astrodbkit Documentation

Release 0.6.6

BDNYC

May 04, 2018

Contents

This documentation describes a toolkit of classes, methods, and functions useful for CRUD operations and analysis of data from an SQL database.

CHAPTER 1

Getting Started

To install **astrodbkit**, do:

```
pip install astrodbkit
```

Alternatively, you can update your existing installation with:

```
pip install --upgrade astrodbkit
```

CHAPTER 2

Creating a Database

To create a database from scratch, do:

```
from astrodbkit import astrodb
dbpath = '/desired/path/to/my_new_database.db'
astrodb.create_database(dbpath)
```

Alternatively, you can download and use the BDNYC Database, which contains the astrometry, photometry, and spectra for the 198 objects in the Filippazzo et al. (2015) sample.

Note: For access to the full dataset, an email request must be made to a BDNYC group admin.

CHAPTER 3

Accessing the Database

To start using the database, launch Python, import the module, then initialize the database with the `astrodb.Database()` class like so:

```
from astrodbkit import astrodb
db = astrodb.Database(dbpath)
```

You are now ready to use the database.

Note: The path to the database can either be the binary database file (typically ending in .db) or an .sql file that contains the schema for the database. In the later case, the database is constructed by referring to the schema and the directory where the tables of data are located (by default, 'tabledata') and a .db file is created. You can use the `save()` method to output a database to a schema file and the individual tables to a specified directory. This workflow can better facilitate version control.

CHAPTER 4

Querying the Database

4.1 Getting Help

It can be daunting to start using the database without a lot of prior knowledge of the contents of the database or the functionality of astrodbkit. For this purpose we have created two utility methods:

```
db.info()
```

Will list the names of the files that have been loaded as well as the contents of the database: every table and the number of sources in it.

And:

```
db.help()
```

Will give a brief overview of what `astrodb.Database()` is and summarizes the more widely used methods. More details on each method can be obtained by using Python's help system, for example:

```
help(db.query)
```

4.2 Specialized Searches

Now that you have the database loaded, you'll want to get some information out of it. There are a variety of ways to extract information with astrodbkit.

The schema for any table can be quickly examined with the `schema()` method:

```
db.schema('sources')
```

You can see an inventory of all data for a specific source by passing an integer id to the `inventory()` method:

```
data = db.inventory(86)
```

This will retrieve the data across all tables with the specified source_id for visual inspection. Setting *fetch=True* will return the data as a dictionary of Astropy tables so that table and column keys can be used to access the results. For example:

```
data['photometry'][['band','magnitude','magnitude_unc']]
```

will return a table of the band, magnitude and uncertainty for all records in the sources table with that source_id.

You can search any table in the database with the `identify()` method by supplying a string, integer, or (ra,dec) coordinates along with the table to search. For example, if I want to find all the records in the SOURCES table in the HR 8799 system:

```
db.search('8799', 'sources')
```

Or all the papers published by Joe Filippazzo:

```
db.search('Fili', 'publications')
```

When supplying coordinates, you can also specify the search *radius* to use, in degrees:

```
db.search((338.673, 40.694), 'sources', radius=5)
```

The `references()` method can be used to search for all entries that match the publication record. For example:

```
db.references('Cruz03')
```

4.3 General Queries

You can also pass SQL queries wrapped in double-quotes (“) to the `query()` method:

```
data = db.query( "SQL_query_goes_here" )
```

For example, you can get an Astropy table of all the records with a spectral type of L2 with:

```
db.query("SELECT * FROM spectral_types WHERE spectral_type=12", fmt='table')
```

By default, this returns the data as a list of arrays. Alternative options for the fmt flag include *‘dict’* for a list of Python dictionaries, *‘table’* for an Astropy Table, and *‘pandas’* for a pandas DataFrame.

Here is a detailed post about how to write a SQL query.

For more general SQL commands beyond SELECT and PRAGMA, you can use the `modify()` method:

```
db.modify("UPDATE spectra SET wavelength_units='A' WHERE id=4")
```

CHAPTER 5

Adding Data

There are two main ways to add data to a database with **astrodbkit**: by passing a properly formatted ascii file or by passing the data directly in a list of lists.

To add data from a file, you want to create a file with the following format:

```
ra|dec|publication_id
123|-34|5
```

Each entry should be its own row, with the first row denoting the columns to be populated. Note that the column names in the ascii file need not be in the same order as the table. Also, only the column names that match will be added and non-matching or missing column names will be ignored. Assuming this file is called **data.txt** in the working directory, we can add this new data to the SOURCES table with:

```
db.add_data('data.txt', 'sources', delim='|')
```

To add the same data without creating the file, you would do the following:

```
data = [['ra', 'dec', 'publication_id'],[123, -34, 5]]
db.add_data(data, 'sources')
```

CHAPTER 6

Saving Results

The `query()` method provides an option to export your query to a file:

```
db.query(my_query, export='results.txt')
```

By default, this will save the results to an ascii file.

VOTables are another way to store data in a format that can be read by other programs, such as TOPCAT. The `votools` module can generate a VOTable from your SQL query. This has been integrated to be called directly by the `query()` method when the filename ends in **.xml** or **.vot**. For example:

```
from astrodbkit import astrodb
db = astrodb.Database('/path/to/database')
txt = 'SELECT s.id, s.ra, s.dec, s.shortname, p.source_id, p.band, p.magnitude FROM␣
↪sources as s ' \
      'JOIN photometry as p ON s.id=p.source_id WHERE s.dec<=-10 AND (p.band IN ("J",
↪"H","Ks","W1"))'
data = db.query(txt, export='votable.xml')
```

You can import and call votools directly, which has additional options you can set.

Note: Special characters (such as accents or greek letters) can cause astropy and thus file output to fail in Python 2. Python 3 handles this differently and will not fail in this instance.

CHAPTER 7

Saving the Full Database

If changes have been made to the database, such as by adding new data or modifying existing entries, you will want to use the `save()` method to dump the contents of the database to ascii files. `save()` writes a schema file and outputs all tables to individual files in a directory of your choice (by default, 'tabledata'). This directory and the schema file can be version controlled with, for example, git, to facilitate tracking changes in the database.

When finished working with the database, the `close()` method will close the connection.

Note: `close()` will prompt to save the database to the default directory.

CHAPTER 8

Contents

8.1 astrodb module

8.2 astrocat module

8.3 votools module

astrodbkit.votools.**dict_tovot**(*tabdata*, *tabname='votable.xml'*, *phot=False*, *binary=True*)
Converts dictionary table **tabdata** to a VOTable with name **tabname**

Parameters

- **tabdata** (*list*) – SQL query dictionary list from running query_dict.execute()
- **tabname** (*str*) – The name of the VOTable to be created
- **phot** (*bool*) – Parameter specifying if the table contains photometry to be merged
- **binary** (*bool*) – Parameter specifying if the VOTable should be saved as a binary. This is necessary for tables with lots of text columns.

astrodbkit.votools.**photaddline**(*tab*, *sourceid*)
Loop through the dictionary list **tab** creating a line for the source specified in **sourceid**

Parameters

- **tab** – Dictionary list of all the photometry data
- **sourceid** – ID of source in the photometry table (source_id)

Returns **tmpdict** – Dictionary with all the data for the specified source

Return type dict

astrodbkit.votools.**photparse**(*tab*)
Parse through a photometry table to group by source_id

Parameters **tab** (*list*) – SQL query dictionary list from running query_dict.execute()

Returns **newtab** – Dictionary list after parsing to group together sources

Return type list

astrodbkit.votools.**table_add**(*tab*, *data*, *col*)

Function to parse dictionary list **data** and add the data to table **tab** for column **col**

Parameters

- **tab** (*Table class*) – Table to store values
- **data** (*list*) – Dictionary list from the SQL query
- **col** (*str*) – Column name (ie, dictionary key) for the column to add

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index