dbgdb Documentation

Release 0.0.5

Pat Daburu

Jul 07, 2018

Contents:

This is a library containing Luigi tasks to help you get that data into (and out of) your GIS databases. The library also comes with a command-line interface based on Click that can be helpful for running tasks individually.

CHAPTER 1

Getting Started

1.1 Prerequisites

1.1.1 Installing the Library

You can use pip to install *dbgdb*.

```
pip install dbgdb
```

1.1.2 GDAL/OGR

You will need to have *ogr2ogr* installed. You can arrange this by installing GDAL.

Note: In this early version we expect *ogr2ogr* to be in your system path. Improvements on that point are forthcoming.

1.2 Using Tasks

This library contains a number of Luigi tasks that you can use in your own pipelines. These include:

- ***dbgdb.tasks.postgres.load.PgLoadTask*** for loading data (like file geodatabases) into your PostgreSQL database;
- ***dbgdb.tasks.postgres.extract.PgExtractTask*** for extrating data from your PostgreSQL database; and
- ***dbgdb.tasks.postgres.drop.PgDropSchemaTask*** if you need to drop an import schema because you're starting over.

1.3 Using the Command Line

Most of the commands you run with the command-line interface (CLI) create Luigi tasks which are then submitted to a Luigi scheduler.

Note: The *-l* parameter indicates that the tasks should be run using the local scheduler. The examples listed below use this parameter. If you want to submit tasks to the Luigi daemon (luigid) you can simply omit this parameter.

1.4 Getting Help

dbgdb has its own command-line help.

```
dbgdb --help
```

1.4.1 Loading Data

You can load a file geodatabase with the *load* subcommand.

```
dbgdb -l load --schema myschema /path/to/your/data.gdb
```

1.4.2 Extracting Data

You can extract all of the data within a schema to an output file with the *extract* subcommand.

```
dbgdb -l extract  --schema myschema /path/to/your/exported/data.db
```

Note: At the moment, we can export to GeoPackage or Spatialite formats. Support for ESRI File Geodatabases (gdb) is still in the works.

1.4.3 Dropping a Schema

If the target schema for your load command already exists, you may notice Luigi reporting there was nothing to do because, from the task's perspective, the work has already been done. If you need to drop a schema, you can use the *drop* subcommand.

```
dbgdb -l drop schema myschema
```

1.5 Resources

Would you like to learn more? Check out the links below.

- Luigi
- Click

CHAPTER 2

API Documentation

2.1 dbgdb

Data pipeline utilities to help you get data into and out of your database.

2.2 dbgdb.db.postgres

This module contains utility functions to help when working with PostgreSQL databases.

dbgdb.db.postgres.**connect**(*url: str*, *dbname: str = None*, *autocommit: bool = False*)
Create a connection to a Postgres database.

Parameters

- **url** – the Postgres instance URL
- **dbname** – the target database name (if it differs from the one specified in the URL)
- **autocommit** – Set the *autocommit* flag on the connection?

Returns a psycopg2 connection

dbgdb.db.postgres.**create_db**(*url: str*, *dbname: str*, *admindb: str = 'postgres'*)
Create a database on a Postgres instance.

Parameters

- **url** – the Postgres instance URL
- **dbname** – the name of the database
- **admindb** – the name of an existing (presumably the main) database

Returns

dbgdb.db.postgres.**create_extensions**(*url: str*)
Create the necessary database extensions.

Parameters **url** – the URL of the database instance

dbgdb.db.postgres.**create_schema**(*url: str*, *schema: str*)
Create a schema in the database.

Parameters

- **url** – the URL of the database instance
- **schema** – the name of the schema

dbgdb.db.postgres.**db_exists**(*url: str*, *dbname: str = None*, *admindb: str = 'postgres'*) → bool
Does a given database on a Postgres instance exist?

Parameters

- **url** – the Postgres instance URL
- **dbname** – the name of the database to test
- **admindb** – the name of an existing (presumably the main) database

Returns *True* if the database exists, otherwise *False*

dbgdb.db.postgres.**drop_schema**(*url: str*, *schema: str*)
Drop a schema from the database.

Parameters

- **url** – the URL of the database instance
- **schema** – the name of the schema

dbgdb.db.postgres.**schema_exists**(*url: str*, *schema: str*)
Does a given schema exist within a Postgres database?

Parameters

- **url** – the Postgres instance URL and database
- **schema** – the name of the schema

Returns *True* if the schema exists, otherwise *False*

dbgdb.db.postgres.**select_schema_tables**(*url: str*, *schema: str*) → Iterable[str]
Select the names of the tables within a given schema.

Parameters

- **url** – the URL of the dat
- **schema** – the name of the schema

2.3 dbgdb.cli

This is the entry point for the command-line interface (CLI) application. It can be used as a handy facility for running the task from a command line.

Note: To learn more about Click visit the project website. There is also a very helpful tutorial video.

To learn more about running Luigi, visit the Luigi project's Read-The-Docs page.

---

class dbgdb.cli.**Info**
Bases: object

This is an information object that can be used to pass data between CLI functions.

__init__()
Initialize self. See help(type(self)) for accurate signature.

dbgdb.cli.**pass_info**(*f*)
pylint: disable=invalid-name

dbgdb.cli.**run**(*tasks: Iterable[luigi.task.Task], info: dbgdb.cli.Info*)
Run tasks on the local scheduler.

Parameters

- **tasks** – the tasks to run
- **info** – the *Info* object containing other parameters

2.4 dbgdb.ogr.postgres

This module contains wrapper functions for work that would generally be handled by OGR/GDAL.

class dbgdb.ogr.postgres.**OgrDrivers**
Bases: enum.Enum

These are the supported OGR drivers.

GeoPackage = 'GPKG'

PostGIS = 'PostgreSQL'

Spatialite = 'SQLITE'

dbgdb.ogr.postgres.**extract**(*outdata: pathlib.Path, schema: str = 'imports', url: str = 'postgresql://postgres@localhost:5432/postgres', driver: dbgdb.ogr.postgres.OgrDrivers = <OgrDrivers.Spatialite: 'SQLITE'>*)
Extract a schema from a PostgreSQL database to a file geodatabase.

Parameters

- **outdata** – the path to the output
- **schema** – the schema to export
- **url** – the URL of the Postgres database instance
- **driver** – the OGR driver to use

dbgdb.ogr.postgres.**load**(*indata: pathlib.Path, url: str = 'postgresql://postgres@localhost:5432/postgres', schema: str = 'imports', overwrite: bool = True, progress: bool = True, use_copy: bool = True, driver: dbgdb.ogr.postgres.OgrDrivers = <OgrDrivers.PostGIS: 'PostgreSQL'>*)
Load a file geodatabase (GDB) into a Postgres database.

Parameters

- **indata** – the path to the file geodatabase
- **url** – the URL of the Postgres instance

- **schema** – the target schema
- **overwrite** – Overwrite existing data?
- **progress** – Show progress?
- **use_copy** – Use COPY instead of INSERT when loading?
- **driver** – the OGR driver to use

2.5 dbgdb.targets.postgres

This module contains PostgreSQL targets.

class dbgdb.targets.postgres.**PgSchemaTarget**(*url: str*, *schema: str*, *dne: bool = False*)
Bases: luigi.target.Target

This is a target that represents a file geodatabase (GDB).

__init__(*url: str*, *schema: str*, *dne: bool = False*)

Parameters

- **url** – the path to the file GDB
- **schema** – the target schema
- **dne** – ("does not exist") this should be *True* if the target's task is considered to be complete if the schema does not exist

connect()
Get a connection to the database.

Returns a connection to the database

exists() → bool
Does the file target schema exist?

Returns *True* if the file geodatabase exists, otherwise *False*

2.6 dbgdb.tasks.postgres.drop

This module contains the PgDropSchemaTask task which you can use to drop a schema.

class dbgdb.tasks.postgres.drop.**PgDropSchemaTask**(**args*, ***kwargs*)
Bases: luigi.task.Task

This task loads a file geodatabase into a database instance.

Variables

- **url** – the URL of the target database
- **schema** – the target schema

output() → dbgdb.targets.postgres.PgSchemaTarget
This task returns a PgSchemaTarget that points to the target schema where the GDB was loaded.

Returns the PostgreSQL schema target

requires()
This task has no requirements.

Returns an empty iteration

run()
Run the task.

schema = <luigi.parameter.Parameter object>

url = <luigi.parameter.Parameter object>

2.7 dbgdb.tasks.postgres.extract

This module contains the `ExtractTask` task which you can use to extract data from your database instance.

class dbgdb.tasks.postgres.extract.**PgExtractTask**(**args*, ***kwargs*)
Bases: `luigi.task.Task`

This task loads a file geodatabase into a database instance.

Variables

- **url** – the URL of the target database
- **schema** – the target schema
- **outdata** – the path to which data should be exported
- **driver** – the driver to use for exporting data

driver = <luigi.parameter.EnumParameter object>

outdata = <luigi.parameter.Parameter object>

output() → luigi.local_target.LocalTarget
This task returns a `PgSchemaTarget` that points to the target schema where the GDB was loaded.

Returns the PostgreSQL schema target

requires()
This task has no requirements.

Returns an empty iteration

run()
Run the task.

schema = <luigi.parameter.Parameter object>

url = <luigi.parameter.Parameter object>

2.8 dbgdb.tasks.postgres.load

This module contains the `LoadTask` task which you can use to load a file geodatabase into your database instance.

class dbgdb.tasks.postgres.load.**PgLoadTask**(**args*, ***kwargs*)
Bases: `luigi.task.Task`

This task loads a file geodatabase into a database instance.

Variables

- **url** – the URL of the target database
- **schema** – the target schema

- **indata** – the path to the input data

indata = <luigi.parameter.Parameter object>

output() → dbgdb.targets.postgres.PgSchemaTarget
This task returns a `PgSchemaTarget` that points to the target schema where the data was loaded.

Returns the PostgreSQL schema target

requires()
This task has no requirements.

Returns an empty iteration

run()
Run the task.

schema = <luigi.parameter.Parameter object>

url = <luigi.parameter.Parameter object>

CHAPTER 3

Python Module Dependencies

The `requirements.txt` file contains this project's module dependencies. You can install these dependencies using `pip`.

```
pip install -r requirements.txt
```

3.1 requirements.txt

```
addict>=2.1.3,<3
click>=6.7,<7
luigi>=2.7.5,<3
luijo>=0.0.17,<0.1
mock>=2.0.0,<3
parameterized>=0.6.1,<1
pip-check-reqs>=2.0.1,<3
psycopg2-binary>=2.7.4,<3
pylint>=1.8.4,<2
pytest>=3.4.0,<4
pytest-cov>=2.5.1,<3
pytest-pythonpath>=0.7.2,<1
setuptools>=38.4.0
Sphinx==1.7.2
sphinx-rtd-theme==0.3.0
testing.postgresql>=1.3.0,<2
tox>=3.0.0,<4
twine>=1.11.0,<2
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

Symbols

C

D

E

G

I

L

O

P

R

S

U