Wendelin: data visualization and prototype of a Pandas-based Inventory API

(Click here to view the Jupyter notebook used as base to build this entire blog post)

Introduction

The Wendelin project started in the beginning of 2015 with Nexedi as consortium leader in charge of managing the development of a big data solution "made in France". Under the Wendelin umbrella there is the Movement Visualization (MOVIS) project. This project is focussed on implementing data visualization features, like pivot tables and charts, using the scientific libraries already present in the Wendelin project and adding even more to help with richer and interactive visualizations.

Motivation

Wendelin is getting more mature as the time goes and it is already being put to test with many prototypes. This time a prototype of the Inventory API was created. The Inventory API is one of the areas of ERP5 that demands the most processing power to do calculations over all the client's stock movements. These inventories also hold valuable information for their companies and they might even request many inventories with different parameters for internal research, along with many tables and charts based on this data for better visualization of important insights (like trends and cycles, for example).

Some inventories are so big that even MariaDB is taking too much time do calculate them. The current implementation of the Inventory API relies on a very complex SQL query on the stock table. This stock table can have millions of rows, depending on the client, and this makes the query very slow.

Talking about visualization of data, this prototype includes integration with PivotTableJs. This tool turns any dataset into a summary table and adds a 2-dimensional drag and drop interface to allow any user to manipulate this summary table, turning it into something very similar to what is found in older versions of Microsoft Excel. With included addons not only a table can be rendered, but various kinds of charts, turning the pivot table into a pivot chart.

In the technical point of view, we had MariaDB as a single point of failure. It's very hard to decentralize and paralelize relational databases in a transaction system. With Wendelin all the stock data that was inside MariaDB can be moved to storage system created specifically for that purpose: NEO is a distributed, redundant and transactional storage designed to be an alternative to ZEO and FileStorage.

What is it?

It's an optional replacement or complement to the original Inventory API that's on early development and will be published soon. It's built on top of Wendelin's Data Array to allow processing of volumes of data much bigger than the available memory. Thanks to Wendelin's Data Array, which is compatible with NumPy's ndarray, we also take advantage of Pandas to create a Data Frame with all the data and have advanced indexing and filtering features.

How to test it?

There are 2 steps before the user can actually use this prototype Pandas-based Inventory API: create the Data Array with stock movement data and add the category information to each stock movement. After these two steps the array is ready for being filtered and the user wishes.

First step: create a Data Array with stock movements information

The first thing to do is actually create and fill a Data Array with data from the stock table. To help import the data a class was created to transform any ERP5 query into a Data Array with an equivalent data type. The SaleOrderModule_zGetQuantityList object is a simple ZSQLMethod that gets and all the rows in the stock table, as simple as:

```
SELECT * FROM stock;
```

In [17]:

```
data = context.sale_order_module.SaleOrderModule_zGetQuantityList()
context.Base_convertResultsToBigArray(
  data
```

```
    reference='WendelinJupyter'
)
```

<Data Array at /erp5/data_array_module/222>

Second step: import category information

Now it's time to import the category information from each stock movement to the Data Array. This is necessary in situations like, for example, to get only the sock movements of a resource that belongs to a specific category. The method Base_fillPandasInventoryCategoryList will take care of querying the catalog in the most efficient way possible to get the category information needed and store it in the Data Array.

In [14]:

```
context.Base_fillPandasInventoryCategoryList(
    'WendelinJupyter',
)
```

<wendelin.bigarray.array_zodb.ZBigArray object at 0x7f5de6d49e50>

Third step: filtering stock movements

Finally the Data Frame is ready to be filtered. Compatibility with the original Inventory API was kept in my mind while developing the prototype: both functions receive the same parameters. The only difference in the prototype is that it returns a Pandas.DataFrame instead of ERP5 objects. With the Data Frame the developer can do further processing to improve the visualization of the data: more filtering, (multilevel) indexing, grouping, sorting and etc.

In [20]:

```
data_frame = context.Base_getInventoryDataFrame(
    is_accountable=True,
    omit_input=True,
    resource_uid=1060068,
    section_uid=1064700,
    from_date='2015-08-11',
    to_date='2015-09-13',
    simulation_state='planned',
    resource_product_line_uid='1048885'
)
data_frame.head()
```

```
         date  explanation_uid  function_uid  funding_uid  is_accountable  \
2  2015-08-12          1112292             0            0               1
32 2015-09-08          1112238             0            0               1
42 2015-09-07          1112240             0            0               1
122 2015-09-03         1112248             0            0               1
132 2015-09-09         1112236             0            0               1

     is_cancellation mirror_date  mirror_node_uid  mirror_section_uid  \
2                  0  2015-08-12          1049627             1049627
32                 0  2015-09-08          1049627             1049627
42                 0  2015-09-07          1049627             1049627
122                0  2015-09-03          1049627             1049627
132                0  2015-09-09          1049627             1049627

     node_uid                    ...                     \
2     1064700                    ...
32    1064700                    ...
42    1064700                    ...
122   1064700                    ...
132   1064700                    ...

              resource_category           node_category  payment_category  \
2    1048885,15668,1048717,15663  1048649,1049113,1048690                 0
32   1048885,15668,1048717,15663  1048649,1049113,1048690                 0
42   1048885,15668,1048717,15663  1048649,1049113,1048690                 0
122  1048885,15668,1048717,15663  1048649,1049113,1048690                 0
132  1048885,15668,1048717,15663  1048649,1049113,1048690                 0

           section_category  mirror_section_category  function_category  \
2    1048649,1049113,1048690  1046951,1049623,1049113                  0
32   1048649,1049113,1048690  1046951,1049623,1049113                  0
42   1048649,1049113,1048690  1046951,1049623,1049113                  0
122  1048649,1049113,1048690  1046951,1049623,1049113                  0
```

```
132  1048649,1049113,1048690  1046951,1049623,1049113                0

    project_category  funding_category payment_request_category  \
2                  0                 0                       0
32                 0                 0                       0
42                 0                 0                       0
122                0                 0                       0
132                0                 0                       0

                                movement_category
2    1048795,1046951,1049627,1064700,1049627,104863...
32   1048795,1046951,1049627,1064700,1049627,104863...
42   1048795,1046951,1049627,1064700,1049627,104863...
122  1048795,1046951,1049627,1064700,1049627,104863...
132  1048795,1046951,1049627,1064700,1049627,104863...

[5 rows x 33 columns]
```

PivotTableJs integration

PivotTableJs is here for the rescue of users and developers who wants to get fast insights from their data. All they have to do is: put all the data in a Pandas.DataFrame and use the external method Base_erp5PivotTableUI. This method is integrated with the ERP5 Jupyter kernel and will render the pivot table user interface with the input data. You can interact with demo pivot table generated by the code below.

In [427]:

```
import random
import pandas as pd


df = context.Base_getInventoryDataFrame(
    is_accountable=True,
    omit_input=True,
    simulation_state='planned',
    resource_uid=1046951
)

# bar chart: sum of quantity vs date by resource_uid/node_uid
columns = ['date', 'resource_uid', 'quantity', 'quantity', 'node_uid']
columns_to_delete = df.columns - columns
for column in columns_to_delete:
    df.drop(column, axis=1, inplace=True)
context.Base_erp5PivotTableUI(df, 'https://localhost:2202/erp5')
```

This gif shows a demonstraction of the PivotTableJs UI running inside a Jupyter notebook from a Wendelin instance. Fed with planned stock movements of the resource with UID 1046951 that are accountable and omitting all the input movements. Then columns are dragged and dropped to organise data in a meaninful way: node_uid and resource_uid are moved to the rows and date to the columns and now there is a "output movements of resource_uid at node_uid over time" table. Next step is modifying the aggregation function to the sum of the quantity column to get the total output of the resource at the given nodes per day. After the data is organised the source is edited: only dates between 2015-08-13 and 2015-09-09 are taken into account and the representation is changed from a simple table to a beautiful line chart.

Conclusion

Wendelin is a project in constant evolution with a simple objective: it aims to bring all the tools known in the scientific community along with their high performance to the ERP5 platform. Scikit-learn, statsmodels and Pandas are already integrated and even more integrations are on the works. Everything backed by Wendelin's NumPy-compatible array to enable you to extend your computation beyond the memory limits of servers without losing the valuable insights achievable through all the most famous scientific libraries in the Python community.