sqlalchemy-redshift Documentation

Release 0.7.2.dev0

Matt George

Dec 11, 2018

Contents

Amazon Redshift dialect for SQLAlchemy.

CHAPTER 1

Installation

The package is available on PyPI:

```
pip install sqlalchemy-redshift
```

CHAPTER 2

Usage

The DSN format is similar to that of regular Postgres:

```
>>> import sqlalchemy as sa
>>> sa.create_engine('redshift+psycopg2://username@host.amazonaws.com:5439/database')
Engine(redshift+psycopg2://username@host.amazonaws.com:5439/database)
```

See the RedshiftDDLCompiler documentation for details on Redshift-specific features the dialect supports.

CHAPTER 3

Releasing

To perform a release, you will need to be an admin for the project on GitHub and on PyPI. Contact the maintainers if you need that access.

You will need to have a *~/.pypirc* with your PyPI credentials and also the following settings:

```
[zest.releaser]
create-wheels = yes
```

To perform a release, run the following:

```
python3.6 -m venv ~/.virtualenvs/dist
workon dist
pip install -U pip setuptools wheel
pip install -U tox zest.releaser
fullrelease  # follow prompts, use semver ish with versions.
```

The releaser will handle updating version data on the package and in CHANGES.rst along with tagging the repo and uploading to PyPI.

CHAPTER 4

0.7.2 (unreleased)

- Update tests to adapt to changes in Redshift and SQLAlchemy (Issue #140)
- Add *header* option to *UnloadFromSelect* command (Issue #156)
- Add support for Parquet and ORC file formats in the COPY command (Issue #151)
- Add official support for Python 3.7 (Issue #153)
- Avoid manipulating search path in table metadata fetch by using system tables directly (Issue #147)

CHAPTER 5

0.7.1 (2018-01-17)

- Fix incompatibility of reflection code with SQLAlchemy 1.2.0+ (Issue #138)

CHAPTER 6

0.7.0 (2017-10-03)

- Do not enumerate *search_path* with external schemas (Issue #120)
- Return constraint name from get_pk_constraint and get_foreign_keys
- Use Enums for Format, Compression and Encoding. Deprecate string parameters for these parameter types (Issue #133)
- Update included certificate with the transitional ACM cert bundle (Issue #130)

CHAPTER 7

0.6.0 (2017-05-04)

- Support role-based access control in COPY and UNLOAD commands (Issue #88)
- Increase max_identifier_length to 127 characters (Issue #96)
- Fix a bug where table names containing a period caused an error on reflection (Issue #97)
- Performance improvement for reflection by caching table constraint info (Issue #101)
- Support BZIP2 compression in COPY command (Issue #110)
- Allow tests to tolerate new default column encodings in Redshift (Issue #114)
- Pull in set of reserved words from Redshift docs (*Issue #94 <https://github.com/sqlalchemy-redshift/sqlalchemy-redshift/issues/94> _*)

CHAPTER 8

0.5.0 (2016-04-21)

- Support reflecting tables with foriegn keys to tables in non-public schemas (Issue #70)
- Fix a bug where DISTKEY and SORTKEY could not be used on column names containing spaces or commas. This is a breaking behavioral change for a command like *__table_args__ = {'redshift_sortkey': ('foo, bar')}*. Previously, this would sort on the columns named *foo* and *bar*. Now, it sorts on the column named *foo, bar*. (Issue #74)

CHAPTER 9

0.4.0 (2015-11-17)

- Change the name of the package to *sqlalchemy_redshift* to match the naming convention for other dialects; the *redshift_sqlalchemy* package now emits a *DeprecationWarning* and references *sqlalchemy_redshift*. The *redshift_sqlalchemy* compatibility package will be removed in a future release. (Issue #58)
- Fix a bug where reflected tables could have incorrect column order for some *CREATE TABLE* statements, particularly for columns with an *IDENTITY* constraint. (Issue #60)
- Fix a bug where reflecting a table could raise a `NoSuchTableError` in cases where its schema is not on the current `search_path` (Issue #64)
- Add python 3.5 to the list of versions for integration tests. (Issue #61)

CHAPTER 10

0.3.1 (2015-10-08)

- Fix breakages to CopyCommand introduced in 0.3.0: Thanks solackerman. (Issue #53)
  - When *format* is omitted, no *FORMAT AS ...* is appended to the query. This makes the default the same as a normal redshift query.
  - fix STATUPDATE as a COPY parameter

CHAPTER 11

0.3.0 (2015-09-29)

- Fix view support to be more in line with SQLAlchemy standards. *get_view_definition* output no longer includes a trailing semicolon and views no longer raise an exception when reflected as *Table* objects. (Issue #46)
- Rename RedShiftDDLCompiler to RedshiftDDLCompiler. (Issue #43)
- Update commands (Issue #52)
  - Expose optional TRUNCATECOLUMNS in CopyCommand.
  - Add all other COPY parameters to CopyCommand.
  - Move commands to their own module.
  - Support inserts into ordered columns in CopyCommand.

CHAPTER 12

0.2.0 (2015-09-04)

- Use SYSDATE instead of NOW(). Thanks bouk. (Issue #15)
- Default to SSL with hardcoded AWS Redshift CA. (Issue #20)
- Refactor of CopyCommand including support for specifying format and compression type. (Issue #21)
- Explicitly require SQLAlchemy >= 0.9.2 for 'dialect_options'. (Issue #13)
- Refactor of UnloadFromSelect including support for specifying all documented redshift options. (Issue #27)
- Fix unicode issue with SORTKEY on python 2. (Issue #34)
- Add support for Redshift `DELETE` statements that refer other tables in the `WHERE` clause. Thanks haleemur. (Issue #35)
- Raise `NoSuchTableError` when trying to reflect a table that doesn't exist. (Issue #38)

CHAPTER 13

0.1.2 (2015-08-11)

- Register postgresql.visit_rename_table for redshift's alembic RenameTable. Thanks bouk. (Issue #7)

CHAPTER 14

0.1.1 (2015-05-20)

- Register RedshiftImpl as an alembic 3rd party dialect.

CHAPTER 15

0.1.0 (2015-05-11)

- First version of sqlalchemy-redshift that can be installed from PyPI

Contents:

DDL Compiler

class sqlalchemy_redshift.dialect.**RedshiftDDLCompiler**(*dialect*, *statement*, *bind=None*, *schema_translate_map=None*, *compile_kwargs=immutabledict({})*)

Handles Redshift-specific `CREATE TABLE` syntax.

Users can specify the *diststyle*, *distkey*, *sortkey* and *encode* properties per table and per column.

Table level properties can be set using the dialect specific syntax. For example, to specify a distribution key and style you apply the following:

```
>>> import sqlalchemy as sa
>>> from sqlalchemy.schema import CreateTable
>>> engine = sa.create_engine('redshift+psycopg2://example')
>>> metadata = sa.MetaData()
>>> user = sa.Table(
...     'user',
...     metadata,
...     sa.Column('id', sa.Integer, primary_key=True),
...     sa.Column('name', sa.String),
...     redshift_diststyle='KEY',
...     redshift_distkey='id',
...     redshift_interleaved_sortkey=['id', 'name'],
... )
>>> print(CreateTable(user).compile(engine))

CREATE TABLE "user" (
    id INTEGER NOT NULL,
```

```
    name VARCHAR,
    PRIMARY KEY (id)
) DISTSTYLE KEY DISTKEY (id) INTERLEAVED SORTKEY (id, name)
```

A single sort key can be applied without a wrapping list:

```
>>> customer = sa.Table(
...     'customer',
...     metadata,
...     sa.Column('id', sa.Integer, primary_key=True),
...     sa.Column('name', sa.String),
...     redshift_sortkey='id',
... )
>>> print(CreateTable(customer).compile(engine))

CREATE TABLE customer (
    id INTEGER NOT NULL,
    name VARCHAR,
    PRIMARY KEY (id)
) SORTKEY (id)
```

Column-level special syntax can also be applied using the column info dictionary. For example, we can specify the ENCODE for a column:

```
>>> product = sa.Table(
...     'product',
...     metadata,
...     sa.Column('id', sa.Integer, primary_key=True),
...     sa.Column('name', sa.String, info={'encode': 'lzo'})
... )
>>> print(CreateTable(product).compile(engine))

CREATE TABLE product (
    id INTEGER NOT NULL,
    name VARCHAR ENCODE lzo,
    PRIMARY KEY (id)
)
```

We can also specify the distkey and sortkey options:

```
>>> sku = sa.Table(
...     'sku',
...     metadata,
...     sa.Column('id', sa.Integer, primary_key=True),
...     sa.Column(
...         'name', sa.String, info={'distkey': True, 'sortkey': True}
...     )
... )
>>> print(CreateTable(sku).compile(engine))

CREATE TABLE sku (
    id INTEGER NOT NULL,
    name VARCHAR DISTKEY SORTKEY,
    PRIMARY KEY (id)
)
```

Dialect

class sqlalchemy_redshift.dialect.**RedshiftDialect**(**args*, ***kw*)

Define Redshift-specific behavior.

Most public methods are overrides of the underlying interfaces defined in `Dialect` and `Inspector`.

create_connect_args(**args*, ***kwargs*)

Build DB-API compatible connection arguments.

Overrides interface `create_connect_args()`.

ddl_compiler

alias of `RedshiftDDLCompiler`

get_columns(*connection*, *table_name*, *schema=None*, ***kw*)

Return information about columns in *table_name*.

Overrides interface `get_columns()`.

get_foreign_keys(*connection*, *table_name*, *schema=None*, ***kw*)

Return information about foreign keys in *table_name*.

Overrides interface `get_pk_constraint()`.

get_indexes(*connection*, *table_name*, *schema*, ***kw*)

Return information about indexes in *table_name*.

Because Redshift does not support traditional indexes, this always returns an empty list.

Overrides interface `get_indexes()`.

get_pk_constraint(*connection*, *table_name*, *schema=None*, ***kw*)

Return information about the primary key constraint on *table_name*.

Overrides interface `get_pk_constraint()`.

get_table_names(*connection*, *schema=None*, ***kw*)

Return a list of table names for *schema*.

Overrides interface `get_table_names()`.

get_table_options(*connection*, *table_name*, *schema*, ***kw*)

Return a dictionary of options specified when the table of the given name was created.

Overrides interface `get_table_options()`.

get_unique_constraints(*connection*, *table_name*, *schema=None*, ***kw*)

Return information about unique constraints in *table_name*.

Overrides interface `get_unique_constraints()`.

get_view_definition(*connection*, *view_name*, *schema=None*, ***kw*)

Return view definition. Given a `Connection`, a string *view_name*, and an optional string *schema*, return the view definition.

Overrides interface `get_view_definition()`.

get_view_names(*connection*, *schema=None*, ***kw*)
Return a list of view names for *schema*.

Overrides interface `get_view_names()`.

Commands

class `sqlalchemy_redshift.commands.`**Compression**
An enumeration.

class `sqlalchemy_redshift.commands.`**CopyCommand**(*to*, *data_location*, *access_key_id=None*, *secret_access_key=None*, *session_token=None*, *aws_account_id=None*, *iam_role_name=None*, *format=None*, *quote=None*, *path_file='auto'*, *delimiter=None*, *fixed_width=None*, *compression=None*, *accept_any_date=False*, *accept_inv_chars=None*, *blanks_as_null=False*, *date_format=None*, *empty_as_null=False*, *encoding=None*, *escape=False*, *explicit_ids=False*, *fill_record=False*, *ignore_blank_lines=False*, *ignore_header=None*, *dangerous_null_delimiter=None*, *remove_quotes=False*, *roundec=False*, *time_format=None*, *trim_blanks=False*, *truncate_columns=False*, *comp_rows=None*, *comp_update=None*, *max_error=None*, *no_load=False*, *stat_update=None*, *manifest=False*)

Prepares a Redshift COPY statement.

Parameters **to** : sqlalchemy.Table or iterable of sqlalchemy.ColumnElement

The table or columns to copy data into

data_location : str

The Amazon S3 location from where to copy, or a manifest file if the *manifest* option is used

access_key_id: str, optional

Access Key. Required unless you supply role-based credentials (`aws_account_id` and `iam_role_name`)

secret_access_key: str, optional

Secret Access Key ID. Required unless you supply role-based credentials (`aws_account_id` and `iam_role_name`)

session_token : str, optional

aws_account_id: str, optional

AWS account ID for role-based credentials. Required unless you supply key based credentials (`access_key_id` and `secret_access_key`)

iam_role_name: str, optional

IAM role name for role-based credentials. Required unless you supply key based credentials (`access_key_id` and `secret_access_key`)

format : Format, optional

Indicates the type of file to copy from

quote : str, optional

Specifies the character to be used as the quote character when using `format=Format.csv`. The default is a double quotation mark (`"`)

delimiter : Field delimiter, optional

defaults to `|`

path_file : str, optional

Specifies an Amazon S3 location to a JSONPaths file to explicitly map Avro or JSON data elements to columns. defaults to `'auto'`

fixed_width: iterable of (str, int), optional

List of (column name, length) pairs to control fixed-width output.

compression : Compression, optional

indicates the type of compression of the file to copy

accept_any_date : bool, optional

Allows any date format, including invalid formats such as `00/00/00 00:00:00`, to be loaded as NULL without generating an error defaults to False

accept_inv_chars : str, optional

Enables loading of data into VARCHAR columns even if the data contains invalid UTF-8 characters. When specified each invalid UTF-8 byte is replaced by the specified replacement character

blanks_as_null : bool, optional

Boolean value denoting whether to load VARCHAR fields with whitespace only values as NULL instead of whitespace

date_format : str, optional

Specified the date format. If you want Amazon Redshift to automatically recognize and convert the date format in your source data, specify `'auto'`

empty_as_null : bool, optional

Boolean value denoting whether to load VARCHAR fields with empty values as NULL instead of empty string

encoding : Encoding, optional

Specifies the encoding type of the load data defaults to `Encoding.utf8`

escape : bool, optional

When this parameter is specified, the backslash character (`\`) in input data is treated as an escape character. The character that immediately follows the backslash character is loaded into the table as part of the current column value, even if it is a character that normally serves a special purpose

explicit_ids : bool, optional

Override the autogenerated IDENTITY column values with explicit values from the source data files for the tables

fill_record : bool, optional

Allows data files to be loaded when contiguous columns are missing at the end of some of the records. The missing columns are filled with either zero-length strings or NULLs, as appropriate for the data types of the columns in question.

ignore_blank_lines : bool, optional

Ignores blank lines that only contain a line feed in a data file and does not try to load them

ignore_header : int, optional

Integer value of number of lines to skip at the start of each file

dangerous_null_delimiter : str, optional

Optional string value denoting what to interpret as a NULL value from the file. Note that this parameter *is not properly quoted* due to a difference between redshift's and postgres's COPY commands interpretation of strings. For example, null bytes must be passed to redshift's `NULL` verbatim as `'\0'` whereas postgres's `NULL` accepts `'\x00'`.

remove_quotes : bool, optional

Removes surrounding quotation marks from strings in the incoming data. All characters within the quotation marks, including delimiters, are retained.

roundec : bool, optional

Rounds up numeric values when the scale of the input value is greater than the scale of the column

time_format : str, optional

Specified the date format. If you want Amazon Redshift to automatically recognize and convert the time format in your source data, specify `'auto'`

trim_blanks : bool, optional

Removes the trailing white space characters from a VARCHAR string

truncate_columns : bool, optional

Truncates data in columns to the appropriate number of characters so that it fits the column specification

comp_rows : int, optional

Specifies the number of rows to be used as the sample size for compression analysis

comp_update : bool, optional

Controls whether compression encodings are automatically applied. If omitted or None, COPY applies automatic compression only if the target table is empty and all the table columns either have RAW encoding or no encoding. If True COPY applies automatic compression if the table is empty, even if the table columns already have encodings other than RAW. If False automatic compression is disabled

max_error : int, optional

If the load returns the `max_error` number of errors or greater, the load fails defaults to 100000

no_load : bool, optional

Checks the validity of the data file without actually loading the data

stat_update : bool, optional

Update statistics automatically regardless of whether the table is initially empty

manifest : bool, optional

Boolean value denoting whether data_location is a manifest file.

class `sqlalchemy_redshift.commands.`**`Encoding`**
An enumeration.

class `sqlalchemy_redshift.commands.`**`Format`**
An enumeration.

class `sqlalchemy_redshift.commands.`**`UnloadFromSelect`**(*select*, *unload_location*, *access_key_id=None*, *secret_access_key=None*, *session_token=None*, *aws_account_id=None*, *iam_role_name=None*, *manifest=False*, *delimiter=None*, *fixed_width=None*, *encrypted=False*, *gzip=False*, *add_quotes=False*, *null=None*, *escape=False*, *allow_overwrite=False*, *parallel=True*, *header=False*)

Prepares a Redshift unload statement to drop a query to Amazon S3 https://docs.aws.amazon.com/redshift/latest/dg/r_UNLOAD_command_examples.html

Parameters **select: sqlalchemy.sql.selectable.Selectable**

The selectable Core Table Expression query to unload from.

unload_location: str

The Amazon S3 location where the file will be created, or a manifest file if the *manifest* option is used

access_key_id: str, optional

Access Key. Required unless you supply role-based credentials (`aws_account_id` and `iam_role_name`)

secret_access_key: str, optional

Secret Access Key ID. Required unless you supply role-based credentials (`aws_account_id` and `iam_role_name`)

session_token : str, optional

aws_account_id: str, optional

AWS account ID for role-based credentials. Required unless you supply key based credentials (`access_key_id` and `secret_access_key`)

iam_role_name: str, optional

IAM role name for role-based credentials. Required unless you supply key based credentials (`access_key_id` and `secret_access_key`)

manifest: bool, optional

Boolean value denoting whether data_location is a manifest file.

delimiter: File delimiter, optional

defaults to '|'

fixed_width: iterable of (str, int), optional

List of (column name, length) pairs to control fixed-width output.

encrypted: bool, optional

Write to encrypted S3 key.

gzip: bool, optional

Create file using GZIP compression.

add_quotes: bool, optional

Quote fields so that fields containing the delimiter can be distinguished.

null: str, optional

Write null values as the given string. Defaults to ''.

escape: bool, optional

For CHAR and VARCHAR columns in delimited unload files, an escape character (`\`) is placed before every occurrence of the following characters: `\r`, `\n`, `\`, the specified delimiter string. If *add_quotes* is specified, `"` and `'` are also escaped.

allow_overwrite: bool, optional

Overwrite the key at unload_location in the S3 bucket.

parallel: bool, optional

If disabled unload sequentially as one file.

header: bool, optional

Boolean value denoting whether to add header line containing column names at the top of each output file. Text transformation options, such as delimiter, add_quotes, and escape, also apply to the header line. *header* can't be used with fixed_width.

`sqlalchemy_redshift.commands.`**`visit_copy_command`**(*element*, *compiler*, ***kw*)

Returns the actual sql query for the CopyCommand class.

`sqlalchemy_redshift.commands.`**`visit_unload_from_select`**(*element*, *compiler*, ***kw*)

Returns the actual sql query for the UnloadFromSelect class.

CHAPTER 16

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index