Diagram widget configuration

Get started

This information is also applicable to the field Visual extra: SQL: Diagram query.

This site only describes the specifics of the diagram widgets, for general widget configuration click here.

This article will only cover the newest version of graphs implemented.

To enable these, toggle the Policy useGoogleChart to false and useNewChart to true.

As of writing (version 8813) these renderings are supported.

- Circle
- Doughnut
- Bar vertical (Columns)
- Bar horizontal
- Area
- Line
- Area (stacked)

Data and Internationalization

An SQL Query may extract almost anything from the database.

This process ignores the security restrictions in the system, so you will have to implement them yourself.

Structure

The structure is the same for all graphs.

The first row of data will be used for headers.

The column named "Title" will be used for titles. On circular diagrams that is the labels, next to the chart, and on xy-graphs that is the labels on the x axis.

The rest of the headers will be used when hovering over the graph, to give information about the data. On xy-graphs it will also be displayed as labels.

The rest of the lines in the data will be added as datapoints in the graph.

If the datapoint is a number, the graph will auto-scale to match the data.

Strings are also supported, but the outcome is unknown.

If a datapoint is null, the graph will bridge the gap, if it is an xy-line-graph, see sample here.

Sample pie/doughnut-graph

Title	TitleEnglish	This moth	Last month
GOOG	Alphabet	100.23	25.23
TSLA	Tesla	10.23	20.23
AAPL	Apple	50.23	50.23

Sample line-graph

Title	TitleEnglish	Alphabet	Tesla	Apple
THIS	This Month	100.23	10.23	50.23
LAST	Last month	25.23	20.23	50.23

Multiply y-axis

Multiple y-axis are supported as of version 9390.

To use this feature, prepend the name of the new axis to the data that should be on that axis, eg `y1|Apple`.

The default y-axis is named `y`. If no axis is specified, this will be used.

You can specify as many axis as ChartJS supports.

Post-Parsing data

There is two possible post-processing methods available. They can't be combined.

Transpose

Start the query with a line containing: `-- @TRANSPOSE`

This transposes (inverts columns and rows) the data fetched by the query, before handing it of to the chart-engine.

Parse

Start the query with a line containing: `-- @PARSEDATASET`

This expects the query to generate data in the following format (headers are optional), and removes the support for links.

X-axis	Group by	Y-axis
2024	Sales	2000
2024	Budget	2000
2025	Sales	2500
2025	Budget	4000
2026	Budget	5000

Internationalization

Add extra columns named "Title", followed by the name of the language, eg *TitleEnglish*.

If a column with this name is found, it will be used, otherwise the grath will fallback to the "Title" column.

If no title-column is found, the first column will be used, for compatibility reasons.

Adding functionality

It is possible to add some simple funktionality to charts, by adding links.

This feature is limited to one link pr x-axis value.

To add a link, add a column named "Link" to the query.

Whatever this column contains will become a link, that can be navigated to, when that x-value is clicked.

Graph defaults

The graphs all reference the js object *tsChartDefaultOptions*, with a couple of tweaks.

To see the defaults, open the terminal in you browser, on a page that has a graph and type the name of the object.

Specifik defaults

These can't be overwritten.

Area stacked

```
options.scales = {
  y: {
    stacked: true
  }
}
```

Area stacked, area and line

```
options.interaction = {
  intersect: false,
  mode: 'index',
}
```

Bars

```
options.indexAxis = 'y'
```

Stacked bars and cols

```
options.scales = {
  x: {
    stacked: true,
  },
  y: {
```

```
    stacked: true,
  },
}
options.plugins = {
  tooltip: {
    callbacks: {
      footer: items => {
        return 'Total: ' + items.reduce((a, b) => a + b.raw, 0)
      },
    },
  },
}
```

Overwriting defaults

To overwrite the default of all charts:

Define an object in js, named tsChartOverwriteOptions.

Set it based on the documentation fund here.

To overwrite the options for a specific chart:

Define an object in js, named tsChartOverwriteOptions.

Add an object with an index matching the chartID.

Set it based on the documentation fund here.

To test your overwrite:

From the console in the browser call *tsChartOptions* with the only parameter being the chartID.

The return will be the final options, that will be used for that chart.

Sample

Force a graph (graph with ID 6) to always have y-axis start at 0.

```
let tsChartOverwriteOptions = {
  chart6: {
    scales: {
      y: {
         beginAtZero: true
      },
    },
  },
}
```

Change the color of the labels to red (graph with ID 4).

```
let tsChartOverwriteOptions = {
  chart4: {
    plugins: {
      legend: {
        labels: {
          color: "rgba(255, 0, 0, 0.9)",
        },
      },
    },
  },
}
```

Colors

The colors used, can be overwritten in the Policy *diagramChartJSColors.*

The structure of the policy is very strict.

The policy has to be formatted as a js array of strings that represent an rgb color.

Thus: It has to start with `[` and and with `,]`.

Each color has to be formatted as an rgb color and wrapped with `'`, and separated by `,` eg `'rgb(255, 255, 255)',` for a white color