<table>
<thead>
<tr>
<th></th>
<th>Installation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Usage</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Running Tests</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Continuous Integration (CI)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Releasing</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.8.15 (unreleased)</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>0.8.14 (2023-04-07)</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>0.8.13 (2023-03-28)</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>0.8.12 (2022-12-08)</td>
<td>19</td>
</tr>
<tr>
<td>10</td>
<td>0.8.11 (2022-07-27)</td>
<td>21</td>
</tr>
<tr>
<td>11</td>
<td>0.8.10 (2022-07-21)</td>
<td>23</td>
</tr>
<tr>
<td>12</td>
<td>0.8.9 (2021-12-15)</td>
<td>25</td>
</tr>
<tr>
<td>13</td>
<td>0.8.8 (2021-11-03)</td>
<td>27</td>
</tr>
<tr>
<td>14</td>
<td>0.8.7 (2021-10-27)</td>
<td>29</td>
</tr>
<tr>
<td>15</td>
<td>0.8.6 (2021-09-22)</td>
<td>31</td>
</tr>
<tr>
<td>16</td>
<td>0.8.5 (2021-08-23)</td>
<td>33</td>
</tr>
<tr>
<td>17</td>
<td>0.8.4 (2021-07-15)</td>
<td>35</td>
</tr>
<tr>
<td>18</td>
<td>0.8.3 (2021-07-07)</td>
<td>37</td>
</tr>
<tr>
<td>19</td>
<td>0.8.2 (2021-01-08)</td>
<td>39</td>
</tr>
<tr>
<td>20</td>
<td>0.8.1 (2020-07-15)</td>
<td>41</td>
</tr>
</tbody>
</table>

- **Contents**

  1. Installation
  2. Usage
  3. Running Tests
  4. Continuous Integration (CI)
  5. Releasing
  6. 0.8.15 (unreleased)
  7. 0.8.14 (2023-04-07)
  8. 0.8.13 (2023-03-28)
  9. 0.8.12 (2022-12-08)
 10. 0.8.11 (2022-07-27)
11. 0.8.10 (2022-07-21)
12. 0.8.9 (2021-12-15)
13. 0.8.8 (2021-11-03)
14. 0.8.7 (2021-10-27)
15. 0.8.6 (2021-09-22)
16. 0.8.5 (2021-08-23)
17. 0.8.4 (2021-07-15)
18. 0.8.3 (2021-07-07)
19. 0.8.2 (2021-01-08)
20. 0.8.1 (2020-07-15)
Amazon Redshift dialect for SQLAlchemy.
CHAPTER 1

Installation

The package is available on PyPI:

```
pip install sqlalchemy-redshift
```

WARNING: This dialect requires either redshift_connector or psycopg2 to work properly. It does not provide it as required, but relies on you to select the distribution you need:

- psycopg2 - standard distribution of psycopg2, requires compilation so few system dependencies are required for it
- psycopg2-binary - already compiled distribution (no system dependencies are required)
- psycopg2cffi - pypy compatible version

See Psycopg2’s binary install docs for more context on choosing a distribution.
The DSN format is similar to that of regular Postgres:

```python
>>> 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.
Tests are ran via tox and can be run with the following command:

```
$ tox
```

However, this will not run integration tests unless the following environment variables are set:

- REDSHIFT_HOST
- REDSHIFT_PORT
- REDSHIFT_USERNAME
- PGPASSWORD (this is the redshift instance password)
- REDSHIFT_DATABASE
- REDSHIFT_IAM_ROLE_ARN

Note that the IAM role specified will need to be associated with redshift cluster and have the correct permissions to create databases and tables as well drop them. Exporting these environment variables in your shell and running tox will run the integration tests against a real redshift instance. Practice caution when running these tests against a production instance.
Continuous Integration (CI)

Project CI is built using AWS CodePipeline and CloudFormation. Please see the ci/ folder and included README.txt for details on how to spin up the project’s CI.
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:

```
python -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.
0.8.15 (unreleased)

- Nothing changed yet.
• Override new upstream postgres method that fails against redshift (Pull #266)
• Fix table reflection broken for non-superusers (Pull #276)
• Fix Broken Reflection for 1.4 FutureEngine (Pull #277)
• Add spectrum support (Pull #263)
• Drop support for Python 3.5
• Fix SQLAlchemy’s "supports_statement_cache" (Pull #259)
• Disable redshift_connector dialect statement cache (Pull #257)
• Support HLLSKETCH Redshift datatypes (Pull #246)
• Disable supports_statement_cache (Pull #249)
• Fix doc, lint CI dependency issues (Pull #250)
• Fix redshift_connector dialect column encoding (Pull #255)
• Support inspection of Redshift datatypes (Pull #242)
0.8.8 (2021-11-03)

- Remove support for Python 2.7; now requires python $\geq 3.4$ (Pull #234)
- Support GEOMETRY, SUPER Redshift datatypes (Pull #235)
• Initial SQLAlchemy 2.0.x support (Pull #237)
• Add RedshiftDialect_redshift_connector (Pull #232)
• Create RedshiftDialectMixin class. Add RedshiftDialect_psycopg2 ffi. (Pull #231)
• Support TIMETZ datatype (Pull #229)
• Fix RelationKey unquoted issue (Pull #228)
0.8.4 (2021-07-15)

- Improve reflection performance by fetching/caching metadata per schema rather than for the entire database (Pull #223)
• SQLAlchemy 1.4.x support (Pull #221)
• Allow supplying multiple role ARNs in COPY and UNLOAD commands. This allows the first role to assume other roles as explained here.
• Support AWS partitions for role-based access control in COPY and UNLOAD commands. This allows these commands to be used, e.g. in GovCloud.
• Add option to drop materialized view with CASCADE (Pull #204)
• Fix invalid SQLAlchemy version comparison (Pull #206)
• Fix for supporting SQLAlchemy 1.3.11+ (Issue #195)
• Added support for materialized views (Issue #202)
• Fix reflection of unique constraints (Issue #199)
• Support for altering column comments in Alembic migrations (Issue #191)
• Import Iterable from collections.abc for Python 3.9 compatibility (Issue #189)
• Add support for Parquet format in UNLOAD command (Issue #187)
0.7.6 (2020-01-17)

- Fix unhashable type error for sortkey reflection in SQLAlchemy >= 1.3.11 (Issue #180)
- Expose supported types for import from the dialect (Issue #181)
- Reflect column comments (Issue #186)
• Extend psycopg2 package version check to also support psycopg2-binary and psycopg2cffi (Issue #178)
• Drop hard dependency on psycopg2 but require package to be present on runtime (Issue #165)
• Switch from info to keyword arguments on columns for SQLAlchemy >= 1.3.0 (Issue #161)
• Add support for column info on redshift late binding views (Issue #159)
• Add support for MAXFILESIZE argument to UNLOAD. (Issue #123)
• Add support for the CREATE LIBRARY command. (Issue #124)
• Add support for the ALTER TABLE APPEND command. (Issue #162)
• Add support for the CSV format to UnloadFromSelect. (Issue #169)
• Update the list of reserved words (adds “az64” and “language”) (Issue #176)
• Add support for REGION argument to COPY and UNLOAD commands. (Issue #90)
• 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)
• Fix incompatibility of reflection code with SQLAlchemy 1.2.0+ (Issue #138)
• 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)
• 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> _)
• Support reflecting tables with foreign 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)
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)
• 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
• 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.
• 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)
• Register postgresql.visit_rename_table for redshift’s alembic RenameTable. Thanks bouk. (Issue #7)
• Register RedshiftImpl as an alembic 3rd party dialect.
First version of SQLAlchemy-redshift that can be installed from PyPI

Contents:

40.1 DDL Compiler

```python
class sqlalchemy_redshift.dialect.RedshiftDDLCompiler(dialect, statement, schema_translate_map=None, render_schema_translate=False, 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:

```python
>>> import sqlalchemy as sa
>>> from sqlalchemy.schema import CreateTable
>>> engine = sa.create_engine('redshift+psycopg2://example')
>>> metadata = sa.MetaData()
>>> user = sa.Table('...', metadata,
...     'id', sa.Integer, primary_key=True),
...     '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,  
nname VARCHAR,  
PRIMARY KEY (id)  )  
) DISTSTYLE KEY DISTKEY (id) INTERLEAVED SORTKEY (id, name)

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

```python
>>> 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,  
nname VARCHAR,  
PRIMARY KEY (id)  )  
) SORTKEY (id)
```

Column-level special syntax can also be applied using Redshift dialect specific keyword arguments. For example, we can specify the ENCODE for a column:

```python
>>> product = sa.Table(  
...     'product',  
...     metadata,  
...     sa.Column('id', sa.Integer, primary_key=True),  
...     sa.Column('name', sa.String, redshift_encode='lzo')  
... )  
>>> print(CreateTable(product).compile(engine))
CREATE TABLE product (  
id INTEGER NOT NULL,  
nname VARCHAR ENCODE lzo,  
PRIMARY KEY (id)  )
```

The TIMESTAMPTZ and TIMETZ column types are also supported in the DDL.

For SQLAlchemy versions < 1.3.0, passing Redshift dialect options as keyword arguments is not supported on the column level. Instead, a column info dictionary can be used:

```python
>>> product_pre_1_3_0 = sa.Table(  
...     'product_pre_1_3_0',  
...     metadata,  
...     sa.Column('id', sa.Integer, primary_key=True),  
...     sa.Column('name', sa.String, info={'encode': 'lzo'})  
... )  
```

We can also specify the distkey and sortkey options:
```python
>>> sku = sa.Table('sku', 
...     metadata, 
...     sa.Column('id', sa.Integer, primary_key=True), 
...     sa.Column('name', sa.String, 
...         redshift_distkey=True, 
...         redshift_sortkey=True)
...)
>>> print(CreateTable(sku).compile(engine))
CREATE TABLE sku (  
id INTEGER NOT NULL,  
    name VARCHAR DISTKEY SORTKEY,  
    PRIMARY KEY (id)
)
```

## 40.2 Dialect

`sqlalchemy_redshift.dialect.RedshiftDialect` alias of `RedshiftDialect_psycopg2`

## 40.3 Commands

```python
class sqlalchemy_redshift.commands.AlterTableAppendCommand(source, target, ignore_extra=False, fill_target=False)
```

Prepares an `ALTER TABLE APPEND` statement to efficiently move data from one table to another, much faster than an INSERT INTO ... SELECT.

CAUTION: This moves the underlying storage blocks from the source table to the target table, so the source table will be empty after this command finishes.

See the documentation for additional restrictions and other information: https://docs.aws.amazon.com/redshift/latest/dg/r_ALTER_TABLE_APPEND.html

**Parameters**

- **source**: `sqlalchemy.Table`
  The table to move data from. Must be an existing permanent table.

- **target**: `sqlalchemy.Table`
  The table to move data into. Must be an existing permanent table.

- **ignore_extra**: `bool`, optional
  If the source table includes columns not present in the target table, discard those columns. Mutually exclusive with `fill_target`.

- **fill_target**: `bool`, optional
  If the target table includes columns not present in the source table, fill those columns with the default column value or NULL. Mutually exclusive with `ignore_extra`.

40.2. Dialect
class sqlalchemy.redshift.commands.Compression
An enumeration.

class sqlalchemy.redshift.commands.CopyCommand
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 or iam_role_arns)

secret_access_key: str, optional

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

session_token : str, optional

iam_role_arns : str or list of strings, optional

Either a single arn or a list of arns of roles to assume when unloading Required unless you supply key based credentials (access_key_id and secret_access_key) or (aws_account_id and iam_role_name) separately.

aws_partition: str, optional

AWS partition to use with role-based credentials. Defaults to 'aws'. Not applicable when using key based credentials (access_key_id and secret_access_key) or role arns (iam_role_arns) directly.
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)
or role arns (iam_role_arns) directly.

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) or role arns
(iam_role_arns) directly.

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 re-
placement 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.

**region**: str, optional

The AWS region where the target S3 bucket is located, if the Redshift cluster isn’t in the same region as the S3 bucket.

```python
class sqlalchemy_redshift.commands.CreateLibraryCommand(library_name, location, access_key_id=None, secret_access_key=None, session_token=None, aws_account_id=None, iam_role_name=None, replace=False, region=None, iam_role_arns=None)
```

Prepares a Redshift CREATE LIBRARY statement. https://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_LIBRARY.html

**Parameters**

**library_name**: str, required

The name of the library to install.

**location**: str, required

The location of the library file. Must be either a HTTP/HTTPS URL or an S3 location.

**access_key_id**: str, optional

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

**secret_access_key**: str, optional

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

**session_token**: str, optional

**iam_role_arns**: str or list of strings, optional

Either a single arn or a list of arns of roles to assume when unloading Required unless you supply key based credentials (access_key_id and secret_access_key) or (aws_account_id and iam_role_name) separately.
aws_partition: str, optional
AWS partition to use with role-based credentials. Defaults to 'aws'. Not applicable when using key based credentials (access_key_id and secret_access_key) or role arns (iam_role_arns) directly.

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) or role arns (iam_role_arns) directly.

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) or role arns (iam_role_arns) directly.

replace: bool, optional, default False
Controls the presence of OR REPLACE in the compiled statement. See the command documentation for details.

region: str, optional
The AWS region where the library’s S3 bucket is located, if the Redshift cluster isn’t in the same region as the S3 bucket.

class sqlalchemy_redshift.commands.Encoding
An enumeration.

class sqlalchemy_redshift.commands.Format
An enumeration.

class sqlalchemy_redshift.commands.RefreshMaterializedView(name)
Prepares a Redshift REFRESH MATERIALIZED VIEW statement. SEE: docs.aws.amazon.com/redshift/latest/dg/materialized-view-refresh-sql-command

This reruns the query underlying the view to ensure the materialized data is up to date.

```python
>>> import sqlalchemy as sa
>>> from sqlalchemy_redshift.dialect import RefreshMaterializedView
>>> engine = sa.create_engine('redshift+psycopg2://example')
>>> refresh = RefreshMaterializedView('materialized_view_of_users')
>>> print(refresh.compile({engine}))
REFRESH MATERIALIZED VIEW materialized_view_of_users
```

This can be included in any execute() statement.
class sqlalchemy_redshift.commands.UnloadFromSelect

Prepares a Redshift unload statement to drop a query to Amazon S3

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` or `iam_role_arns`).

- **secret_access_key**: str, optional
  - Secret Access Key ID. Required unless you supply role-based credentials (`aws_account_id` and `iam_role_name` or `iam_role_arns`).

- **session_token**: str, optional

- **iam_role_arns**: str or list of strings, optional
  - Either a single arn or a list of arns of roles to assume when unloading. Required unless you supply key based credentials (`access_key_id` and `secret_access_key`) or (`aws_account_id` and `iam_role_name`) separately.

- **aws_partition**: str, optional
  - AWS partition to use with role-based credentials. Defaults to 'aws'. Not applicable when using key based credentials (`access_key_id` and `secret_access_key`) or role arns (`iam_role_arns`) directly.

- **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`) or role arns (`iam_role_arns`) directly.

- **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) or role arns (iam_role_arns) directly.

**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.

**region: str, optional**

The AWS region where the target S3 bucket is located, if the Redshift cluster isn’t in the same region as the S3 bucket.

**max_file_size: int, optional**

Maximum size (in bytes) of files to create in S3. This must be between 5 * 1024**2 and 6.24 * 1024**3. Note that Redshift appears to round to the nearest KiB.

**format**

Indicates the type of file to unload to.

```python
sqlalchemy_redshift.commands.compile_refresh_materialized_view(element, compiler, **kw)
```

Formats and returns the refresh statement for materialized views.
sqlalchemy_redshift.commands.visit_alter_table_append_command(element, compiler, **kw)

Returns the actual SQL query for the AlterTableAppendCommand class.

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

Returns the actual SQL query for the CopyCommand class.

sqlalchemy_redshift.commands.visit_create_library_command(element, compiler, **kw)

Returns the actual SQL query for the CreateLibraryCommand class.

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

Returns the actual SQL query for the UnloadFromSelect class.
CHAPTER 41

Indices and tables

- genindex
- modindex
- search
S

sqlalchemy_redshift.commands, 83
### Index

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>AlterTableAppendCommand (class in sqlalchemy_redshift.commands), 83</td>
</tr>
<tr>
<td>C</td>
<td>compile_refresh_materialized_view() (in module sqlalchemy_redshift.commands), 90</td>
</tr>
<tr>
<td>C</td>
<td>Compression (class in sqlalchemy_redshift.commands), 83</td>
</tr>
<tr>
<td>C</td>
<td>CopyCommand (class in sqlalchemy_redshift.commands), 84</td>
</tr>
<tr>
<td>C</td>
<td>CreateLibraryCommand (class in sqlalchemy_redshift.commands), 87</td>
</tr>
<tr>
<td>E</td>
<td>Encoding (class in sqlalchemy_redshift.commands), 88</td>
</tr>
<tr>
<td>F</td>
<td>Format (class in sqlalchemy_redshift.commands), 88</td>
</tr>
<tr>
<td>R</td>
<td>RedshiftDDLCompiler (class in sqlalchemy_redshift.dialect), 81</td>
</tr>
<tr>
<td>R</td>
<td>RedshiftDialect (in module sqlalchemy_redshift.dialect), 83</td>
</tr>
<tr>
<td>R</td>
<td>RefreshMaterializedView (class in sqlalchemy_redshift.commands), 88</td>
</tr>
<tr>
<td>S</td>
<td>sqlalchemy_redshift.commands (module), 83</td>
</tr>
<tr>
<td>U</td>
<td>UnloadFromSelect (class in sqlalchemy_redshift.commands), 88</td>
</tr>
<tr>
<td>V</td>
<td>visit_alter_table_append_command() (in module sqlalchemy_redshift.commands), 90</td>
</tr>
<tr>
<td>V</td>
<td>visit_copy_command() (in module sqlalchemy_redshift.commands), 91</td>
</tr>
<tr>
<td>V</td>
<td>visit_create_library_command() (in module sqlalchemy_redshift.commands), 91</td>
</tr>
<tr>
<td>V</td>
<td>visit_unload_from_select() (in module sqlalchemy_redshift.commands), 91</td>
</tr>
</tbody>
</table>