dbf_to_sql Documentation

Release 0.0.1

James Wang

Apr 22, 2017

Contents

Contents:

CHAPTER 1

Introduction

Motivation

FoxPro is a dying technology, but unfortunately, some organizations still have all of their data in it. This tool is meant to help ease transition into more modern databases (specifically sqlite3).

Synopsis

This module essentially glues together dbf and SQLAlchemy to convert FoxPro tables into SQL tables (specifically, sqlite3 – although this should be easy to adapt and might work out of the box for dialects that SQLAlchemy supports).

At this stage, I would not suggest using this for any dialect other than sqlite (which the Converter defaults to anyhow). If you need to use this with other SQL dialects, use other tools to convert the sqlite3 database to something else.

Requires dbf and SQLAlchemy (both of which are open source and available through pip install).

```
pip install dbf
pip install sqlalchemy
```

Version Warning

Testing for this module has primarily come through my own usage. Although it has been "tested," there's nothing quite like real data to muck up a theoretically sound program.

I'm not looking to use this module to do data validation or clean up horribly mangled databases, but I suspect there will be common errors (especially in domain-specific applications, which FoxPro is often used for), which I haven't yet tripped over. If you feel there's such a case, please open an issue or send me an email.

Usage

You can convert directories of DBF tables directly:

```
import dbf_to_sql

fp2sql = dbf_to_sql.Converter('sqldata.db')
fp2sql.convert_dbfs('foxpro_app/data/')
```

Or just individual tables:

```
fp2.sql.convert_dbf('clients.dbf')
```

CHAPTER 2

Module Documentation

mod *dbf_to_sql* - Converts FoxPro Tables to SQL

This module essentially glues together dbfpy and sqlalchemy to convert FoxPro tables into SQL tables (specifically, sqlite – although this should be easy to adapt and might work out of the box for dialects that sqlalchemy supports, it has not been tested for those cases).

Requires dbf and sqlalchemy (both of which are available through pip install)

class `dbf_to_sql.`**`Converter`**(*sqldb*[, *sql='sqlite'*])

Converts a FoxPro database (directory of .dbf files) to an sqlite3 database stored on disk (will overwrite if already exists).

Theoretically can work for any sql dialect that sqlalchemy supports, but in reality has just been tested for sqlite (and is expected to break in subtle to not-so-subtle ways for other dialects). Use for other dialects at your own risk.

Example:

```
fp2sql = Converter('sqldata.db')
fp2sql.convert_dbfs('foxpro_app/data/')
```

Alternatively:

```
fp2.sql.convert_dbf('clients.dbf')
```

CHAPTER 3

Methods

dbf_to_sql.**convert_dbf**(*foxpro_source*)
: Reads .dbf file and converts to specified sql database, preserving the schema of the foxpro database (as best it can).

dbf_to_sql.**convert_dbfs**(*foxpro_dir*)
: Reads FoxPro database (given as a directory containing .dbf files).

classmethod dbf_to_sql.**convert_type**(*typ_tuple*)
: Takes a field_info tuple (called from dbf.Table.field_info) and converts dbfpy type information into sqlalchemy type information.

classmethod dbf_to_sql.**fix_record**(*read_string*, *col_type*)
: Takes record data as a read_string from dbf and an sqlalchemy datatype (e.g. sqlalchemy.Integer). Performs simple typecasting depending on type, and tries to handle bytestrings.

  Needless to say, this won't save you from corrupted/badly mangled data.

CHAPTER 4

Indices and tables

- genindex
- search

Python Module Index

d

Index