+++ title = "Microsoft SQL Server" description = "Guide for using Microsoft SQL Server in Grafana" keywords = ["grafana", "MSSQL", "Microsoft", "SQL", "guide", "Azure SQL Database"] aliases = ["/docs/grafana/latest/features/datasources/mssql/"] weight = 900 +++

Using Microsoft SQL Server in Grafana

Grafana ships with a built-in Microsoft SQL Server (MS SQL) data source plugin that allows you to query and visualize data from any Microsoft SQL Server 2005 or newer, including Microsoft Azure SQL Database. This topic explains options, variables, querying, and other options specific to the MS SQL data source. Refer to [Add a data source]({{< relref "add-a-data-source.md" >}}) for instructions on how to add a data source to Grafana. Only users with the organization admin role can add data sources.

Data source options

To access data source settings, hover your mouse over the **Configuration** (gear) icon, then click **Data Sources**, and then click the 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 MS SQL instance. If you omit the port, then the driver default is used (0). You can specify multiple connection properties such as ApplicationIntent using ';' character to separate each property.
`Database`	Name of your MS SQL database.
`Authentication`	Authentication mode. Either using SQL Server Authentication or Windows Authentication (single sign on for Windows users).
`User`	Database user's login/username
`Password`	Database user's password
`Encrypt`	This option determines whether or to which extent a secure SSL TCP/IP connection will be negotiated with the server, default `false`.
`Max open`	The maximum number of open connections to the database, default `unlimited`.
`Max idle`	The maximum number of connections in the idle connection pool, default `2`.
`Max lifetime`	The maximum amount of time in seconds a connection may be reused, default `14400`/4 hours.

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

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 MS SQL user with restricted permissions.

Example:

```
CREATE USER grafanareader WITH PASSWORD 'password'
GRANT SELECT ON dbo.YourTable3 TO grafanareader
```

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

Known Issues

If you're using an older version of Microsoft SQL Server like 2008 and 2008R2 you may need to disable encryption to be able to connect. If possible, we recommend you to use the latest service pack available for optimal compatibility.

Query Editor

{{< figure src="/static/img/docs/v51/mssql_query_editor.png" class="docs-image--no-shadow" >}}

You will find the MSSQL query editor in the metrics tab in Graph, Singlestat or Table panel's edit mode. You enter edit mode by clicking the panel title, then edit. The editor allows you to define a SQL query to select data to be visualized.

1. Select *Format as* `Time series` (for use in Graph or Singlestat panel's among others) or `Table` (for use in Table panel among others).
2. This is the actual editor where you write your SQL queries.
3. Show help section for MSSQL below the query editor.
4. Show actual executed SQL query. Will be available first after a successful query has been executed.
5. Add an additional query where an additional query editor will be displayed.

Macros

To simplify syntax and to allow for dynamic parts, like date range filters, the query can contain macros.

Macro example	Description
`$__time(dateColumn)`	Will be replaced by an expression to rename the column to *time*. For example, *dateColumn as time*
`$__timeEpoch(dateColumn)`	Will be replaced by an expression to convert a DATETIME column type to Unix timestamp and rename it to *time*. For example, *DATEDIFF(second, '1970-01-01', dateColumn) AS time*
`$__timeFilter(dateColumn)`	Will be replaced by a time range filter using the specified column name. For example, *dateColumn BETWEEN '2017-04-21T05:01:17Z' AND '2017-04-21T05:06:17Z'*
`$__timeFrom()`	Will be replaced by the start of the currently active time selection. For example, *'2017-04-21T05:01:17Z'*
`$__timeTo()`	Will be replaced by the end of the currently active time selection. For example, *'2017-04-21T05:06:17Z'*
`$__timeGroup(dateColumn,'5m'[, fillvalue])`	Will be replaced by an expression usable in GROUP BY clause. Providing a *fillValue* of *NULL* or *floating value* will automatically fill empty series in timerange with that value. For example, *CAST(ROUND(DATEDIFF(second, '1970-01-01', time_column)/300.0, 0) as bigint)*300*.
`$__timeGroup(dateColumn,'5m', 0)`	Same as above but with a fill parameter so missing points in that series will be added by grafana and 0 will be used as value.
`$__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.

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.

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.

Example database table:

```
CREATE TABLE [event] (
  time_sec bigint,
  description nvarchar(100),
  tags nvarchar(100),
)
```

```
CREATE TABLE [mssql_types] (
  c_bit bit, c_tinyint tinyint, c_smallint smallint, c_int int, c_bigint bigint,
c_money money, c_smallmoney smallmoney, c_numeric numeric(10,5),
  c_real real, c_decimal decimal(10,2), c_float float,
  c_char char(10), c_varchar varchar(10), c_text text,
  c_nchar nchar(12), c_nvarchar nvarchar(12), c_ntext ntext,
  c_datetime datetime,  c_datetime2 datetime2, c_smalldatetime smalldatetime, c_date
date, c_time time, c_datetimeoffset datetimeoffset
)

INSERT INTO [mssql_types]
SELECT
  1, 5, 20020, 980300, 1420070400, '$20000.15', '£2.15', 12345.12,
  1.11, 2.22, 3.33,
  'char10', 'varchar10', 'text',
  N'☺nchar12☺', N'☺nvarchar12☺', N'☺text☺',
```

```
  GETDATE(), CAST(GETDATE() AS DATETIME2), CAST(GETDATE() AS SMALLDATETIME),
CAST(GETDATE() AS DATE), CAST(GETDATE() AS TIME), SWITCHOFFSET(CAST(GETDATE() AS
DATETIMEOFFSET), '-07:00')
```

Query editor with example query:

{{< figure src="/static/img/docs/v51/mssql_table_query.png" max-width="500px" class="docs-image--no-shadow" >}}

The query:

```
SELECT * FROM [mssql_types]
```

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

```
SELECT
  c_bit as [column1], c_tinyint as [column2]
FROM
  [mssql_types]
```

The resulting table panel:

{{< figure src="/static/img/docs/v51/mssql_table_result.png" max-width="1489px" class="docs-image--no-shadow" >}}

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') as time,
  min("value_double"),
  'min' as metric
FROM test_data
WHERE $__timeFilter(time_date_time)
```

```
GROUP BY $__timeGroup(time_date_time, '5m')
ORDER BY 1
```

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) as time,
  sum(value) as value,
  hostname
FROM test_data
WHERE
  $__timeFilter(createdAt)
GROUP BY
  $__timeGroup(createdAt, '5m', 0),
  hostname
ORDER BY 1
```

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 name 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 $__timeGroup(time_date_time, '5m')
ORDER BY 1
```

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.

Check out the [Templating]({{< relref "../variables/_index.md" >}}) documentation 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 MS SQL 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]
```

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 allow 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 __text, id __value FROM host
```

You can also create nested variables. For example, if you had another variable named `region`. Then 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 Variables in Queries

> *From Grafana 4.3.0 to 4.6.0, template variables are always quoted automatically so if it is a string value do not wrap them in quotes in where clauses.*
>
> *From Grafana 5.0.0, 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 time,
  aint value
FROM table
WHERE $__timeFilter(atimestamp) and hostname in($hostname)
ORDER BY atimestamp
```

`[[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
```

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'`. Do disable quoting, use the csv formatting option for variables:

`${servers:csv}`

Read more about variable formatting options in the [Variables]({{< relref "../variables/variable-types/_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.

Columns:

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.

Example database tables:

```
CREATE TABLE [events] (
  time_sec bigint,
  description nvarchar(100),
  tags nvarchar(100),
)
```

We also use the database table defined in Time series queries.

Example query using time column with epoch values:

```
SELECT
  time_sec as time,
  description as [text],
  tags
FROM
  [events]
WHERE
  $__unixEpochFilter(time_sec)
ORDER BY 1
```

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

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

```
SELECT
  time_sec as time,
  time_end_sec as timeend,
  description as [text],
  tags
FROM
  [events]
WHERE
  $__unixEpochFilter(time_sec)
ORDER BY 1
```

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

```
SELECT
  time,
  measurement as text,
  convert(varchar, valueOne) + ',' + convert(varchar, valueTwo) as tags
FROM
  metric_values
WHERE
  $__timeFilter(time_column)
ORDER BY 1
```

Stored procedure support

Stored procedures have been verified to work. However, please note that we haven't done anything special to support this, so there might be edge cases where it won't work as you would expect. Stored procedures should be supported in table, time series and annotation queries as long as you use the same naming of columns and return data in the same format as describe above under respective section.

Please note that any macro function will not work inside a stored procedure.

Examples

{{< figure src="/static/img/docs/v51/mssql_metrics_graph.png" class="docs-image--no-shadow docs-image--right" >}} For the following examples, the database table is defined in Time series queries. Let's say that we want to visualize four series in a graph panel, such as all combinations of columns `valueOne`, `valueTwo` and `measurement`. Graph panel to the right visualizes what we want to achieve. To solve this, we need to use two queries:

First query:

```
SELECT
  $__timeGroup(time, '5m') as time,
  measurement + ' - value one' as metric,
  avg(valueOne) as valueOne
FROM
  metric_values
WHERE
  $__timeFilter(time)
GROUP BY
  $__timeGroup(time, '5m'),
  measurement
ORDER BY 1
```

Second query:

```
SELECT
  $__timeGroup(time, '5m') as time,
  measurement + ' - value two' as metric,
  avg(valueTwo) as valueTwo
FROM
  metric_values
```

```
GROUP BY
  $__timeGroup(time, '5m'),
  measurement
ORDER BY 1
```

Stored procedure using time in epoch format

We can define a stored procedure that will return all data we need to render 4 series in a graph panel like above. In this case the stored procedure accepts two parameters `@from` and `@to` of `int` data types which should be a timerange (from-to) in epoch format which will be used to filter the data to return from the stored procedure.

We're mimicking the `$__timeGroup(time, '5m')` in the select and group by expressions, and that's why there are a lot of lengthy expressions needed - these could be extracted to MS SQL functions, if wanted.

```
CREATE PROCEDURE sp_test_epoch(
  @from int,
  @to   int
)       AS
BEGIN
  SELECT
    cast(cast(DATEDIFF(second, {d '1970-01-01'}, DATEADD(second,
DATEDIFF(second,GETDATE(),GETUTCDATE()), time))/600 as int)*600 as int) as time,
    measurement + ' - value one' as metric,
    avg(valueOne) as value
  FROM
    metric_values
  WHERE
    time >= DATEADD(s, @from, '1970-01-01') AND time <= DATEADD(s, @to, '1970-01-
01')
  GROUP BY
    cast(cast(DATEDIFF(second, {d '1970-01-01'}, DATEADD(second,
DATEDIFF(second,GETDATE(),GETUTCDATE()), time))/600 as int)*600 as int),
    measurement
  UNION ALL
  SELECT
    cast(cast(DATEDIFF(second, {d '1970-01-01'}, DATEADD(second,
DATEDIFF(second,GETDATE(),GETUTCDATE()), time))/600 as int)*600 as int) as time,
    measurement + ' - value two' as metric,
    avg(valueTwo) as value
  FROM
    metric_values
  WHERE
    time >= DATEADD(s, @from, '1970-01-01') AND time <= DATEADD(s, @to, '1970-01-
01')
  GROUP BY
    cast(cast(DATEDIFF(second, {d '1970-01-01'}, DATEADD(second,
DATEDIFF(second,GETDATE(),GETUTCDATE()), time))/600 as int)*600 as int),
    measurement
  ORDER BY 1
END
```

Then we can use the following query for our graph panel.

```
DECLARE
  @from int = $__unixEpochFrom(),
  @to int = $__unixEpochTo()

EXEC dbo.sp_test_epoch @from, @to
```

Stored procedure using time in datetime format

We can define a stored procedure that will return all data we need to render 4 series in a graph panel like above. In this case the stored procedure accepts two parameters `@from` and `@to` of `datetime` data types which should be a timerange (from-to) which will be used to filter the data to return from the stored procedure.

We're mimicking the `$__timeGroup(time, '5m')` in the select and group by expressions and that's why there's a lot of lengthy expressions needed - these could be extracted to MS SQL functions, if wanted.

```
CREATE PROCEDURE sp_test_datetime(
  @from datetime,
  @to   datetime
)       AS
BEGIN
  SELECT
    cast(cast(DATEDIFF(second, {d '1970-01-01'}, time)/600 as int)*600 as int) as
time,
    measurement + ' - value one' as metric,
    avg(valueOne) as value
  FROM
    metric_values
  WHERE
    time >= @from AND time <= @to
  GROUP BY
    cast(cast(DATEDIFF(second, {d '1970-01-01'}, time)/600 as int)*600 as int),
    measurement
  UNION ALL
  SELECT
    cast(cast(DATEDIFF(second, {d '1970-01-01'}, time)/600 as int)*600 as int) as
time,
    measurement + ' - value two' as metric,
    avg(valueTwo) as value
  FROM
    metric_values
  WHERE
    time >= @from AND time <= @to
  GROUP BY
    cast(cast(DATEDIFF(second, {d '1970-01-01'}, time)/600 as int)*600 as int),
    measurement
  ORDER BY 1
END
```

Then we can use the following query for our graph panel.

```
DECLARE
  @from datetime = $__timeFrom(),
  @to datetime = $__timeTo()

EXEC dbo.sp_test_datetime @from, @to
```

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: MSSQL
    type: mssql
    url: localhost:1433
    database: grafana
    user: grafana
    jsonData:
      maxOpenConns: 0 # Grafana v5.4+
      maxIdleConns: 2 # Grafana v5.4+
      connMaxLifetime: 14400 # Grafana v5.4+
    secureJsonData:
      password: 'Password!'
```