+++ title = "PostgreSQL" description = "Guide for using PostgreSQL in Grafana" keywords = ["grafana", "postgresql", "guide"] aliases = ["/docs/grafana/latest/features/datasources/postgres/"] weight = 1200 +++

PostgreSQL data source

Grafana ships with a built-in PostgreSQL data source plugin that allows you to query and visualize data from a PostgreSQL compatible database. This topic explains options, variables, querying, and other options specific to this data source. For instructions about how to add a data source to Grafana, refer to [Add a data source]({{< relref "add-a-data-source.md" >}}). Only users with the organization admin role can add data sources.

PostgreSQL settings

To access PostgreSQL settings, hover your mouse over the **Configuration** (gear) icon, then click **Data Sources**, and then click the PostgreSQL data source.

Name	Description
`Name`	The data source name. This is how you refer to the data source in panels and queries.
`Default`	Default data source means that it will be pre-selected for new panels.
`Host`	The IP address/hostname and optional port of your PostgreSQL instance. *Do not* include the database name. The connection string for connecting to Postgres will not be correct and it may cause errors.
`Database`	Name of your PostgreSQL database.
`User`	Database user's login/username
`Password`	Database user's password
`SSL Mode`	Determines whether or with what priority a secure SSL TCP/IP connection will be negotiated with the server. When SSL Mode is disabled, SSL Method and Auth Details would not be visible.
`SSL Auth Details Method`	Determines whether the SSL Auth details will be configured as a file path or file content. Grafana v7.5+
`SSL Auth Details Value`	File path or file content of SSL root certificate, client certificate and client key
`Max open`	The maximum number of open connections to the database, default `unlimited` (Grafana v5.4+).
`Max idle`	The maximum number of connections in the idle connection pool, default `2` (Grafana v5.4+).
`Max lifetime`	The maximum amount of time in seconds a connection may be reused, default `14400`/4 hours (Grafana v5.4+).
`Version`	Determines which functions are available in the query builder (only available in Grafana 5.3+).
`TimescaleDB`	A time-series database built as a PostgreSQL extension. When enabled, Grafana uses `time_bucket` in the `$__timeGroup` macro to display TimescaleDB specific aggregate functions in the query builder (only available in Grafana 5.3+).

Min time interval

A lower limit for the [$__interval]({{< relref "../variables/variable-types/global-variables/#__interval" >}}) and [$__interval_ms]({{< relref "../variables/variable-types/global-variables/#__interval_ms" >}}) variables. Recommended to be set to write frequency, for example `1m` if your data is written every minute. This option can also be overridden/configured in a dashboard panel under data source options. It's important to note that this value **needs** to be formatted as a number followed by a valid time identifier, e.g. `1m` (1 minute) or `30s` (30 seconds). The following time identifiers are supported:

Identifier	Description
`y`	year
`M`	month
`w`	week
`d`	day
`h`	hour
`m`	minute
`s`	second
`ms`	millisecond

Database user permissions (Important!)

The database user you specify when you add the data source should only be granted SELECT permissions on the specified database and tables you want to query. Grafana does not validate that the query is safe. The query could include any SQL statement. For example, statements like `DELETE FROM user;` and `DROP TABLE user;` would be executed. To protect against this we **highly** recommend you create a specific PostgreSQL user with restricted permissions.

Example:

```
CREATE USER grafanareader WITH PASSWORD 'password';
GRANT USAGE ON SCHEMA schema TO grafanareader;
GRANT SELECT ON schema.table TO grafanareader;
```

Make sure the user does not get any unwanted privileges from the public role.

Query editor

{{< figure src="/static/img/docs/v53/postgres_query_still.png" class="docs-image--no-shadow" animated-gif="/static/img/docs/v53/postgres_query.gif" >}}

You find the PostgreSQL query editor in the metrics tab in Graph or Singlestat panel's edit mode. You enter edit mode by clicking the panel title, then edit.

The query editor has a link named `Generated SQL` that shows up after a query has been executed, while in panel edit mode. Click on it and it will expand and show the raw interpolated SQL string that was executed.

Select table, time column and metric column (FROM)

When you enter edit mode for the first time or add a new query Grafana will try to prefill the query builder with the first table that has a timestamp column and a numeric column.

In the FROM field, Grafana will suggest tables that are in the `search_path` of the database user. To select a table or view not in your `search_path` you can manually enter a fully qualified name (schema.table) like `public.metrics`.

The Time column field refers to the name of the column holding your time values. Selecting a value for the Metric column field is optional. If a value is selected, the Metric column field will be used as the series name.

The metric column suggestions will only contain columns with a text datatype (char,varchar,text). If you want to use a column with a different datatype as metric column you may enter the column name with a cast: `ip::text`. You may also enter arbitrary SQL expressions in the metric column field that evaluate to a text datatype like `hostname || ' ' || container_name`.

Columns, window, and aggregation functions (SELECT)

In the `SELECT` row you can specify what columns and functions you want to use. In the column field you may write arbitrary expressions instead of a column name like `column1 * column2 / column3`.

The available functions in the query editor depend on the PostgreSQL version you selected when configuring the data source. If you use aggregate functions you need to group your resultset. The editor will automatically add a `GROUP BY time` if you add an aggregate function.

The editor tries to simplify and unify this part of the query. For example:

The above will generate the following PostgreSQL `SELECT` clause:

```
avg(tx_bytes) OVER (ORDER BY "time" ROWS 5 PRECEDING) AS "tx_bytes"
```

You may add further value columns by clicking the plus button and selecting `Column` from the menu. Multiple value columns will be plotted as separate series in the graph panel.

Filter data (WHERE)

To add a filter click the plus icon to the right of the `WHERE` condition. You can remove filters by clicking on the filter and selecting `Remove`. A filter for the current selected timerange is automatically added to new queries.

Group by

To group by time or any other columns click the plus icon at the end of the GROUP BY row. The suggestion dropdown will only show text columns of your currently selected table but you may manually enter any column. You can remove the group by clicking on the item and then selecting `Remove`.

If you add any grouping, all selected columns need to have an aggregate function applied. The query builder will automatically add aggregate functions to all columns without aggregate functions when you add groupings.

Gap filling

Grafana can fill in missing values when you group by time. The time function accepts two arguments. The first argument is the time window that you would like to group by, and the second argument is the value you want Grafana to fill missing items with.

Text editor mode (RAW)

You can switch to the raw query editor mode by clicking the hamburger icon and selecting `Switch editor mode` or by clicking `Edit SQL` below the query.

> *If you use the raw query editor, be sure your query at minimum has `ORDER BY time` and a filter on the returned time range.*

Macros

Macros can be used within a query to simplify syntax and allow for dynamic parts.

Macro example	Description
`$__time(dateColumn)`	Will be replaced by an expression to convert to a UNIX timestamp and rename the column to `time_sec`. For example, *UNIX_TIMESTAMP(dateColumn) as time_sec*
`$__timeEpoch(dateColumn)`	Will be replaced by an expression to convert to a UNIX timestamp and rename the column to `time_sec`. For example, *UNIX_TIMESTAMP(dateColumn) as time_sec*
`$__timeFilter(dateColumn)`	Will be replaced by a time range filter using the specified column name. For example, *dateColumn BETWEEN FROM_UNIXTIME(1494410783) AND FROM_UNIXTIME(1494410983)*
`$__timeFrom()`	Will be replaced by the start of the currently active time selection. For example, *FROM_UNIXTIME(1494410783)*
`$__timeTo()`	Will be replaced by the end of the currently active time selection. For example, *FROM_UNIXTIME(1494410983)*
`$__timeGroup(dateColumn,'5m')`	Will be replaced by an expression usable in GROUP BY clause. For example, *cast(cast(UNIX_TIMESTAMP(dateColumn)/(300) as signed)*300 as signed),*
`$__timeGroup(dateColumn,'5m', 0)`	Same as above but with a fill parameter so missing points in that series will be added by grafana and 0 will be used as value.
`$__timeGroup(dateColumn,'5m', NULL)`	Same as above but NULL will be used as value for missing points.
`$__timeGroup(dateColumn,'5m', previous)`	Same as above but the previous value in that series will be used as fill value if no value has been seen yet NULL will be used (only available in Grafana 5.3+).
`$__timeGroupAlias(dateColumn,'5m')`	Will be replaced identical to $__timeGroup but with an added column alias (only available in Grafana 5.3+).
`$__unixEpochFilter(dateColumn)`	Will be replaced by a time range filter using the specified

	column name with times represented as Unix timestamp. For example, *dateColumn > 1494410783 AND dateColumn < 1494497183*
`$__unixEpochFrom()`	Will be replaced by the start of the currently active time selection as Unix timestamp. For example, *1494410783*
`$__unixEpochTo()`	Will be replaced by the end of the currently active time selection as Unix timestamp. For example, *1494497183*
`$__unixEpochNanoFilter(dateColumn)`	Will be replaced by a time range filter using the specified column name with times represented as nanosecond timestamp. For example, *dateColumn > 1494410783152415214 AND dateColumn < 1494497183142514872*
`$__unixEpochNanoFrom()`	Will be replaced by the start of the currently active time selection as nanosecond timestamp. For example, *1494410783152415214*
`$__unixEpochNanoTo()`	Will be replaced by the end of the currently active time selection as nanosecond timestamp. For example, *1494497183142514872*
`$__unixEpochGroup(dateColumn,'5m', [fillmode])`	Same as $__timeGroup but for times stored as Unix timestamp (only available in Grafana 5.3+).
`$__unixEpochGroupAlias(dateColumn,'5m', [fillmode])`	Same as above but also adds a column alias (only available in Grafana 5.3+).

We plan to add many more macros. If you have suggestions for what macros you would like to see, please open an issue in our GitHub repo.

Table queries

If the `Format as` query option is set to `Table` then you can basically do any type of SQL query. The table panel will automatically show the results of whatever columns and rows your query returns.

Query editor with example query:

The query:

```
SELECT
  title as "Title",
  "user".login as "Created By",
  dashboard.created as "Created On"
FROM dashboard
INNER JOIN "user" on "user".id = dashboard.created_by
WHERE $__timeFilter(dashboard.created)
```

You can control the name of the Table panel columns by using regular `as` SQL column selection syntax.

The resulting table panel:

 postgres table

Time series queries

If you set Format as to *Time series*, then the query must have a column named time that returns either a SQL datetime or any numeric datatype representing Unix epoch in seconds. In addition, result sets of time series queries must be sorted by time for panels to properly visualize the result.

A time series query result is returned in a [wide data frame format]({{< relref "../developers/plugins/data-frames.md#wide-format" >}}). Any column except time or of type string transforms into value fields in the data frame query result. Any string column transforms into field labels in the data frame query result.

> *For backward compatibility, there's an exception to the above rule for queries that return three columns including a string column named metric. Instead of transforming the metric column into field labels, it becomes the field name, and then the series name is formatted as the value of the metric column. See the example with the metric column below.*

You can optionally customize the default series name formatting using instructions in [Reference: Standard field definitions]({{< relref "../panels/reference-standard-field-definitions.md#display-name" >}}).

Example with `metric` column:

```
SELECT
  $__timeGroup("time_date_time",'5m'),
  min("value_double"),
  'min' as metric
FROM test_data
WHERE $__timeFilter("time_date_time")
GROUP BY time
ORDER BY time
```

Data frame result:

```
+---------------------+-----------------+
| Name: time          | Name: min       |
| Labels:             | Labels:         |
| Type: []time.Time   | Type: []float64 |
+---------------------+-----------------+
| 2020-01-02 03:05:00 | 3               |
| 2020-01-02 03:10:00 | 6               |
+---------------------+-----------------+
```

Example using the fill parameter in the $__timeGroup macro to convert null values to be zero instead:

```
SELECT
  $__timeGroup("createdAt",'5m',0),
  sum(value) as value,
  hostname
```

```
FROM test_data
WHERE
  $__timeFilter("createdAt")
GROUP BY time, hostname
ORDER BY time
```

Given the data frame result in the following example and using the graph panel, you will get two series named *value 10.0.1.1* and *value 10.0.1.2*. To render the series with a name of *10.0.1.1* and *10.0.1.2* , use a [Reference: Standard field definitions]({{< relref "../panels/reference-standard-field-definitions.md#display-name" >}}) display value of `${__field.labels.hostname}` .

Data frame result:

```
+---------------------+---------------------------+---------------------------+
| Name: time          | Name: value               | Name: value               |
| Labels:             | Labels: hostname=10.0.1.1 | Labels: hostname=10.0.1.2 |
| Type: []time.Time   | Type: []float64           | Type: []float64           |
+---------------------+---------------------------+---------------------------+
| 2020-01-02 03:05:00 | 3                         | 4                         |
| 2020-01-02 03:10:00 | 6                         | 7                         |
+---------------------+---------------------------+---------------------------+
```

Example with multiple columns:

```
SELECT
  $__timeGroup("time_date_time",'5m'),
  min("value_double") as "min_value",
  max("value_double") as "max_value"
FROM test_data
WHERE $__timeFilter("time_date_time")
GROUP BY time
ORDER BY time
```

Data frame result:

```
+---------------------+-----------------+-----------------+
| Name: time          | Name: min_value | Name: max_value |
| Labels:             | Labels:         | Labels:         |
| Type: []time.Time   | Type: []float64 | Type: []float64 |
+---------------------+-----------------+-----------------+
| 2020-01-02 03:04:00 | 3               | 4               |
| 2020-01-02 03:05:00 | 6               | 7               |
+---------------------+-----------------+-----------------+
```

Templating

Instead of hard-coding things like server, application and sensor name in your metric queries you can use variables in their place. Variables are shown as dropdown select boxes at the top of the dashboard. These dropdowns make it easy to change the data being displayed in your dashboard.

Refer to [Templates and variables]({{< relref "../variables/_index.md" >}}) for an introduction to the templating feature and the different types of template variables.

Query variable

If you add a template variable of the type `Query`, you can write a PostgreSQL query that can return things like measurement names, key names or key values that are shown as a dropdown select box.

For example, you can have a variable that contains all values for the `hostname` column in a table if you specify a query like this in the templating variable *Query* setting.

```
SELECT hostname FROM host
```

A query can return multiple columns and Grafana will automatically create a list from them. For example, the query below will return a list with values from `hostname` and `hostname2`.

```
SELECT host.hostname, other_host.hostname2 FROM host JOIN other_host ON host.city =
other_host.city
```

To use time range dependent macros like `$__timeFilter(column)` in your query the refresh mode of the template variable needs to be set to *On Time Range Change*.

```
SELECT event_name FROM event_log WHERE $__timeFilter(time_column)
```

Another option is a query that can create a key/value variable. The query should return two columns that are named `__text` and `__value`. The `__text` column value should be unique (if it is not unique then the first value is used). The options in the dropdown will have a text and value that allows you to have a friendly name as text and an id as the value. An example query with `hostname` as the text and `id` as the value:

```
SELECT hostname AS __text, id AS __value FROM host
```

You can also create nested variables. Using a variable named `region`, you could have the hosts variable only show hosts from the current selected region with a query like this (if `region` is a multi-value variable then use the `IN` comparison operator rather than `=` to match against multiple values):

```
SELECT hostname FROM host  WHERE region IN($region)
```

Using `__searchFilter` to filter results in Query Variable

> *Available from Grafana 6.5 and above*

Using `__searchFilter` in the query field will filter the query result based on what the user types in the dropdown select box. When nothing has been entered by the user the default value for `__searchFilter` is `%`.

> *Important that you surround the `__searchFilter` expression with quotes as Grafana does not do this for you.*

The example below shows how to use `__searchFilter` as part of the query field to enable searching for `hostname` while the user types in the dropdown select box.

Query

```
SELECT hostname FROM my_host  WHERE hostname LIKE '$__searchFilter'
```

Using Variables in Queries

From Grafana 4.3.0 to 4.6.0, template variables are always quoted automatically. If your template variables are strings, do not wrap them in quotes in where clauses.

From Grafana 4.7.0, template variable values are only quoted when the template variable is a `multi-value`.

If the variable is a multi-value variable then use the `IN` comparison operator rather than `=` to match against multiple values.

There are two syntaxes:

`$<varname>` Example with a template variable named `hostname`:

```
SELECT
  atimestamp as time,
  aint as value
FROM table
WHERE $__timeFilter(atimestamp) and hostname in($hostname)
ORDER BY atimestamp ASC
```

`[[varname]]` Example with a template variable named `hostname`:

```
SELECT
  atimestamp as time,
  aint as value
FROM table
WHERE $__timeFilter(atimestamp) and hostname in([[hostname]])
ORDER BY atimestamp ASC
```

Disabling quoting for multi-value variables

Grafana automatically creates a quoted, comma-separated string for multi-value variables. For example: if `server01` and `server02` are selected then it will be formatted as: `'server01', 'server02'`. To disable quoting, use the csv formatting option for variables:

`${servers:csv}`

Read more about variable formatting options in the [Variables]({{< relref "../variables/_index.md#advanced-formatting-options" >}}) documentation.

Annotations

[Annotations]({{< relref "../dashboards/annotations.md" >}}) allow you to overlay rich event information on top of graphs. You add annotation queries via the Dashboard menu / Annotations view.

Example query using time column with epoch values:

```
SELECT
  epoch_time as time,
  metric1 as text,
  concat_ws(', ', metric1::text, metric2::text) as tags
FROM
  public.test_data
WHERE
  $__unixEpochFilter(epoch_time)
```

Example region query using time and timeend columns with epoch values:

> *Only available in Grafana v6.6+.*

```
SELECT
  epoch_time as time,
  epoch_time_end as timeend,
  metric1 as text,
  concat_ws(', ', metric1::text, metric2::text) as tags
FROM
  public.test_data
WHERE
  $__unixEpochFilter(epoch_time)
```

Example query using time column of native SQL date/time data type:

```
SELECT
  native_date_time as time,
  metric1 as text,
  concat_ws(', ', metric1::text, metric2::text) as tags
FROM
  public.test_data
WHERE
  $__timeFilter(native_date_time)
```

Name	Description
`time`	The name of the date/time field. Could be a column with a native SQL date/time data type or epoch value.
`timeend`	Optional name of the end date/time field. Could be a column with a native SQL date/time data type or epoch value. (Grafana v6.6+)
`text`	Event description field.
`tags`	Optional field name to use for event tags as a comma separated string.

Alerting

Time series queries should work in alerting conditions. Table formatted queries are not yet supported in alert rule conditions.

Configure the data source with provisioning

It's now possible to configure data sources using config files with Grafana's provisioning system. You can read more about how it works and all the settings you can set for data sources on the [provisioning docs page]({{< relref "../administration/provisioning/#datasources" >}})

Here are some provisioning examples for this data source.

```
apiVersion: 1

datasources:
  - name: Postgres
    type: postgres
    url: localhost:5432
    database: grafana
    user: grafana
    secureJsonData:
      password: 'Password!'
    jsonData:
      sslmode: 'disable' # disable/require/verify-ca/verify-full
      maxOpenConns: 0 # Grafana v5.4+
      maxIdleConns: 2 # Grafana v5.4+
      connMaxLifetime: 14400 # Grafana v5.4+
      postgresVersion: 903 # 903=9.3, 904=9.4, 905=9.5, 906=9.6, 1000=10
      timescaledb: false
```

> ***Note:*** *In the above code, the* `postgresVersion` *value of* `10` *refers to version PotgreSQL 10 and above.*

If you encounter metric request errors or other issues:

- Make sure your data source YAML file parameters exactly match the example. This includes parameter names and use of quotation marks.
- Make sure the `database` name is not included in the `url`.