pyqaxe Documentation

Release 0.1.2

Matthew Spellings

Jul 28, 2019

Contents:

pyqaxe is a library to facilitate unifying data access from a variety of sources. The basic idea is to expose data through custom tables and adapters using python's *sqlite3* module.

```
cache = pyqaxe.Cache()
cache.index(pyqaxe.mines.Directory())
cache.index(pyqaxe.mines.GTAR())

for (positions,) in cache.query(
    'select data from gtar_records where name = "position"'):
    pass # do something with positions array
```

CHAPTER 1

Installation

Install from PyPI:

```
pip install pyqaxe
```

Alternatively, install from source using the typical distutils procedure:

```
python setup.py install
```

CHAPTER 2

Examples

Usage examples go in the *examples* directory.

CHAPTER 3

Documentation

class pyqaxe.**Cache**(*location=':memory:'*, *read_only=False*)

A queryable cache of data found in one or more datasets

Cache objects form the core around which the functionality of pyqaxe is built. They reference an sqlite database at a particular location; this can either be ':memory:' (default) to build an in-memory database or a filename to create persistent storage of the cached contents.

The database is populated by indexing data sources, or *mines*, which may expose files for other mines to work with or create additional tables and associated conversion functions.

Caches and their mines can be reconsituted in a separate process by simply opening a new *Cache* object pointing to the same file location.

Caches can be opened in *read-only* mode which prevents modifications to the underlying database. Data can be selected from read-only databases, but indexing mines will not work.

Caches can be used as context managers. When the context exits, the cache (and all of its open file handles) will be closed automatically.

Cache objects create the following tables in the database:

- mines: The data sources that have been indexed by this object
- files: The files (or file-like objects) that have been exposed by indexed mines

The **mines** table has the following columns:

- pickle: A pickled representation of the mine
- update_time: The last time the mine was indexed

The **files** table has the following columns:

- path: The path of the file being referenced
- mine_id: Integer ID of the mine that provides the file

close()

Close the connection to the database.

get_cache_size()
Return the maximumnumber of files to keep open.

classmethod get_opened_cache(*unique_id*)
Return a currently-opened cache by its unique identifier.

This method allows entries stored in the database to reference living *Cache* objects by their persistent identifier, which is useful for running additional queries on the database or retrieving opened file objects.

index(*mine*, *force=False*)
Index a new mine.

Mines may add entries to the table of files or create additional tables. If a mine is new to this database, it will be indexed regardless of the *force* argument.

Parameters

- **mine** – Mine to index
- **force** – If True, force the mine to index its contents (usually implies some IO operations)

Returns The mine object that was indexed

insert_file(*conn*, *mine_id*, *path*, *mtime=None*, *parent=None*)
Insert a new entry into the files table.

named_mines
A dictionary mapping active mine type names to objects.

open_file(*row*, *mode='r'*, *named=False*)
Open an entry from the files table.

Pass this function an entire row from the files table, just as it is (i.e. *select * from files where ...*). Dispatches its work to the mine that owns the file. Returns a file-like object.

ordered_mines
A list of each active mine, in order of indexing.

query(**args*, ***kwargs*)
Run a query on the database.

See `sqlite3.Connection.query()` for details.

set_cache_size(*value*)
Set the maximum number of files to keep open.

class pyqaxe.mines.directory.**Directory**(*root='.'*, *exclude_regexes=()*, *exclude_suffixes=()*, *relative=False*)
A simple recursive directory browser.

Directory populates the files table by recursively searching all subdirectories of a given root directory.

Parameters

- **root** – Base directory to begin searching
- **exclude_regexes** – Iterable of regex patterns that should be excluded from addition to the list of files upon a successful search
- **exclude_suffixes** – Iterable of suffixes that should be excluded from addition to the list of files
- **relative** – Whether to store absolute or relative paths (see below)

Relative paths: Directory can store relative, rather than absolute, paths to files. To use absolute paths, set *relative=False* in the constructor (default). To make the paths be relative to the current working directory, set *relative=True*. To have the paths be relative to the *Cache* object that indexes this mine, set *relative=cache* for that cache object.

Examples:

```
cache.index(Directory(exclude_regexes=[r'/\..*']))
cache.index(Directory(exclude_suffixes=['txt', 'zip']))
```

class pyqaxe.mines.gtar.**GTAR**(*exclude_frames_regexes=('\.',)*)

Interpret getar-format files.

GTAR parses zip, tar, and sqlite-format archives in the getar format (https://libgetar.readthedocs.io) to expose trajectory data. The getar files themselves are opened upon indexing to find which records are available in each file, but the actual data contents are read on-demand.

Parameters **exclude_frames_regexes** – Iterable of regex patterns of quantity names that should be excluded as columns from *gtar_frames* table (see below)

GTAR objects create the following table in the database:

- gtar_records: Contains links to data found in all getar-format files
- gtar_frames: Contains sets of data stored by index for all getar-format files

The **gtar_records** table has the following columns:

- path: path within the archive of the record
- gtar_group: *group* for the record
- gtar_index: *index* for the record
- name: *name* for the record
- file_id: files table identifier for the archive containing this record
- data: exposes the data of the record. Value is a string, bytes, or array-like object depending on the stored format.

The **gtar_frames** table's columns depend on which records are found among the indexed files. For each unique index, it lists all quantities found among all archives as columns (note that some quantity names may need to be surrounded by quotes) up to that index. gtar_frames contains the following additional columns:

- gtar_index: *index* for the record
- file_id: files table identifier for the archive containing this record

: cache.query('SELECT box, position FROM gtar_frames')

GTAR objects register a **gtar_frame** collation that can be used to sort indices in the standard GTAR way, rather than sqlite's default string comparison:

```
cache.query('SELECT data FROM gtar_records WHERE name = "position" '
            'ORDER BY gtar_index COLLATE gtar_frame')
```

Note: Consult the libgetar documentation to find more details about how records are encoded.

classmethod get_cache_size()

Return the maximum number of files to keep open.

classmethod set_cache_size(*value*)
Set the maximum number of files to keep open.

class pyqaxe.mines.glotzformats.**GlotzFormats**(*exclude_regexes=()*, *exclude_suffixes=()*)
Expose frames of glotzformats-readable trajectory formats.

GlotzFormats parses trajectory files and exposes them with a common interface. Files are opened once upon indexing to query the number of frames and data are read on-demand as frame data are selected.

GlotzFormats objects create the following table in the database:

- glotzformats_frames: Contains entries for each frame found in all trajectory files

The **glotzformats_frames** table has the following columns:

- file_id: files table identifier for the archive containing this record
- cache_id: *Cache* unique identifier for the archive containing this record
- frame: Integer (0-based) corresponding to the frame index within the trajectory
- box: Glotzformats box object for the frame
- types: Glotzformats types object for the frame
- positions: Glotzformats positions object for the frame
- velocities: Glotzformats velocities object for the frame
- orientations: Glotzformats orientations object for the frame
- shapedef: Glotzformats shapedef object for the frame

Note: Consult the glotzformats documentation to find more details about the encoding of the various data types listed here.

---

class pyqaxe.mines.tarfile.**TarFile**(*target=None*, *exclude_regexes=()*, *exclude_suffixes=()*, *relative=False*)
Expose the files within one or more tar-format archives.

TarFile populates the files table from one or more "source" tar archives. These archives can be entries that have been found by previously-indexed mines (*target=None*) or a single file that exists somewhere in the filesystem (*target='/path/to/file.tar'*).

Parameters

- **target** – Optional single tar file to open. If not given, expose records found inside all tar archives
- **exclude_regexes** – Iterable of regex patterns that should be excluded from addition to the list of files upon a successful search
- **exclude_suffixes** – Iterable of suffixes that should be excluded from addition to the list of files
- **relative** – Whether to use absolute or relative paths for *target* argument (see below)

Relative paths: *TarFile* can store the target tar archive location as a relative, rather than absolute, path. To use paths exactly as they are given, set *relative=False* in the constructor (default). To make the path be relative to the current working directory, set *relative=True*. To have the path be relative to the *Cache* object that indexes this mine, set *relative=cache* for that cache object.

Links: *TarFile* can be used to expose bundles of links to files on the filesystem. When indexing the *TarFile*, if a link is found and the file it references exists, that file will be added to the files table. Relative link pathss are interpreted with respect to the tar file they come from.

Examples:

```
cache.index(TarFile('archive.tar', relative=True))
cache.index(TarFile(exclude_regexes=[r'/\..*']))
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index