agate-sql Documentation

Release 0.5.4 (beta)

Christopher Groskopf

May 21, 2018

Contents

agate-sql adds SQL read/write support to agate.

Important links:

- agate http://agate.rtfd.org
- Documentation: http://agate-sql.rtfd.org
- Repository: https://github.com/wireservice/agate-sql
- Issues: https://github.com/wireservice/agate-sql/issues

CHAPTER 1

Install

To install:

```
pip install agate-sql
```

For details on development or supported platforms see the agate documentation.

Warning: You'll need to have the correct sqlalchemy drivers installed for whatever database you plan to access. For instance, in order to read/write tables in a Postgres database, you'll also need to `pip install psycopg2`.

CHAPTER 2

Usage

agate-sql uses a monkey patching pattern to add SQL support to all `agate.Table` instances.

```
import agate
import agatesql
```

Importing `agatesql` attaches new methods to `agate.Table`. For example, to import a table named `doctors` from a local postgresql database named `hospitals` you will use `from_sql()`:

```
new_table = agate.Table.from_sql('postgresql:///hospitals', 'doctors')
```

To save this table back to the database:

```
new_table.to_sql('postgresql:///hospitals', 'doctors')
```

The first argument to either function can be any valid sqlalchemy connection string. The second argument must be a database name. (Arbitrary SQL queries are not supported.)

That's all there is to it.

CHAPTER 3

API

agatesql.table.**from_sql**(*cls*, *connection_or_string*, *table_name*)

Create a new agate.Table from a given SQL table. Types will be inferred from the database schema.

Monkey patched as class method `Table.from_sql()`.

Parameters

- **connection_or_string** – An existing sqlalchemy connection or connection string.
- **table_name** – The name of a table in the referenced database.

agatesql.table.**from_sql_query**(*self*, *query*)

Create an agate table from the results of a SQL query. Note that column data types will be inferred from the returned data, not the column types declared in SQL (if any). This is more flexible than *from_sql()* but could result in unexpected typing issues.

Parameters **query** – A SQL query to execute.

agatesql.table.**to_sql**(*self*, *connection_or_string*, *table_name*, *overwrite=False*, *create=True*, *create_if_not_exists=False*, *insert=True*, *prefixes=[]*, *db_schema=None*, *constraints=True*, *unique_constraint=[]*, *chunk_size=None*)

Write this table to the given SQL database.

Monkey patched as instance method `Table.to_sql()`.

Parameters

- **connection_or_string** – An existing sqlalchemy connection or a connection string.
- **table_name** – The name of the SQL table to create.
- **overwrite** – Drop any existing table with the same name before creating.
- **create** – Create the table.
- **create_if_not_exists** – When creating the table, don't fail if the table already exists.
- **insert** – Insert table data.
- **prefixes** – Add prefixes to the insert query.

- **db_schema** – Create table in the specified database schema.

:param constraints Generate constraints such as `nullable` for table columns.

:param unique_constraint The names of the columns to include in a UNIQUE constraint.

:param chunk_size Write rows in batches of this size. If not set, rows will be written at once.

`agatesql.table.`**`to_sql_create_statement`**(*self*, *table_name*, *dialect=None*, *db_schema=None*, *constraints=True*, *unique_constraint=[]*)

Generates a CREATE TABLE statement for this SQL table, but does not execute it.

Parameters

- **table_name** – The name of the SQL table to create.
- **dialect** – The dialect of SQL to use for the table statement.
- **db_schema** – Create table in the specified database schema.

:param constraints Generate constraints such as `nullable` for table columns.

:param unique_constraint The names of the columns to include in a UNIQUE constraint.

`agatesql.table.`**`sql_query`**(*self*, *query*, *table_name='agate'*)

Convert this agate table into an intermediate, in-memory sqlite table, run a query against it, and then return the results as a new agate table.

Multiple queries may be separated with semicolons.

Parameters

- **query** – One SQL query, or multiple queries to be run consecutively separated with semicolons.
- **table_name** – The name to use for the table in the queries, defaults to `agate`.

3.1 Authors

The following individuals have contributed code to agate-sql:

- Christopher Groskopf
- Adrian Klaver
- James McKinney
- Chris Keller
- git-clueless
- z2s8
- Jake Zimmerman
- Shige Takeda

3.2 Changelog

3.2.1 0.5.4

Dialect-specific:

- Add support for CrateDB.
- Eliminate SQLite warning about Decimal numbers.

3.2.2 0.5.3 - January 28, 2018

- Add `chunk_size` option to `Table.to_sql()` to write rows in batches.
- Add `unique_constraint` option to `Table.to_sql()` to include in a UNIQUE constraint.

Dialect-specific:

- Specify precision and scale for `DECIMAL` (MS SQL, MySQL, Oracle).
- Set length of `VARCHAR` to `1` even if maximum length is `0` (MySQL).
- Set type to `TEXT` if maximum length is greater than 21,844 (MySQL).

3.2.3 0.5.2 - April 28, 2017

- Add `create_if_not_exists` flag to `Table.to_sql()`.

3.2.4 0.5.1 - February 27, 2017

- Add `prefixes` option to *`to_sql()`* to add expressions following the INSERT keyword, like OR IGNORE or OR REPLACE.
- Use `TIMESTAMP` instead of `DATETIME` for DateTime columns.

3.2.5 0.5.0 - December 23, 2016

- `VARCHAR` columns are now generated with proper length constraints (unless explicilty disabled).
- Tables can now be created from query results using *`from_sql_query()`*.
- Add support for running queries directly on tables with *`sql_query()`*.
- When creating tables, `NOT NULL` constraints will be created by default.
- SQL create statements can now be generated without being executed with *`to_sql_create_statement()`*

3.2.6 0.4.0 - December 19, 2016

- Modified `example.py` so it no longer depends on Postgres.
- It is no longer necessary to run `agatesql.patch()` after importing agatesql.
- Upgrade required agate to `1.5.0`.

3.2.7 0.3.0 - November 5, 2015

- Add `overwrite` flag to `Table.to_sql()`.
- Removed Python 2.6 support.
- Updated agate dependency to version 1.1.0.
- Additional SQL types are now supported. (#4, #10)

3.2.8 0.2.0 - October 22, 2015

- Add explicit patch function.

3.2.9 0.1.0 - September 22, 2015

- Initial version.

3.3 License

3.4 Indices and tables

- genindex
- modindex
- search

Index