Creating a SQL Chart

Creating a SQL Chart

opCharts has the ability to create custom charts from data contained in external SQL databases. SQL charts are not currently supported in the chart creator. To make an SQL chart you will need to edit /usr/local/omk/conf/charts/charts.json. The default configuration comes with several examples for SQL data sets. The layout of an SQL chart is this:

```
{
         "parameters" : {
            "time_period" : "time_difference",
            "end_date" : "7-Mar-2014 14:03:01",
            "start_date" : "7-Mar-2014 13:48:01"
         },
         "datasets" : [
            {
               "parameters" : {
                  "aggregation_function" : 1,
                  "lineType" : "column",
                  "query" : "select * from nodes",
                  "value_column" : "sum",
                  "axis" : "0",
                  "groupby" : [
                     "group_column"
                  ]
               },
               "data_source" : "local_mysql",
               "options" : {
                  "datasetTitle" : "Groups"
               },
               "name" : "sqlquery_dataset",
               "model" : "sql_query"
            }
         ],
         "options" : {
            "titleText" : ""
         },
         "name" : "SQL Test",
         "model_view" : "non-time-chart"
      },
```

Dataset:

Looking at an individual dataset will help us understand:

```
 {
               "parameters" : {
                  "query" : "select * from nodes",
                  "groupby" : [
                     "group_column"
                  ],
                  "aggregation_function" : 1,
                  "value_column" : "unused",
                  "lineType" : "column",
                  "axis" : "0",
               },
               "data_source" : "local_mysql",
               "options" : {
                  "datasetTitle" : "Groups"
               },
               "name" : "sqlquery_dataset",
               "model" : "sql_query"
}
```

Breaking this down:

From the data_source "local_mysql", run the "sql_query" that is "select * from nodes", group the result by "group_column" and COUNT the number or rows in each group. Display the results in a column graph on the 0 axis.

query:

The SQL to run. This SQL can contain almost anything you would like.

NOTES:

- do not use single quotes, please use escaped double quotes, eg: \"value\" instead of 'value'
- do not use SELECT *, specify the columns you need, * will not work.

query subsitutions

Currently supported subsitutions are:

query token	substituted value
user.customer	The customer name of the current user is assigned to, if the user does not have a customer the query can fail. Administrators have an option to set this value in the advanced menu.
time.start	unix timestamp of the starting time selected from the advanced menu, default value is 15min before now
time.end	unix timestamp of the ending time selected from the advanced menu, default value is now

For the timestamps the time column being queried may need to be converted into a unix timestamp to make a comparison valid. MySQL's timestamp function is UNIX_TIMESTAMP(timecolumn)

Example: SELECT * FROM interface WHERE UNIX_TIMESTAMP(lastUpdate) >= time.start AND UNIX_TIMESTAMP(lastUpdate) < time.end AND customerName = user.customer

groupby:

If the data returned from the SQL statement needs to be grouped (for summing or counting, works much like SQL GROUP BY) use this field to specify the group, as an array, order matters. This works in tandem with the aggregation function to produce results. Just like in an SQL GROUP BY each column requires a function to aggregate it's result.

value_column:

The column in the dataset to run the aggregation function on. If you are not sure what the column name will be, use SELECT column AS some_unique_name. "some_unique_name" can then be used as the value column

aggregation_function:

PASSTHROUGH => 0, COUNT => 1, SUM => 2, MAX => 3, MIN => 4, AVG => 5

This function should be run on each entry in the "group" to produce one row. The value_column will be read and have this function run on it, **unless** PASSTHROUGH or COUNT are used. For PASSTHROUGH no grouping or aggregation are done. For COUNT the number of rows in each group is tallied. both PASSTHROUGH or COUNT ignore the value_column setting.

lineType:

column is most likely, other options are available, open the chart creator (go to charts and click new chart), there is a drop down with the options.

name:

this is of little consequence, functionally not used at all right now but a way for you to name the dataset for later recognition.

model (formerly 'type'):

must be sql_query for the above parameters to work.

Back to the chart

The name is the unique name that identifies this chart when using it elsewhere (like creating a dashboard). What is model_view for?

model_view (also formerly 'type')

```
"model_view" : "non-time-chart"

"model_view" : "graph"
```

2 options exist for this. "non-time-chart" means the data will not be an "over time" graph, but a snapshot of the data at a specific time. This is the most likely candidate for an SQL chart. "graph" is a data over time view, the time base is in unix epoc, the SQL query must return the time column in this format and "time_column" must be specified in the dataset telling it what the column is that holds the time value.