mwsql

Release 0.1.5

Slavina Stefanova

Feb 10, 2024

CONTENTS:

CHAPTER

ONE

OVERVIEW

`mwsql` provides utilities for working with Wikimedia SQL dump files. It supports Python 3.9 and later versions.

`mwsql` abstracts the messiness of working with SQL dump files. Each Wikimedia SQL dump file contains one database table. The most common use case for `mwsql` is to convert this table into a more user-friendly Python `Dump class` instance. This lets you access the table's metadata (db names, field names, data types, etc.) as attributes, and its content – the table rows – as a generator, which enables processing of larger-than-memory datasets due to the inherent lazy/delayed execution of Python generators.

`mwsql` also provides a method to convert SQL dump files into CSV. You can find more information on how to use `mwsql` in the usage examples.

1.1 Installation

You can install `mwsql` with `pip`:

```
$ pip install mwsql
```

1.2 Basic Usage

```
>>> from mwsql import Dump
>>> dump = Dump.from_file('simplewiki-latest-change_tag_def.sql.gz')
>>> dump.head(5)
['ctd_id', 'ctd_name', 'ctd_user_defined', 'ctd_count']
['1', 'mw-replace', '0', '10453']
['2', 'visualeditor', '0', '309141']
['3', 'mw-undo', '0', '59767']
['4', 'mw-rollback', '0', '71585']
['5', 'mobile edit', '0', '234682']
>>> dump.dtypes
{'ctd_id': int, 'ctd_name': str, 'ctd_user_defined': int, 'ctd_count': int}
>>> rows = dump.rows(convert_dtypes=True)
>>> next(rows)
[1, 'mw-replace', 0, 10453]
```

1.3 Known Issues

1.3.1 Encoding errors

Wikimedia SQL dumps use utf-8 encoding. Unfortunately, some fields can contain non-recognized characters, raising an encoding error when attempting to parse the dump file. If this happens while reading in the file, it's recommended to try again using a different encoding. `latin-1` will sometimes solve the problem; if not, you're encouraged to try with other encodings. If iterating over the rows throws an encoding error, you can try changing the encoding. In this case, you don't need to recreate the dump – just pass in a new encoding via the `dump.encoding` attribute.

1.3.2 Parsing errors

Some Wikimedia SQL dumps contain string-type fields that are sometimes not correctly parsed, resulting in fields being split up into several parts. This is more likely to happen when parsing dumps containing file names from Wikimedia Commons or containing external links with many query parameters. If you're parsing any of the other dumps, you're unlikely to run into this issue.

In most cases, this issue affects a relatively very small proportion of the total rows parsed. For instance, Wikimedia Commons `page` dump contains approximately 99 million entries, out of which ~13.000 are incorrectly parsed. Wikimedia Commons `page links` on the other hand, contains ~760M records, and only 20 are wrongly parsed.

This issue is most commonly caused by the parser mistaking a single quote (or apostrophe, as they're identical) within a string for the single quote that marks the end of said string. There's currently no known workaround other than manually removing the rows that contain more fields than expected, or if they are relatively few, manually merging the split fields.

Future versions of `mwsql` will improve the parser to correctly identify when single quotes should be treated as string delimiters and when they should be escaped. For now, it's essential to be aware that this problem exists.

1.4 Project information

`mwsql` is released under the GPLv3. You can find the complete documentation at Read the Docs. If you run into bugs, you can file them in our issue tracker. Have ideas on how to make `mwsql` better? Contributions are most welcome – we have put together a guide on how to get started.

CHAPTER

TWO

USAGE EXAMPLES

2.1 Loading a dump file

Wikimedia SQL dump files are publicly available and can be downloaded from the web. They can also be directly accessed through Wikimedia environments like PAWS or Toolforge.

`mwsql` includes a load utility for easy (down)loading of dump files – All you need to know is which file you need. For this example, we want to download the latest `pages` dump from the Simple English Wikipedia. If we go to https://dumps.wikimedia.org/simplewiki/latest/, we see that this file is called `simplewiki-latest-page.sql.gz`. Instead of manually downloading it, we can do the following:

```
>>> from mwsql import load
>>> dump_file = load('simplewiki', 'page')
```

If you *are not* in a Wikimedia hosted environment, the file will now be downloaded to your current working directory, and you will see a progress bar:

```
>>> dump_file = load('simplewiki', 'page')
Downloading https://dumps.wikimedia.org/simplewiki/latest/simplewiki-latest-page.sql.gz
Progress:     92% [19.0 / 20.7] MB
```

If you *are* in a Wikimedia hosted environment, the file is already available to you and does not need downloading. The syntax is the same, however:

```
>>> dump_file = load('simplewiki', 'page')
```

In both cases, `dump_file` will be a PathObject that points to the file.

2.2 Loading a dump file from a different date

The default behavior of the `load` function is to load the file from the latest dump. If you want to use a file from an earlier date, you can specify this by passing the date as a string to the `date` parameter:

```
>>> dump_file = load('simplewiki', 'page', '20210720')
```

2.3 Peeking at a dump file

Before diving into the data contained in the dump, you may want to look at its raw contents. You can do so by using the `head` function:

```
>>> from mwsql import head
>>> head(dump_file)
-- MySQL dump 10.18  Distrib 10.3.27-MariaDB, for debian-linux-gnu (x86_64)
--
-- Host: 10.64.32.82    Database: simplewiki
-- ------------------------------------------------------
-- Server version   10.4.19-MariaDB-log

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;
```

By default, the `head` function prints the first 10 lines. This can be changed to anything you want by specifying it in the function call:

```
>>> from mwsql import head
>>> head(dump_file, 5)
-- MySQL dump 10.18  Distrib 10.3.27-MariaDB, for debian-linux-gnu (x86_64)
--
-- Host: 10.64.32.82    Database: simplewiki
-- ------------------------------------------------------
-- Server version   10.4.19-MariaDB-log
```

2.4 Creating a dump object from file

The main use of the `mwsql` library is to parse an SQL dump file and turn it into a Python object that is easier to work with.

```
>>> from mwsql import Dump
>>> dump = Dump.from_file(file_path)
```

The file that `file_path` refers to can be either a `.sql` or a `.sql.gz` file. Now that we have instantiated a Dump object, we can access its attributes:

```
>>> dump = Dump.from_file('simplewiki-latest-page.sql.gz')
>>> dump
Dump(database=simplewiki, name=page, size=21654225)
>>> dump.col_names
['page_id', 'page_namespace', 'page_title', 'page_restrictions', 'page_is_redirect',
↪'page_is_new', 'page_random', 'page_touched', 'page_links_updated', 'page_latest',
↪'page_len', 'page_content_model', 'page_lang']
>>> dump.encoding
'utf-8'
```

There are other attributes a well, such as `dtypes` or `primary_key`. See the Module Reference for a complete list.

2.5 Displaying the rows

The most interesting part of an SQL table is arguably its entries (rows.) We can take a look at them by using the `head` method. Note that this is different than the `head` *function* we used to peek at a file *before* we turned it into a Dump object.

```
>>> dump_file = load('simplewiki', 'change_tag_def')
>>> dump = Dump.from_file(dump_file)
>>> dump
Dump(database=simplewiki, name=change_tag_def, size=2133)
>>> dump.head()
['ctd_id', 'ctd_name', 'ctd_user_defined', 'ctd_count']
['1', 'mw-replace', '0', '10453']
['2', 'visualeditor', '0', '309141']
['3', 'mw-undo', '0', '59767']
['4', 'mw-rollback', '0', '71585']
['5', 'mobile edit', '0', '234682']
['6', 'mobile web edit', '0', '227115']
['7', 'very short new article', '0', '28794']
['8', 'visualeditor-wikitext', '0', '20529']
['9', 'mw-new-redirect', '0', '30423']
['10', 'visualeditor-switched', '0', '18009']
```

By default, the `head` method prints the `col_names`, followed by the first ten rows. You can change this by passing how many rows you'd like to see as a parameter:

```
>>> dump.head(3)
['ctd_id', 'ctd_name', 'ctd_user_defined', 'ctd_count']
['1', 'mw-replace', '0', '10453']
['2', 'visualeditor', '0', '309141']
['3', 'mw-undo', '0', '59767']
```

2.6 Iterating over rows

If we want to access the rows, all we need to do is create a generator object by using the Dump's `rows` method.

```
>>> dump_file = load('simplewiki', 'change_tag_def')
>>> dump = Dump.from_file(dump_file)
>>> dump
Dump(database=simplewiki, name=change_tag_def, size=2133)
>>> rows = dump.rows()
>>> for _ in range(5):
        print(next(rows))
['1', 'mw-replace', '0', '10453']
['2', 'visualeditor', '0', '309141']
['3', 'mw-undo', '0', '59767']
['4', 'mw-rollback', '0', '71585']
['5', 'mobile edit', '0', '234682']
```

2.7 Converting to Python dtypes

Note that in the above example, *all* values were printed as strings – even those that seem to be of a different dtype. We can tell the `rows` method that we would like to convert numeric types to int or float by setting the `convert_dtypes` parameter to `true`:

```
>>> rows = dump.rows(convert_dtypes=True)
>>> for _ in range(5):
        print(next(rows))
[1, 'mw-replace', 0, 10453]
[2, 'visualeditor', 0, 309141]
[3, 'mw-undo', 0, 59767]
[4, 'mw-rollback', 0, 71585]
[5, 'mobile edit', 0, 234682]
```

2.8 Exporting as CSV

You can export the dump as a CSV file by using the `to_csv` method and specifying a `file path` for the output file:

```
>>> dump_file = Dump.from_file(some_file)
>>> dump.to_csv('some_folder/outfile.csv')
```

While this may take some time for larger files, you don't risk running out of memory as neither the input nor the output file is ever loaded into RAM in one big chunk.

CHAPTER

THREE

HOW TO CONTRIBUTE

First of all, thank you for considering contributing to `mwsql`! The intent of this document is to help get you started. Don't be afraid to reach out with questions – no matter how "silly." Just open a PR whether you have made any significant changes or not, and we'll try to help. You can also open an issue to discuss any changes you want to make before you start.

3.1 Basic Guidelines

- Contributions of any size are welcome! Fixed a typo? Changed a docstring? No contribution is too small.
- Try to limit each pull request to *one* change only.
- *Always* add tests and docs for your code.
- Make sure your proposed changes pass our CI. Until it's green, you risk not getting any feedback on it.
- Once you've addressed review feedback, make sure to bump the pull request with a comment so we know you're done.

3.2 Local Dev Environment

To start, create a virtual environment and activate it. If you don't already have a preferred way of doing this, you can take a look at some commonly used tools: pew, virtualfish, and virtualenvwrapper.

Next, get an up to date checkout of the `mwsql` repository via `SSH`:

```
$ git clone git@github.com:blancadesal/mwsql.git
```

or if you want to use git via `https`:

```
$ git clone https://github.com/blancadesal/mwsql.git
```

Change into the newly created directory and install an editable version of `mwsql`:

```
$ cd mwsql
$ pip install -e .
```

pip will install all the necessary dependencies for you, no need to install from a separate `requirements.txt` file.

3.3 Dev dependencies

The only dependency you *really* need to install is tox. It will handle everyhing else for you, including running tests, formatting and linting through pre-commit, and building and serving the latest version of the documentation. Below are a few examples of how to run tox:

```
$ tox
# This will run the full pytest suite, as well as pre-commit.
# These are the same tests that are run by the CI

$ tox -e pre-commit
# This will run the pre-commit linting and formatting checks

$ tox -e docs
# This will run the documentation build process

$ tox -e serve-docs
# Live-serve docs with Sphinx autobuild
```

3.4 Code style

- We use flake8 to enforce PEP 8 conventions, isort to sort our imports, and we use the black formatter with a line length of 88 characters. Static typing is enforced using mypy. Code that does not follow these conventions won't pass our CI. These tools are configured in either `tox.ini` or `pyproject.toml`.
- Make sure your docstrings are formatted using the Sphinx-style format like in the example below:

```
def add_one(number: int) -> int:
    """
    Add one to a number.

    :param number: A very important parameter.
    :type number: int
    :rtype: int
    """
```

- As long as you run the tox suite before submitting a PR, you should be fine. Tox runs all the tools above by calling pre-commit. It also runs the whole pytest suite (see Tests below) across all supported Python versions, the same as the CI workflow.

```
$ tox
```

- See the section above how to run pre-commit on its own via tox

3.5 Tests

- We use pytest for testing. For the sake of consistency, write your asserts as `actual == expected`:

```
def test_add_one():
   assert func(2) == 3
   assert func(4) == 5
```

- You can run the test suite either through tox or directly with pytest:

```
$ python -m pytest
```

3.6 Docs

- Use semantic newlines in `.rst` files (reStructuredText files):

```
This is a sentence.
This is another sentence.
```

- If you start a new section, add two blank lines before and one blank line after the header, except if two headers follow immediately after each other:

```
Last line of previous section.


Header of New Top Section
-------------------------

Header of New Section
^^^^^^^^^^^^^^^^^^^^^

First line of new section.
```

- If you add a new feature, include one or more usage examples in `examples.rst`.

CHAPTER

FOUR

MODULE REFERENCE

4.1 mwsql.dump

A set of utilities for processing MediaWiki SQL dump data.

class mwsql.dump.**Dump**(*database: str | None*, *table_name: str | None*, *col_names: List[str]*, *col_sql_dtypes: Dict[str, str]*, *primary_key: str | None*, *source_file: str | Path*, *encoding: str*)

Class for parsing an SQL dump file and processing its contents.

property dtypes: Dict[str, type]

Mapping between col_names and native Python dtypes.

Returns
A mapping from the column names in a SQL table to their respective Python data types. Example: {"ct_id": int}

Return type
Dict[str, type]

property encoding: str

The encoding used to read the dump file.

Getter
Returns the current encoding

Setter
Sets the encoding to a new value

Returns
Text encoding

Return type
str

classmethod from_file(*file_path: str | Path*, *encoding: str = 'utf-8'*) → T

Initialize Dump object from dump file.

Parameters

- **cls** (Dump) – A Dump class instance
- **file_path** (*PathObject*) – Path to source SQL dump file. Can be a .gz or an uncompressed file
- **encoding** (*str, optional*) – Text encoding, defaults to "utf-8" If you get an encoding error when processing the file, try setting this parameter to 'Latin-1'

Returns
A Dump class instance

Return type
Dump

head(*n_lines: int = 10*, *convert_dtypes: bool = False*) → None

Display first n rows.

Parameters

- **n_lines** (*int, optional*) – Number of rows to display, defaults to 10
- **convert_dtypes** (*bool, optional*) – Optionally, shows numerical types as int or float instead of all str. Defaults to False.

rows(*convert_dtypes: bool = False*, *strict_conversion: bool = False*, ***fmtparams: Any*) → Iterator[List[Any]]

Create a generator object from the rows.

Parameters

- **convert_dtypes** (*bool, optional*) – When set to True, numerical types are converted from str to int or float. Defaults to False.
- **strict_conversion** (*bool, optional*) – When True, raise exception Error on bad input when converting from SQL dtypes to Python dtypes. Defaults to False.
- **fmtparams** – Any kwargs you want to pass to the csv.reader() function that does the actual parsing.

Yield
A generator used to iterate over the rows in the SQL table

Return type
Iterator[List[Any]]

to_csv(*file_path: str | Path*, ***fmtparams: Any*) → None

Write Dump object to CSV file.

Parameters
file_path (*PathObject*) – The file to write to. Will be created if it doesn't already exist. Will be overwritten if it does exist.

4.2 mwsql.utils

Utility functions used to download, open and display the contents of Wikimedia SQL dump files.

mwsql.utils.**download_file**(*url: str*, *file_name: str*) → Path | None

Download a file from a URL and show a progress indicator. Return the path to the downloaded file.

Parameters

- **url** (*str*) – URL to download from
- **file_name** (*str*) – name of the file to download

Returns
path to the downloaded file

Return type
Optional[Path]

mwsql.utils.**head**(*file_path: str | Path*, *n_lines: int = 10*, *encoding: str = 'utf-8'*) → None

Display first n lines of a file. Works with both .gz and uncompressed files. Defaults to 10 lines.

Parameters

- **file_path** (*PathObject*) – The path to the file
- **n_lines** (*int, optional*) – Lines to display, defaults to 10
- **encoding** (*str, optional*) – Text encoding, defaults to "utf-8"

mwsql.utils.**load**(*database: str*, *filename: str*, *date: str = 'latest'*, *extension: str = 'sql'*) → str | Path | None

Load a dump file from a Wikimedia public directory if the user is in a supported environment (PAWS, Toolforge…). Otherwise, download dump file from the web and save in the current working directory. In both cases,the function returns a path-like object which can be used to access the file. Does not check if the file already exists on the path.

Parameters

- **database** (*str*) – The database backup dump to download a file from, e.g. 'enwiki' (English Wikipedia). See a list of available databases here: https://dumps.wikimedia.org/backup-index-bydb.html
- **filename** (*str*) – The name of the file to download, e.g. 'page' loads the file {database}-{date}-page.sql.gz
- **date** (*str, optional*) – Date the dump was generated, defaults to "latest". If "latest" is not used, the date format should be "YYYYMMDD"
- **extension** (*str*) – The file extension. Defaults to 'sql'

Returns
Path to dump file

Return type
Optional[PathObject]

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

m

INDEX