olapy Documentation

Release 0.6.2

Abilian SAS

Jul 31, 2019

Contents

1 Installation **3**
- 1.1 Install from PyPI . . . 3
- 1.2 Install from Github . . . 3
- 1.3 Initialization . . . 3
- 1.4 Olapy-data . . . 4

2 Quick Start **5**
- 2.1 OlaPy as XMLA server . . . 5
- 2.2 Olapy as a library . . . 6

3 Advanced Olapy options **7**

4 Cubes creation **9**
- 4.1 OLAPY CUBES RULES . . . 9
- 4.2 Examples: . . . 9

5 Cube customization **13**

6 Running olapy with Database **15**
- 6.1 Environnement variable . . . 15
- 6.2 Database string connection . . . 16
- 6.3 Olapy config file . . . 16

7 OlaPy ETL **17**

8 API Documentation **19**
- 8.1 Package `olapy.core.services.xmla` . . . 19
- 8.2 Package `olapy.core.mdx.executor` . . . 19
- 8.3 Package `olapy.etl.etl` . . . 19

9 Indices and tables **21**

OlaPy is an OLAP engine based on Python, which gives you a set of tools for the development of reporting and analytical applications, multidimensional analysis, and browsing of aggregated data with MDX and XMLA support. It can be found in GitHub. or PyPI.

- It is fast and uses in-memory technology and various techniques (aggregation and real-time computation) to provide sub-second responses.
- It includes an ETL layer (Extract Transform Load) for better data handling.
- It support most common databases (Postgres, MySql, Oracle, SQL Server) and CSV file format (only CSV right now) to construct cubes.

Contents:

CHAPTER 1

Installation

1.1 Install from PyPI

You can install it directly from the Python Package Index:

```
pip install olapy
```

1.2 Install from Github

The project sources are stored in Github repository.

Download from GitHub:

```
git clone git://github.com/abilian/olapy.git
```

Then install:

```
cd olapy
python setup.py install
```

1.3 Initialization

Before running olapy, you have to initialize it with:

```
olapy init
```

1.3.1 Testing

OlaPy is configured to run units and integrations tests suites. Before running tests, make sure you have installed all development requirements with:

```
pip install -r dev-requirements.txt
```

and then run:

```
tox
```

Test other databases

The default database used with tests is sqlite, if you want to run tests against mysql or postgres, you need to install the appropriate driver and export a connection string like this

```
export SQLALCHEMY_DATABASE_URI = { dialect+driver://username:password@host:port/
↪database }
```

take a look to SQLAlchemy documentation for more information.

1.4 Olapy-data

After Olapy initialization, you can take a look to *olapy-data* folder located under:

```
~/olapy-data                        for Linux/Mac user

C:\User\{USER_NAME}\olapy-data      for Windows
```

This folder contains some required files to configure olapy and some demo cubes under /cubes folder, we will deeply discuss about this in the *Cubes* and *Cube Customization*

CHAPTER 2

Quick Start

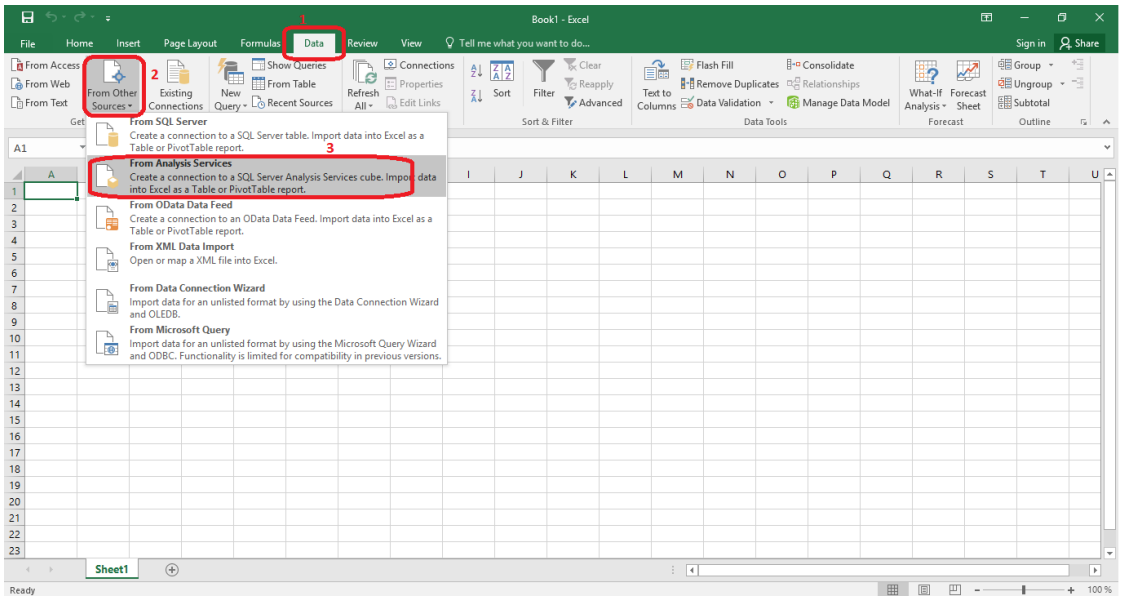
2.1 OlaPy as XMLA server

After *installation*, you can run the Olapy server with:

```
olapy runserver
```

All you have to do now is to try it with an Excel spreadsheet:

open excel, open new spreadsheet and go to : Data -> From Other Sources -> From Analysis Services



After that, Excel will ask you the server name: put `http://127.0.0.1:8000/` and click next, then you can chose one of default olapy demo cubes (sales, foodmart. . .) and finish.

That's it! Now you can play with your data.

2.2 Olapy as a library

If you want to use olapy as a library to execute MDX queries, start by importing the MDX engine:

```
from olapy.core.mdx.executor import MdxEngine
```

In our example, we're going to use sales demo cube:

```
executor = MdxEngine() # instantiate the MdxEngine
executor.load_cube('sales') # load sales cube
```

We set an MDX query:

```
query = """
SELECT
Hierarchize({[Measures].[Amount]}) ON COLUMNS
FROM [sales]
"""
```

and execute it:

```
df = executor.execute_mdx(query)['result']
print(df)
```

Result:

	Amount
0	1023

CHAPTER 3

Advanced Olapy options

Olapy is easy configurable. When using the `olapy runserver` command, you can pass a lot of options to it:

```
-st     Cubes source type (db|csv), DEFAULT : csv only
-sa     SQL Alchemy URI to connect to database , **DON'T PUT THE DATABASE NAME !**
-wf     Write logs into a file or display them into the console. log file location,
        by default under olapy-data folder
-lf     If you want to change log file location.
-od     Olapy-Data folder location
-h      Host ip adresse
-p      Host port
-dbc    Database configuration file path, Default : ~/olapy-data/olapy-config.yml
-cbf    Cube config file path, default : ~/olapy-data/cube/cubes-config.yml

-tf     File path or DB table name if you want to construct cube from a single file␣
↪(or table)
-c      To explicitly specify columns (construct cube from a single file), columns␣
↪order matters
-m      To explicitly specify measures (construct cube from a single file)
```

Here is an example of `olapy runserver` with all options:

```
olapy runserver -sa=postgresql://postgres:root@localhost:5432
                -wf=False
                -lf=/home/{USER_NAME}/Documents/olapy_logs
                -od=/home/{USER_NAME}/Documents
                -st=db,csv
                -h=0.0.0.0
                -p=8000
```

Here is an example of `olapy runserver` with a simple csv file:

```
olapy runserver -tf=/home/moddoy/olapy-data/cubes/sales/Data.csv
                -c=City,Licence
                -m=Amount,Count
```

CHAPTER 4

Cubes creation

To add new cube, put your CSV files in a folder (folder name <=> cube name), **make sure that they follow** *OLAPY CUBES RULES*, and move that folder under `olapy-data/cubes`, thus, the path to your cube will be:

- `~/olapy-data/cubes/{YOUR_CUBE}/{YOU_CSV_FILES}` for Mac/Linux,
- `C:\\User\\{USER_NAME}\\olapy-data\\{YOUR_CUBE}\\{YOU_CSV_FILES}` for Windows.

4.1 OLAPY CUBES RULES

NOTE : THE SAME THING IF YOU WANT TO WORK WITH DATABASES

Here are the rules to apply to your tables so that can works perfectly with olapy:

1) Make sure that your tables follow the star schema
2) The fact table should be named 'Facts'
3) Each table id columns, must be the same in facts table, example (product_id column from product table must be product_id in Facts table,
4) Avoid 'id' for id columns name, you should use something_id for example
5) The columns name must be in a good order (hierarchy) (example : Continent -> Country -> City. . .)

take a look to the default cubes structure (sales and foodmart).

Here are two examples of table structures that follows olapy rules:

4.2 Examples:

4.2.1 Cube 1

Geography table

Geo_id	Continent	Country
0001	America	Canada
...		
00526	Europe	France

Facts table

Geo_id	Prod_id	Amount	Count
0001	111111	5000	20
...			
0011	222222	1000	40

Product table

Prod_id	Company	Name
111111	Ferrero	Nutella
...		
222222	Nestle	KitKat

4.2.2 Cube 2

Here we don't use id column name in tables.

Geography table

Continent	Country
America	Canada
...	
Europe	France

Facts table

Continent	Company	Amount	Count
America	Ferrero	5000	20
...			
Europe	Nestle	1000	40

Product table

Company	Name
Ferrero	Nutella
...	
Nestle	KitKat

CHAPTER 5

Cube customization

If you don't want to follow olapy cubes rules and you want to customize your cube construction, you can use a configuration file, you can find the default example in

```
~/olapy-data/cubes/cubes-config.xml for mac/linux

C:\\User\\{USER_NAME}\\olapy-data\\cubes\\cubes-config.xml for windows
```

Here is an examples of configuration:

Assuming we have two tables as follows under 'custom_cube' folder

table 1: stats (which is the facts table)

departement_id	amount	monthly_salary	total monthly cost
111	1000	2000	3000
...			

table 2: organization (which is a dimension)

id	type	name	acronym	other colums.....
111	humanitarian	humania	for better life	
...	...			

you can use a configuration file like this to construct cube and access to it with excel:

```
# if you want to set an authentication mechanism to access cube,
# user must set a token with login url like 'http://127.0.0.1/admin
# default password = admin
xmla_authentication : False

# cube name <==> db name
name : custom_cube
#csv | postgres | mysql ...
```

(continues on next page)

(continued from previous page)

```
source : csv

# star building customized star schema
facts :
  table_name : stats
  keys:
    departement_id : organization.id

  measures :
    # by default, all number type columns in facts table, or you can specify them here
    - amount
    - monthly_salary

# star building customized dimensions display in excel from the star schema
dimensions:
  #  IMPORTANT , put here facts table also
  - name : stats
    displayName : stats

  - name : organization
    displayName : Organization
    columns :
      - name : id
      - name : type
      - name : name
        column_new_name : full_name
```

CHAPTER 6

Running olapy with Database

As we said in the previous section, Olapy uses CSV files as source type by default when using the `olapy runserver` command, so how can we work with databases ? Well, you need to provide some database information (login, password, etc. . .) to Olapy so it can connect to your database management system.

The command to run Olapy with databases is

```
olapy runserver -st=csv,db
```

Here, Olapy gets cubes from csv and database (of course if you want only database use `-st=db` . . .)

You have three possibilities to configure olapy with database:

6.1 Environnement variable

At startup, Olapy looks for an environment variable called *SQLALCHEMY_DATABASE_URI* which is the connection string that holds your database credentials and its something like:

```
SQLALCHEMY_DATABASE_URI = mysql://root:root@localhost:3306
```

To use this method, just before starting Olapy with `olapy runserver`, use the following command:

```
export SQLALCHEMY_DATABASE_URI = mysql://root:root@localhost:3306 for mac/linux

set SQLALCHEMY_DATABASE_URI = mysql://root:root@localhost:3306 for windows
```

and then start Olapy with the option `-st=csv,db` of course.

NOTE don't put the database name in the connection string, you will select the database after from Excel.

`SQLALCHEMY_DATABASE_URI = mysql://root:root@localhost:3306/my_database` -> this will not work

6.2 Database string connection

This is simple as running Olapy with the `-sa` option:

```
olapy runserver -st=csv,db -sa=mysql://root:root@localhost:3306
```

and the same rule **don't put the database name**.

6.3 Olapy config file

The third way to configure a database connection is using a file configuration named `olapy-config.yml` under `olapy-data` folder. A default/demo `olapy-config` file is created after installing olapy under `olapy-data`.

You can modify this file according to your configuration:

```
connection_string : postgresql+psycopg2://postgres:root@localhost:5432
```

take a look to SQLAlchemy documentation for more information.

CHAPTER 7

OlaPy ETL

NOTE: this part is working only with Python 3.5+.

OlaPy ETL can be used if you have an excel file (one sheet) contains all your data in order to let OlaPy make necessary transformations relative to its rules on your data.

To use OlaPy ETL, after installing OlaPy with *python setup.py install* use the following command:

```
etl --input_file_path=<EXCEL FILE PATH> --config_file=<CONFIG FILE PATH> [OPTIONAL] --
↪output_cube_path=<PATH WHERE TO GENERATE THE CUBE>
```

config_file describe how to create the cube, here an example of the configuration file, consider this excel sheet:

Count	Continent	Country	Year	Month	Day
84	America	Canada	2010	January 2010	January 1,2010
841	America	Canada	2010	January 2010	January 2,2010
2	America	United States	2010	January 2010	January 3,2010

and we want to divide it into three table, we use a configuration file like this:

```
Facts:      [Count] # just measures
Geography:  [Continent, Country]
Date:       [Year, Month, Day]
```

and you save it as yaml file (.yml).

CHAPTER 8

API Documentation

8.1 Package olapy.core.services.xmla

to import the package use:

```
import olapy.core.services.xmla
```

8.2 Package olapy.core.mdx.executor

to import the package use:

```
import olapy.core.mdx.executor
```

8.3 Package olapy.etl.etl

to import the package use:

```
import olapy.etl.etl
```

CHAPTER 9

Indices and tables

- genindex
- modindex
- search