Spectrify Documentation

Release 1.0.1

The Narrativ Company, Inc.

Jul 12, 2018

Contents

Contents:

CHAPTER 1

Spectrify

A simple yet powerful tool to move your data from Redshift to Redshift Spectrum.

- Free software: MIT license
- Documentation: https://spectrify.readthedocs.io.

1.1 Features

One-liners to:

- Export a Redshift table to S3 (CSV)
- Convert exported CSVs to Parquet files in parallel
- Create the Spectrum table on your Redshift cluster
- **Perform all 3 steps in sequence**, essentially "copying" a Redshift table Spectrum in one command.

S3 credentials are specified using boto3. See http://boto3.readthedocs.io/en/latest/guide/configuration.html

Redshift credentials are supplied via environment variables, command-line parameters, or interactive prompt.

1.2 Install

```
$ pip install spectrify
```

1.3 Command-line Usage

Export Redshift table *my_table* to a folder of CSV files on S3:

Convert exported CSVs to Parquet:

Create Spectrum table from S3 folder:

Transform Redshift table by performing all 3 steps in sequence:

1.4 Python Usage

Export to S3:

```
from spectrify.export import RedshiftDataExporter
RedshiftDataExporter(sa_engine, s3_config).export_to_csv('my_table')
```

Convert exported CSVs to Parquet:

```
from spectrify.convert import ConcurrentManifestConverter
from spectrify.utils.schema import SqlAlchemySchemaReader
sa_table = SqlAlchemySchemaReader(engine).get_table_schema('my_table')
ConcurrentManifestConverter(sa_table, s3_config).convert_manifest()
```

Create Spectrum table from S3 parquet folder:

```
from spectrify.create import SpectrumTableCreator
from spectrify.utils.schema import SqlAlchemySchemaReader
sa_table = SqlAlchemySchemaReader(engine).get_table_schema('my_table')
SpectrumTableCreator(sa_engine, dest_schema, dest_table_name, sa_table, s3_config).
↪create()
```

Transform Redshift table by performing all 3 steps in sequence:

```
from spectrify.transform import TableTransformer
transformer = TableTransformer(engine, 'my_table', s3_config, dest_schema, dest_table_
↪name)
transformer.transform()
```

1.5 Contribute

Contributions always welcome! Read our guide on contributing here: http://spectrify.readthedocs.io/en/latest/contributing.html

1.6 License

CHAPTER 2

Installation

2.1 Stable release

To install Spectrify, run this command in your terminal:

```
$ pip install spectrify
```

This is the preferred method to install Spectrify, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Spectrify can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/hellonarrativ/spectrify
```

Or download the tarball:

```
$ curl  -OL https://github.com/hellonarrativ/spectrify/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

3.1 Installation

Assuming you have Python and pip installed, you can simply run the following:: pip install spectrify

3.2 Basic Usage

Spectrify can be used as a command-line tool to "copy" an entire table from Redshift to Redshift Spectrum.

> spectrify –host=example-url.redshift.aws.com –user=myuser –db=mydb transform my_table 's3://example-bucket/my_table'

This will perform the following:

- Unload the table to s3://example-bucket/my_table/csv/
- Properly encode the data into Parquet files in s3://example-bucket/my_table/spectrum/
- Create a table of the same name in the spectrum schema in your Redshift cluster

Note that the invocation above creates a single partition, and uses a max CSV file size of 1GB, which for our data translates into parquet files of around 800MB. By default, the Parquet files are compressed using gzip compression.

3.3 Customizing Spectrify

Spectrify can also be used as a code library to create your own Spectrum data pipeline. To give a real-world use case: say you would like to export your data weekly, with each week's data residing in a separate partition. You would need to modify Spectrify's default behavior in the following ways:

- In Redshift, unload only the records from the previous week
- In S3, store CSVs for each week into a separate folder

- In S3, store each week's Parquet files in a separate folder
- In Redshift Spectrum, Add a new partition instead of creating a new table

A full code listing for this example can be found in the repository.

3.3.1 Unloading a Subset of a Table

In Spectrify, the class `RedshiftDataExporter` responsible for unloading records. By modifying the SQL statement used to perform the unload, you can instruct Spectrify to export a subset of a table. In the scenario above, it might look like this:

```
from spectrify.export import RedshiftDataExporter

class WeeklyDataExporter(RedshiftDataExporter):
    """This class overrides the export query in the following ways:
       - Exports only records with timestamp_col between start_date and end_date
       - Features a smaller MAXFILESIZE (256MB)
    """
    UNLOAD_QUERY = """
    UNLOAD ($$
      SELECT *
      FROM {table_name}
      WHERE timestamp_col >= '{start_date}' AND timestamp_col < '{end_date}'
    $$)
    TO %(s3_path)s
    CREDENTIALS %(credentials)s
    ESCAPE MANIFEST GZIP ALLOWOVERWRITE
    MAXFILESIZE 256 MB;
    """

    def __init__(self, *args, **kwargs):
        self.start_date = kwargs.pop('start_date')
        self.end_date = kwargs.pop('end_date')
        super().__init__(*args, **kwargs)

    def get_query(self, table_name):
        return self.UNLOAD_QUERY.format(
            table_name=table_name,
            start_date=self.start_date,
            end_date=self.end_date,
        )
```

3.3.2 Customizing S3 Data Locations

In order to perform operations involving S3, Spectrify requires you to pass in a `S3Config` object. At its simplest, an S3Config simply points to directories where the CSV and Parquet/Spectrum files should be stored. You can specify these directories easily by creating a SimpleS3Config as shown below:

```
from spectrify.utils.s3 import SimpleS3Config

csv_path = 's3://my-temp-bucket/my-table'
spectrum_path = 's3://my-spectrum-bucket/my-table'
s3_config = SimpleS3Config(csv_path, spectrum_path)
```

3.3.3 Creating New Partitions

In Spectrify, the class `SpectrumTableCreator` is responsible for creating Redshift Spectrum external tables. If you already have a table created, you can modify the class to create a new partition instead. It might look like this:

```
class SpectrumPartitionCreator(SpectrumTableCreator):
    """Instead of issuing a CREATE TABLE statement, this subclass creates a
    new partition.
    """
    create_query = """
    ALTER TABLE {spectrum_schema}.{dest_table}
    ADD partition(partition_key='{start_date}')
    LOCATION '{partition_path}';
    """

    def __init__(self, *args, **kwargs):
        self.start_date = kwargs.pop('start_date')
        self.end_date = kwargs.pop('end_date')
        super().__init__(*args, **kwargs)

    def format_query(self):
        partition_path = self.s3_config.spectrum_dir
        return self.create_query.format(
            spectrum_schema=self.schema_name,
            dest_table=self.table_name,
            start_date=self.start_date,
            partition_path=partition_path,
        )
```

3.3.4 Incorporating Customizations

The classes above describe how to modify individual aspects of Spectrify. The following class demonstrates how you can bring these pieces together with the rest of Spectrify's functionality. The class `TableTransformer` encompasses all pieces of the conversion from Redshift to Redshift Spectrum. Here we override the export and table creation steps with our own partition strategy:

```
class WeeklyDataTransformer(TableTransformer):
    """The TableTransformer does 3 things:
      - Export Redshift data to CSV
      - Convert CSV to Parquet
      - Create a Spectrum table from Parquet files

    This subclass overrides the default behavior in the following ways:
      - Exports only last week of data (via WeeklyDataExporter)
      - Adds a partition instead of creating new table (via SpectrumPartitionCreator)
    """
    def __init__(self, *args, **kwargs):
        self.start_date = kwargs.pop('start_date')
        self.end_date = kwargs.pop('end_date')
        super().__init__(*args, **kwargs)

    def export_redshift_table(self):
        """Overrides the export behavior to only export the last week's data"""
        exporter = WeeklyDataExporter(
            self.engine,
            self.s3_config,
```

(continues on next page)

(continued from previous page)

```
            start_date=self.start_date,
            end_date=self.end_date
        )
        exporter.export_to_csv(self.table_name)

    def create_spectrum_table(self):
        """Overrides create behavior to add a new partition to an existing table"""
        creator = SpectrumPartitionCreator(
            self.engine,
            self.spectrum_schema,
            self.spectrum_name,
            self.sa_table,
            self.s3_config,
            start_date=self.start_date,
            end_date=self.end_date
        )
        creator.create()
```

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/hellonarrativ/spectrify/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" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

Spectrify could always use more documentation, whether as part of the official Spectrify 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/hellonarrativ/spectrify/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 *spectrify* for local development.

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

```
$ git clone git@github.com:your_name_here/spectrify.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 spectrify -p `which python3`  # or python2, if you prefer
$ cd spectrify/
$ pip install -e .
```

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 spectrify tests
$ python setup.py test or 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 to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4 and 3.5, and 3.6. Check https://travis-ci.org/hellonarrativ/spectrify/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_spectrify
```

CHAPTER 5

Credits

5.1 Development Lead

- Colin Nichols <engineering@narrativ.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 1.0.1 (2018-07-12)

- Loosen version requirement for PyArrow
- Add example script
- Update documentation

6.2 1.0.0 (2018-04-20)

- Move functionality into classes to make customizing behavior easier
- Add support for DATE columns
- Add support for DECIMAL/NUMERIC columns
- Upgrade to pyarrow v0.9.0

6.3 0.4.1 (2018-03-25)

- Fix exception when source table is not in schema public

6.4 0.4.0 (2018-02-25)

- Upgrade to pyarrow v0.8.0
- Verify Redshift column types are supported before attempting conversion
- Bugfix: Properly clean up multiprocessing.pool resource

6.5 0.3.0 (2017-10-30)

- Support 16- and 32-bit integers
- Packaging updates

6.6 0.2.1 (2017-09-27)

- Fix Readme

6.7 0.2.0 (2017-09-27)

- First release on PyPI.

6.8 0.1.0 (2017-09-13)

- Didn't even make it to PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search