SimpleSQLite Documentation

Release 1.3.0

Tsuyoshi Hombashi

Jun 20, 2021

TABLE OF CONTENTS

CHAPTER

ONE

SIMPLESQLITE

1.1 Summary

SimpleSQLite is a Python library to simplify SQLite database operations: table creation, data insertion and get data as other data formats. Simple ORM functionality for SQLite.

1.2 Features

- Automated SQLite table creation from data
- **Support various data types of record(s) insertion into a table:**
  - `dict`
  - `namedtuple`
  - `list`
  - `tuple`
- **Create table(s) from:**
  - CSV file/text
  - JSON file/text
  - pandas.DataFrame instance
  - tabledata.TableData instance loaded by pytablereader
- **Get data from a table as:**
  - pandas.DataFrame instance
  - tabledata.TableData instance

- Simple object-relational mapping (ORM) functionality

CHAPTER

TWO

INSTALLATION

2.1 Install from PyPI

```
pip install SimpleSQLite
```

2.2 Install from PPA (for Ubuntu)

```
sudo add-apt-repository ppa:thombashi/ppa
sudo apt update
sudo apt install python3-simplesqlite
```

CHAPTER

THREE

DEPENDENCIES

- Python 3.6+
- Python package dependencies (automatically installed)

3.1 Optional Dependencies

- **loguru**
  - Used for logging if the package installed
- pandas
- pytablereader

CHAPTER

FOUR

EXAMPLES

4.1 Create table

4.1.1 Create a table from data matrix

create_table_from_data_matrix() method create a table in a SQLite database from data matrix. Data matrix required one of the types: dict/namedtuple()/list/tuple.

Sample Code

Listing 1: Create a table in a SQLite database from data matrix

```
from simplesqlite import SimpleSQLite


table_name = "sample_table"
con = SimpleSQLite("sample.sqlite", "w")

# create table -----
data_matrix = [[1, 1.1, "aaa", 1, 1], [2, 2.2, "bbb", 2.2, 2.2], [3, 3.3,
↪"ccc", 3, "ccc"]]
con.create_table_from_data_matrix(
    table_name,
    ["attr_a", "attr_b", "attr_c", "attr_d", "attr_e"],
    data_matrix,
)

# display data type for each column in the table -----
print(con.schema_extractor.fetch_table_schema(table_name).dumps())

# display values in the table -----
print("records:")
result = con.select(select="*", table_name=table_name)
for record in result.fetchall():
    print(record)
```

Output

```
.. table:: sample_table

    +---------+-------+-----------+--------+------+-----+
```

(continues on next page)

(continued from previous page)

```
    |Attribute| Type  |PRIMARY KEY|NOT NULL|UNIQUE|Index|
    +=========+=======+===========+========+======+=====+
    |attr_a   |INTEGER|           |        |      |     |
    +---------+-------+-----------+--------+------+-----+
    |attr_b   |REAL   |           |        |      |     |
    +---------+-------+-----------+--------+------+-----+
    |attr_c   |TEXT   |           |        |      |     |
    +---------+-------+-----------+--------+------+-----+
    |attr_d   |REAL   |           |        |      |     |
    +---------+-------+-----------+--------+------+-----+
    |attr_e   |TEXT   |           |        |      |     |
    +---------+-------+-----------+--------+------+-----+


records:
(1, 1.1, 'aaa', 1.0, '1')
(2, 2.2, 'bbb', 2.2, '2.2')
(3, 3.3, 'ccc', 3.0, 'ccc')
```

4.1.2 Create a table from CSV

create_table_from_csv() method create a table from a CSV (Comma Separated Values) file/text.

Sample Code

Listing 2: Create a table in a SQLite database from CSV

```
from simplesqlite import SimpleSQLite

with open("sample_data.csv", "w") as f:
    f.write("\n".join([
        '"attr_a","attr_b","attr_c"',
        '1,4,"a"',
        '2,2.1,"bb"',
        '3,120.9,"ccc"',
    ]))

# create table ---
con = SimpleSQLite("sample.sqlite", "w")
con.create_table_from_csv("sample_data.csv")

# output ---
table_name = "sample_data"
print(con.fetch_attr_names(table_name))
result = con.select(select="*", table_name=table_name)
for record in result.fetchall():
    print(record)
```

Output

```
['attr_a', 'attr_b', 'attr_c']
(1, 4.0, u'a')
```

(continues on next page)

(continued from previous page)

```
(2, 2.1, u'bb')
(3, 120.9, u'ccc')
```

4.1.3 Create table(s) from JSON

create_table_from_json() method can create a table from a JSON file/text.

Sample Code 1

Listing 3: Create multiple tables in a SQLite database from multiple table data in a JSON file

```
from simplesqlite import SimpleSQLite

file_path = "sample_data_multi.json"

# create sample data file ---
with open(file_path, "w") as f:
    f.write("""{
        "table_a" : [
            {"attr_b": 4, "attr_c": "a", "attr_a": 1},
            {"attr_b": 2.1, "attr_c": "bb", "attr_a": 2},
            {"attr_b": 120.9, "attr_c": "ccc", "attr_a": 3}
        ],
        "table_b" : [
            {"a": 1, "b": 4},
            {"a": 2 },
            {"a": 3, "b": 120.9}
        ]
    }""")

# create table ---
con = SimpleSQLite("sample.sqlite", "w")
con.create_table_from_json(file_path)

# output ---
for table_name in con.fetch_table_names():
    print("table: " + table_name)
    print(con.fetch_attr_names(table_name))
    result = con.select(select="*", table_name=table_name)
    for record in result.fetchall():
        print(record)
    print()
```

Output

Listing 4: Output of the sample code 1

```
table: table_b
['a', 'b']
(1, '4')
(2, 'NULL')
```

(continues on next page)

(continued from previous page)

```
(3, '120.9')

table: table_a
['attr_a', 'attr_b', 'attr_c']
(1, 4.0, 'a')
(2, 2.1, 'bb')
(3, 120.9, 'ccc')
```

Sample Code 2

Listing 5: Create a table in a SQLite database from a table data in a JSON file

```
from simplesqlite import SimpleSQLite

file_path = "sample_data_single.json"

# create sample data file ---
with open(file_path, "w") as f:
    f.write("""[
        {"attr_b": 4, "attr_c": "a", "attr_a": 1},
        {"attr_b": 2.1, "attr_c": "bb", "attr_a": 2},
        {"attr_b": 120.9, "attr_c": "ccc", "attr_a": 3}
    ]""")

# create table ---
con = SimpleSQLite("sample.sqlite", "w")
con.create_table_from_json(file_path)

# output ---
for table_name in con.fetch_table_names():
    print("table: " + table_name)
    print(con.fetch_attr_names(table_name))
    result = con.select(select="*", table_name=table_name)
    for record in result.fetchall():
        print(record)
    print()
```

Output

Listing 6: Output of the sample code 2

```
table: sample_data_single
['attr_a', 'attr_b', 'attr_c']
(1, 4.0, 'a')
(2, 2.1, 'bb')
(3, 120.9, 'ccc')
```

4.1.4 Create a table from pandas DataFrame

create_table_from_dataframe() method can create a table from a `pandas.DataFrame` instance.

Sample Code

Listing 7: Create a table in a SQLite database from pandas.DataFrame

```
from simplesqlite import SimpleSQLite
import pandas

con = SimpleSQLite("pandas_df.sqlite")

con.create_table_from_dataframe(pandas.DataFrame(
    [
        [0, 0.1, "a"],
        [1, 1.1, "bb"],
        [2, 2.2, "ccc"],
    ],
    columns=['id', 'value', 'name']
), table_name="pandas_df")
```

Output

Listing 8: Output sqlite database file

```
$ sqlite3 pandas_df.sqlite
sqlite> .schema
CREATE TABLE 'pandas_df' (id INTEGER, value REAL, name TEXT);
```

4.1.5 Create table(s) from Excel

You can extract tabular data from an Excel file by `pytablereader.ExcelTableFileLoader` class defined in pytablereader module. And you can create a table from extracted data by using *create_table_from_tabledata()* method.

Sample Code

Listing 9: Create a table in a SQLite database from an Excel file

```
import pytablereader
import simplesqlite
import xlsxwriter
```

(continues on next page)

(continued from previous page)

```
file_path = "sample_data.xlsx"

# create sample data file ---
workbook = xlsxwriter.Workbook(file_path)

worksheet = workbook.add_worksheet("samplesheet1")
table = [
    ["", "", "", ""],
    ["", "a", "b", "c"],
    ["", 1, 1.1, "a"],
    ["", 2, 2.2, "bb"],
    ["", 3, 3.3, "cc"],
]
for row_idx, row in enumerate(table):
    for col_idx, item in enumerate(row):
        worksheet.write(row_idx, col_idx, item)

worksheet = workbook.add_worksheet("samplesheet2")

worksheet = workbook.add_worksheet("samplesheet3")
table = [
    ["", "", ""],
    ["", "", ""],
    ["aa", "ab", "ac"],
    [1, "hoge", "a"],
    [2, "", "bb"],
    [3, "foo", ""],
]
for row_idx, row in enumerate(table):
    for col_idx, item in enumerate(row):
        worksheet.write(row_idx, col_idx, item)

workbook.close()

# create table ---
con = simplesqlite.SimpleSQLite("sample.sqlite", "w")

loader = pytablereader.ExcelTableFileLoader(file_path)
for table_data in loader.load():
    con.create_table_from_tabledata(table_data)

# output ---
for table_name in con.fetch_table_names():
    print("table: " + table_name)
    print(con.fetch_attr_names(table_name))
    result = con.select(select="*", table_name=table_name)
    for record in result.fetchall():
        print(record)
    print()
```

Output

```
table: samplesheet1
['a', 'b', 'c']
(1.0, 1.1, 'a')
(2.0, 2.2, 'bb')
(3.0, 3.3, 'cc')

table: samplesheet3
['aa', 'ab', 'ac']
(1.0, 'hoge', 'a')
(2.0, '', 'bb')
(3.0, 'foo', '')
```

4.1.6 Create table(s) from Google Sheets

`GoogleSheetsTableLoader` class and *`create_table_from_tabledata()`* method can create a table from Google Spreadsheet.

Required packages:

- oauth2client
- pyOpenSSL

See also:

https://gspread.rtfd.io/en/latest/oauth2.html

Sample Code

Listing 10: Create a table in a SQLite database from Google Sheets

```
import simplesqlite
import pytablereader as ptr


credentials_file = "sample-xxxxxxxxxxx.json"

# create table ---
con = simplesqlite.SimpleSQLite("sample.sqlite", "w")

loader = ptr.GoogleSheetsTableLoader(credentials_file)
loader.title = "samplebook"

for table_data in loader.load():
    con.create_table_from_tabledata(table_data)

# output ---
for table_name in con.fetch_table_names():
    print("table: " + table_name)
    print(con.fetch_attr_names(table_name))
    result = con.select(select="*", table_name=table_name)
    for record in result.fetchall():
        print(record)
    print()
```

4.2 Insert records into a table

insert()/*insert_many()* method can insert record(s) into a table. Record is one of the dict/namedtuple()/list/tuple.

4.2.1 Insert dictionary

Sample Code

```
from simplesqlite import SimpleSQLite

table_name = "sample_table"
con = SimpleSQLite("sample.sqlite", "w")
con.create_table_from_data_matrix(
    table_name,
    ["attr_a", "attr_b", "attr_c", "attr_d", "attr_e"],
    [[1, 1.1, "aaa", 1,   1]])

con.insert(
    table_name,
    record={
        "attr_a": 4,
        "attr_b": 4.4,
        "attr_c": "ddd",
        "attr_d": 4.44,
        "attr_e": "hoge",
    })
con.insert_many(
    table_name,
    records=[
        {
            "attr_a": 5,
            "attr_b": 5.5,
            "attr_c": "eee",
            "attr_d": 5.55,
            "attr_e": "foo",
        },
        {
            "attr_a": 6,
            "attr_c": "fff",
        },
    ])

result = con.select(select="*", table_name=table_name)
for record in result.fetchall():
    print(record)
```

Output

```
(1, 1.1, 'aaa', 1, 1)
(4, 4.4, 'ddd', 4.44, 'hoge')
```

(continues on next page)

(continued from previous page)

```
(5, 5.5, 'eee', 5.55, 'foo')
(6, None, 'fff', None, None)
```

4.2.2 Insert list/tuple/namedtuple

Sample Code

```
from collections import namedtuple
from simplesqlite import SimpleSQLite

table_name = "sample_table"
con = SimpleSQLite("sample.sqlite", "w")
con.create_table_from_data_matrix(
    table_name,
    ["attr_a", "attr_b", "attr_c", "attr_d", "attr_e"],
    [[1, 1.1, "aaa", 1, 1]],
)

# insert namedtuple
SampleTuple = namedtuple("SampleTuple", "attr_a attr_b attr_c attr_d attr_e
↪")

con.insert(table_name, record=[7, 7.7, "fff", 7.77, "bar"])
con.insert_many(
    table_name,
    records=[(8, 8.8, "ggg", 8.88, "foobar"), SampleTuple(9, 9.9, "ggg", 9.
↪99, "hogehoge")],
)

# print
result = con.select(select="*", table_name=table_name)
for record in result.fetchall():
    print(record)
```

Output

```
(1, 1.1, 'aaa', 1, 1)
(7, 7.7, 'fff', 7.77, 'bar')
(8, 8.8, 'ggg', 8.88, 'foobar')
(9, 9.9, 'ggg', 9.99, 'hogehoge')
```

4.3 Update a record in a table

update() method can update record(s) in a table.

Sample Code

```
from simplesqlite import SimpleSQLite
from simplesqlite.query import Where


table_name = "sample_table"
con = SimpleSQLite("sample.sqlite", "w")

data_matrix = [
    [1, "aaa"],
    [2, "bbb"],
]
con.create_table_from_data_matrix(
    table_name,
    ["key", "value"],
    data_matrix)

print("---- before update ----")
for record in con.select(select="*", table_name=table_name).fetchall():
    print(record)
print()

con.update(table_name, set_query="value = 'ccc'", where=Where(key="key",␣
↪value=1))

print("---- after update ----")
for record in con.select(select="*", table_name=table_name).fetchall():
    print(record)
```

Output

```
---- before update ----
(1, 'aaa')
(2, 'bbb')

---- after update ----
(1, 'ccc')
(2, 'bbb')
```

4.4 Get Data from a Table

4.4.1 Get Data from a table as OrderedDict

select_as_dict() method can get data from a table in a SQLite database as a `collections.OrderedDict` list.

Sample Code

```
from simplesqlite import SimpleSQLite

con = SimpleSQLite("sample.sqlite", "w", profile=True)

con.create_table_from_data_matrix(
    "sample_table",
    ["a", "b", "c", "d", "e"],
    [
        [1, 1.1, "aaa", 1,   1],
        [2, 2.2, "bbb", 2.2, 2.2],
        [3, 3.3, "ccc", 3,   "ccc"],
    ])

for record in con.select_as_dict(table_name="sample_table"):
    print(record)
```

Output

```
OrderedDict([('a', 1), ('b', 1.1), ('c', 'aaa'), ('d', 1), ('e', 1)])
OrderedDict([('a', 2), ('b', 2.2), ('c', 'bbb'), ('d', 2.2), ('e', 2.2)])
OrderedDict([('a', 3), ('b', 3.3), ('c', 'ccc'), ('d', 3), ('e', 'ccc')])
```

4.4.2 Get Data from a table as pandas DataFrame

select_as_dataframe() method can get data from a table in a SQLite database as a `pandas.Dataframe` instance.

Sample Code

```
from simplesqlite import SimpleSQLite

con = SimpleSQLite("sample.sqlite", "w", profile=True)

con.create_table_from_data_matrix(
    "sample_table",
    ["a", "b", "c", "d", "e"],
    [
        [1, 1.1, "aaa", 1,   1],
        [2, 2.2, "bbb", 2.2, 2.2],
        [3, 3.3, "ccc", 3,   "ccc"],
    ])

print(con.select_as_dataframe(table_name="sample_table"))
```

Output

```
$ sample/select_as_dataframe.py
   a    b    c    d    e
0  1  1.1  aaa  1.0    1
1  2  2.2  bbb  2.2  2.2
2  3  3.3  ccc  3.0  ccc
```

4.5 ORM functionality

Sample Code

```
from simplesqlite import connect_memdb
from simplesqlite.model import Integer, Model, Real, Text


class Sample(Model):
    foo_id = Integer(primary_key=True)
    name = Text(not_null=True, unique=True)
    value = Real()


def main():
    con = connect_memdb()

    Sample.attach(con)
    Sample.create()
    Sample.insert(Sample(name="abc", value=0.1))
    Sample.insert(Sample(name="xyz", value=1.11))
    Sample.insert(Sample(name="bar", value=2.22))

    print(Sample.fetch_schema().dumps())
    print("records:")
    for record in Sample.select():
        print("    {}".format(record))

    return 0


if __name__ == "__main__":
    sys.exit(main())
```

Output

```
.. table:: sample

    +---------+-------+-----------+--------+------+-----+
    |Attribute| Type  |PRIMARY KEY|NOT NULL|UNIQUE|Index|
    +=========+=======+===========+========+======+=====+
    |foo_id   |INTEGER|X          |        |      |     |
    +---------+-------+-----------+--------+------+-----+
    |name     |TEXT   |           |X       |X     |     |
    +---------+-------+-----------+--------+------+-----+
```

(continues on next page)

(continued from previous page)

```
    |value    |REAL   |           |        |      |     |
    +---------+-------+-----------+--------+------+-----+


records:
    Sample: foo_id=1, name=abc, value=0.1
    Sample: foo_id=2, name=xyz, value=1.11
    Sample: foo_id=3, name=bar, value=2.22
```

4.6 Profiling

get_profile() method can get profile of query execution time.

Sample Code

```
from simplesqlite import SimpleSQLite


con = SimpleSQLite("sample.sqlite", "w", profile=True)
data_matrix = [
    [1, 1.1, "aaa", 1,   1],
    [2, 2.2, "bbb", 2.2, 2.2],
    [3, 3.3, "ccc", 3,   "ccc"],
]
con.create_table_from_data_matrix(
    "sample_table",
    ["a", "b", "c", "d", "e"],
    data_matrix,
    index_attrs=["a"])

for profile in con.get_profile():
    print(profile)
```

Output

```
SqliteProfile(query="CREATE INDEX IF NOT EXISTS sample_table_a_index ON␣
↪sample_table('a')", cumulative_time=0.021904945373535156, count=1)
SqliteProfile(query="CREATE TABLE IF NOT EXISTS 'sample_table' ('a'␣
↪INTEGER, 'b' REAL, 'c' TEXT, 'd' REAL, 'e' TEXT)", cumulative_time=0.
↪015315055847167969, count=1)
SqliteProfile(query="DROP TABLE IF EXISTS 'sample_table'", cumulative_
↪time=0.011831998825073242, count=1)
SqliteProfile(query="SELECT name FROM sqlite_master WHERE TYPE='table'",␣
↪cumulative_time=0.0004591941833496094, count=6)
SqliteProfile(query="SELECT * FROM 'sample_table'", cumulative_time=4.
↪220008850097656e-05, count=1)
```

4.7 Make an in-memory database

connect_memdb() function can create a SQLite database in memory. This function return *SimpleSQLite* instance, the instance can execute methods as well as a *SimpleSQLite* instance opened with write mode.

Sample Code

Listing 11: Make an in-memory database and create a table in the database

```
import simplesqlite


table_name = "sample_table"
con = simplesqlite.connect_memdb()

# create table -----
data_matrix = [[1, 1.1, "aaa", 1, 1], [2, 2.2, "bbb", 2.2, 2.2], [3, 3.3,
↪"ccc", 3, "ccc"]]
con.create_table_from_data_matrix(
    table_name,
    ["attr_a", "attr_b", "attr_c", "attr_d", "attr_e"],
    data_matrix,
)

# display data type for each column in the table -----
print(con.schema_extractor.fetch_table_schema(table_name).dumps())

# display values in the table -----
print("records:")
result = con.select(select="*", table_name=table_name)
for record in result.fetchall():
    print(record)
```

Output

```
.. table:: sample_table

    +---------+-------+-----------+--------+------+-----+
    |Attribute| Type  |PRIMARY KEY|NOT NULL|UNIQUE|Index|
    +=========+=======+===========+========+======+=====+
    |attr_a   |INTEGER|           |        |      |     |
    +---------+-------+-----------+--------+------+-----+
    |attr_b   |REAL   |           |        |      |     |
    +---------+-------+-----------+--------+------+-----+
    |attr_c   |TEXT   |           |        |      |     |
    +---------+-------+-----------+--------+------+-----+
    |attr_d   |REAL   |           |        |      |     |
    +---------+-------+-----------+--------+------+-----+
    |attr_e   |TEXT   |           |        |      |     |
    +---------+-------+-----------+--------+------+-----+


records:
```

(continues on next page)

(continued from previous page)

```
(1, 1.1, 'aaa', 1.0, '1')
(2, 2.2, 'bbb', 2.2, '2.2')
(3, 3.3, 'ccc', 3.0, 'ccc')
```

CHAPTER

FIVE

REFERENCE

5.1 SimpleSQLite class

class simplesqlite.**SimpleSQLite**(*database_src: Union[sqlite3.Connection, simplesqlite.core.SimpleSQLite, str]*, *mode: str = 'a'*, *delayed_connection: bool = True*, *max_workers: Optional[int] = None*, *profile: bool = False*)

Wrapper class for `sqlite3` module.

Parameters

- **database_src** (*str*) – SQLite database source. Acceptable types are: (1) File path to a database to be connected. (2) sqlite3.Connection instance. (3) SimpleSQLite instance
- **mode** (*str*) – Open mode.
- **delayed_connection** (*bool*) – Delaying connection to a database until access to the database the first time, if the value is `True`.
- **max_workers** (*int*) – Maximum number of workers to generate a table. In default, the same as the total number of CPUs.
- **profile** (*bool*) – Recording SQL query execution time profile, if the value is `True`.

See also:

connect() get_profile()

check_connection() → None

Raises ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.

Sample Code

```
import simplesqlite

con = simplesqlite.SimpleSQLite("sample.sqlite", "w")

print("---- connected to a database ----")
con.check_connection()

print("---- disconnected from a database ----")
con.close()
try:
    con.check_connection()
```

(continues on next page)

(continued from previous page)

```
except simplesqlite.NullDatabaseConnectionError as e:
    print(e)
```

Output

```
---- connected to a database ----
---- disconnected from a database ----
null database connection
```

close() → None
Commit and close the connection.

See also:

sqlite3.Connection.close()

commit() → None

See also:

sqlite3.Connection.commit()

connect(*database_path: str*, *mode: str = 'a'*) → None
Connect to a SQLite database.

Parameters

- **database_path** (*str*) – Path to the SQLite database file to be connected.
- **mode** (*str*) – "r": Open for read only. "w": Open for read/write. Delete existing tables when connecting. "a": Open for read/write. Append to the existing tables.

Raises

- **ValueError** – If database_path is invalid or *mode* is invalid.
- ***simplesqlite.DatabaseError*** – If the file is encrypted or is not a database.
- ***simplesqlite.OperationalError*** – If unable to open the database file.

property connection: Optional[sqlite3.Connection]
sqlite3.Connection instance of the connected database. :rtype: sqlite3.Connection

Type return

create_index(*table_name: str*, *attr_name: str*) → None

Parameters

- **table_name** (*str*) – Table name that contains the attribute to be indexed.
- **attr_name** (*str*) – Attribute name to create index.

Raises

- **IOError** – If the open *mode* is neither "w" nor "a".
- ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.
- ***simplesqlite.TableNotFoundError*** – If the table not found in the database.

create_index_list(*table_name: str*, *attr_names: Sequence[str]*) → None

Parameters

- **table_name** (*str*) – Table name that exists attribute.
- **attr_names** (*list*) – List of attribute names to create indices. Ignore attributes that are not existing in the table.

See also:

create_index()

create_table(*table_name: str*, *attr_descriptions: Sequence[str]*) → bool

Parameters

- **table_name** (*str*) – Table name to create.
- **attr_descriptions** (*list*) – List of table description.

Raises

- ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.
- **IOError** – If the open *mode* is neither "w" nor "a".

create_table_from_csv(*csv_source: str*, *table_name: str = ''*, *attr_names: Sequence[str] = ()*, *delimiter: str = ','*, *quotechar: str = '"'*, *encoding: str = 'utf-8'*, *primary_key: Optional[str] = None*, *add_primary_key_column: bool = False*, *index_attrs: Optional[Sequence[str]] = None*) → None

Create a table from a CSV file/text.

Parameters

- **csv_source** (*str*) – Path to the CSV file or CSV text.
- **table_name** (*str*) – Table name to create. Using CSV file basename as the table name if the value is empty.
- **attr_names** (*list*) – Attribute names of the table. Use the first line of the CSV file as attributes if `attr_names` is empty.
- **delimiter** (*str*) – A one-character string used to separate fields.
- **quotechar** (*str*) – A one-character string used to quote fields containing special characters, such as the `delimiter` or `quotechar`, or which contain new-line characters.
- **encoding** (*str*) – CSV file encoding.
- **primary_key** (*str*) – Primary key of the creating table.
- **index_attrs** (*tuple*) – List of attribute names that creating indices.

Raises **ValueError** – If the CSV data is invalid.

Dependency Packages

- pytablereader

Example *Create a table from CSV*

See also:

create_table_from_data_matrix() csv.reader() pytablereader.CsvTableFileLoader.load() pytablereader.CsvTableTextLoader.load()

create_table_from_data_matrix(*table_name: str*, *attr_names: Sequence[str]*, *data_matrix: Any*, *primary_key: Optional[str] = None*, *add_primary_key_column: bool = False*, *index_attrs: Optional[Sequence[str]] = None*, *type_hints: Optional[Sequence[Optional[Type[typepy.type._base.AbstractType]]]] = None*) → None

Create a table if not exists. Moreover, insert data into the created table.

Parameters

- **table_name** (*str*) – Table name to create.
- **attr_names** (*list*) – Attribute names of the table.
- **data_matrix** (List of dict/namedtuple()/list/tuple) – Data to be inserted into the table.
- **primary_key** (*str*) – Primary key of the creating table.
- **index_attrs** (*tuple*) – List of attribute names that creating indices.

Raises

- **simplesqlite.NameValidationError** – If the name is invalid for a SQLite table name.
- **simplesqlite.NameValidationError** – If the name is invalid for a SQLite attribute name.
- **ValueError** – If the data_matrix is empty.

Example *Create a table from data matrix*

See also:

create_table() insert_many() create_index_list()

create_table_from_dataframe(*dataframe*, *table_name: str = ''*, *primary_key: Optional[str] = None*, *add_primary_key_column: bool = False*, *index_attrs: Optional[Sequence[str]] = None*) → None

Create a table from a pandas.DataFrame instance.

Parameters

- **dataframe** (*pandas.DataFrame*) – DataFrame instance to convert.
- **table_name** (*str*) – Table name to create.
- **primary_key** (*str*) – Primary key of the creating table.
- **index_attrs** (*tuple*) – List of attribute names that creating indices.

Examples *Create a table from pandas DataFrame*

create_table_from_json(*json_source: str*, *table_name: str = ''*, *primary_key: Optional[str] = None*, *add_primary_key_column: bool = False*, *index_attrs: Optional[Sequence[str]] = None*) → None

Create a table from a JSON file/text.

Parameters

- **json_source** (*str*) – Path to the JSON file or JSON text.
- **table_name** (*str*) – Table name to create.

- **primary_key** (*str*) – Primary key of the creating table.
- **index_attrs** (*tuple*) – List of attribute names that creating indices.

Dependency Packages

- pytablereader

Examples *Create table(s) from JSON*

See also:

`pytablereader.JsonTableFileLoader.load()` `pytablereader.JsonTableTextLoader.load()`

create_table_from_tabledata(*table_data: tabledata._core.TableData*, *primary_key: Optional[str] = None*, *add_primary_key_column: bool = False*, *index_attrs: Optional[Sequence[str]] = None*) → None

Create a table from `tabledata.TableData`.

Parameters

- **table_data** (*tabledata.TableData*) – Table data to create.
- **primary_key** (*str*) – Primary key of the creating table.
- **index_attrs** (*tuple*) – List of attribute names that creating indices.

See also:

create_table_from_data_matrix()

property database_path: Optional[str]

File path of the connected database. :rtype: str

Examples

```
>>> from simplesqlite import SimpleSQLite
>>> con = SimpleSQLite("sample.sqlite", "w")
>>> con.database_path
'/tmp/sample.sqlite'
>>> con.close()
>>> print(con.database_path)
None
```

Type return

delete(*table_name: str*, *where: Optional[Union[str, simplesqlite.query.Where, simplesqlite.query.And, simplesqlite.query.Or]] = None*) → Optional[sqlite3.Cursor]

Send a DELETE query to the database.

Parameters

- **table_name** (*str*) – Table name of executing the query.
- **where** (*str*/*Where*/*And*/*Or*) – WHERE clause for the query.

drop_table(*table_name: str*) → None

Parameters **table_name** (*str*) – Table name to drop.

Raises

- ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.

- **IOError** – If the open *mode* is neither "w" nor "a".

execute_query(*query: Union[str, simplesqlite.query.QueryItem]*, *caller: Optional[Tuple] = None*) → Optional[sqlite3.Cursor]

Send arbitrary SQLite query to the database.

Parameters

- **query** – Query to executed.
- **caller** (*tuple*) – Caller information. Expects the return value of `logging.Logger.findCaller()`.

Returns The result of the query execution.

Return type sqlite3.Cursor

Raises

- ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.
- ***simplesqlite.OperationalError*** – If failed to execute a query.

Warning: This method can execute an arbitrary query. i.e. No access permissions check by *mode*.

fetch_attr_names(*table_name: str*) → List[str]

Returns List of attribute names in the table.

Return type list

Raises

- ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.
- ***simplesqlite.TableNotFoundError*** – If the table not found in the database.
- ***simplesqlite.OperationalError*** – If failed to execute a query.

Example

```
import simplesqlite

table_name = "sample_table"
con = simplesqlite.SimpleSQLite("sample.sqlite", "w")
con.create_table_from_data_matrix(
    table_name,
    ["attr_a", "attr_b"],
    [[1, "a"], [2, "b"]])

print(con.fetch_attr_names(table_name))

try:
    print(con.fetch_attr_names("not_existing"))
except simplesqlite.TableNotFoundError as e:
    print(e)
```

Output

```
['attr_a', 'attr_b']
'not_existing' table not found in /tmp/sample.sqlite
```

fetch_attr_type(*table_name: str*) → Dict[str, str]

Returns Dictionary of attribute names and attribute types in the table.

Return type dict

Raises

- ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.
- ***simplesqlite.TableNotFoundError*** – If the table not found in the database.
- ***simplesqlite.OperationalError*** – If failed to execute a query.

fetch_num_records(*table_name: str*, *where: Optional[Union[str,* simplesqlite.query.Where, simplesqlite.query.And, simplesqlite.query.Or*]] = None*) → Optional[int]

Fetch the number of records in a table.

Parameters

- **table_name** (*str*) – Table name to get number of records.
- **where** (str/*Where*/*And*/*Or*) – WHERE clause for the query.

Returns Number of records in the table. None if no value matches the conditions, or the table not found in the database.

Return type int

fetch_sqlite_master() → List[Dict]

Get sqlite_master table information as a list of dictionaries.

Returns sqlite_master table information.

Return type list

Raises ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.

Sample Code

```
import json

from simplesqlite import SimpleSQLite

con = SimpleSQLite("sample.sqlite", "w")
data_matrix = [
    [1, 1.1, "aaa", 1,   1],
    [2, 2.2, "bbb", 2.2, 2.2],
    [3, 3.3, "ccc", 3,   "ccc"],
]
con.create_table_from_data_matrix(
    "sample_table",
    ["a", "b", "c", "d", "e"],
    data_matrix,
    index_attrs=["a"])
```

(continues on next page)

(continued from previous page)

```
print(json.dumps(con.fetch_sqlite_master(), indent=4))
```

Output

```
[
    {
        "tbl_name": "sample_table",
        "sql": "CREATE TABLE 'sample_table' ('a' INTEGER, 'b' REAL, 'c
↪' TEXT, 'd' REAL, 'e' TEXT)",
        "type": "table",
        "name": "sample_table",
        "rootpage": 2
    },
    {
        "tbl_name": "sample_table",
        "sql": "CREATE INDEX sample_table_a_index ON sample_table('a')
↪",
        "type": "index",
        "name": "sample_table_a_index",
        "rootpage": 3
    }
]
```

fetch_table_names(*include_system_table: bool = False*, *include_view: bool = True*) → List[str]

Returns List of table names in the database.

Return type list

Raises

- ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.
- ***simplesqlite.OperationalError*** – If failed to execute a query.

Sample Code

```
from simplesqlite import SimpleSQLite

con = SimpleSQLite("sample.sqlite", "w")
con.create_table_from_data_matrix(
    "hoge",
    ["attr_a", "attr_b"],
    [[1, "a"], [2, "b"]])
print(con.fetch_table_names())
```

Output

```
['hoge']
```

fetch_value(*select: str*, *table_name: str*, *where: Optional[Union[str, simplesqlite.query.Where, simplesqlite.query.And, simplesqlite.query.Or]] = None*, *extra: Optional[str] = None*) → Optional[int]

Fetch a value from the table. Return `None` if no value matches the conditions, or the table not found in the database.

Parameters

- **select** (*str*) – Attribute for SELECT query
- **table_name** (*str*) – Table name of executing the query.
- **where** (`str`/*Where*/*And*/*Or*) – `WHERE` clause for the query.

Returns Result of execution of the query.

Raises

- ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.
- ***simplesqlite.OperationalError*** – If failed to execute a query.

fetch_view_names() → List[str]

Returns List of table names in the database.

Return type list

get_profile(*profile_count: int = 50*) → List[Any]

Get profile of query execution time.

Parameters **profile_count** (*int*) – Number of profiles to retrieve, counted from the top query in descending order by the cumulative execution time.

Returns Profile information for each query.

Return type list of `namedtuple()`

Raises

- ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.
- ***simplesqlite.OperationalError*** – If failed to execute a query.

Example *Profiling*

has_attr(*table_name: str*, *attr_name: Optional[str]*) → bool

Parameters

- **table_name** (*str*) – Table name that the attribute exists.
- **attr_name** (*str*) – Attribute name to be tested.

Returns `True` if the table has the attribute.

Return type bool

Raises ***simplesqlite.TableNotFoundError*** – If the table not found in the database.

Sample Code

```
import simplesqlite

table_name = "sample_table"
con = simplesqlite.SimpleSQLite("sample.sqlite", "w")
con.create_table_from_data_matrix(
    table_name,
    ["attr_a", "attr_b"],
    [[1, "a"], [2, "b"]])

print(con.has_attr(table_name, "attr_a"))
print(con.has_attr(table_name, "not_existing"))
try:
    print(con.has_attr("not_existing", "attr_a"))
except simplesqlite.DatabaseError as e:
    print(e)
```

Output

```
True
False
'not_existing' table not found in /tmp/sample.sqlite
```

has_attrs(*table_name: str*, *attr_names: Sequence[str]*) → bool

Parameters

- **table_name** (*str*) – Table name that attributes exists.
- **attr_names** – Attribute names to tested.

Returns True if the table has all of the attribute.

Return type bool

Raises ***simplesqlite.TableNotFoundError*** – If the table not found in the database.

Sample Code

```
import simplesqlite

table_name = "sample_table"
con = simplesqlite.SimpleSQLite("sample.sqlite", "w")
con.create_table_from_data_matrix(
    table_name,
    ["attr_a", "attr_b"],
    [[1, "a"], [2, "b"]])

print(con.has_attrs(table_name, ["attr_a"]))
print(con.has_attrs(table_name, ["attr_a", "attr_b"]))
print(con.has_attrs(table_name, ["attr_a", "attr_b", "not_existing"]))
try:
    print(con.has_attr("not_existing", ["attr_a"]))
except simplesqlite.DatabaseError as e:
    print(e)
```

Output

```
True
True
False
'not_existing' table not found in /tmp/sample.sqlite
```

has_table(*table_name: str*, *include_view: bool = True*) → bool

Parameters **table_name** (*str*) – Table name to be tested.

Returns True if the database has the table.

Return type bool

Sample Code

```
from simplesqlite import SimpleSQLite

con = SimpleSQLite("sample.sqlite", "w")
con.create_table_from_data_matrix(
    "hoge",
    ["attr_a", "attr_b"],
    [[1, "a"], [2, "b"]])

print(con.has_table("hoge"))
print(con.has_table("not_existing"))
```

Output

```
True
False
```

has_view(*view_name: str*) → bool

Parameters **table_name** (*str*) – Table name to be tested.

Returns True if the database has the table.

Return type bool

insert(*table_name: str*, *record: Any*, *attr_names: Optional[Sequence[str]] = None*) → None

Send an INSERT query to the database.

Parameters

- **table_name** (*str*) – Table name of executing the query.
- **record** (dict/namedtuple()/list/tuple) – Record to be inserted.

Raises

- **IOError** – If the open *mode* is neither "w" nor "a".
- ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.
- ***simplesqlite.OperationalError*** – If failed to execute a query.

Example *Insert records into a table*

insert_many(*table_name: str*, *records: Sequence[Union[Dict, Sequence]]*, *attr_names: Optional[Sequence[str]] = None*) → int

Send an INSERT query with multiple records to the database.

Parameters

- **table** (*str*) – Table name of executing the query.
- **records** (list of dict/namedtuple()/list/tuple) – Records to be inserted.

Returns Number of inserted records.

Return type int

Raises

- **IOError** – If the open *mode* is neither "w" nor "a".
- ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.
- ***simplesqlite.TableNotFoundError*** – If the table not found in the database.
- ***simplesqlite.OperationalError*** – If failed to execute a query.

Example *Insert records into a table*

is_connected() → bool

Returns True if the connection to a database is valid.

Return type bool

Examples

```
>>> from simplesqlite import SimpleSQLite
>>> con = SimpleSQLite("sample.sqlite", "w")
>>> con.is_connected()
True
>>> con.close()
>>> con.is_connected()
False
```

property mode: Optional[str]

Connection mode: "r"/"w"/"a". :rtype: str

See also:

connect()

Type return

rollback() → None

See also:

sqlite3.Connection.rollback()

select(*select: Union[str, simplesqlite.query.AttrList]*, *table_name: str*, *where: Optional[Union[str, simplesqlite.query.Where, simplesqlite.query.And, simplesqlite.query.Or]] = None*, *extra: Optional[str] = None*) → Optional[sqlite3.Cursor]

Send a SELECT query to the database.

Parameters

- **select** – Attribute for the SELECT query.
- **table_name** (*str*) – Table name of executing the query.
- **where** (str/*Where*/*And*/*Or*) – WHERE clause for the query.
- **extra** (*str*) – Any other SQL clause for the query.

Returns Result of the query execution.

Return type sqlite3.Cursor

Raises

- ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.
- ***simplesqlite.TableNotFoundError*** – If the table not found in the database.
- ***simplesqlite.OperationalError*** – If failed to execute a query.

select_as_dataframe(*table_name: str*, *columns: Optional[Sequence[str]] = None*, *where: Optional[Union[str,* simplesqlite.query.Where*,* simplesqlite.query.And*,* simplesqlite.query.Or*]] = None*, *extra: Optional[str] = None*)

Get data in the database and return fetched data as a `pandas.Dataframe` instance.

Parameters

- **table_name** (*str*) – Table name of executing the query.
- **columns** – Column names to get data. If the value is `None`, get data from all of the columns in the table.
- **where** – WHERE clause for the query.
- **extra** – Any other SQL clause for the query.

Returns Table data as a `pandas.Dataframe` instance.

Return type pandas.DataFrame

Raises

- ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.
- ***simplesqlite.TableNotFoundError*** – If the table not found in the database.
- ***simplesqlite.OperationalError*** – If failed to execute a query.

Example *Get Data from a table as pandas DataFrame*

Note: `pandas` package required to execute this method.

---

select_as_dict(*table_name: str*, *columns: Optional[Sequence[str]] = None*, *where: Optional[Union[str,* simplesqlite.query.Where*,* simplesqlite.query.And*,* simplesqlite.query.Or*]] = None*, *extra: Optional[str] = None*) → Optional[List[OrderedDict[str, Any]]]

Get data in the database and return fetched data as a `collections.OrderedDict` list.

Parameters

- **table_name** (*str*) – Table name of executing the query.

- **columns** (*list*) – Column names to get data. If the value is None, get data from all of the columns in the table.
- **where** (str/*Where*/*And*/*Or*) – WHERE clause for the query.
- **extra** (*str*) – Any other SQL clause for the query.

Returns Table data as collections.OrderedDict instances.

Return type list of collections.OrderedDict

Raises

- ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.
- ***simplesqlite.TableNotFoundError*** – If the table not found in the database.
- ***simplesqlite.OperationalError*** – If failed to execute a query.

Example *Get Data from a table as OrderedDict*

select_as_memdb(*table_name: str*, *columns: Optional[Sequence[str]] = None*, *where: Optional[Union[str,* simplesqlite.query.Where*,* simplesqlite.query.And*,* simplesqlite.query.Or*]] = None*, *extra: Optional[str] = None*)

Get data in the database and return fetched data as a in-memory *SimpleSQLite* instance.

Parameters

- **table_name** (*str*) – Table name of executing the query.
- **columns** – Column names to get data. If the value is None, get data from all of the columns in the table.
- **where** (str/*Where*/*And*/*Or*) – WHERE clause for the query.
- **extra** (*str*) – Any other SQL clause for the query.

Returns Table data as a *SimpleSQLite* instance that connected to in memory database.

Return type *SimpleSQLite*

Raises

- ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.
- ***simplesqlite.TableNotFoundError*** – If the table not found in the database.
- ***simplesqlite.OperationalError*** – If failed to execute a query.

select_as_tabledata(*table_name: str*, *columns: Optional[Sequence[str]] = None*, *where: Optional[Union[str,* simplesqlite.query.Where*,* simplesqlite.query.And*,* simplesqlite.query.Or*]] = None*, *extra: Optional[str] = None*, *type_hints: Optional[Dict[str, Optional[Type[typepy.type._base.AbstractType]]]] = None*) → tabledata._core.TableData

Get data in the database and return fetched data as a tabledata.TableData instance.

Parameters

- **table_name** (*str*) – Table name of executing the query.
- **columns** – Column names to get data. If the value is None, get data from all of the columns in the table.
- **where** (str/*Where*/*And*/*Or*) – WHERE clause for the query.

- **extra** (*str*) – Any other SQL clause for the query.

Returns Table data as a `tabledata.TableData` instance.

Return type tabledata.TableData

Raises

- ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.
- ***simplesqlite.TableNotFoundError*** – If the table not found in the database.
- ***simplesqlite.OperationalError*** – If failed to execute a query.

Note: `pandas` package required to execute this method.

---

set_row_factory(*row_factory: Optional[Callable]*) → None

Set row_factory to the database connection.

property total_changes: int

See also:

`sqlite3.Connection.total_changes`

update(*table_name: str*, *set_query: Optional[str]*, *where: Optional[Union[str,* simplesqlite.query.Where*,* simplesqlite.query.And*,* simplesqlite.query.Or*]] = None*) → Optional[sqlite3.Cursor]

Execute an UPDATE query.

Parameters

- **table_name** (`str`) – Table name of executing the query.
- **set_query** (`str`) – `SET` clause for the update query.
- **where** (`str`/*Where*/*And*/*Or* , optional) – `WHERE` clause for the update query. Defaults to `None`.

Raises

- **IOError** – If the open *mode* is neither `"w"` nor `"a"`.
- ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.
- ***simplesqlite.TableNotFoundError*** – If the table not found in the database.
- ***simplesqlite.OperationalError*** – If failed to execute a query.

validate_access_permission(*valid_permissions: Sequence[str]*) → None

Parameters **valid_permissions** (`list`/`tuple`) – List of permissions that access is allowed.

Raises

- **ValueError** – If the *mode* is invalid.
- **IOError** – If the *mode* not in the `valid_permissions`.
- ***simplesqlite.NullDatabaseConnectionError*** – If not connected to a SQLite database file.

verify_attr_existence(*table_name: str*, *attr_name: str*) → None

Parameters

- **table_name** (*str*) – Table name that the attribute exists.
- **attr_name** (*str*) – Attribute name to tested.

Raises

- ***simplesqlite.AttributeNotFoundError*** – If attribute not found in the table
- ***simplesqlite.TableNotFoundError*** – If the table not found in the database.

Sample Code

```
from simplesqlite import (
    SimpleSQLite,
    DatabaseError,
    AttributeNotFoundError
)

table_name = "sample_table"
con = SimpleSQLite("sample.sqlite", "w")
con.create_table_from_data_matrix(
    table_name,
    ["attr_a", "attr_b"],
    [[1, "a"], [2, "b"]])

con.verify_attr_existence(table_name, "attr_a")
try:
    con.verify_attr_existence(table_name, "not_existing")
except AttributeNotFoundError as e:
    print(e)
try:
    con.verify_attr_existence("not_existing", "attr_a")
except DatabaseError as e:
    print(e)
```

Output

```
'not_existing' attribute not found in 'sample_table' table
'not_existing' table not found in /tmp/sample.sqlite
```

verify_table_existence(*table_name: str*, *allow_view: bool = True*) → None

Parameters **table_name** (*str*) – Table name to be tested.

Raises

- ***simplesqlite.TableNotFoundError*** – If the table not found in the database.
- **simplesqlite.NameValidationError** – If the name is invalid for a SQLite table name.

Sample Code

```
import simplesqlite
```

(continues on next page)

(continued from previous page)

```
table_name = "sample_table"
con = simplesqlite.SimpleSQLite("sample.sqlite", "w")
con.create_table_from_data_matrix(
    table_name,
    ["attr_a", "attr_b"],
    [[1, "a"], [2, "b"]])

con.verify_table_existence(table_name)
try:
    con.verify_table_existence("not_existing")
except simplesqlite.DatabaseError as e:
    print(e)
```

Output

```
'not_existing' table not found in /tmp/sample.sqlite
```

5.2 Query classes

class simplesqlite.query.**Table**(*value: str*)

Parameters **name** (*str*) – Table name.

Returns String that suitable for table name of a SQLite query.

Examples

```
>>> from simplesqlite.query import Table
>>> Table("length")
'length'
>>> Table("length(cm)")
'[length(cm)]'
>>> Table("string length")
"'string length'"
```

to_query() → str

class simplesqlite.query.**Attr**(*name: str*, *operation: str = ''*)

Parameters

- **name** (*str*) – Attribute name.
- **operation** (*str*) – Used as a SQLite function if the value is not empty.

Returns String that suitable for attribute name of a SQLite query.

Return type str

Examples

```
>>> from simplesqlite.query import Attr
>>> Attr("key")
```

(continues on next page)

(continued from previous page)

```
'key'
>>> Attr("a+b")
'[a+b]'
>>> Attr("key", operation="SUM")
'SUM(key)'
```

classmethod sanitize(*name: str*) → str

to_query() → str

class simplesqlite.query.**AttrList**(*names: Sequence[str]*, *operation: str = ''*)

Parameters

- **names** (*list/tuple*) – Attribute names.
- **operation** (*str*) – Used as a SQLite function if the value is not empty.

Examples

```
>>> from simplesqlite.query import AttrList
>>> AttrList(["key", "a+b"]))
['key', '[a+b]']
>>> AttrList(["key", "a+b"], operation="AVG")
['AVG(key)', 'AVG([a+b])']
```

See also:

Attr

append(*item: Union[str,* simplesqlite.query.Attr*]*) → None
Append object to the end of the list.

classmethod sanitize(*names: Sequence[str]*) → List[str]

to_query() → str

class simplesqlite.query.**Value**(*value: Any*)

Parameters **value** (*str*) – Value associated with a key.

Returns String that suitable for a value of a key. Return "NULL" if the value is None.

Return type str

Examples

```
>>> from simplesqlite.query import Value
>>> Value(1.2)
'1.2'
>>> Value("value")
"'value'"
>>> Value(None)
'NULL'
```

to_query() → str

class simplesqlite.query.**Where**(*key: str*, *value: Any*, *cmp_operator: str = '='*)
WHERE query clause.

Parameters

- **key** (*str*) – Attribute name of the key.
- **value** – Value of the right hand side associated with the key.
- **cmp_operator** (*str*) – Comparison operator of WHERE query.

Raises ***simplesqlite.SqlSyntaxError*** – If **a)** `cmp_operator` is invalid operator. Valid operators are as follows: "=", "==", "!=", "<>", ">", ">=", "<", "<=". **b)** the `value` is `None` and the `cmp_operator` is not "="/"!=".

Examples

```
>>> from simplesqlite.query import Where
>>> Where("key", "hoge")
"key = 'hoge'"
>>> Where("value", 1, cmp_operator=">")
'value > 1'
```

property key: str

to_query() → str

property value: str

class simplesqlite.query.**Select**(*select: Union[str,* simplesqlite.query.AttrList*]*, *table: str*, *where: Optional[Union[str,* simplesqlite.query.Where, simplesqlite.query.And, simplesqlite.query.Or*]] = None*, *extra: Optional[str] = None*)

`SELECT` query clause.

Parameters

- **select** – Attribute for SELECT query.
- **table** (*str*) – Table name of executing the query.
- **where** (*str*) – Add a WHERE clause to execute query, if the value is not `None`.
- **extra** (*extra*) – Add additional clause to execute query, if the value is not `None`.

Raises

- **ValueError** – `select` is empty string.
- **simplesqlite.NameValidationError** – If the name is invalid for a SQLite table name.

Examples

```
>>> from simplesqlite.query import Select, Where
>>> Select(select="value", table="example")
'SELECT value FROM example'
>>> Select(select="value", table="example", where=Where("key", 1))
'SELECT value FROM example WHERE key = 1'
>>> Select(select="value", table="example", where=Where("key", 1),␣
↪extra="ORDER BY value")
'SELECT value FROM example WHERE key = 1 ORDER BY value'
```

to_query() → str

class simplesqlite.query.**And**(*where_list: List*)

`AND` query clause.

Parameters **where_list** (list of `str`/*Where*/*And*/*Or*) – Query items that concatenating with `AND`.

to_query() → str

class simplesqlite.query.**Or**(*where_list: List*)

OR query clause.

Parameters **where_list** (list of str/*Where*/*And*/*Or*) – Query items that concatenating with OR.

to_query() → str

5.3 Functions

simplesqlite.**append_table**(*src_con: SimpleSQLite*, *dst_con: SimpleSQLite*, *table_name: str*) → bool

Append a table from source database to destination database.

Parameters

- **src_con** (SimpleSQLite) – Connection to the source database.
- **dst_con** (SimpleSQLite) – Connection to the destination database.
- **table_name** (*str*) – Table name to append.

Returns True if the append operation succeed.

Return type bool

Raises

- ***simplesqlite.TableNotFoundError*** – If the table not found in the database.
- **ValueError** – If attributes of the table are different from each other.

simplesqlite.**copy_table**(*src_con: SimpleSQLite*, *dst_con: SimpleSQLite*, *src_table_name: str*, *dst_table_name: str*, *is_overwrite: bool = True*) → bool

Copy a table from source to destination.

Parameters

- **src_con** (SimpleSQLite) – Connection to the source database.
- **dst_con** (SimpleSQLite) – Connection to the destination database.
- **src_table_name** (*str*) – Source table name to copy.
- **dst_table_name** (*str*) – Destination table name.
- **is_overwrite** (*bool*) – If True, overwrite existing table.

Returns True if the copy operation succeed.

Return type bool

Raises

- ***simplesqlite.TableNotFoundError*** – If the table not found in the database.
- **ValueError** – If attributes of the table are different from each other.

simplesqlite.**connect_memdb**(*max_workers: Optional[int] = None*) → *simplesqlite.core.SimpleSQLite*

Returns Instance of an in memory database.

Return type *SimpleSQLite*

Example *Make an in-memory database*

5.4 Errors

exception simplesqlite.**DatabaseError**
Bases: sqlite3.DatabaseError

Exception raised for errors that are related to the database.

See also:

- sqlite3.DatabaseError

exception simplesqlite.**NullDatabaseConnectionError**
Bases: *simplesqlite.error.DatabaseError*

Exception raised when executing an operation of *SimpleSQLite* instance without connection to a SQLite database file.

exception simplesqlite.**TableNotFoundError**
Bases: *simplesqlite.error.DatabaseError*

Exception raised when accessed the table that not exists in the database.

exception simplesqlite.**AttributeNotFoundError**
Bases: *simplesqlite.error.DatabaseError*

Exception raised when accessed the attribute that not exists in the table.

exception simplesqlite.**SqlSyntaxError**
Bases: Exception

Exception raised when a SQLite query syntax is invalid.

exception simplesqlite.**OperationalError**(**args*, ***kwargs*)
Bases: sqlite3.OperationalError

Exception raised when failed to execute a query.

CHAPTER

SIX

CHANGELOG

https://github.com/thombashi/SimpleSQLite/releases

CHAPTER

SEVEN

SPONSORS

Become a sponsor

CHAPTER

EIGHT

INDICES AND TABLES

- genindex

CHAPTER

NINE

LINKS

- GitHub repository
- Issue tracker
- pip: A tool for installing python packages

9.1 Related project

- sqlitebiter: A CLI tool to create a SQLite database from CSV/JSON/Excel/Google-Sheets by using SimpleSQLite

CHAPTER

TEN

INDICES AND TABLES

- genindex

INDEX

O

R

S

T

U

V

W