SQLSorcery

Apr 28, 2021

Contents

CHAPTER 1

README

SQLSorcery: Dead simple wrapper for pandas and sqlalchemy

SQLSorcery is just some good old fashion syntactic sugar . It really doesn't do anything new. It just makes doing it a little bit easier. It started as a connection wrapper for SQLAlchemy to cut down on the need for boilerplate code that was used to keep the database credentials secret, connect to the database, and then pass the connection to Pandas for queries and inserts.

It wasn't much code, but needing to cut and paste it for each new project seemed like a recipe for bugs. So here we are. We've added more utility methods to the module as well as added all of the basic dialects of SQL that SQLAlchemy supports.

In many cases, the methods available are less robust than the underlying libraries and are more of a shortcut. When you need something that is outside the bounds of the defaults you can always drop back down into Pandas or SQLAlchemy to get more functionality/flexibility.

1.1 Getting Started

1.1.1 Install this library

```
$ pipenv install sqlsorcery
```

By default, **sqlsorcery** does not install the sql dialect specific python drivers. To install these, you can specify the dialects as a comma separated list of each dialect you will need drivers for.

```
$ pipenv install sqlsorcery[mssql]
```

OR

```
$ pipenv install sqlsorcery[mysql,postgres]
```

1.1.2 Setup .env file with credentials

For use with a single database:

```
DB_SERVER=
DB_PORT=
DB=
DB_SCHEMA=
DB_USER=
DB_PWD=
```

Otherwise, refer to the documentation for instructions.

1.2 Examples

1.2.1 Query a table

```
from sqlsorcery import MSSQL


sql = MSSQL()
df = sql.query("SELECT * FROM my_table")
print(df)
```

1.2.2 Query from a *.sql* file

```
from sqlsorcery import MSSQL


sql = MSSQL()
df = sql.query_from_file("filename.sql")
print(df)
```

1.2.3 Insert into a table

```
from sqlsorcery import MSSQL
import pandas as pd


sample_data = [
    { "name": "Test 1", "value": 98 },
    { "name": "Test 2", "value": 100 },
]

df = pd.DataFrame(sample_data)
```

(continues on next page)

(continued from previous page)

```
sql = MSSQL()
sql.insert_into("table_name", df)
```

1.2.4 Execute a stored procedure

```
from sqlsorcery import MSSQL


sql = MSSQL()
sql.exec_sproc("sproc_name")
```

1.3 Documentation

Documentation and tutorials available at sqlsorcery.readthedocs.io

CHAPTER 2

LICENSE

CHAPTER 3

Installation

While you can install SQLSorcery with pip via pypi we encourage the use of pipenv because SQLSorcery takes advantage of environment variables for connection secrets.

If you choose to install directly with pip, you will want to install a module like python-dotenv to handle your .env file variables or manage them some other secure way. This cookbook assumes use of pipenv and all examples assume you are running your code through the pipenv shell.

3.1 Installing with Pipenv

```
$ pipenv install sqlsorcery
```

By default sqlsorcery does not install the sql dialect specific python drivers. To install these, you can specify the dialects as a comma separated list of each dialect you will need drivers for.

```
$ pipenv install sqlsorcery[mssql]
```

OR

```
$ pipenv install sqlsorcery[mysql,postgres]
```

3.2 Additional drivers

Both Microsoft SQL and Oracle require additional system level drivers in order to function.

3.2.1 MS SQL

You can find directions for installing MS SQL ODBC drivers for various systems on docs.microsoft.com

Example for installing in a Python Dockerfile:

```
FROM python:3
WORKDIR /code
RUN wget https://packages.microsoft.com/debian/9/prod/pool/main/m/msodbcsql17/
↪msodbcsql17_17.4.1.1-1_amd64.deb
RUN apt-get update
RUN apt-get install -y apt-utils
RUN apt-get install -y unixodbc unixodbc-dev
RUN pip install pipenv
COPY Pipfile .
RUN pipenv install --skip-lock
RUN yes | dpkg -i msodbcsql17_17.4.1.1-1_amd64.deb
COPY ./ .
ENTRYPOINT ["pipenv", "run", "python", "main.py"]
```

3.2.2 Oracle

You can find directions for installing Oracle instant-client drivers for various systems on oracle.com

Example for installing in a Python Dockerfile:

```
FROM python:3
WORKDIR /code
RUN mkdir -p /opt/oracle
RUN wget https://download.oracle.com/otn_software/linux/instantclient/193000/
↪instantclient-basic-linux.x64-19.3.0.0.0dbru.zip -P /opt/oracle
RUN cd /opt/oracle && unzip instantclient-basic-linux.x64-19.3.0.0.0dbru.zip
RUN ln -s /opt/oracle/instantclient_19_3/libclntsh.so.19.1 /opt/oracle/libclntsh.so
ENV LD_LIBRARY_PATH="/opt/oracle/instantclient_19_3:${LD_LIBRARY_PATH}"
RUN apt-get update
RUN apt-get install -y libaio1
RUN pip install pipenv
COPY Pipfile .
RUN pipenv install --skip-lock
COPY ./ .
ENTRYPOINT ["pipenv", "run", "python", "main.py"]
```

CHAPTER 4

Environment

Storing your progam's config in the environment is a common practice for several reasons:

1. You can add your *.env* to your *.gitignore* to ensure credentials and other project secrets don't end up exposed in your remote repository.
2. It makes it easier to add environment specific variables (development vs. production)
3. Separates config from code. Config varies across deploys, code does not.

One way to leverage environment variables as your config is to store them in a *.env* file. This file acts like a dictionary of key/value pairs which exist outside of the program.

Pipenv has native support for loading these variables at runtime. SQLSorcery leverages these for config by default, but you can also set them manually in your environment or use an alternative library like python-dotenv if preferred.

4.1 Setup a *.env* file

SQLSorcery takes a convention over configuration approach for locating the correct environment variables. Environment variables are specified with a generic (*DB_*) or specific (*MS_*, *OR_*, etc.) prefix for each connection parameter.

SQLSorcery looks for connection params in the following order:

1. specified at object instantiation
2. using the specific dialect prefix
3. using the generic *DB_* prefix

4.1.1 Generic env vars

This is the simplest method for specifying connection variables and works best when you need to connect to a single SQL database.

```
DB_SERVER=
DB_PORT=
DB=
DB_SCHEMA=
DB_USER=
DB_PWD=
```

4.1.2 Dialect specific env vars

For a list of each dialect's required variables see each dialect listed below. This is also useful when doing database-to-database ETL work or when doing cross-database analysis in *pandas* because you can combine the params in your *.env* file for easy management.

Microsoft (MS SQL)

```
MS_SERVER=
MS_PORT=
MS_DB=
MS_SCHEMA=
MS_USER=
MS_PWD=
MS_DRIVER=(optional if not using a system with a single driver install)
```

MySQL

```
MY_SERVER=
MY_PORT=
MY_DB=
MY_USER=
MY_PWD=
```

Oracle (PL/SQL)

```
OR_SERVER=
OR_PORT=
OR_SCHEMA=
OR_SID=
OR_USER=
OR_PWD=
```

PostgreSQL

```
PG_SERVER=
PG_PORT=
PG_DB=
PG_SCHEMA=
PG_USER=
PG_PWD=
```

SQLite

SQLite only requires a filepath to connect. It is generally unnecessary to specify via an env var.

CHAPTER 5

Queries

SQLSorcery is designed to simplify data analysis and script based ETL. A common need in both is the ability to run queries against database tables or views.

5.1 Connect to the database

If using a *.env* file:

```
from sqlsorcery import PostgreSQL

sql = PostgreSQL()
```

If specifying at object instantiation:

```
from sqlsorcery import PostgreSQL

sql = PostgreSQL(server="ip or url", port="port number", db="database name", schema=
↪"schema name", user="username", pwd="password")
```

Warning: It is generally inadvisable to specify connection variables directly in your code.

5.2 Query from a string

Reads a database table into a pandas dataframe and prints to console:

```
from sqlsorcery import PostgreSQL

sql = PostgreSQL()
```

(continues on next page)

(continued from previous page)

```
df = sql.query("SELECT * FROM tablename")
print(df)
```

5.3 Query from a .sql file

If you had a *.sql* file with the following query named *user_location.sql*:

```
SELECT
      u.id
    , u.username
    , l.latitude
    , l.longitude
    , l.ip_address
    , u.is_staff
FROM users u
INNER JOIN location l
    ON u.id = l.user_id
WHERE u.is_staff = false
```

You could query with it like so:

```
from sqlsorcery import PostgreSQL

sql = PostgreSQL()
df = sql.query_from_file("user_location.sql")
print(df)
```

5.4 Query a view

If that previous *.sql* file was a view in the database called *vw_user_location* you could query it like so:

```
from sqlsorcery import PostgreSQL

sql = PostgreSQL()
df = sql.query("SELECT * FROM vw_user_location")
print(df)
```

CHAPTER 6

ETL

SQLSorcery is also useful for simple script based ETL actions.

Note: Keep in mind memory constraints when attempting bulk insertions. You can also improve performance by batching inserts using the `chunksize` param. A sane default is batches of *1000*.

6.1 Insert csv to table

Insert ~1 million IMDB ratings into a MySQL table.

```
from sqlsorcery import MySQL
import pandas as pd

sql = MySQL()
df = pd.read_csv("title.ratings.tsv", sep="\t")
sql.insert_into("ratings", df)
```

6.2 Copy table between databases

Copy the contents of a query in one database to another:

```
from sqlsorcery import MSSQL, PostgreSQL

ms = MSSQL()
pg = PostgreSQL()

df = pg.query("SELECT * FROM tablename")
ms.insert_into("new_table", df)
```

6.3 Query API endpoint and load into table

```
import requests
import pandas as pd
from sqlsorcery import SQLite

sql = SQLite(path="example.db")

response = requests.get("https://swapi.co/api/people/").json()
next_page = response["next"]

while next_page:
    response = requests.get(next_page).json()
    results = response["results"]
    next_page = response["next"]

    df = pd.DataFrame(results)
    df["film_appearances"] = len(df["films"])
    df = df[["name", "gender", "film_appearances"]]
    sql.insert_into("star_wars", df)
```

6.4 Update table values

It is often necessary to modify existing records in a table after loading. There are several ways to accomplish this in SQLSorcery depending on your use case including issuing raw commands or embedding within a stored procedure.

6.4.1 Via SQLAlchemy

```
import datetime
from sqlsorcery import MSSQL
import pandas as pd


sql = MSSQL()
df = pd.read_csv("daily_ratings.csv")
sql.insert_into("ratings_cache", df)
table = sql.table("ratings_cache")
# Adds today's date as the datestamp to all records
table.update().values(datestamp=datetime.date.today())
```

OR you could specify an additional `WHERE` clause

```
# If you wanted to override a specific rating
table.update().where(table.c.name=="Top Gun").values(avgRating="10")
```

6.4.2 Via pandas

With this scenario you would just modify the dataframe in memory before inserting into the database. This has trade-offs for performance as well as traceability.

```
import datetime
from sqlsorcery import MSSQL
import pandas as pd


sql = MSSQL()
df = pd.read_csv("daily_ratings.csv")
df["datestamp"] = datetime.date.today()
sql.insert_into("ratings_cache", df)
```

6.4.3 Via command

```
from sqlsorcery import MSSQL
import pandas as pd


sql = MSSQL()
df = pd.read_csv("daily_ratings.csv")
sql.insert_into("ratings_cache", df)
sql.exec_cmd("UPDATE ratings_cache SET datestamp = GETDATE()")
```

6.5 Truncate a table

It is often desirable to empty a table's contents before loading additional records during an ETL process. This is commonly used in conjuntion with a cache table which will be further transformed after the raw data is loaded into the database.

There are several ways to accomplish this in SQLSorcery depending on your use case.

6.5.1 Drop and replace during insert

```
from sqlsorcery import MSSQL
import pandas as pd

sql = MSSQL()
df = pd.read_csv("daily_ratings.csv")
sql.insert_into("ratings_cache", df, if_exists="replace")
```

6.5.2 Truncate all records

Most databases support `TRUNCATE TABLE` statements which differ from `DELETE FROM` statements in how logging and diskspace is handled. A truncate will also reset any identity column on the table.

```
from sqlsorcery import MSSQL
import pandas as pd

sql = MSSQL()
sql.truncate("ratings_cache")
df = pd.read_csv("daily_ratings.csv")
sql.insert_into("ratings_cache", df)
```

6.5.3 Delete all records

This will flush the table's contents, but will not reset the values in the identity column (such as an id or primary key). This is useful if you will want the insert to fail if the schema has changed.

```
from sqlsorcery import MSSQL
import pandas as pd

sql = MSSQL()
sql.delete("ratings_cache")
df = pd.read_csv("daily_ratings.csv")
sql.insert_into("ratings_cache", df)
```

6.5.4 Delete specific records

You might also find it necessary to only delete a subset of records. To do so you can drop down into *SQLAlchemy* to pass a `WHERE` clause.

```
import datetime
from sqlsorcery import MSSQL
import pandas as pd


sql = MSSQL()
table = sql.table("ratings_cache")
table.delete().where(table.c.datestamp == datetime.date.today())
df = pd.read_csv("daily_ratings.csv")
sql.insert_into("ratings_cache", df)
```

6.6 Execute a stored procedure

The following command will execute a stored procedure called *sproc_upsert_ratings* which merges data from a daily cache table of movie ratings into longitudinal table which stores all the daily results over time.

```
from sqlsorcery import MSSQL
import pandas as pd

sql = MSSQL()
df = pd.read_csv("daily_ratings.csv")
sql.insert_into("ratings_cache", df, if_exists="replace")
sql.exec_sproc("sproc_upsert_ratings")
```

The content of this stored procedure might look like:

```
IF OBJECT_ID('sproc_upsert_ratings') IS NULL
    EXEC('CREATE PROCEDURE sproc_upsert_ratings AS SET NOCOUNT ON;')
GO

ALTER PROCEDURE dbo.sproc_upsert_ratings AS
BEGIN
    SET NOCOUNT ON;

    MERGE dbo.factRatings AS target
```

(continues on next page)

(continued from previous page)

```
    USING dbo.ratings_cache AS source
    ON (target.id = source.id)
    WHEN MATCHED THEN
        UPDATE SET name = source.Name
            ,avgRating = source.avgRating
            ,numVotes = source.numVotes
    WHEN NOT MATCHED THEN
        INSERT (id, name, avgRating, numVotes)
        VALUES (source.id, source.name, source.avgRating, source.numVotes)
END;
```

Note: If your stored procedure does not return a result, you can/should pass the *autocommit=True* param. For more information on autocommit see *SQLAlchemy's documentation <https://docs.sqlalchemy.org/en/13/core/connections.html#understanding-autocommit>*

6.7 Execute any arbitrary command

Any valid SQL command can be passed raw to be executed. This is a catch all for things like function calls, create, or drop commands, etc.

6.7.1 Create a table from SQL command string

```
from sqlsorcery import MSSQL

sql = MSSQL()

table = """
    CREATE TABLE star_wars (
        name VARCHAR(100) NULL,
        gender VARCHAR(25) NULL,
        film_appearances INT NULL
    )
"""
sql.exec_cmd(table)
```

6.7.2 Create a table from a .sql file

Assuming you have a *.sql* file named *table.auth_user.sql*:

```
CREATE TABLE IF NOT EXISTS auth_user (
    id SERIAL NOT NULL CONSTRAINT auth_user_pkey PRIMARY KEY,
    password VARCHAR(128) NOT NULL,
    last_login TIMESTAMP WITH TIME ZONE,
    is_superuser BOOLEAN NOT NULL,
    username VARCHAR(150)NOT NULL CONSTRAINT auth_user_username_key UNIQUE,
    first_name VARCHAR(30) NOT NULL,
    last_name VARCHAR(150) NOT NULL,
    email VARCHAR(254) NOT NULL,
    is_staff BOOLEAN NOT NULL,
```

(continues on next page)

(continued from previous page)

```
    is_active BOOLEAN NOT NULL,
    date_joined TIMESTAMP WITH TIME ZONE NOT NULL
);

ALTER TABLE auth_user OWNER TO admin;

CREATE INDEX IF NOT EXISTS auth_user_username_idx ON auth_user (username);
```

You can execute it like so:

```
from sqlsorcery import MSSQL

sql = MSSQL()
sql.exec_cmd_from_file("table.auth_user.sql")
```

6.7.3 Drop a table from SQL command string

```
from sqlsorcery import MSSQL

sql = MSSQL()
sql.exec_cmd("DROP TABLE star_wars")
```

Note: Keep in mind this is merely an example of the types of commands that can be sent through raw. A cleaner way to drop a table is `sql.table('star_wars').drop()`.

CHAPTER 7

API

class sqlsorcery.**Connection**

Bases: object

Base class for sql connections containing shared class methods.

Note: This parent class is not meant to be called publicly and should only be used for inheritance in the specific connection types.

delete(*tablename*)

Deletes all records in a given table. Does not reset identity columns.

Parameters **tablename** (*string*) – Name of the table to delete records for

exec_cmd(*command*)

Executes an arbitrary sql command on the database.

Note: **Security Warning**: This command is vulnerable to SQL-injection. Do not use in conjunction with arbitrary user input.

Parameters **command** (*string*) – The SQL command to be executed

exec_cmd_from_file(*filename*)

Executes an arbitrary sql command provided from a .sql file.

Parameters **filename** (*string*) – Path to .sql file containing a query

Returns Resulting dataset from query

Return type Pandas.DataFrame

exec_sproc(*stored_procedure*, *autocommit=False*)

Executes a stored procedure

Note: **Security Warning**: This command leverages interpolated strings and as such is vulnerable to SQL-injection. Do not use in conjunction with arbitrary user input.

Parameters

- **stored_procedure** (*string*) – The name of the stored procedure to be executed.
- **autocommit** (*boolean*) – Determines how to handle transactions (default=False)

Returns Stored procedure results

Return type SQLAlchemy.ResultProxy

get_columns(*table*)
Returns the column definitions for a given table.

Parameters **table** (*string*) – The name of the table to inspect. Do not include the schema prefix.

Returns A list of column definition dictionaries

Return type list

get_view_definition(*view*)
Returns the view definition (DDL) for a given SQL view.

Parameters **view** (*string*) – The name fo the view to inspect. Do not include the schema prefix.

Returns Multi-line string of the view definition text

Return type string

insert_into(*table*, *df*, *if_exists='append'*, *chunksize=None*, *dtype=None*)
Inserts the data in a pandas dataframe into a specified sql table

Parameters

- **table** (*string*) – Name of sql table to insert data into
- **df** – DataFrame to be inserted
- **if_exists** (*string*) – How to behave if the table already exists. Possible options: *fail*, *append*, *replace*. Default = *append*
- **chunksize** (*int*) – Size of batch for inserts (default is all at once)
- **dtype** – Explicitly specify the data type for columns

Returns None

query(*sql_query*, *params=None*)
Executes the given sql query

Parameters

- **sql_query** (*string*) – SQL query string
- **params** (*list or dict*) – list or dict of parameters to pass to sql query

Note: See PEP249 for possible paramstyles.

Returns Resulting dataset from query

Return type

Pandas.DataFrame

query_from_file(*filename*)
Executes the given sql query from a provided sql file

Parameters **filename** (*string*) – Path to .sql file containing a query

Returns Resulting dataset from query

Return type

Pandas.DataFrame

table(*tablename*)
Returns a SQLAlchemy table object for further manipulation such as updates.

Parameters **tablename** (*string*) – Name of the table to return

Returns A table

Return type SQLAlchemy.Table

truncate(*tablename*)
Truncates a given table. Faster than a delete and reseeds identity values.

Note: **Security Warning**: This command leverages interpolated strings and as such is vulnerable to SQL-injection. Do not use in conjunction with arbitrary user input. Instead, use .delete()

Parameters **tablename** (*string*) – Name of the table to truncate

class sqlsorcery.**MSSQL**(*schema=None*, *port=None*, *server=None*, *db=None*, *user=None*, *pwd=None*, *driver=None*)
Bases: *sqlsorcery.Connection*

Child class that inherits from Connection with specific configuration for connecting to MS SQL.

Initializes an MS SQL database connection

Note: When object is instantiated without params, SQLSorcery will attempt to pull the values from the environment. See the README for examples of setting these correctly in a .env file.

Parameters

- **schema** (*string*) – Database object schema prefix
- **server** (*string*) – IP or URL of database server
- **db** (*string*) – Name of database
- **user** (*string*) – Username for connecting to the database
- **pwd** (*string*) – Password for connecting to the database. **Security Warning**: always pass this in with environment variables when used in production.
- **driver** (*string*) – Name of MS SQL driver installed in system

class sqlsorcery.**MySQL**(*server=None*, *port=None*, *db=None*, *user=None*, *pwd=None*)
Bases: *sqlsorcery.Connection*

Child class that inherits from Connection with specific configuration for connecting to a MySQL database.

Initializes a MySQL database connection

Note: When object is instantiated without params, SQLSorcery will attempt to pull the values from the environment. See the README for examples of setting these correctly in a .env file.

Parameters

- **server** (*string*) – IP or URL of database server
- **port** (*string*) – Port number
- **db** (*string*) – Name of database
- **user** (*string*) – Username for connecting to the database
- **pwd** (*string*) – Password for connecting to the database. **Security Warning**: always pass this in with environment variables when used in production.

class sqlsorcery.**Oracle**(*schema=None*, *server=None*, *port=None*, *sid=None*, *user=None*, *pwd=None*)
Bases: *sqlsorcery.Connection*

Child class that inherits from Connection with specific configuration for connecting to Oracle PL/SQL.

Initializes an Oracle database connection

Note: When object is instantiated without params, SQLSorcery will attempt to pull the values from the environment. See the README for examples of setting these correctly in a .env file.

Parameters

- **schema** (*string*) – Database object schema prefix
- **server** (*string*) – IP or URL of database server
- **port** (*string*) – Port number
- **sid** (*string*) – Database site identifier
- **user** (*string*) – Username for connecting to the database
- **pwd** (*string*) – Password for connecting to the database. **Security Warning**: always pass this in with environment variables when used in production.

class sqlsorcery.**PostgreSQL**(*schema=None*, *server=None*, *port=None*, *db=None*, *user=None*, *pwd=None*)
Bases: *sqlsorcery.Connection*

Child class that inherits from Connection with specific configuration for connecting to PostgreSQL.

Initializes a PostgreSQL database connection

Note: When object is instantiated without params, SQLSorcery will attempt to pull the values from the environment. See the README for examples of setting these correctly in a .env file.

Parameters

- **schema** (*string*) – Database object schema prefix
- **server** (*string*) – IP or URL of database server
- **port** (*string*) – Port number
- **db** (*string*) – Name of database
- **user** (*string*) – Username for connecting to the database
- **pwd** (*string*) – Password for connecting to the database. **Security Warning**: always pass this in with environment variables when used in production.

class sqlsorcery.**SQLite**(*path=None*)

Bases: *sqlsorcery.Connection*

Child class that inherits from Connection with specific configuration for connecting to a SQLite database file.

Initializes a SQLite database connection

Parameters **path** – Path to the .db file

CHAPTER 8

Index

Python Module Index

S

Index