

By default, the database is started in read-only mode, which is why we need to pass the `--permissions w` flag. This should start a bottle application with the following endpoints available:

- `/pddb`
- `/pddb/find/<table>`
- `/pddb/insert/<table>`
- `/pddb/upsert/<table>`
- `/pddb/delete/<table>`

The parameters to those endpoints can be passed as a GET query string, or via POST. For example, to insert a new record, the user can simply visit the following URL once the database engine is running:

```
http://127.0.0.1:8080/pddb/insert/table_name?Name=John
```

Likewise, the user can find the inserted record by visiting:

```
http://127.0.0.1:8080/pddb/find/table_name
```

Matching conditions can also be added:

```
http://127.0.0.1:8080/pddb/find/table_name?Name=John
```

Performing an update is a little more complicated. Rather than exposing multiple API's, a single API is used and the parameters are parsed to understand the user's desired operation. So, instead of using `/pddb/upsert/table_name?column_name=column_value`, the user must use `/pddb/upsert/table_name?record__column_name=column_value&where__condition_name=condition_va`
Essentially, prepend `record__` or `where__` to let the database engine know which pair of key-value corresponds to what parameter. For example, to change the name `John` to `Jane` in our record, we can simply visit:

```
http://127.0.0.1:8080/pddb/upsert/table_name?record__Name=Jane&where__Name=John
```

Note that this also applies to the rest of the API's, even though the parameters being parsed default to the most obvious choice. For example, `/pddb/find` assumes that the parameters in the query string correspond to the equivalent of a `WHERE` in `SQL`. However, the find query can also be written as:

```
http://127.0.0.1:8080/pddb/find/table_name?where__Name=John
```

The usefulness of this does not appear evident until more complex queries are used, such as `WHERE-NOT`:

```
http://127.0.0.1:8080/pddb/find/table_name?where_not__Name=John
```

While admittedly a bit quirky, these very simple API's allow for any application in any language to interface with the database engine by performing very simple GET requests. The user does not need to worry about exposing an API or interfacing with a database in another process or server, which gives more time to developing the critical parts of the project first.

CHAPTER 4

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/omtinez/pddb/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/omtinez/pddb/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.

4.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/pddb.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 pddb
$ cd pddb/
$ 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 pddb tests
$ python setup.py test
$ 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.

4.3 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 wherever appropriate in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/omtinez/pddb/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

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

CHAPTER 5

Credits

5.1 Development Lead

- Oscar Martinez <omtinez@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.0 (2016-01-10)

- First release on PyPI.

6.2 0.1.1 (2016-01-10)

- Fix case sensitive tests
- Disable testing for all versions except 2.7, 3.3 and 3.4
- Update docs

6.3 0.2.0 (2016-01-12)

- Do not attempt to drop table if it does not exist
- Improved Windows compatibility

6.4 0.3.0 (2016-01-17)

- Delete database folder only if empty after dropping all tables
- Improved Windows compatibility

6.5 0.3.1 (2016-01-18)

- Added auto_cast option
- String type enforcing enabled
- Unicode and byte type support

6.6 0.3.2 (2016-02-14)

- Added ability to use custom bottle route methods

6.7 0.3.3 (2016-02-15)

- Delete CSV files if dataframe is empty when saving
- Expose tostr() method as a public method

CHAPTER 7

Indices and tables

- genindex
- modindex
- search