TOC

Composer 24

Manipulate Data in the Composer Data Store

Composer allows you to manipulate the data read from your data store. You can enable distinct counts and Data Sharpening, change the formatting of numeric and time fields, convert appropriate existing data store fields into time fields, and use existing data store fields to create derived fields and custom metrics.

See the following topics:

- Use Data Sharpening
- Distinct Counts
- Configure Number Formatting - Data Sources
- Configure Date And Time Formatting - Data Sources
- Maintain Custom Metrics
- Maintain Derived Fields
- Configure Date And Time Fields

Use Data Sharpening

Data Sharpening™ is Composer's patented technique to deliver fast and responsive visuals for large volumes of data. Conceptually, Data Sharpening is similar to the way large image files or streaming video files display in a browser. When you start to load the image file, you see a blurry approximation of the image. But as the file loads in the background, the image sharpens until the entire image eventually comes into clear focus.

When you create or modify a visual for a large data set in Composer, Data Sharpening immediately displays a partial or approximate rendering of the data. It then continuously updates the visual with more and more data until the fully sharpened result is available.

While the visual sharpens, you can continue to interact with it, zooming into more detail or changing the Group By attribute without waiting for the entire query to return. Essentially, you can continue your big data exploration without waiting for long-running queries over billions of rows of data to complete. Composer adjusts on-the-fly based on your input. One thing to keep in mind—Data Sharpening may not always be needed when visualizing your data. If Composer is able to complete its query of the data quickly (within a few seconds), you will simply see the final result rendered in the visual. Data Sharpening is a tool that is leveraged when the visual may not render immediately due to the volume of the data being queried.

See the following topics:

- Data Sharpening Prerequisites
- When Data Sharpening Occurs
- Enable Data Sharpening And Configure Its Defaults
- Enable Data Sharpening For Cloudera Impala Data Sources

Data Sharpening Prerequisites

Before you use Data Sharpening, verify that your data source configuration has enabled it. See Enable Data Sharpening And Configure Its Defaults.

Before Composer can perform Data Sharpening with a data source, a "playable" time field is required. Composer attempts to automatically detect this playable field from your data source when it is defined. To determine what makes a time attribute "playable," refer to the table below. In addition, an appropriate time attribute must be specified in your data source's global default settings on the Visuals page of the data source configuration. The granularity of this time field is referred to as the *driving time field granularity* (DTFG) and plays an important role in determining whether and how Data Sharpening is executed (see When Data Sharpening Occurs). See Enable Data Sharpening And Configure Its Defaults to learn how to enable Data Sharpening for your connected data sources.

Data Sharpening setup differs slightly depending on the data source. Although a playable time field is required for sharpening to occur, the time field requirement is based on the data source. The table below lists the time field requirements for different data stores supported in Composer.

Data Store	Time Field Requirement
Amazon Redshift	Sort Key (only the first sort key is selected)
Cloudera Impala, Hive	Partitioned time field (The time field that is partitioned needs to be configured from the "Fields" page. A single partitioned column is needed for Data Sharpening to work in Impala or Hive sources.)
Search-based sources (Cloudera Search, Elasticsearch, Apache Solr)	Indexed time field*. Composer automatically detects for indices.
SQL-based sources (MySQL, Oracle, PostgreSQL, SQL Server)	Indexed time field*. Composer automatically detects for indices.

Note: The indexed time field should already be set in the data source, so no additional configuration is needed in Composer.

When Data Sharpening Occurs

When Composer connects to your data source for the first time, it runs an initial query to return a sample of the data set—approximately 100 rows of data—to provide an initial time range. Meanwhile, it continues to run a comprehensive query to obtain the actual MIN/MAX range based on the entire data set. In this instance, based on the results of the sample query, Data Sharpening may not activate because the time range and time granularity results fall short of the criteria for sharpening to execute. However, after Composer completes the full query, Data Sharpening works as expected as long as the correct parameters have been applied and the time criteria are met. Depending on the size of your data set, the comprehensive query process may take a few minutes to complete its execution (additional constraints include the size and type of the data source and other factors such as competing resources on the database and resource and performance limitations).

Bear in mind that Composer connects to and runs queries in your original data source and can be resource intensive. The full query runs in the background at the same time as a series of microqueries that sample data across partitions and refine estimates.

When you create a visual, Composer determines whether Data Sharpening is necessary based on the visual style selected and the time attribute parameters that are set for it.

- For non-trend visuals (such as bars, donut charts, and heat maps), the granularity of the driving time field must be less than 10% of the range that is set in the time bar (determined by the MIN/MAX range set in the data source). The minimum granularity used by Composer will *always* be **minutes**. Thus, even if your driving time field granularity (DTFG) is **seconds**, Composer will use **minutes** when performing this 10% rule calculation.

- For trend visuals (such as line and bars trend and line trend attribute value charts), Composer runs an internal check to determine whether Data Sharpening should execute. Similar to the non-trend visuals, a 10% criteria is used, but it is slightly modified for the trend visual scenario. If the granularity of the driving time field for the source is less than 10% of the time granularity used in the particular trend visual, then Data Sharpening executes.

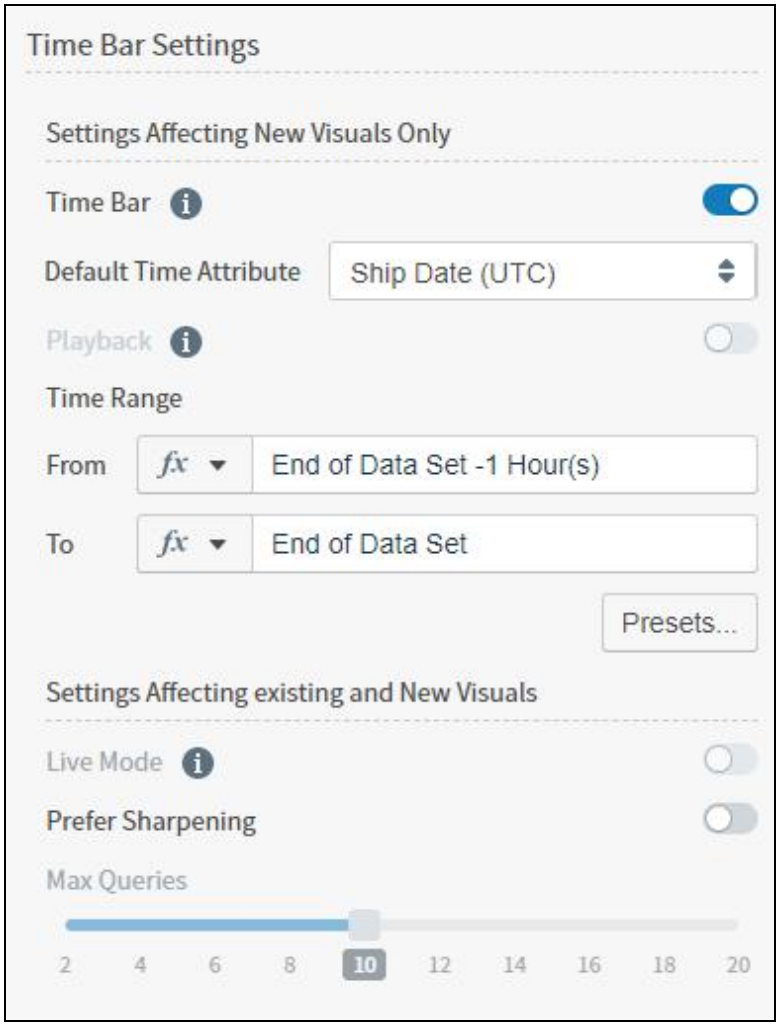
The bottom line is that Composer tries to perform Data Sharpening when warranted based on the size of the data set, the time attributes available, and the time granularity that is set. If Composer determines that results can be rendered in the visual quickly without Data Sharpening, it does so. Otherwise, it attempts to use Data Sharpening to return near instantaneous result sets that are refined over time until the query completes.

Enable Data Sharpening and Configure Its Defaults

To enable and configure Data Sharpening for a data source configuration, you need to enable the time settings in the data source's settings page. Specifically, a playable time field must be specified in the Global Settings tab of the data source configuration. However, Cloudera Impala data sources require additional configuration as described in Enable Data Sharpening For Cloudera Impala Data Sources.

Configure Data Sharpening in a data source configuration

1. Log into Composer (either as an administrator or as a user who has been assigned to a group with data source management privileges).
2. Select **Sources** on the UI menu () or the top-level navigation menu, or select the **Sources** box on the Home page. The Sources page appears.
3. In the table on the Sources page, locate and select the data source configuration you want to edit.
4. Select the **Global Settings** tab. The time bar, search, and Data Sharpening settings for the data source appears.



5. Make sure **Time Bar** is enabled to access Data Sharpening settings. If Time Bar is disabled, you can't configure Data Sharpening settings. For more information about time bar default settings, see Configure Time Bar Defaults.

6. Slide the **Prefer Sharpening** switch to the right to enable Data Sharpening for the data source.

7. Optionally, use the **Max Queries** slider to specify the maximum number of queries used for Data Sharpening. The default maximum is 10 queries.

8. When your changes are complete, select **Save Settings** to save your changes.

Distinct Counts

Distinct count functionality determines the number of unique values in a column or expression within a selected table by comparing all the records pulled from the data store by a data source configuration. When distinct counts are used, unique value results are returned when analyzing data. For example, distinct counts could return the number of:

- Unique customers in a sales database
- Unique UPC codes for a category of products
- The number of trucks in a company's fleet

For example, given a single collection and string field with the following three values:

1. Apple
2. Orange
3. Apple

The distinct count returns 2, since there are only two distinct values (“Apple” and “Orange”), while an ordinary count returns 3 to reflect the total number of records. SQL-based connectors might produce a query that looks like this:

```
select count(distinct myField) from myCollection
```

Support for this feature by connector is shown in the following table.

Key:Y - Supported; **N** - Not Supported; N/A - not applicable

Connector	Supported?	Notes
Amazon Redshift	**Y**	
Amazon S3	**Y**	
Apache Drill	**Y**	

Connector	Supported?	Notes
Apache Phoenix	Y	
Apache Phoenix Query Server (QS)	Y	
Apache Solr	Y	
BigQuery	Y	If you need to access a BigQuery partition, explicitly include an alias for the built in partition column in your select clause, such as `select *, _PARTITIONTIME as pt from projectId.datasetId.tableId`.
Business Central Jet	Y	
Cloudera Impala	Y	Cloudera Impala connectors can receive only a single distinct count field in a query.
Cloudera Search	Y	
Couchbase	Y	
Dremio	Y	
Elasticsearch 7.0	Y	
Elasticsearch 8.0	Y	
File Upload	Y	
HDFS	Y	
Hive	Y	
Jira	Y	
MemSQL	Y	
Microsoft SQL Server	Y	
MongoDB	Y	
MySQL	Y	
Oracle	Y	
PostgreSQL	Y	
Python	Y	
Real Time Sales	Y	
Salesforce	Y	
SAP Hana	Y	
SAP S/4HANA	Y	
SAP IQ	Y	
Spark SQL	Y	

Connector	Supported?	Notes
Snowflake	Y	
Teradata	Y	
TIBCO DV	Y	
Trino	Y	
File Upload (Upload API)	Y	
Vertica	Y	

Configure Number Formatting - Data Sources

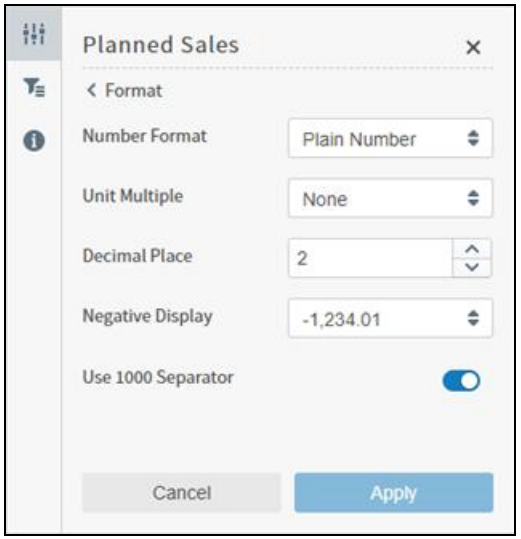
As a user with data source privileges, configure number formats for Number fields in a data source at creation, or later when you edit an existing source. This defines the default format for number fields as displayed in visuals, data details of visuals, and sample format fields for consistency across your organization.

There are three places to control the display of numbers in a data source.

- Setting the locale for users to display correct formats, based on a user's geographical location. For example, users based in the United States and users based in Italy would see different currency formats due to the local formatting of the fields. To define the location settings for users within your environment, ask your administrator or see Specify A User's Regional Settings.
- Configuring the formats for number fields in a data source configuration, as described in the rest of this topic. Bear in mind that defining number formats depends on the region (location) settings configured for your environment.
- Finally, you can enable users to override the formatting you've defined at the data source and for most visuals at the visual level. Find more visual formatting information in the following topics:
  - Configure Date And Time Formatting For Visuals
  - Configure Number Formatting For Visuals
  - Format Time Table Data Using The Table Context Menu
  - Format Numeric Table Data Using The Table Context Menu
  - Available Visual Types

Specify the format for number fields in a data source configuration

1. Log in as a user with the **Administer Sources** or **Create New Data Sources** privilege, or write permission for this source.
2. Select **Sources** on the UI menu (). The Sources page appears.
3. Edit the appropriate data source configuration and access the **Fields** tab of the data source.
4. Locate the number field you want to modify. The **Data Type** column on the **Fields** tab must define the field as a **Number** field.
5. In the sidebar menu, select the settings () button to open the **Settings** work area. Select the edit () button in the **Format** work area to edit the date and time format for this field.



6. Select one of the following number formats in the drop-down list in the Number Format box. The other fields on the Format dialog change based on the number format you select.

i. **Plain Number**: Select this format to display the field as plain number values. Additional format information you can select includes:

Format Option	Description
Unit Multiple	Select the unit of measure you want to use for the field from the drop-down list. Valid values are **None**, **Thousands (K)**, **Millions (M)**, **Billions (B)**, and **Trillions (T)**. The unit multiple is used in visuals. For example, a value of 1,500,000 would show as 1.5M in visuals.
Decimal Place	Specify the number of decimal places used in the data.
Negative Display	Select the desired negative value format from the drop-down list. Valid values are the dash (-) in front of a negative value or parentheses surrounding negative values. For example, value of negative 30 could show as -30 or (30) in visuals.
Use 1000 Separator	If you want commas used to separate number values into thousands, millions, billions, and trillions, check the **Use 1000 Separator** box. For example, if this box is checked, 2500 appears as 2,500 in visuals.

ii. **Percentage**: Select this format to display the field as percentage values. Additional format information you can select includes:

Format Option	Description
Decimal Place	Specify the number of decimal places used in the data.
Negative Display	Select the desired negative value format from the drop-down list. Valid values are the dash (-) in front of a negative value or parentheses surrounding negative values. For example, value of negative 30.25 percent could show as -30.25% or (30.25%) in visuals.
Use 1000 Separator	If you want commas used to separate number values into thousands, millions, billions, and trillions, check the **Use 1000 Separator** box.

iii. **Money**: Select this format to display the field as currency values. Additional format information you can select includes:

Format Option	Description
Symbol	Select the currency symbol you want used for money values in visuals.
Unit Multiple	Select the unit of measure you want to use for the field from the drop-down list. Valid values are **None**, **Thousands (K)**, **Millions (M)**, **Billions (B)**, and **Trillions (T)**. The unit multiple is used in visuals. For example, a value of 1,500,000 would show as 1.5M in visuals.
Decimal Place	Specify the number of decimal places used in the data.
Negative Display	Select the desired negative value format from the drop-down list. Valid values are the dash (-) in front of a negative value or parentheses surrounding negative values. For example, value of negative 30 dollars and 25 cents could show as -$30.25 or ($30.25) in visuals.
Use 1000 Separator	If you want commas used to separate number values into thousands, millions, billions, and trillions, check the **Use 1000 Separator** box.

iv. **Storage**: Select this format to display the field as computer storage values. Additional format information you can select includes:

Format Option	Description
Unit Multiple	Select the unit of measure you want to use for the field from the drop-down list. Valid values are **Bytes (B)**, **Kilobytes (KB)**, **Megabytes (MB)**, **Gigabytes (GB)**, **Terabytes (TB)**, **Petabytes (PB)**, and **Exabytes (EB)**. The unit multiple is used in visuals. For example, a value of 950 kilobytes would show as 950KB in visuals.
Decimal Place	Specify the number of decimal places used in the data.

v. **Scientific Notation**: Select this format to display the field as scientific decimals. Additional format information you can select includes:

Format Option	Description
Decimal Place	Specify the number of decimal places used in the data.

7. Select **Apply** to apply to apply your changes, then **Save** to save your changes.

Configure Date and Time Formatting - Data Sources

As a user with data source privileges, configure date and time for Time fields in a data source at creation, or later when you edit an existing source. This defines the default format for time fields as displayed in visuals, data details of visuals, and sample format fields for consistency across your organization.

There are three places to control the format of date and time information.

- Set the locale for users to display the appropriate order of fields, based on a user's geographical location. For example, users based in the United States and users based in Italy see different date order formats due to the local formatting of the fields. To define the location settings for users within your environment, ask your administrator or see Specify A User's Regional Settings.
- Configure the formats for Time fields in a data source configuration, as described in the rest of this topic. The order dates and times are presented in depend on the region (location) settings for your environment: you're changing the format of each time or date field to suit your organization's needs.
- Finally, you can enable users to override the formatting you've defined at the data source and for most visuals at the visual level. Find more visual formatting information in the following topics:
  - Configure Date And Time Formatting For Visuals
  - Configure Number Formatting For Visuals
  - Format Time Table Data Using The Table Context Menu
  - Format Numeric Table Data Using The Table Context Menu
  - Available Visual Types

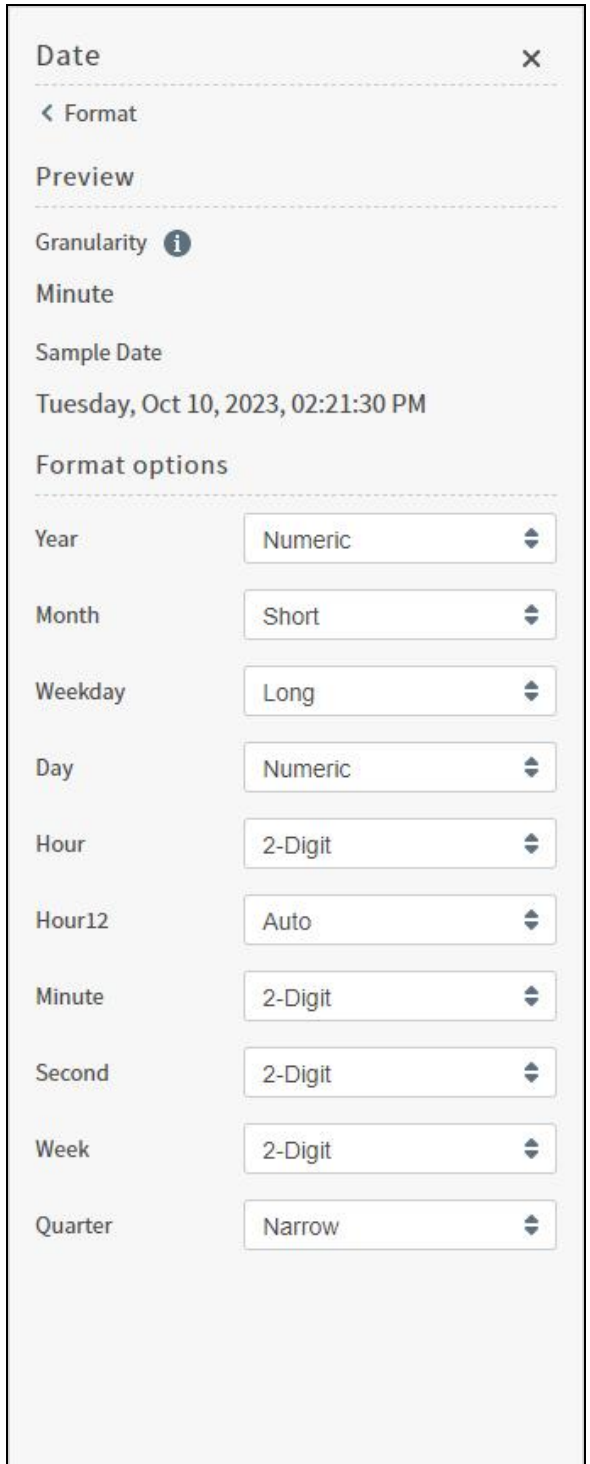
Note: If you change the format for dates and times for a field used by current visuals in your data source, the formats used in the visual reflect those changes. If you adjust the granularity of a field in your data source, the change is reflected in the granularity modal of the visual after the change.

Specify the format for Time fields in a data source configuration

1. Log in as a user with the **Administer Sources** or **Create New Data Sources** privilege, or write permission for this source.
2. Select **Sources** on the UI menu ( ). The Sources page appears.
3. Select to edit the appropriate data source configuration, and switch to the **Fields** tab.

4. Locate the time field you want to modify. The **Data Type** column on the **Fields** tab must define the field as a **Time** field.

5. In the Settings sidebar menu, select the edit () button to open the **Format** work area.


Date
Format
Preview
Granularity
Minute
Sample Date
Tuesday, Oct 10, 2023, 02:21:30 PM
Format options
Year
Numeric
Month
Short
Weekday
Long
Day
Numeric
Hour
2-Digit
Hour12
Auto
Minute
2-Digit
Second
2-Digit
Week
2-Digit
Quarter
Narrow

Not all fields may be available for all time formats.

Format Option	Description
Year	Select **None**, **Numeric** (2024), **2-Digit** (24), **Narrow**, or **Short**.
Day Of Year	Select **None** or **Short**.
Quarter	Select **None**, **Range**, **Narrow**, **Numeric** (1-4), or **2-Digit** (01-04). Selections for Quarter are applied only if the time granularity is set to Quarter. Not all options may be available for all time fields.
Month	Select **None**, **Numeric** (1-12), **2-Digit** (01-12), **Long** (January, March), **Short** (Jan, Mar), or **Narrow** (J, M).
Week	Select **None**, **WeekStart**, **WeekEnd**, **Range**, **Narrow** (w1-w53), **Numeric** (1-53), or **2-Digit** (01-53).
Weekday	Select **Long** (Friday, Sunday), **Short** (Fri, Sun), **Narrow** (F, S), or **None**. The default is **None**, to show no day of the week.
Day	Select **None**, **Numeric** (1-7), **2-Digit** (01-07), **Narrow**, or **Short**.
Hour	Select **Numeric** (1-24) or **2-Digit** (01-24).
Hour12	Select a 12 or 24 hour format option for Hour12. ■ **Auto** displays the hour format defined in the source data. ■ **True** overrides the source format, displaying the hours in 12 hour format, with an appended **AM** or **PM**. ■ **False** overrides the source format, displaying the hours in 24 hour format.
Minute	Select **Numeric** (0-59) or **2-Digit** (00-59).
Second	Select **Numeric** (0-59) or **2-Digit** (00-59).

6. The changes you make are reflected in the **Sample Date** preview field.

7. When you are finished formatting your date and time values, select **Apply** and examine your updates. If they are correct, select **Save** to save these changes to the source.

Configure Date and Time Fields

Composer allows you to use number and attribute fields as time fields by creating a derived field you can use as time data. See Convert Attributes to Time Fields in Data Source Field Specifications.

Supported Date and Time Formats

Composer allows you to use number and attribute fields as time fields by creating a derived field you can use as time data. See Convert Attributes to Time Fields in Data Source Field Specifications.

Fiscal Calendars

Note: Fiscal Calendars functionality is disabled by default. To enable, contact technical support for assistance.

Important: This is an experimental feature.

Use fiscal calendars in your environment to view your data based on a calendar system you define in Composer. Use the REST API endpoint `/api/calendars` to define one or more fiscal calendars to use in your sources.

API documentation is provided with your Composer installation at this link: *`https://<composer-URL>/composer/swagger-ui.html`*.

Once defined, you can assign a fiscal calendar for use in specific sources and convert a time field into a new a derived field you can use as a time field in the data source.

- Define a Fiscal Calendar
- Use a Fiscal Calendar

Define a Fiscal Calendar

Note: Fiscal Calendars functionality is disabled by default. To enable, contact technical support for assistance.

Important: This is an experimental feature.

You can define multiple fiscal calendars for use in your environment, enabling your users to create dashboards and visuals that reflect data as structured in your preferred calendar time frame.

Create a Fiscal Calendar

Administrators and users who are assigned to a group with the Administer Calendars privilege can use the REST API endpoint `/api/calendars` to define one or more fiscal calendars to define calendars for your users.

API documentation is provided with your Composer installation at this link: *`https://<composer-URL>/composer/swagger-ui.html`*.

When you create a fiscal calendar, you can define which month of your quarters is the longest month, and optionally shift the month your year begins. Send an array for the **CalendarResource** `type` as `MONTH_SHIFT`, `FISCAL_445`, or include both.

The name you select for a calendar is displayed in the Composer user interface.

Set the first month of your fiscal year

If you optionally include and define a `MONTH_SHIFT`, use an integer value ranging from `-11` to `11` to set the start month of your fiscal year. Do not use `0`: it is an invalid selection.

- For February of the previous calendar year, use `-11`.
- For November of the current calendar year, use `11`.
- If no `MONTH_SHIFT` is provided, the default start month of the year is January.

Set the five week month of your fiscal year

When you define the fiscal calendar, you can set which month of the quarters will be five weeks long using **quarterType**.

- `QUARTER_445_WEEKS` - the last month of each quarter is the five week month.
- `QUARTER_454_WEEKS` - the middle month of each quarter is the five week month.

- `QUARTER_544_WEEKS` - the first month of each quarter is the five week month.

There are several more settings used to define your fiscal calendar. See the REST API for more information.

Once you've created one or more fiscal calendars, users who can create and update sources can select appropriate calendars to use for a source, and define a derived field **Time (FISCAL)** field that uses a selected calendar.

SeeUse a Fiscal Calendar.

Use a Fiscal Calendar

Note: Fiscal Calendars functionality is disabled by default. To enable, contact technical support for assistance.

Important: This is an experimental feature.

After a user with appropriate privileges has defined one or more fiscal calendars to your environment, you can use these calendars in any of your sources.

Select an alternative calendar

1. Open and edit an existing source or create a new source.
2. Navigate to the Global Settings tab and select an available calendar listed under **Alternative Calendars Settings**.

   Note: The default calendar is used for all sources unless you specifically select an alternative calendar.
3. Select one or more calendars for your source as needed. Clear the checkbox for a calendar to not use that specific calendar with this source and prevent fiscal time field conversion.
4. After completing your changes, select **Save Settings** to make available your selected calendars to new and existing visuals that use this source.

Create a Time (FISCAL) field for your selected calendar

1. After you have selected one or more calendars for your source, navigate to the Fields tab to convert a time field to a time field that uses one of the calendars you have added to this source.
2. Select the time field you want to use in the list of available fields.
3. Select the **Convert** option in the Data Details work area on the Settings tab. Select the option presented, **Convert to Fiscal Time** to open a conversion dialog.
4. Enter a **Label** for the new derived field, and select a **Fiscal Calendar** from the list of available options. **Save** your changes.
5. Your new derived field is added to the list of available fields with a data type of **Time (Fiscal)**. It is similar to standard time fields, with the exception that you cannot define a **Time Zone** for this new derived field.

Note: Alternatively, you can select **Add Derived Field** from the Fields tab and create the field using the `to_chrono_datetime` function. Include the `calendar_id` and `dateTime` field in the editor.

Once created, you can use this new field in visuals, expressions, filters, and using presets as needed.

- Objects that use a fiscal time field are marked with a **FISCAL** indicator.
- When you export data that relies on fiscal calendars, the data is presented in the appropriate date and time format.
- If you remove an alternative calendar from your source, you cannot make new derived fields based on that calendar.

Convert Attributes to Time Fields in Data Source Field Specifications

If your data contains time-related fields (attributes) that are not stored in a recognized time format, you can convert the fields to time fields using the Fields tab of the data source configuration. As long as any string field contains date or time data, you change it to a time pattern recognized by Composer.

Important: As of Composer v7.10, implicit field type conversions are no longer supported. When you want to convert one field type to another, a derived field is created with the new field type you selected. When you upgrade from an earlier version of Composer, some of the fields in your data sources may be converted into a derived field supported by the upgrade. See Source Field Migration.

Note: The process described here is **not** the recommended process. Instead, Composer recommends that you convert the data using a derived field, as described in Convert Attributes To Time Fields Using Derived Fields.

Composer uses Java's SimpleDateFormat for time conversions. See SimpleDateFormat.

Note: Group functionality does **not** work for fields for which the type is manually set to **Time**. You cannot specify these fields as the Group, Group By, or Trend fields for a visual. You **can** use these fields in filters and apply them as filters on the time bar. However, you may find that the results shown in your visual are incorrect. This may happen because the manually configured time formats are different and, consequently, not in lexicographic order. For example, suppose you have two strings:

- String `20230801` (August 1, 2023) matches the time format `yyyyMMdd`.
- String `08012024` (August 1, 2024) matches the time format `MMddyyyy`.

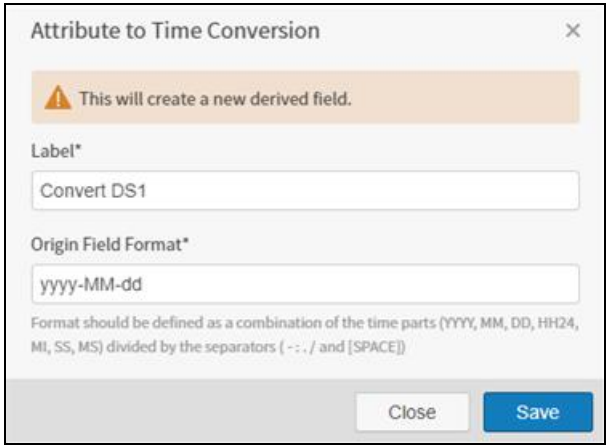
When filtering time data by these fields, Composer treats the time values as numbers. So, when filtered in ascending order, 08012024 will sort before 20230801, which is not correct (August 1, 2024 occurred after August 1, 2023, not before). The resulting visual will not be correct.

Convert an attribute field to a time field as a derived field in your data source configuration

1. Make sure you are logged in as a Composer administrator.
2. Select **Sources** on the UI menu ( ). The Sources page appears.
3. In the sources table on the Sources page, locate and select the data source configuration you want to edit.
4. Select the Fields tab.
5. Locate and select the time field (Data Type: Attribute) in the list of fields.

6. Select **Convert** in the Data Type area of the Settings tab for the field and select **Convert to Time**.

7. Rename the field in the **Label** field, and define time granularity in the **Origin Field Format**.



Valid time parts include:

i. YYYY - four digit year

ii. MM - two digit month

iii. DD - two digit day

iv. HH - two digit 24-hour format

v. MI - two digit minute

vi. SS - two digit seconds

vii. MS - two digit miliseconds

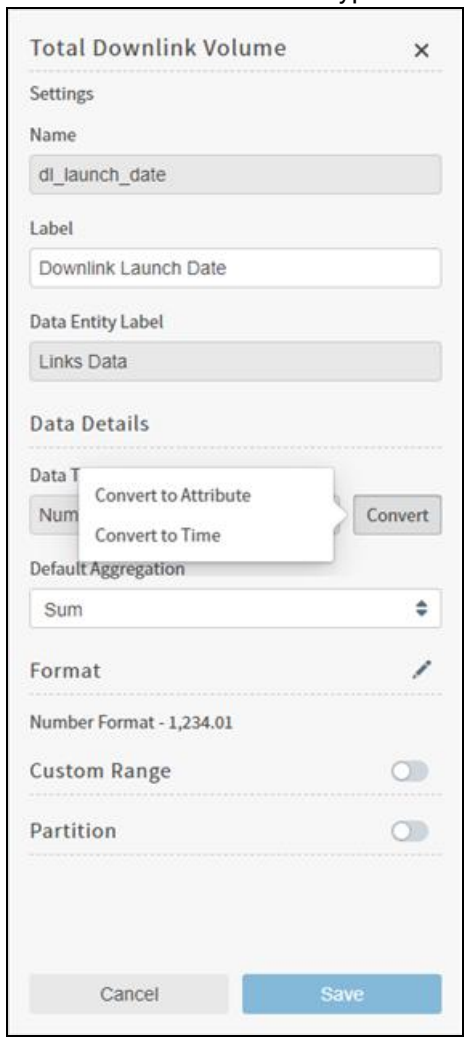
> **Note:** The information returned is limited by the information available in the original field. For example, if a field's data is stored in hours, you will get values up to the hour level. If you request granularity not available, zeros are returned for information not available, for example, 0 minutes, 0 seconds, and 0 milliseconds.

8. When your changes are complete, select **Save** create the derived field.

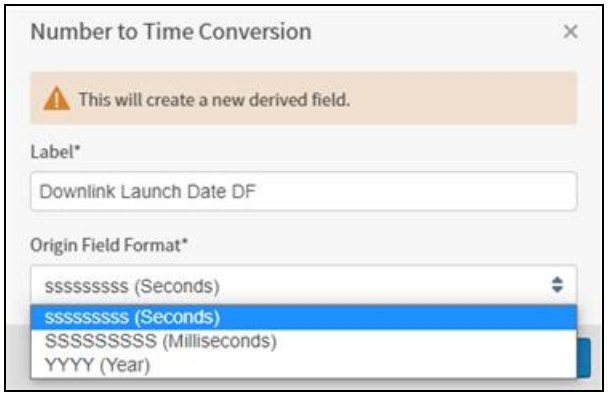
Convert a number field to a time field as a derived field in your data source configuration

1. Make sure you are logged in as a Composer administrator.
2. Select **Sources** on the UI menu (). The Sources page appears.
3. In the sources table on the Sources page, locate and select the data source configuration you want to edit.
4. Select the Fields tab.
5. Locate and select the time field (Data Type: Number) in the list of fields.

6. Select **Convert** in the Data Type area of the Settings tab for the field and select **Convert to Time**.



7. Rename the field in the **Label** field, and select an available time granularity in the **Origin Field Format**.



Valid time parts include:

i. YYYY - four digit year

ii. MM - two digit month

iii. DD - two digit day

iv. HH - two digit 24-hour format

v. MI - two digit minute

vi. SS - two digit seconds

vii. MS - two digit miliseconds

> **Note:** The information returned is limited by the information available in the original field. For example, if a field's data is stored in hours, you will get values up to the hour level. If you request granularity not available, zeros are returned for information not available, for example, 0 minutes, 0 seconds, and 0 milliseconds.

8. When your changes are complete, select **Save** create the derived field.

Convert Attributes to Time Fields Using Derived Fields

If your data contains time-related fields (attributes) that are not stored in a recognized time format, you can convert them to time fields using a derived field. After the derived field is defined, you can use it instead of the original attribute in your Composer visuals. This is the preferred method because the data in the derived field is constructed as data is read from the data store.

Convert an attribute field to a time field in a derived field

1. Log in as a Composer administrator or user with the ability to modify a data source configuration.
2. Start creating a derived field as described in Create And Modify Derived Fields.
3. Use the `TEXT_TO_TIME` function in a row-level expression in the derived field to convert your field to a time field. See Text Functions.
4. Test and save the derived field. See Create And Modify Derived Fields.

Source Field Migration

When you upgrade from an earlier version of Composer to v7.10 or later, several field types in your existing sources are migrated and converted to supported field types. Composer no longer supports implicit field type conversions: when you change the field type for a field, you effectively create a new derived field. See Create And Modify Derived Fields.

Field Type Conversion: Previous Releases to Current Release

Scenario: You have a `date_year` field defined as an integer in your original database.

- Composer (previous releases):
  - `date_year` is defined as a number or integer (dependent on your version).
  - You set the `type` as Time with Custom Pattern `yyyy`.
  - Composer treats the field as a `time` field.
- Composer, on upgrade, will now have two fields for each affected field in your source.
  - Field 1: `date_year_original`. The Label text is appended with `Original`. The field type is set to `number` and the field is not visible (Visible toggle disabled in the Fields tab).
  - Field 2: `date_year` as in your original source. The Label text and other settings are unchanged. This is a derived field, using the expression `year_to_time(date_year_original)`

Note: If you create a new data source that uses the `date_year` integer from your source, you can create a derived field to change the `number` into `time`, using the expression `year_to_time(date_year)`.

Possible field type conversions and limitations performed on upgrade are listed in table below. This is not an all-inclusive list:

Data Type conversion	Derived Field Expression	Requires Derived Field Support
ATTRIBUTE to NUMBER	`text_to_num(field_name)`	Yes
ATTRIBUTE to TIME	`text_to_time(field_name, 'pattern')`	Yes

NUMBER to TIME	`year_to_time(date_year)`
	`ms_to_time(date_milliseconds)`
	`unix_time_to_time(date_seconds)`
		No

NUMBER to ATTRIBUTE	`num_to_text(field_name)`	Yes

Data Type conversion	Derived Field Expression	Requires Derived Field Support
TIME to NUMBER	`time_to_unix_time(field_name)`	Yes

Maintain Custom Metrics

Custom metrics, formerly called *calculations*, serve as additional metrics to use with your visuals to help you analyze your data. When you define a data source configuration, you can create custom metrics for it. After custom metrics are defined, they are listed with the other metrics in your visuals and dashboards that use the data source and can be used just as any other metric.

You can use custom metrics to aggregate data in different ways, whether from an entire data source or from a selected subset and, as needed, applying arithmetic or trigonometric operators to them. Because a custom metric represents aggregated data, it must be created using an aggregate function. Data can be aggregated on a column-wide scope, a table scope, or a window scope. For more information, see Supported Aggregation Functions and .

Scope of data refers to the domain of data included in a custom metric - whether a whole column from the data source, or just part of the column. Composer always excludes data from a custom metric if it is excluded from the visual by a filter even if that data would otherwise be included in the scope of the custom metric. You can further control what data is included by choosing metric functions that match the scope you want.

You can also use row-level functions and constant values to build your formula for a custom metric. In this way, you can create custom metrics of percentages, differences, averages, and the like. For more information about row-level functions, see Supported Row-Level Functions.

Custom metrics can be filtered. Filters are created using the SQL-like expressions WHERE and TRANSFORM and may also use date and time filter aggregate functions. For more information about the use of filters in custom metrics, see Apply Filters To Custom Metrics.

For information on creating and deleting custom metrics, see the following links:

- Custom Metrics Editor
- Create And Modify Custom Metrics
- Supported Aggregation Functions
- Supported Row-Level Functions
- Apply Filters To Custom Metrics
- Delete Custom Metrics
- Custom Metric Examples

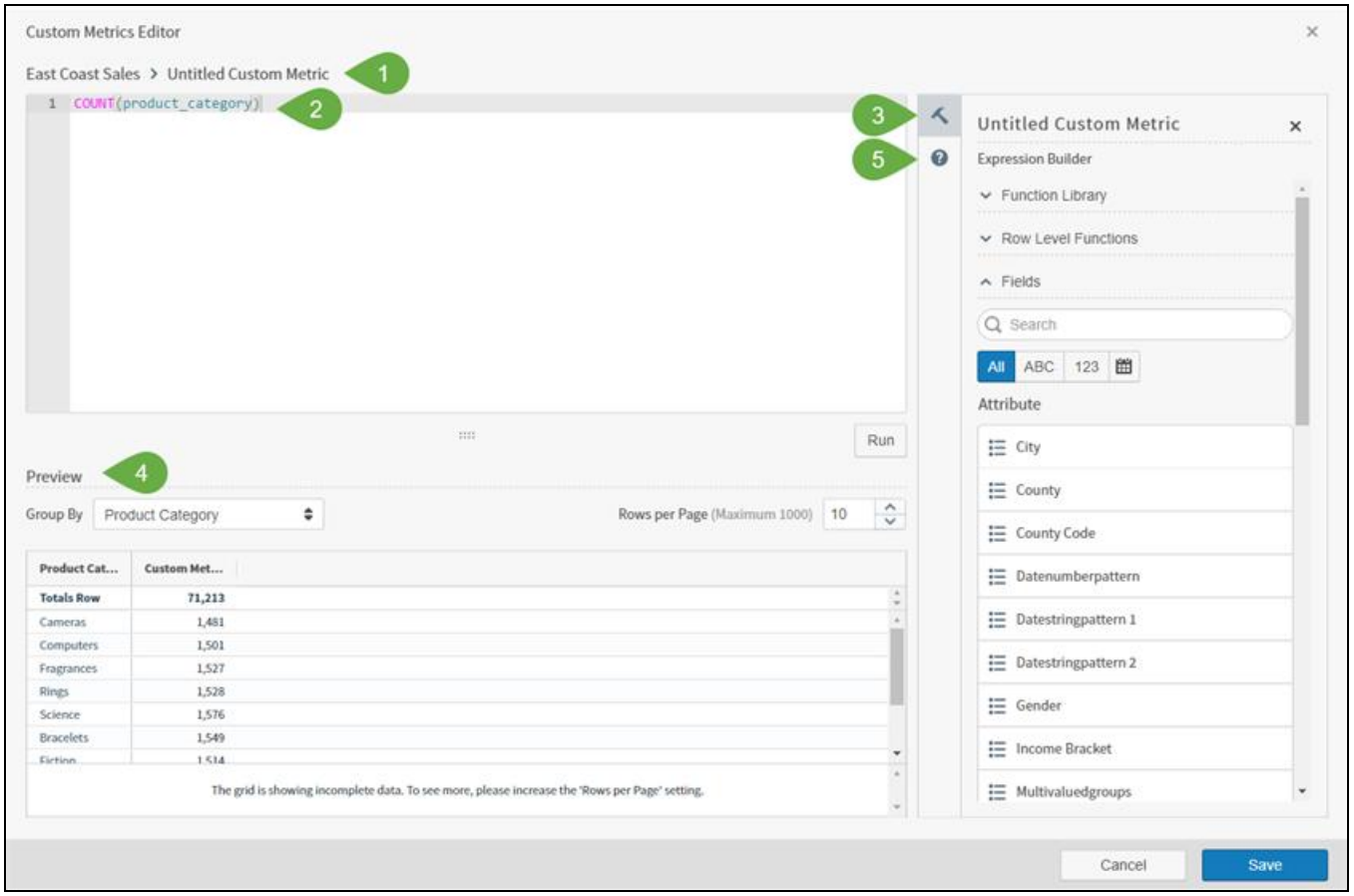
Custom Metrics Editor

Composer provides a Custom Metrics Editor to help you create and test custom metrics for a data source.

To define custom metrics for a data source, you must have:

- Read permission for the data source and the **Edit Calculation** privilege, or
- Write permission for the data source.

The Custom Metrics Editor is shown below. For every source added, the custom metric **Volume**, using a `Count(*)` expression is created.



The numbered regions are:

1. Custom Metric Label: Mandatory, fewer than 255 characters long.
2. Editing space: Build your expression in this space. Syntax highlighting improves the readability of your expression.
3. Expression Builder work area: Includes a Function Library, Row Level Functions, and Fields to help you build your custom metric.
4. Preview space: Shows a preview of the results of your expression.
5. Calculation Help: Provides more detailed information about the types of metrics Composer supports.

Create a custom metric by defining a formula composed of metrics and attributes that include row level functions and aggregation functions. You can include existing custom metrics in your new custom metric formula, and control the visibility of each metric.

Custom Metrics Editor Features:

- Syntax highlighting improves the readability of your expressions to provide visual clues about the items being used and their validity.

  Parts of an expression are highlighted in different colors or with different text treatments:

  - function names, both row level and aggregate
  - fields and metrics
  - keywords such as CASE and IN
  - example parameters
  - date period constants such as `year`
  - values such as numbers and `true` or `false`
  - strings
  - arithmetic operators such as `+`

  References to fields that do not exist in the data source or are otherwise not usable in an expression are interpreted as values. These are shown in black, alerting you to possible typos or other issues.

- Limit the number of records shown in **Preview** by adjusting the **Rows per Page**.
- The Function Library, Row Level Functions and Fields sections always appear in the Expression Builder.

- Autocomplete functionality is included: Type two letters and a list of possible auto-completions appears.
  a. Function completions provide the type of function, the name of the function, the description of the function, and an example of how the function is used, including parameters.
  b. Field completions are available by typing either the field ID or the field label. The field ID appears in square brackets next to the field label when the two are different. Field completions provide the type of the field, the field label and ID.
  c. Metric completions provide the metric label and the expression that will be inserted into the editor when the metric is selected.
- Syntax and validation errors are shown in the Preview area.
- When you're creating or editing an expression, select **Cancel** to return your expression to the initial state.
- Composer disables the **Save** button, preventing you from saving a custom metric unless it has a successfully run expression.
- When you close the editor with unsaved changes, Composer displays a confirmation message.
  - Select **Cancel** to continue editing or changing the text of the label for your custom metric.
  - Select **Discard** to discard your changes. If you are editing an existing item, your last saved version remains in Composer.
  - If you select **Discard** while creating a new expression, Composer returns you to the source work area.

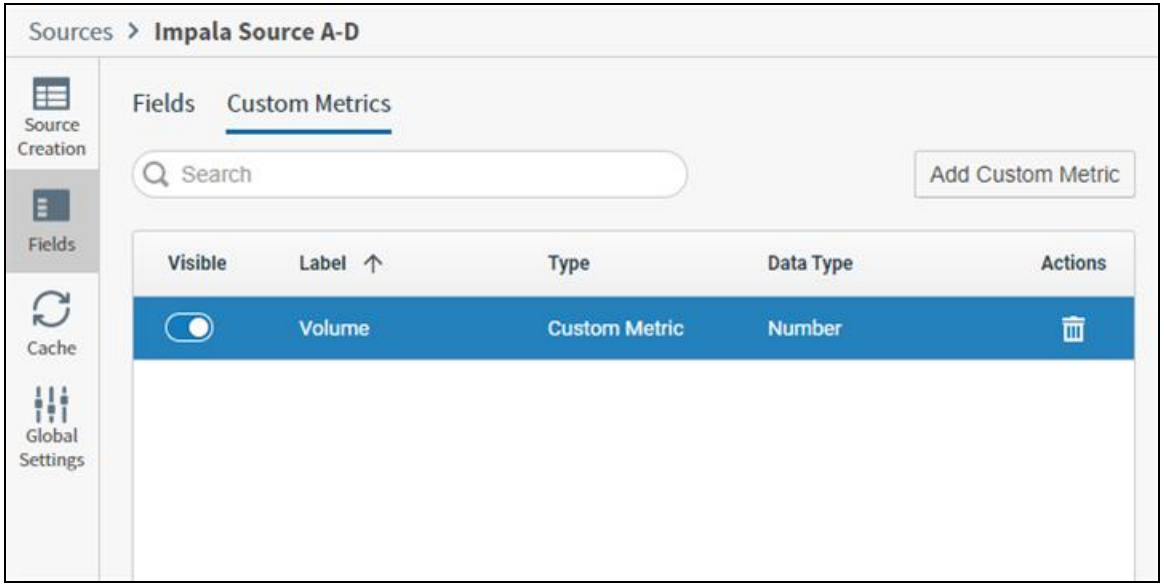
Access the Custom Metrics Editor in the following ways:

- Access The Custom Metrics Editor From A Data Source Configuration
- Access The Custom Metrics Editor From The Metric Selection Dialog
- Access The Custom Metrics Editor From The Group Selection Dialog
- Access The Custom Metrics Editor From The Color Sidebar
- Access The Custom Metrics Editor From The Filters Sidebar

Access the Custom Metrics Editor from a Data Source Configuration

Access the Custom Metrics Editor from a data source configuration

1. Edit the data source configuration in the UI. See Edit A Data Source Configuration.
2. Select the **Fields** tab.
3. Select **Custom Metrics** to open the Custom Metrics work area.



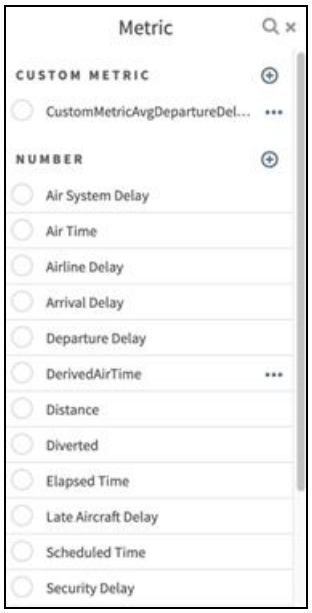
4. Select **Add Custom Metric** to add a custom metric. Select an existing custom metric to modify the metric.

   The Custom Metrics Editor appears.

Access the Custom Metrics Editor from the Metric Selection Dialog

Access the Custom Metrics Editor from the metric selection dialog of a visual

1. On a visual, select the metric label (y-axis label) to view the Metric selection dialog.



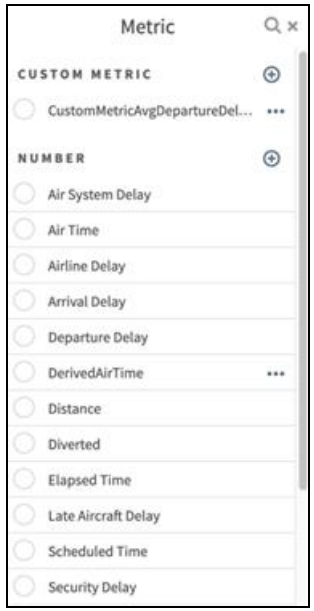
2. Select the  icon in the Number or Custom Metric section of the dialog. A menu opens with two options: **Add Derived Field** and **Add Custom Metric**.

3. Select **Add Custom Metric** to access the Custom Metrics Editor and create a custom metric.

You can also access the Custom Metrics Editor when you edit any custom metric listed on the Metric selection dialog. See the following steps.

Edit a custom metric

1. On a visual, select the metric label (y-axis label) to view the Metric selection dialog.



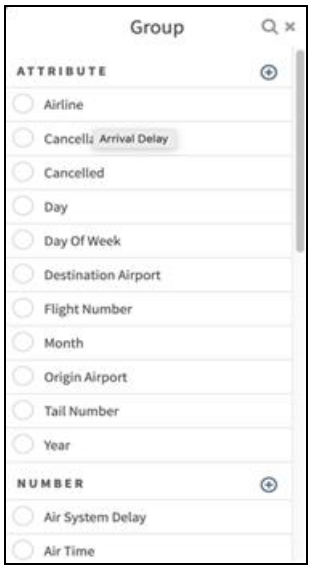
2. Locate a custom metric listed on the Metric selection dialog and select the ellipsis (•••) next to it. The following menu appears:

3. Select **Edit** on the menu to edit the custom metric. The Custom Metrics Editor appears.

Access the Custom Metrics Editor from the Group Selection Dialog

Access the Custom Metrics Editor from the Group selection dialog of a visual

1. On a visual, select the group label (x-axis label) to view the Group selection dialog.



2. Select  in the Attributes, Number or Time section of the dialog. A menu opens with two options: **Add Derived Field** and **Add Custom Metric**.

3. Select **Add Custom Metric** to access the Custom Metrics Editor and create a custom metric.

Access the Custom Metrics Editor from the Color Sidebar

Access the Custom Metrics Editor from the Color sidebar of a visual

1. Select ••• and then **Color** from the visual drop-down menu to access the Color sidebar.

2. Select the **Color Attribute** box on the Color sidebar. A list of attribute, number, and time fields you can select for the color attribute appears in the Color sidebar.

3. Select ⊕ in the Attribute, Number, or Time fields section of the sidebar. A menu opens with two options: **Add Derived Field** and **Add Custom Metric**.

4. Select **Add Custom Metric** to access the Custom Metrics Editor and create a custom metric.

You can also access the Custom Metrics Editor when you edit any custom metric listed on the Color sidebar.

1. Select ••• and then Color from the menu to access the Color sidebar.

2. Select the **Color Metric** box on the Color sidebar. A list of fields you can select for the color metric appears in the Color sidebar.

3. Locate a custom metric listed on the Color sidebar and select the ellipsis (•••) next to it. The following menu appears:

4. Select **Edit** on the menu to edit the custom metric. The Custom Metrics Editor appears.

Access the Custom Metrics Editor from the Filters Sidebar

Access the Custom Metrics Editor from the Filters sidebar of a visual or dashboard

1. Access the visual filter sidebar or the dashboard filter sidebar.

i. To access the visual filter sidebar, select the filter icon ( ) on the visual or select on the sidebar menu. If the visual is in a dashboard, select **Settings** from the visual menu ( ), then select the filter icon on the sidebar menu.

ii. To access the dashboard filter sidebar, select the dashboard filter icon ( ) next to the dashboard title. The dashboard-level filter icon is available only when all the visuals are from the same data source.

   The Filters sidebar appears.

2. Select the add icon at the top of the Attribute, Number, or Time sections on the Row or Group tab. A menu opens with two options: **Add Derived Field** and **Add Custom Metric**.

3. Select **Add Custom Metric** to access the Custom Metrics Editor and create a custom metric.

You can also access the Custom Metrics Editor when you edit any custom metric listed on the Filters sidebar. See the following steps.

Edit a custom metric

1. Select the filter icon to access the Filters sidebar for a visual or dashboard.

2. Select **Add Filter**.

3. Locate a custom metric listed on the Filters sidebar and select the ellipsis ( ) next to it. The following menu appears:

4. Select **Edit** on the menu to edit the custom metric. The Custom Metrics Editor appears.

Create and Modify Custom Metrics

Custom metrics are created and modified using the Custom Metrics Editor.

Create or modify a custom metric

1. Access the Custom Metrics Editor in any of the ways described in Custom Metrics Editor.
2. Enter a label for the custom metric in the space labeled **Untitled Custom Metric**.

   > **Note:** If you are using non-Latin characters in your functions, the label for your custom metric should start with a letter or an underscore (_) symbol followed by one or more letters, numbers, underscore, or period characters. Symbols other than the underscore (_) or period (.) are not allowed.

3. Enter the expression for the custom metric in the editing space. Expressions should follow standard mathematical and logical syntax and are resolved using the standard order of operations. You can manually key in expressions or you can select the elements of your expressions from the menus at the left. The expression should not be assigned to a variable because the resolved value of the expression is assigned to the custom metric, which serves as the 'variable' to which the value is assigned. That is, enter `a / b` rather than `x = a / b`.

   Row-level expressions and aggregate functions can be used in custom metrics. In addition, custom metrics can be filtered (using date and time filter aggregate functions and SQL-like expressions).

4. To test your custom metric, select **Run**.

   The editor attempts to calculate your custom metric. Any errors are reported. Any results are shown in the Preview area of the Custom Metrics Editor.

5. When you are finished creating or modifying your custom metric definition, select **Save**.

Supported Aggregation Functions

Use aggregate functions in custom metrics. Custom metrics can also include the use of row-level functions, conditional CASE expressions, statistical functions, an expansive list of arithmetic functions, as well as FIRST and LAST values. In addition, they can be filtered (using date and time filter functions and SQL-like expressions).

Data can be aggregated using column, table, or window aggregation functions. Each is explained in the links below. Aggregation functionality is explained in the links below.

- Conditional CASE Expressions
- Supported Statistical Functions
- Arithmetic Functions
- Column Aggregation Functions
- Table Aggregation Functions
- Window Aggregation Functions
- Date And Time Filter Aggregation Functions

Column Aggregation Functions

Column aggregation functions aggregate data using all the data displayed on a visual. They group results in the same way that the visual itself groups its data. For example, if your visual shows data grouped by gender (male and female), then column aggregation functions return two results, one for males and one for females. Only data included in the visual by any filters that have been applied are included in the results.

The following table describes the supported column aggregation functions.

Function	Parameter Type	Description
`AVG(<field>)`	numeric	Returns the average of a column (field), grouped in the same manner as the visual data.
`COUNT(<field>)`	any	Returns the numeric count of values in a column (field), grouped in the same manner as the visual data. This aggregate function normally ignores null values for the specified field. Consequently, the result of this aggregate function may not be the same as the actual number of records in the data. Use the wildcard character (*) for `<field>` to include null values for the field in the count.
`COUNTD(<field>)`	any	Returns the numeric count of unique values in a column (field), grouped in the same manner as the visual data. This aggregate function normally ignores null values for the specified field. Consequently, the result of this aggregate function may not be the same as the actual number of records in the data. Use the wildcard character (*) for `<field>` to include null values for the field in the count.
`MAX(<field>)`	numeric	Returns the maximum value of a column (field), grouped in the same manner as the visual data.
`MIN(<field>)`	numeric	Returns the minimum value of a column (field), grouped in the same manner as the visual data.
`SUM(<field>)`	numeric	Returns the sum of a column (field), grouped in the same manner as the visual data.
`FIRST_VALUE(<field>)`	any	Returns the first value of a given expression in the group, as when the expression is sorted in the ascending order.
`LAST_VALUE(<field>)`	any	Returns the last value of a given expression in the group, as when the expression is sorted in the ascending order.
`STDDEV_POP(<field>)`	numeric	Computes the population standard deviation and returns the square root of the population variance.
`STDDEV_SAMP(<field>)`	numeric	Computes the cumulative sample standard deviation and returns the square root of the sample variance.

Function	Parameter Type	Description
`VAR_POP(<field>)`	numeric	Returns the population standard variance of a given expression.
`VAR_SAMP(<field>)`	numeric	Returns the sample variance of a given expression.
`MEDIAN(<field>)`	numeric	Computes the median value across the group.

Example

Suppose you have the following fields and data in a data source:

name	gender	city	earned	spent
Alan	M	Rockville	$10	$2
Bob	M	Rockville	$8	$3
Carol	F	Rockville	$5	$5
Darlene	F	Reston	$4	$6
Ed	M	Reston	$2	$8

To use this data set to create a custom metric called `Leftover` (a group's leftover money), use the following formula.

```
SUM( earned ) - SUM( spent )
```

If you used the `Leftover` custom metric in a visual grouping by gender using the data above, you would get the results shown below.

Gender	Leftover
F	-2
M	7
Total	5

Males have $7, derived from (10+8+2) - (2+3+8). Females have -$2 left over, derived from (5+4) - (5+6). If you used the same custom metric in a visual grouping by city, you would see Rockville having $13, from (10+8+5) - (2+3+5), and Reston having -$8, from (4+2) - (6+8).

Table Aggregation Functions

Table aggregation functions are broader in scope than column aggregation functions. Table aggregation functions use all data from a field and produce a single, ungrouped value. You typically do not use the result directly in a visual, since it is ungrouped.

For example, if you have sales records grouped by gender, a `TableSUM` custom metric returns the total sales of all records as one value, regardless of the group in consideration. The `TableSUM` result would include both the male and female data values. Consequently, males and females would appear to have the same sales if the `TableSUM` result was included in the visual.

Table aggregation functions are typically used to calculate percentages of a whole or average values.

The following table describes the supported table aggregation functions.

Function	Type	Description
`TableAVG(<field>)`	numeric	Returns the average of a column (field), regardless of how the visual is grouped.
`TableCOUNT(<field>)`	any	Returns the numeric count of values in a column (field), regardless of how the visual is grouped. This aggregate function normally ignores null values for the specified field. Consequently, the result of this aggregate function may not be the same as the actual number of records in the data. Use the wildcard character (*) for `<field>` to include null values for the field in the count.
`TableCOUNTD(<field>)`	any	Returns the numeric count of unique values in a column (field), regardless of how the visual is grouped. This aggregate function normally ignores null values for the specified field. Consequently, the result of this aggregate function may not be the same as the actual number of records in the data. Use the wildcard character (*) for `<field>` to include null values for the field in the count.
`TableMAX(<field>)`	numeric	Returns the maximum value of a column (field), regardless of how the visual is grouped.
`TableMIN(<field>)`	numeric	TableMIN returns the minimum value of a column (field), regardless of how the visual is grouped.
`TableSUM(<field>)`	numeric	TableSUM returns the sum of a column (field), regardless of how the visual is grouped.

Example

Suppose you have the following fields and data:

name	gender	city	earned	spent
Alan	M	Rockville	$10	$2
Bob	M	Rockville	$8	$3
Carol	F	Rockville	$5	$5

name	gender	city	earned	spent
Darlene	F	Reston	$4	$6
Ed	M	Reston	$2	$8

Using this data, you can create a custom metric containing individual earnings as a percentage of total earnings with the following formula:

```
SUM(earned) / TableSUM(earned) * 100
```

in which:

- `SUM(earned)` calculates the sum earnings for each individual.
- `TableSUM(earned)` calculates the total earnings of all records in the data.
- The quotient of `SUM(earned) / TableSUM(earned)` is multiplied by 100 to convert the result into a percentage.

Shown on a table using the example data above, the results would look like this:

Name	% of Whole
Alan	34.48
Bob	27.59
Carol	17.24
Darlene	13.79
Ed	6.90
Total	100

Window Aggregation Functions

Window aggregation functions are a middle case between column and table aggregation functions. They provide a snapshot or window into a subset of data, depending on the groupings used by the visual. Each window function such as `WindowSUM` or `WindowAVG` requires a numeric field to aggregate followed by a list of one or more attributes. The function aggregates the data and groups the results based on these attributes *if the attributes are present in the visual*. Attributes absent from the visual are ignored from the aggregation.

Derived fields can be used in window aggregation functions.

For example, an aggregation `WindowAVG( profits, gender, city )` returns the average profits in the data, grouped by gender and city *if gender and city are represented in the visual*. If gender happens to be absent from the visual, then it is dropped from the aggregation. Effectively, the average profits would then be grouped only by city.

The following table describes the supported window aggregation functions.

Function	Type	Description
`WindowAVG(<field>,<attr1>[,<attr2>]...)`	numeric	Returns the average of a column (field), grouped by the specified attributes.
`WindowCOUNT(<field>,<attr1>[,<attr2>]... )`	any	Returns the numeric count of values in a column (field), grouped by the specified attributes. This aggregate function normally ignores null values for the specified field. Consequently, the result of this aggregate function may not be the same as the actual number of records in the data. Use the wildcard character (*) for `<field>` to include null values for the field in the count.
`WindowCOUNTD(<field>,<attr1>[,<attr2>]...)`	any	Returns the numeric count of unique values in a column (field), grouped by the specified attributes. This aggregate function normally ignores null values for the specified field. Consequently, the result of this aggregate function may not be the same as the actual number of records in the data. Use the wildcard character (*) for `<field>` to include null values for the field in the count.
`WindowMAX(<field>,<attr1>[,<attr2>]...)`	numeric	Returns the maximum value of a column (field), grouped by the specified attributes.
`WindowMIN(<field>,<attr1>[,<attr2>]...)`	numeric	Returns the minimum value of a column (field), grouped by the specified attributes.
`WindowSUM(<field>,<attr1>[,<attr2>]...)`	numeric	Returns the sum of a column (field), grouped by the specified attributes.

Example

Suppose you have the following fields and data:

name	gender	city	earned	spent
Alan	M	Rockville	$10	$2
Bob	M	Rockville	$8	$3
Carol	F	Rockville	$5	$5
Darlene	F	Reston	$4	$6
Ed	M	Reston	$2	$8

To create a custom metric containing a group's contribution to just gender, rather than to the whole, use the following formula.

```
SUM(earned)/WindowSUM(earned,gender) * 100
```

Using this custom metric in a pivot table with the example data set shown above produces results similar to the ones shown below.

City	Gender	Volume	% of Each Gender's Earnings
Reston	F	1	44.44
	M	1	10
Rockville	F	1	55.56
	M	2	90
Total		5	100

In this pivot table, each city's total earnings (`SUM(earned)`) is shown as a percentage of each gender's total earnings. If gender had been absent from the visual, the cities' total earnings would have been shown as totals of the whole, rather than of each gender.

Date and Time Filter Aggregation Functions

To filter a custom metric using dates or times, you must already have a time attribute configured in your data source. The following date and time functions can only be used after WHERE in your custom metric.

Date field options use common time formats such as YTD, MMDDYYYY, and YoY.

The following date and time filter aggregation functions are supported.

Supported Date and Time Functions	
Function	Description
`DATE()`	**Deprecated.** Use `NOW()` instead.

`DateADD('<time_period>',<interval>,'<date>')`	**Deprecated.** Use `TIME_ADD` instead.
	For example, consider this `DateADD` specification:
	`DateADD('YEAR', 1, '2021-01-01')`
	Use this `TIME_ADD` specification instead:
	`TIME_ADD('YEAR', 1, '2021-01-01')`
	In a second example, consider this `DateADD` specification:
	`DateADD('MONTH', 1, DATE())`
	Use this `TIME_ADD` specification instead:
	`TIME_ADD('YEAR', 1, NOW())`

`DateSUB('<time_period>',<interval>,'<date>')`	**Deprecated.** Use `TIME_ADD` instead, specifying a negative number for `interval`. For example, consider this `DateADD` specification: `DateSUB('YEAR', 1, '2021-01-01')`

Supported Date and Time Functions	
Function	Description
	Use this `TIME_ADD` specification instead: `TIME_ADD('YEAR', -1, '2021-01-01'` TIME_ADD supports negative `interval` numbers for subtraction.
`NOW()`	Obtains the current date and time for the derived field. `NOW()` functionality is available when you use a supported connector. Set the `calculations.rle.now.function` property in the `query-engine.properties` file to `true` and restart the query engine microservice. See Query Engine Properties.
`PreviousPeriod(<offset>,<numPeriods>)`	This function is supported only within a TRANSFORM clause used for filtering the custom metric. The period returned is of the same length as the currently represented period, but not immediately prior to it. Instead, it counts back in `<numPeriods>` periods of time, measured in units named by `<offset>`. The following time `<offset>` values are supported: `YEAR`, `QUARTER`, `MONTH`, `WEEK`, `DAY`, `HOUR`, `MINUTE`, `SECOND`, `MILLISECOND`. See PreviousPeriod Function.
`TIME()`	**Deprecated.** Use `NOW()` instead.
`TIME_ADD('<time-period>',<interval>, <date-time-field>)`	Adds an interval value to the `<timepart>` of the date-time field: In the following example, 7 is added to the hour in the field called `date_time_field`: `TIME_ADD ('HOUR', +7, date_time_field)`

Date Filter Functions

Specific parameters are needed for the `DateADD` and `DateSub` functions. The following table describes them.

Parameter	Value
time_period	Supported time periods (with corresponding interval range): YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND, MILLISECOND
interval	Whole number integer. Negative numbers are supported for subtraction.
date	■ Current day operator: date() ■ Standard date and time formats supported include: ◦ yyyy-MM-dd HH:mm:ss ◦ MM/dd/yy hh:mm aa ◦ yyyy For all supported formats, see Convert Attributes to Time Fields in Data Source Field Specifications.

PreviousPeriod Function

The PreviousPeriod function is used for comparing data values between different time periods. This function can be used when you need to compare one time period to another of equivalent size for variance custom metrics. For example, comparing results from the current month to the previous month or the current week to the same week one year ago.

Note: Note that this function only works when the date field used in the formula is selected on the time bar.

To use this function, the TRANSFORM SQL-like expression must be used in the custom metric to convert the date range for a specified time attribute. For example:

```
SUM(Sales) TRANSFORM saledate = PreviousPeriod('month',1)
```

If the saledate time period is March 2015, the custom metric returns SUM(Sales) where the saledate is February 2015.

Note: If the data is grouped by the same field for which a PreviousPeriod transformation is performed and it is grouped by days but transformed by units of months, quarters, or years, null values are returned when the previous period does not have matching days for the current period. For example, if the current period is the month of March and PreviousPeriod('month',1) is used for the transformation, null values are produced for February 29-31, 2015 because those days are not valid days (although they are valid days for March 2015). Composer attempts to preserve the day-of-month correspondence between the two periods.

Specific parameters must be specified in PreviousPeriod functions. The following table describes them.

Parameter	Value
`offset`	The time granularity for the previous period (includes `YEAR`, `QUARTER`, `MONTH`, `WEEK`, `DAY`, `HOUR`, `MINUTE`, `SECOND`, `MILLISECOND`).
`numPeriods`	The argument specifying the number of periods to go back in time.

Supported SQL-Like Expressions

Composer's custom metrics support the following SQL-like expressions:

Expression	Description

WHERE	Use WHERE to filter by a condition. Data will only be included in the custom metric if the condition that follows is true. For example:
	`COUNT(weatherdelay) WHERE airportcode IN ('LAX', 'ORD', 'IAD')`
	Row-level functions and expressions can be used in WHERE clauses in custom metrics. In a custom metric, WHERE clauses allow you to specify a formula without first creating a derived field. The WHERE clause must be in the leftmost part of the custom metric expression, but it can be expressed with a row-level function or any of the aggregate functions available for custom metrics. In the following example, the total planned sales is calculated for men.
	`SUM(plannedsales) WHERE UPPER(gender) = 'MALE'`

AND	Use AND to form a conjunctive condition. Data is only included in the custom metric if it meets both of the conditions connected by AND. The following example calculates the sum of `deicing` only if the `broadphaseofflight` includes LANDING and the `airportcode` is YYZ. `SUM(deicing) WHERE broadphaseofflight IN 'LANDING' AND airportcode='YYZ'`
OR	Use OR to for a disjunctive condition. Data is included in the custom metric if it meets either of the conditions connected by OR. The following example calculates the sum of `deicing` if the `broadphaseofflight` includes LANDING or the `airportcode` is YYZ. `SUM(deicing) WHERE broadphaseofflight IN 'LANDING' OR airportcode='YYZ'`
BETWEEN...AND	Use BETWEEN to filter using a range of values. The following example counts the number of distinct records for `weatherdelay` that have `cancelledflight` counts between 2 and 10. `COUNTD(weatherdelay) WHERE cancelledflight BETWEEN 2 AND 10`
IN	Use IN to filter using a set of values. Data is included in the custom metric only if a data field matches one of the listed values. The following example calculates the sum of `weatherdelay` only for records in which the `airportcode` field is LAX, ORD, or IAD.

Expression	Description
	`SUM(weatherdelay) WHERE airportcode IN ('LAX','ORD','IAD')`
NOT IN	Use NOT IN to filter using a set of values. Data is included in the aggregation only if a data field does not match one of the listed values. The following example calculates the sum of `weatherdelay` only for records in which the `airportcode` field is *not* LAX, ORD, or IAD. `SUM(weatherdelay) WHERE airportcode NOT IN ('LAX','ORD','IAD')`
TRANSFORM	Use TRANSFORM to filter based on a derived date. To derive a date with TRANSFORM, you must already have a time attribute configured in your data source. The following example calculates the sum of `weatherdelay` only for records in which the `eventdate` is for the previous period. In other words, if the visual is examining two weeks of data for `weatherdelay`, this calculation will provide data about the two weeks prior to that. `SUM(weatherdelay) TRANSFORM eventdate=PreviousPeriod()` To work correctly, data must be available for the periods of time considered.

Delete Custom Metrics

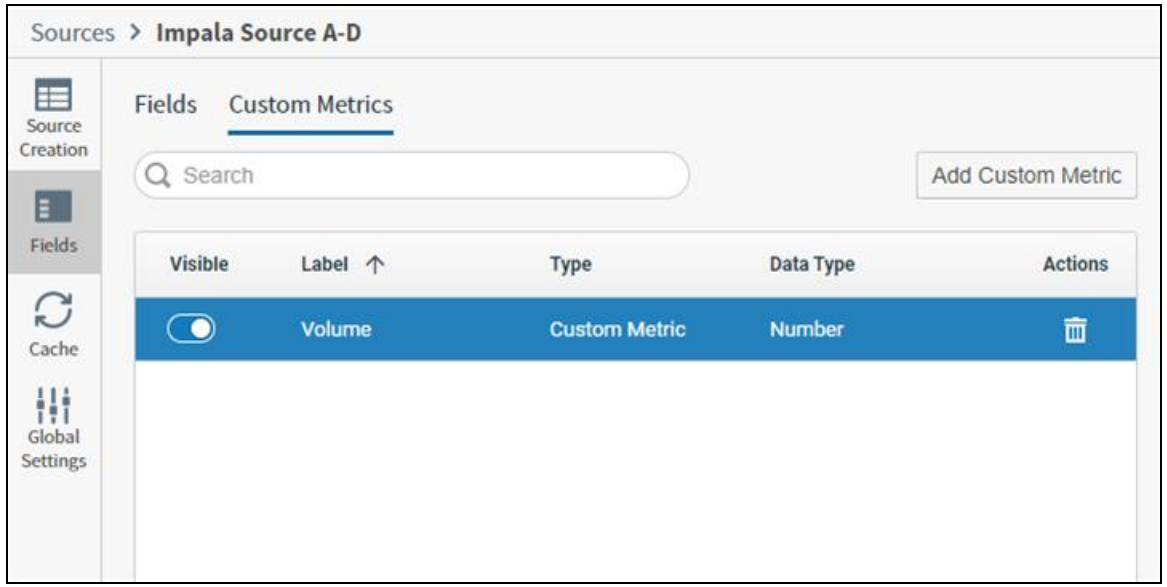
You can delete custom metrics from a data source in the Custom Metrics tab on the Fields tab, or by deleting the source entirely.

Important: If you delete a source, your custom metric can't be used by associated visuals, materialized views, actions, or chart defaults.

Note: If you try to delete a visual, filter snippet, dashboard, dashboard link, source, or source field, Composer displays an error message naming any objects dependent on the item you're trying to delete. You can delete the item after you've removed the association from the dependent object. See Fields Usage.

Delete custom metrics from a data source while editing the data source configuration

1. Edit the data source configuration in the UI. See Edit A Data Source Configuration.
2. Select the **Fields** tab.
3. Select **Custom Metrics** to open the Custom Metrics tab.



4. Locate the custom metric in the Custom Metrics table that you would like to remove.

5. Select the delete icon ( ) in the Actions column.

6. Select **Delete** in the pop-up confirmation dialog. The custom metric is deleted if it is not in use.

Apply Filters to Custom Metrics

You can apply filters to your custom metrics. This means that attributes from the data can be applied as filter parameters to further refine and narrow your results. Specifying a filter lets you perform a calculation within a predefined range, for example, determining sales results within a certain period of time or within a certain demographic.

This topic covers the application of filters to custom metrics and provides instructions covering the operators and syntax for integrating a filter into your custom metrics.

Filter Syntax

To create a custom metric that includes filters, operators or expressions are required.

When integrating a filter within a custom metric, keep in mind that Composer supports certain operators and follows a logical structure. In general, creating a filtered custom metric entails selecting the appropriate function variable, integrating the desired attribute or metric, using the appropriate filter operators and Date function, and entering a date in the correct format. Composer custom metrics use one of the following general structures:

```
<agg-function> WHERE <field> <operator> <values>
<agg-function> TRANSFORM <field> PreviousPeriod([offset,numPeriods])
```

where:

- `<agg-function>` is any of the aggregate functions described in Supported Aggregation Functions.
- WHERE or TRANSFORM is the appropriate SQL-like expression used for custom metric filtering. TRANSFORM is used specifically with the `PreviousPeriod` date and time filter aggregation function. WHERE and TRANSFORM filters are always applied to the broadest possible expression. For example, the following two expressions are both valid, but the first applies the filters to both SUMs, while the second expression applies the filter only to `SUM(Sales)`.

  ```
  SUM(Profit)/SUM(Sales) WHERE zipcode IN (90210,94107,92101)
  ```

  ```
  SUM(Profit)/(SUM(Sales) WHERE zipcode IN (90210,94107,92101))
  ```

  WHERE clauses can include row-level functions and expressions.

- `<field>` is a field (metric or attribute) in the data.

- `<operator>` is one of the operators shown in the following table:

Filters		
Capability	Operators	Notes

Filtering on attribute options	NULL check: IS NULL, IS NOT NULL
	Single value: =, !=, STARTS WITH, ENDS WITH, CONTAINS
	Multiple values: IN(), NOT IN()
		■ Non-numeric fields such as Name, Address and State can be used as a filter.
		■ References to an attribute in a field must be enclosed in single quotation marks. For example:
		`SUM(Sales) WHERE State = 'Florida'`
		■ Values specified for STARTS WITH, ENDS WITH, and CONTAINS are case-sensitive. The following example would show sales for states that include the lowercase letters `ia`, such as California and Louisiana (and others):
		`SUM(Sales) WHERE State CONTAINS 'ia'`

Filtering on date field options	Date range operator: BETWEEN	■ Standard date formats (see Convert Attributes To Time Fields In Data Source Field Specifications) are supported (such as 'yyyy-mm-dd', 'dd/mm/yyyy' including 'hh-mm-ss') . ■ Date ('01-01-2015') must be enclosed in single quotation marks. ■ Supported time periods are YEAR, MONTH, WEEK, or DAY
Filtering on numeric fields and custom metric options	>, <, =, !=	■ References to a numeric field must be enclosed in single quotation marks ■ References can be made to other custom metrics (using the custom metric label)

- `<value>` is an appropriate value, based on the requirements of the SQL-like expression. Values can be specific numbers or dates, or one of the date and time filter aggregation functions.

Examples

The following table provides examples of filtered custom metrics.

Filtered Custom Metric Examples	
Capability	Example

Filtering on Attribute options	`SUM(Sales) WHERE State = 'California'`
	
	`SUM(Profit)/SUM(Sales) WHERE zipcode IN (90210,94107,92101)`

Filtering on Date field options	`MIN(Margin) WHERE Sale_Date BETWEEN '2013-01-01' AND DateADD(MONTH, 6, '01-01-2014'))`

Filtering on numeric fields and custom metric options	`SUM(Sales)/(SUM(Sales) WHERE User_Income = '0 to $25000')`
	
	To calculate year over year growth:
	
	`((SUM(price) WHERE sale_date BETWEEN '2014-01-01' AND '2015-01-01') - (SUM(price) WHERE sale_date BETWEEN '2013-01-01' AND '2014-01-01')) / (SUM(price) WHERE sale_date BETWEEN '2012-01-01' AND '2013-01-01') * 100`

Comparing the difference between the current period to last period	Compare this period's deliveries to last period's deliveries: `SUM(deliveries) TRANSFORM delivery_date = PreviousPeriod()`
Using a row-level expression or function in a filter	`SUM(plannedsales) WHERE maxage - age > 30` `SUM(plannedsales) WHERE UPPER(gender) = 'MALE'`

Custom Metric Examples

The following commonly used custom metrics demonstrate custom metric syntax. These examples work with your data source only if your data source contains fields of the same name and type. Change the fields to make them work with your data sources.

The following custom metric is the sum of a metric (`totallatearrivals`) between a specific date (`2018-01-01`) and the current date:

```
SUM(totallatearrivals) WHERE eventdate BETWEEN '2018-01-01' AND DATE()
```

in which:

- `totallatearrivals` can be replaced by your own metric
- `eventdate` can be replaced by your own date attribute
- `2018-01-01` can be replaced by a different specific date

The following aggregated field formula is the sum of a metric (`totallatearrivals`) from the previous period:

```
SUM(totallatearrivals) TRANSFORM eventdate = PreviousPeriod()
```

in which:

- `totallatearrivals` can be replaced by your own metric
- `eventdate` can be replaced by your own date attribute

The following custom metric produces the difference between the current sum of a metric (`totallatearrivals`) and the sum of the same metric from the previous period:

```
SUM(totallatearrivals) - (SUM(totallatearrivals) TRANSFORM eventdate = PreviousPeriod())
```

in which:

- `totallatearrivals` can be replaced by your own metric

Conditional CASE Expressions

You can include the use of CASE expressions (singular or nested) in your custom metrics, much as you would row-level case and SQL case expressions. These capabilities include:

Conditions in `when`:

- Condition must be an aggregate-level expression.
- Can compare both metrics and groups. Group values, or any other non-numeric values can be used through `FIRST_VALUE` / `LAST_VALUE` functions.
- Use existing metrics from the request or add new metrics.
- Row-level expressions can be used inside aggregation functions.
- Use `AND` / `OR` operators to build complex conditions.
- `where` and `transform` clauses can be used in conditions to modify aggregate expressions.

To return results in `then`that include custom metric expressions, including calculated sub-queries.

- Must be an aggregate-level expression.
- All result values must be of the same type.
- Can be a numeric or non-numeric value.
- `where` and `transform` clauses can be used in conditions to modify aggregate expressions.

Additionally, you can nest case functions if needed, both for conditions and results. If you use case expressions as part of an arithmetic expression, it must be enclosed in parentheses.

Note: Only one result branch is returned from the expression, but all branches will be evaluated simultaneously, regardless of which condition is met first.

Metrics

In Composer, a metric is a numeric or attribute field that has been aggregated. Numeric fields can be aggregated using aggregation methods Distinct Count, Count, SUM, AVG, MIN, MAX, and LAST VALUE. An attribute field can only be aggregated using Distinct Count and Count. See Distinct Counts. See also Metric Aggregation Functions.

After a numeric or attribute field becomes a metric by aggregation, it can be used in the y-axis of visuals (that support a y-axis). The aggregation of a metric can also be changed in tables using the table context menu and the Table Settings sidebar. See Change Metric Aggregation In Tables.

Arithmetic Functions

Custom metrics support the following arithmetic functions in your aggregate functions.

The following table describes the supported arithmetic functions.

Function	Type	Description
`POWER(numeric1, numeric2)`	numeric	Returns numeric1 raised to the power of numeric2.
`MOD(numeric1, numeric2)`	numeric	Returns the remainder (modulus) of numeric1 divided by numeric2. The result is negative only if numeric1 is negative.
`SQRT(numeric)`	numeric	Returns the square root of numeric.
`LN(numeric)`	numeric	Returns the natural logarithm (base e) of numeric.
`LOG10(numeric)`	numeric	Returns the base 10 logarithm of numeric.
`EXP(numeric)`	numeric	Returns e raised to the power of numeric.
`CEIL(numeric)`	numeric	Rounds numeric up, returning the smallest integer that is greater than or equal to numeric.
`FLOOR(numeric)`	numeric	Rounds numeric down, returning the largest integer that is less than or equal to numeric.
`ROUND(numeric1, numeric2)`	numeric	Rounds numeric1 to optionally numeric2 (if not specified 0) places right to the decimal point.
`RAND(seed)`	numeric	Generates a random double between 0 and 1 inclusive, optionally initializing the random number generator with *seed*.
`SIGN(numeric)`	numeric	Returns the signum of *numeric*.
`SIN(numeric)`	numeric	Returns the sine of numeric.
`COS(numeric)`	numeric	Returns the cosine of numeric.
`TAN(numeric)`	numeric	Returns the tangent of numeric.
`COT(numeric)`	numeric	Returns the cotangent of numeric.
`ASIN(numeric)`	numeric	Returns the arcsine of numeric.
`ACOS(numeric)`	numeric	Returns the arc cosine of numeric.
`ATAN(numeric)`	numeric	Returns the arctangent of numeric.
`PI()`	numeric	Returns a value that is closer than any other value to pi.
`DEGREES(numeric)`	numeric	Converts numeric from radians to degrees.
`RADIANS(numeric) >`	numeric	Converts numeric from degrees to radians.

Metric Aggregation Functions

Composer provides a set of metric functions that are used to group (aggregate) data. The following aggregation methods can be selected for metrics in your visuals. See also Metrics.

Aggregation Function	What Is Returned
AVG	The average of the data values for the field. This function is available only for numeric fields.
DISTINCT COUNT	The total number of unique values for the field. This function is available only for attribute and numeric fields.
COUNT	The total number of values for the field. This function is available only for attribute and numeric fields.
MIN	The lowest value in all the data values for the field. This function is available only for numeric fields.
MAX	The highest value in all the data values for the field. This function is available only for numeric fields.
SUM	The total of all the data values for the field. This function is available only for numeric fields.
LAST VALUE	The last value in all the data values for the field, sorted by the time attribute selected for the time bar. If the latest date and time for the time attribute is exactly the same in multiple records, the last value for the field is the maximum value of the field in the records with the latest date and time. See LAST VALUE Examples. This function is available only for numeric fields.
NO AGGREGATION	No Aggregation is available if you group and sort by the same field. When you use No Aggregation, you can select the sort Order of as Alphabetical (A-Z) or Reverse Alphabetical (Z-A).

Suppose you have the following raw data:

Name	Gender	Age
Johnny	Male	10
Adam	Male	12
Mina	Female	11
Jenny	Female	13
Ann	Female	15

When this data is aggregated by gender, only two records (one for males and one for females) are returned and the aggregation must somehow determine what value to return for the age of the different genders. To do this, the aggregation requires input (using a metric function) about how the age should be returned. For example, if you elected to aggregate the data by gender and return the average age using the AVG metric function, the resulting data would be:

Gender	Age Returned	Aggregation Calculation	Aggregation Logic
Male	11	10 + 12 = 22 /2 = 11	Johnny's and Adam's ages are summed and divided by 2 (two males).

Gender	Age Returned	Aggregation Calculation	Aggregation Logic
Female	13	11 + 13 + 15 = 39 /3 = 13	Mina's, Jenny's, and Ann's ages are summed and divided by 3 (three females).

If you elected to aggregate the data by gender and return the minimum age using the MIN metric function, the resulting data would be:

Gender	Age Returned	Aggregation Logic
Male	10	Johnny's and Adam's ages are evaluated and the lower of the two ages is returned.
Female	11	Mina's, Jenny's, and Ann's ages are evaluated and the lower of the three ages is returned.

LAST VALUE Examples

The LAST VALUE examples in this section use the following data:

Record #	Gender	Country	Price	Items	Sale_Date
1	Male	US	10	6	2019-01-03
2	Male	US	20	5	2019-01-02
3	Male	UK	30	4	2019-01-01
4	Male	UK	40	3	2019-01-01
5	Male	UA	50	2	2019-01-02
6	Male	UA	60	1	2019-01-03
7	Female	US	1	7	2019-01-04
8	Female	US	11	6	2019-01-03
9	Female	US	21	5	2019-01-02
10	Female	UK	31	4	2019-01-01
11	Female	UK	41	3	2019-01-01
12	Female	UA	51	2	2019-01-02
13	Female	UA	61	1	2019-01-03
14	Female	UA	71	0	2019-01-04

Examples: Grouping By One Field

Suppose you aggregate this data by Gender and request that the last value for Price be returned based on the Sale_Date. The results would be:

Gender	Price Returned	Aggregation Logic
Male	60	In all the records for males, two records have the latest date (2019-01-03) - records #1 and #6. Therefore, the prices in both records are compared and the maximum price is returned. The result is 60 from record #6.
Female	71	In all the records for females, two records have the latest date (2019-01-04) - records #7 and #14. Therefore, the prices in both records are compared and the maximum price is returned. The result is 71 from record #14.

Suppose you aggregate this data by Country and request that the last value for Price be returned based on the Sale_Date. The results would be:

Country	Price Returned	Aggregation Logic
US	1	In all the records for the US, the record with the latest date is for the female with a sale date of 2019-01-04 (record #7). The price in that record is 1.
UK	41	All the records for the UK are for 2019-01-01. Therefore, the prices in all records are compared and the maximum price is returned. The result is 41 from record #11.
UA	71	In all the records for the UA, the record with the latest date is for a female with a sale date of 2019-01-04 (record #14). The price in that record is 71.

Example: Grouping By Two Fields

Suppose you aggregate this data by Gender and then by Country and request that the last value for Price be returned based on the Sale_Date. The results would be:

Gender	Country	Price Returned	Aggregation Logic
Male	US	10	The two records for US males are compared and the latest record has a sale date of 2019-01-03 (record #1). The price in that record is 10.
Male	UK	40	The two records for UK males have the same sale dates (2019-01-01). Therefore, the prices in all UK male records are compared and the maximum price is returned. The result is 40 from record #4.
Male	UA	60	The two records for UA males are compared and the latest record has a sale date of 2019-01-03 (record #6). The price in that record is 60.
Female	US	1	The three records for US females are compared and the latest record has a sale date of 2019-01-04 (record #1). The price in that record is 1.
Female	UK	41	The two records for UK females have the same sale dates (2019-01-01). Therefore, the prices in all UK female records are compared and the maximum price is returned. The result is 41 from record #11.

Gender	Country	Price Returned	Aggregation Logic
Female	UA	71	The three records for UA females are compared and the latest record has a sale date of 2019-01-03 (record #1). The price in that record is 10.

Example: Grouping By Two LAST VALUE Metrics

Suppose you aggregate this data by Gender and request that the last value for Price and the last value for Items be returned based on the Sale_Date. The results would be:

Gender	Price Returned	Items Returned	Aggregation Logic
Male	60	6	In all the records for males, two records have the latest date (2019-01-03) - records #1 and #6. The prices and item counts in both records are compared and the maximum price and item count are returned. The returned results are a price of 60 from record #6 and 6 items from record #1.
Female	71	7	In all the records for females, two records have the latest date (2019-01-04) - records #7 and #14. The prices and item counts in both records are compared and the maximum price and item count are returned. The returned results are a price of 71 from record #14 and 7 items from record #7.

Interpolation for User Attributes in Expressions

You can use interpolation in Composer to incorporate user attributes to customize and provide a personalized user experience based on those attributes.

The syntax of incorporating these attributes is the same as elsewhere in your software: `${attribute_name|default_value}`. When you use these special markers in your expressions, it is replaced with the user attribute value of the user using your software or the default value if that attribute is blank: `${User.department|'sales'}` and `${User.composerUserName|'default'}`.

You can use user attributes in interpolation in:

Derived Fields: Change parts of the expression based on the user who is opening a dashboard.

Custom Metrics: Change metric-level calculations based on the user, such as using a different aggregation function with the same custom metric.

How Interpolations in Expressions are Processed

Important: You must include a default value in each interpolation marker when you use interpolation in expressions.

All special markers in your expression are replaced by the value of referenced user's attribute. If the attribute is blank for the user, or they don't have one, the default value you defined is used instead. After the substitution takes place, the resulting expression must meet syntax rules for validity.

No special processing is applied to the user attribute values. If a value contains a textual value, Composer will treat it as a string literal only if the resulting expression wraps this textual value in quote marks. For example, the value itself must contain quotes(`'New York'`), or the interpolation marker must be quoted (`'${User.city|Washington}'`). This also applies to other types of values, including numbers and timestamps.

Additionally, no special processing is applied to multi-valued user attributes. If you add a user attribute value that contains several values separated by commas or any other separators, these values are put in the place of the interpolation marker as-is. You can use this, for example, as a list of function arguments.

Interpolation Processing Differences When Used in Expressions

When you use interpolation in expressions, the requirements are different than when use din other places in Composer. The mandatory default value is required in interpolation markers to ensure you always pass a valid expression, regardless of the existence of the current user's attribute. This prevents confusing errors when these expressions are used in some parts of the dashboard and visuals configuration. If you use an interpolation marker in a case where an empty value is acceptable, you can leave the default section empty: (`${User.attribute|}`).

Interpolation in Expressions - Limitations

You need to keep several limitations in mind when writing an expression.

- The data type of the whole expression must not change based on the substituted value. For example, if your expression returns a `NUMBER` value, it must do so if it uses the user attribute value or the default value. If your expression doesn't do this, an error is returned.

- A type of expression used can't be changed.
  - If you create a row-level expression, such as a derived field, the result should remain a row-level expression.
  - If you create an aggregate-level expression, such as a custom metric, the result should remain an aggregate-level expression.

- Secure user attributes can't be used; when an attribute is marked as secure, the default value is substituted by Composer instead.

Interpolation in Expressions - Default Values

Important: You must include a default value in each interpolation marker when you use interpolation in expressions.

When you build your expression, you can use a default value that is a different type than the values expected in the user attribute. For example, user attributes can reference data source fields, but the default value can be just a string or number literal.

```
concat('Hello ', ${User.nameField|'World'})
```

The main requirement is that your expression is valid in both cases. In the example above, the function must accept field references and string literals as arguments.

For more information, see: Use Interpolations in Expressions

Use Interpolations in Expressions

Simple Interpolation Expressions

The simplest way you can use interpolation in expressions is to substitute different textual and numeric literal values as parts of function calls or in arithmetic expressions.

- `sum(sales)*${User.saleCoefficient|1.0}`
- `concat(field, '${User.name|John}')`

In a more advanced scenario, substitute parts of the expression itself, such as a function name, or an operator:

- `${User.aggFunction|sum}(sales)` - The `User.aggFunction` can include values such as `min`, `max`, and so on.
- `sales ${User.operator|*} 2` - The `User.operator` can include values such as `*`, `/`, and so on.

Multiple Interpolation Expressions

User attribute values aren't limited to holding only one component of an expression, such as literals, functions, and operator. Your user attribute values can contain bigger parts, up to a full expression.

In this example, a user attribute contains a `WHERE` clause to restrict the data a user can see:

```
sum(sales) ${User.restriction|}
```

The `User.restriction` for some users can contain a value such as `WHERE transaction_date between '2023-01-01 00:00:00' and NOW()`

Consequently, the metric value is restricted for this user, while other users see a metric that includes the whole span of time that includes the data.

Interpolation in Expressions - Limitations

You need to keep several limitations in mind when writing an expression.

- The data type of the whole expression must not change based on the substituted value. For example, if your expression returns a `NUMBER` value, it must do so if it uses the user attribute value or the default value. If your expression doesn't do this, an error is returned.

- A type of expression used can't be changed.
  - If you create a row-level expression, such as a derived field, the result should remain a row-level expression.
  - If you create an aggregate-level expression, such as a custom metric, the result should remain an aggregate-level expression.
- Secure user attributes can't be used; when an attribute is marked as secure, the default value is substituted by Composer instead.

Interpolation in Expressions - Default Values

Important: You must include a default value in each interpolation marker when you use interpolation in expressions.

When you build your expression, you can use a default value that is a different type than the values expected in the user attribute. For example, user attributes can reference data source fields, but the default value can be just a string or number literal.

```
concat('Hello ', ${User.nameField|'World'})
```

The main requirement is that your expression is valid in both cases. In the example above, the function must accept field references and string literals as arguments.

Maintain Derived Fields

Derived fields are supported by certain connectors that come out-of-the-box in Composer. To see what functions are available, see Supported Row-Level Functions.

Support for this feature by connector is shown in the following table.

Key:Y - Supported; **N** - Not Supported; N/A - not applicable

Connector	Supported?	Notes
Amazon Redshift	**Y**	
Amazon S3	**Y**	
Apache Drill	**Y**	
Apache Phoenix	**Y**	Apache Phoenix and Apache Phoenix Query Server connectors support row-level expressions (derived fields) with the following limitations: ■ The filter IS NULL does not work properly on grouped fields. ■ The LOCATE text row-level function only supports a constant as a argument. ■ A COALESCE conditional row-level function specified with and empty argument does not work properly. ■ If the CASE conditional row-level function returns a null value as a an argument of another function, a NullPointerException may occur. ■ The LPAD and RPAD text row-level functions are not supported.
Apache Phoenix Query Server (QS)	**Y**	
Apache Solr	**N**	
BigQuery	**Y**	If you need to access a BigQuery partition, explicitly include an alias for the built in partition column in your select clause, such as `select *, _PARTITIONTIME as pt from projectId.datasetId.tableId`.
Business Central Jet	**Y**	
Cloudera Impala	**Y**	
Cloudera Search	**N**	
Couchbase	**Y**	

Connector	Supported?	Notes
Dremio	N	
Elasticsearch 7.0	Y	
Elasticsearch 8.0	Y	
File Upload	Y	
HDFS	Y	
Hive	Y	
Jira	Y	
MemSQL	Y	
Microsoft SQL Server	Y	
MongoDB	Y	MongoDB connectors support derived fields with some exceptions. See the discussion in Manage The MongoDB Connector.
MySQL	Y	
Oracle	Y	
PostgreSQL	Y	
Python	Y	
Real Time Sales	N/A	
Salesforce	Y	
SAP Hana	Y	
SAP S/4HANA	Y	
SAP IQ	Y	
Spark SQL	Y	
Snowflake	Y	
Teradata	Y	
TIBCO DV	Y	
Trino	N	
File Upload (Upload API)	Y	
Vertica	Y	

A derived field is an in-memory column for your data table that is populated with results from calculations performed on data already in your table. You can create derived fields using row-level expressions that are built using row-level functions.

These calculations are performed at the level of a row, that is, a record, and do not include other data from your table that is outside of that particular row. If a source supports derived fields, then you can use them as arguments for aggregate functions when creating other calculations.

Derived fields can be created from other derived fields.

Examples

Your data source has records that list the revenue generated and the term of employment but does not have an average of the two. You can use a derived field to create an average of the two for each record. Use the following formula:

```
(revenue/lengthofemployment)
```

Your data source continues values that have been brought in as text strings. In order to cross-reference this data with the time values, you need to change the text to a numeric value. Use the following formula as a base:

```
TEXT_TO_NUM (LTRIM (Field_A, '$')) SUM(TEXT_TO_NUM(SUBSTRING("$124456.00", 2, 10)))
```

Your data source contains records that list the start of employment and termination of employment for your company. You want to find the differences between these time values to average out the length of employment. Use the following formula as a base:

```
TIME_DIFF (timePart, startTime : Time, endTime : Time) : Numeric
```

Composer supports row-level functions in derived fields. See Supported Row-Level Functions.

Derived fields can be hidden. See Hide Fields.

For information on maintaining derived fields, see the following links:

- Derived Field Editor
- Create And Modify Derived Fields
- Supported Row-Level Functions
- Delete Derived Fields
- Hide Fields

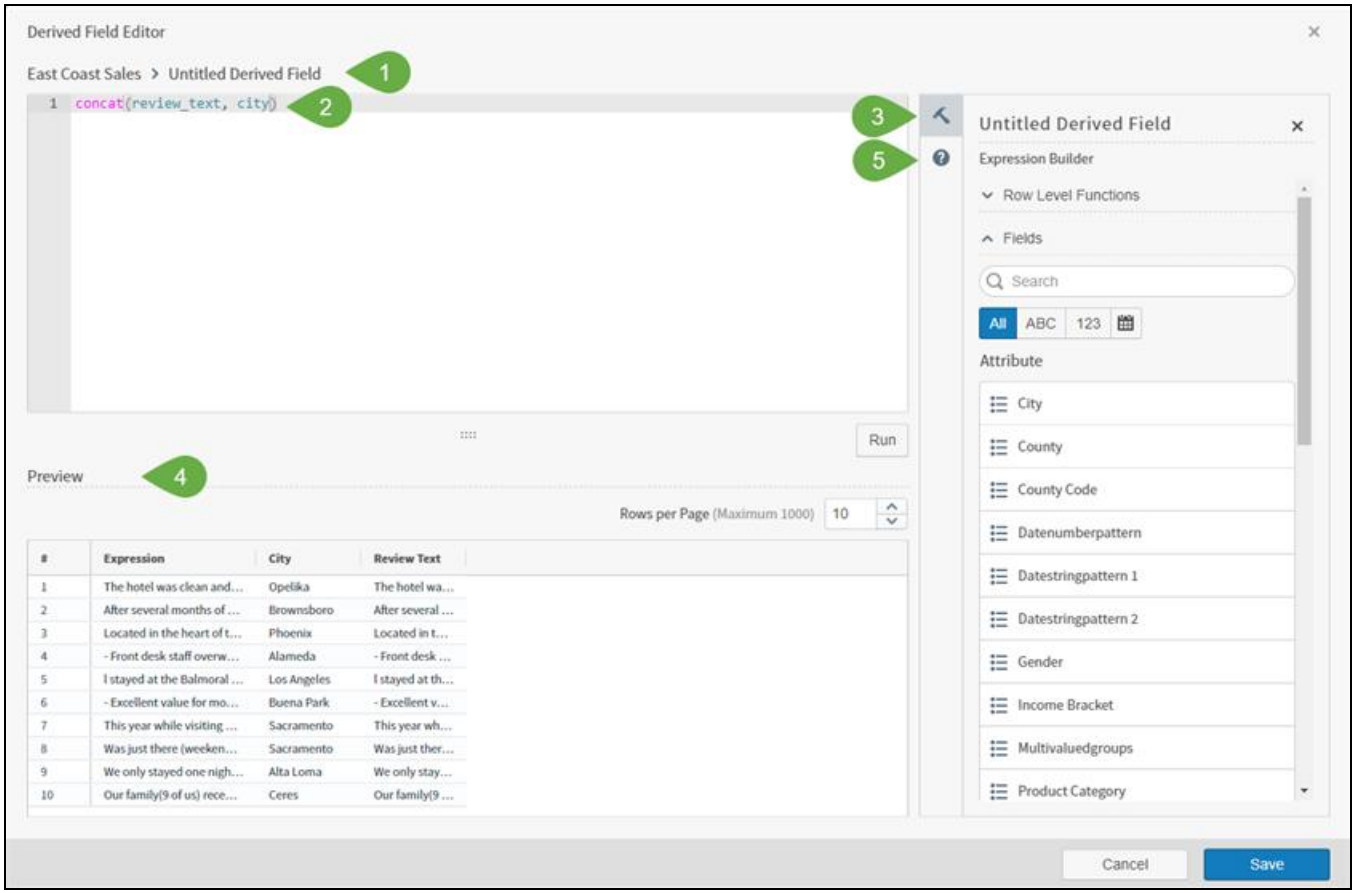
Derived Field Editor

Composer provides a Derived Field Editor to help you create and test derived fields for a data source.

To define derived fields for a data source, you must have:

- Read permission for the data source and the Edit Calculations privilege, or
- Write permission for the data source.

The Derived Field Editor:



The numbered regions are:

1. Derived Field Label: Mandatory, fewer than 255 characters long.
2. Editing space: Build your expression in this space. Syntax highlighting improves the readability of your expression.
3. Expression Builder: This tool includes Row Level Functions and Fields to help you build your derived field.
4. Preview space: Shows a preview of the results of your expression.
5. Calculation Help: Provides more detailed information about the types of expressions Composer supports.

Create a derived field by defining a formula composed of metrics and attributes that include row level functions. You can also include existing derived fields in your new derived field formula. Composer automatically assigns data types to your derived fields.

Derived Field Editor Features:

- Syntax highlighting improves the readability of your expressions to provide visual clues about the items being used and their validity.

  Parts of an expression are highlighted in different colors or with different text treatments:

  - function names, both row level and aggregate
  - fields and metrics
  - keywords such as CASE and IN
  - example parameters
  - date period constants such as `year`
  - values such as numbers and `true` or `false`
  - strings
  - arithmetic operators such as `+`

  References to fields that do not exist in the data source or are otherwise not usable in an expression are interpreted as values. These are shown in black, alerting you to possible typos or other issues.

- Row Level Functions and Fields sections always appear in the Expression Builder.
- Autocomplete functionality is included: Type two letters and a list of possible auto-completions appears.

a. Function completions provide the type of function, the name of the function, the description of the function, and an example of how the function is used, including parameters.

b. Field completions are available by typing either the field ID or the field label. The field ID appears in square brackets next to the field label when the two are different. Field completions provide the type of the field, the field label and ID, expressions for derived fields, and information about whether the field is hidden (as applicable).

c. Metric completions provide the metric label and the expression that will be inserted into the editor when the metric is selected.

- Syntax and validation errors are shown in the Preview area.
- When you're creating or editing an expression, select **Cancel** to return your expression to the initial state.
- Composer disables the **Save** button, preventing you from saving a derived field unless it has a successfully run expression.
- When you close the editor with unsaved changes, Composer displays a confirmation message.
  - Select **Cancel** to continue editing or changing the text of the label for your derived field.
  - Select **Discard** to discard your changes. If you are editing an existing item, your last saved version remains in Composer.
  - If you select **Discard** while creating a new expression, Composer returns you to the source work area.

Access the Derived Field Editor in the following ways:

- Access The Derived Field Editor From A Data Source
- Access The Derived Field Editor From The Metric Selection Dialog
- Access The Derived Field Editor From The Group Selection Dialog
- Access The Derived Field Editor From The Color Sidebar
- Access The Derived Field Editor From The Filters Sidebar

Access the Derived Field Editor from a Data Source

Access the Derived Field Editor from a data source

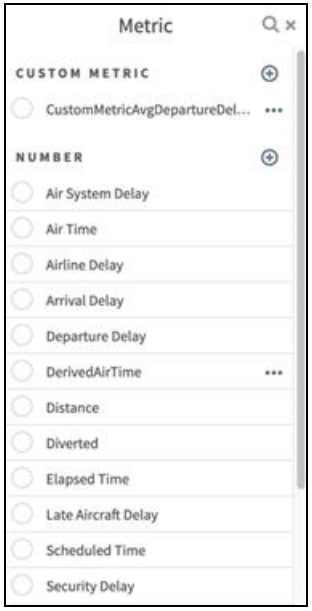
1. Edit the data source in the UI. See Edit A Data Source Configuration.
2. Select the **Fields** tab.
3. Select **Add Derived Field** at the top of the Fields table to add a derived field. To modify an existing derived field, select derived field then the edit () expression icon in the Settings side bar menu.

   The Derived Field Editor appears.

Access the Derived Field Editor from the Metric Selection Dialog

Access the Derived Field Editor from the metric selection dialog of a visual

1. On a visual, select the metric label (y-axis label) to view the Metric selection dialog.



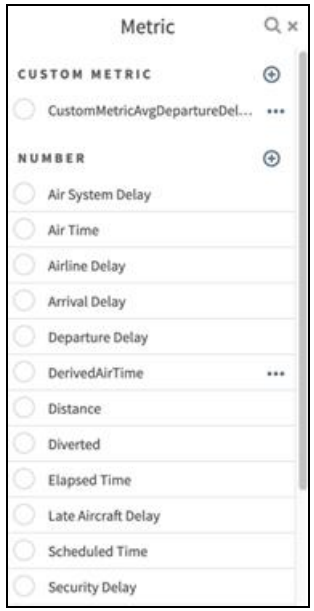
2. Select the  icon in the Number or Custom Metric section of the dialog. A menu opens with two options: **Add Derived Field** and **Add Custom Metric**.

3. Select **Add Derived Field** to access the Derived Field Editor and create a derived field.

You can also access the Derived Field Editor when you edit any derived field listed on the Metric selection dialog. See the following steps.

Edit a derived field

1. On a visual, select the metric label (y-axis label) to view the Metric selection dialog.



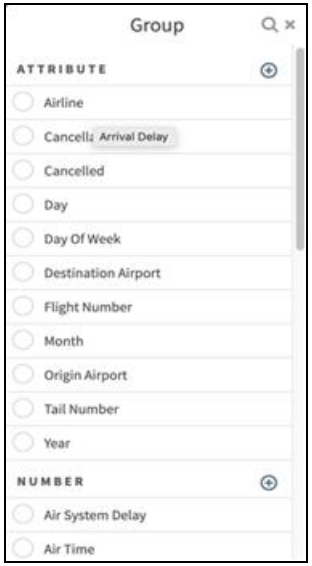
2. Locate a derived field listed on the Metric selection dialog and select the ellipsis (•••) next to it. The following menu appears:

3. Select **Edit** on the menu to edit the derived field. The Derived Field Editor appears.

Access the Derived Field Editor from the Group Selection Dialog

Access the Derived Field Editor from the Group selection dialog of a visual

1. On a visual, select the group label (x-axis label) to view the Group selection dialog.



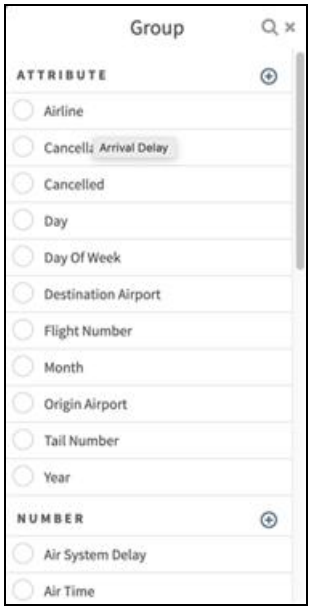
2. Select  in the Attributes, Number or Time section of the dialog. A menu opens with two options: **Add Derived Field** and **Add Custom Metric**.

3. Select **Add Derived Field** to access the Derived Field Editor and create a derived field.

You can also access the Derived Field Editor when you edit any derived field listed on the Group selection dialog. See the following steps.

Edit a derived field

1. On a visual, select the group label (x-axis label) to view the Group selection dialog.



2. Locate a derived field listed on the Group selection dialog and select the ellipsis (•••) next to it. The following menu appears:

3. Select **Edit** on the menu to edit the derived field. The Derived Field Editor appears.

Access the Derived Field Editor from the Color Sidebar

Access the Derived Field Editor from the Color sidebar of a visual

1. Select ••• and then **Color** from the visual drop-down menu to access the Color sidebar.

2. Select the **Color Attribute** box on the Color sidebar. A list of attribute, number, and time fields you can select for the color attribute appears in the Color sidebar.

3. Select ⊕ in the Attribute, Number, or Time fields section of the sidebar. A menu opens with two options: **Add Derived Field** and **Add Custom Metric**.

4. Select **Add Derived Field** to access the Derived Field Editor and create a derived field.

You can also access the Derived Field Editor when you edit any derived field listed on the Color sidebar. See the following steps.

Edit a derived field

1. Select ••• and then Color from the menu to access the Color sidebar.

2. Select the **Color Metric** box on the Color sidebar. A list of fields you can select for the color metric appears in the Color sidebar.

3. Locate a derived field listed on the Color sidebar and select the ellipsis (•••) next to it. The following menu appears:

4. Select **Edit** on the menu to edit the derived field. The Derived Field Editor appears.

Access the Derived Field Editor from the Filters Sidebar

Access the Derived Field Editor from the Filters sidebar of a visual, filter snippet, or dashboard

1. Access the filter sidebar or the dashboard filter sidebar.

i. To access the filter sidebar, select the filter icon ( ) on the visual or filter snippet, or select on the sidebar menu. If the visual is in a dashboard, select **Settings** from the menu ( ), then select on the sidebar menu.

i. To access the dashboard filter sidebar, select the dashboard filter icon ( ) next to the dashboard title. The dashboard-level filter icon is available only when all the visuals are from the same data source.

The Filters sidebar appears.

2. Select the add icon at the top of the Attribute, Number, or Time sections on the Row or Group tab. A menu opens with two options: **Add Derived Field** and **Add Custom Metric**.

3. Select **Add Derived Field** to access the Derived Field Editor and create a derived field.

You can also access the Derived Field Editor when you edit any derived field listed on the Filters sidebar. See the following steps.

Edit a derived field

1. Select the filter icon to access the Filters sidebar.

2. Select **Add Filter**.

3. Locate a derived field listed on the Filters sidebar and select the ellipsis ( ) next to it. The following menu appears:

4. Select **Edit** on the menu to edit the derived field. TheDerived Field Editor appears, depending on the field you selected.

Create and Modify Derived Fields

Derived fields are created and modified using the Derived Field Editor.

Create or modify a derived field

1. Access the Derived Field Editor in any of the ways described in Derived Field Editor.
2. Enter a name for the derived field in the space labeled **Untitled Derived Field**.

   Note: If you are using non-Latin characters in your functions, the name for your derived field should start with a letter or an underscore (_) symbol followed by one or more letters, numbers, underscore, or period characters. Symbols other than the underscore (_) or period (.) are not allowed.
3. Enter the expression for the derived field in the editing space. Expressions should follow standard mathematical and logical syntax and are resolved using the standard order of operations. You can manually key in expressions or you can select the elements of your expressions from the menus at the left. The expression should not be assigned to a variable because the resolved value of the expression is assigned to the custom metric, which serves as the *variable* to which the value is assigned. That is, enter `a / b` rather than `x = a / b`.

   Row-level expressions can be used in derived fields.
4. To test your derived field, select **Run**.

   The editor attempts to run your calculation. Any errors are reported. Any results are shown in the Preview area of the Derived Field Editor.
5. When you are finished with your derived field definition, select **Save**.
6. After creating or modifying the derived field and leaving the derived field editor, if you return to the Fields tab of the data source configuration, you can hide the field. See Hide Fields.

Delete Derived Fields

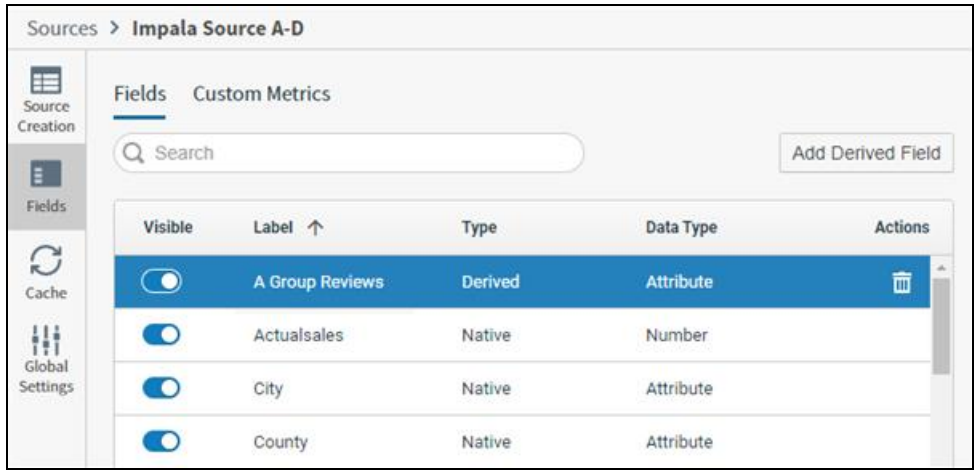
You can delete derived fields from a data source on the Fields tab, or by deleting the source entirely.

Important: If you delete a source, your derived field can't be used by associated visuals, materialized views, actions, or chart defaults.

Note: If you try to delete a visual, filter snippet, dashboard, dashboard link, source, or source field, Composer displays an error message naming any objects dependent on the item you're trying to delete. You can delete the item after you've removed the association from the dependent object. See Fields Usage.

Delete derived fields from a data source while editing the data source configuration

1. Edit the data source configuration in the UI. See Edit A Data Source Configuration.
2. Select the **Fields** tab.
3. Locate the derived field in the Fields table that you would like to remove.



4. Select the delete icon ( ) in the Actions column.
5. Select **Delete** in the pop-up confirmation dialog. The derived field is deleted if it is not in use.

Row-Level Expressions

A row-level expression is a mathematical expression involving a single record (row) in the data. They are used to calculate derived fields but can also be used in custom metrics and admin-defined functions.

Row-level expressions are created using attributes and metrics from the data and row-level functions.

Note: Be careful not to create row-level expressions using fields in a data source that have had their field data type changed. Doing so may generate errors for the row-level expression. Instead, use the original field (with its original field type) to create a derived field of the field type you need and then use the new derived field in your row-level expressions.

Supported Row-Level Functions

Use row-level functions in the row-level expressions you use to create the following calculation types:

- derived fields
- custom metrics
- admin-defined functions

The row-level functions described in this section are fully supported by all three types. For information about the aggregate functions and SQL-like expressions you can use in custom metrics, see Supported Aggregation Functions and Supported SQL-Like Expressions.

Row-level functions are divided into the following categories:

- Arithmetic Functions
- Conditional Functions
- Logical Functions
- Numerical Functions
- Relational Functions
- Text Functions
- Time Functions

Arithmetic Functions

Operator	Description
- (SUBTRACT)	Subtracts one numeric value from another
* (MULTIPLY)	Multiply one number by another
/ (DIVIDE)	Divide one number by another
+ (ADD)	Addition: adds two numbers

Supported Statistical Functions

Type	Parameter Type	Description
STDDEV_POP	numeric	Computes the population standard deviation and returns the square root of the population variance.
STDDEV_SAMP	numeric	Computes the cumulative sample standard deviation and returns the square root of the sample variance.
VAR_POP	numeric	Returns the population standard variance of a given expression.
VAR_SAMP	numeric	Returns the sample variance of a given expression.
MEDIAN	numeric	Computes the median value across the group.

Conditional Functions

Operator	Description
CASE	Use the CASE function is the same manner as standard SQL CASE functions. CASE can be used to list a series of conditions and return an appropriate value for the first condition that is met.
COALESCE	Use the COALESCE function in the same manner as standard SQL COALESCE functions. COALESCE can be used to return the first non-null value in a list of values. The COALESCE function also supports aggregate metrics with complex calculations using arithmetic functions (for example `COALESCE(max(sales) * 1.3, 0)`), so a default value can be used if null values are returned.

Logical Functions

Operator	Description
AND	Evaluates to TRUE if both boolean expressions are TRUE.
BETWEEN	Evaluates to TRUE if the operand is within a range.
IN	Evaluates to TRUE if the operand is equal to one of a list of expressions.
NOT	Reverses the value of any other Boolean operator.
OR	Evaluates to TRUE if either boolean expression is TRUE.

Numerical Functions

Function	Description	Example
CEIL	Returns the smallest integer value that is not less than the passed Value	CEIL(value : Numeric) : Numeric `CEIL(Field_A)`
FLOOR	Returns the largest integer value that is not greater than the passed value	FLOOR(value : Numeric) : Numeric `FLOOR(Field_A)`
NUM_TO_TEXT	Converts the numeric expression to text	NUM_TO_TEXT(value : Numeric) `NUM_TO_TEXT(Field_A)`
ROUND	Rounds a numeric value to the number of decimals specified	ROUND(Field_A, 0)
UNIX_TIME_TO_TIME	Converts the numeric expression to time	UNIX_TIME_TO_TIME(Field_Milliseconds /1000)

Relational Functions

Operator	Description
!=	Checks whether the values of two operands are not equal. If the values are not equal, then the condition is true.
<	Checks whether the value of the left operand is less than the value of the right operand. If it is, then the condition is true.
<=	Checks whether the value of the left operand is less than or equal to the value of the right operand. If it is, then the condition is true.
=	Checks whether the values of two operands are equal. If the values are equal, then the condition is true.
>	Checks whether the value of the left operand is greater than the value of the right operand. If it is, then the condition is true.
>=	Checks whether the value of the left operand is greater than or equal to the value of right operand. If it is, then condition is true.

You can also combine less than (<) and greater than (>) functions using logical AND processing in the same statement. For example, the following are valid statements:

```
saledate > 2020-10-28 AND saledate < 2020-10-30
```

```
saledate > 2020-10-28 AND saledate < 2020-10-30 AND state = 'CA'
```

In each of these examples, the individual relational functions must all be true for the full statement to be true. In the second example, the sale date must be greater than October 28, 2020 and less than October 30, 2020 and the sale must take place in the state of California.

Text Functions

Function	Description	Example
CONCAT	Returns a text that is the result of concatenating two or more text values.	`CONCAT(Field_FirstName, ' ,', Field_LastName)`
LENGTH	Returns the number of characters of the specified string.	`LENGTH(SUBSTRING('$12456.00', 2, 10))`
LOCATE	Finds the first occurrence of substring in a string, starting at position.	`LOCATE('Mr.', CONCAT(Field_FirstName, ' ,', Field_LastName), 0)`
LOWER	Returns the argument in lowercase.	`LOWER(SUBSTRING(Field_A, 0, 3 ))`
LPAD	Returns the text argument, left-padded with the text specified by *padString* to a length of Length characters.	`LPAD(SUBSTRING(Field_A, 0, 15), 3, 'abc')`
LTRIM	Returns a text value after removing leading blanks.	`LTRIM(SUBSTRING(Field_A, 0, 5))`
RPAD	Returns the Text argument, right-padded with the text specified by *padString* to a length of Length characters.	`RPAD(SUBSTRING(Field_A, 0, 15), 3, 'abc')`
RTRIM	Returns a text value after removing trailing blanks.	`RTRIM(SUBSTRING(Field_A, 0, 5))`
SUBSTRING	Returns the substring of String value which begins at position defined by Start and is Length characters long.	`SUBSTRING(Field_A, 4, 3)`
TEXT_TO_NUM	Converts the text string to numeric.	`TEXT_TO_NUM(LTRIM(Field_A))`
TEXT_TO_TIME	Converts the text expression to time according to the specified format. This function requires input in the form of an attribute or string field containing data that could be parsed as a time field and the format for the time field. Valid formats must be enclosed in single quotation marks and can only use the following syntax elements: `YYYY` (for years), `MM` (for months), `DD` (for days), `HH24` (for hours), `MI` (for	`TEXT_TO_TIME(Field_A,'YYYY-MM-DD HH24:MI:SS')`

Function	Description	Example
	minutes), `SS` (for seconds), and `MS` (for milliseconds). Separators in the syntax that are allowed are `-` (dashes), `:` (colons), `.` (periods), `/` (backslashes), and spaces.	
UPPER	Returns the argument in uppercase.	`UPPER(SUBSTRING(Field_A, 0, 3))`

Time Functions

The following time functions are supported. Valid values for `<timepart>` vary, based on the Composer connector selected, but can include YEAR, QUARTER, MONTH, WEEK, WEEK_OF_YEAR, WEEK_OF_MONTH, DAY, DAY_OF_YEAR, DAY_OF_MONTH, DAY_OF_WEEK, HOUR, MINUTE, SECOND, or MILLISECOND. Review the documentation for the Composer connector for any deviations from this list. Note that the WEEK_OF_YEAR function calculates the week from January 1, not from the week containing January 1.

Function	Description
EXTRACT	Extracts the `<timepart>` of the `<datetime>` field: `extract(<timepart>,<datetime>)`

NOW	Obtains the current date and time for the derived field. `NOW()` functionality is available when you use a supported connector. Set the `calculations.rle.now.function` property in the `query-engine.properties` file to `true` and restart the query engine microservice. See Query Engine Properties.
	`CASE WHEN [date column] = ''`
	`NOW()`
	`ELSE [date column]`
	`END`

TIME_ADD	Adds an interval value to the `<timepart>` of the `<datetime>` field:
	`time_add(<timepart>, <interval>, <datetime>)`
	In the following example, 7 is added to the hour in the field called `date_time_field`:
	`TIME_ADD (hour, +7, date_time_field)`

TIME_DIFF	Returns the time difference between two time fields in the unit you request: `time_diff('<timepart>', <end_date_field>, <start_date_field>)` In the following example, the difference between the values of the ENDDATE and STARTDATE fields is returned in days:

Function	Description
	`time_diff('DAY', ENDDATE, STARTDATE)`
TIME_TO_UNIX_TIME	Returns the value of a `<datetime>` field as a Unix time stamp: `time_to_unix_time(<datetime>)`
TRUNCATE_TIME	Rounds (Truncates) the `<datetime>` field value down to the granularity specified by `<timepart>`: `truncate_time(<timepart>,<datetime>)`

Fields Usage

Composer doesn't allow you to remove a field from a source definition if any objects rely on this field. For example, if your native field `field_1` is used in a derived field expression such as `year_to_time(field_1)`, you can't remove it from the source.

The table below is a list of objects where a field can be used.

Composer Object Type Referencing the Field	Can't Delete from Source Configuration	Can Delete from Source Configuration
derived field	If used in derived fields.	
custom metric	If used in custom metrics.	
global settings > time controls	If used by global defaults, such as time controls.	
global settings	If used by global defaults, such as time controls or global filters.	
row security	If used in row security.	
column security		If used in column security.
visual	If used by a visual.	
key set	If a key set exists for the data source.	
filter set	If a filter set exists for the data source.	
row level filters (dashboards and visuals)	If used in a row level filter.	
field link (cross-source link)	If used in a cross-source link as part of dashboard interactions.	
same source link		If the same source link is in a publisher or subscriber link.
actions	If used in an action.	
partition configuration	If used in a partition configuration.	
join configuration	If used in a join configuration.	

Note: If you try to delete a visual, filter snippet, dashboard, dashboard link, source, or source field, Composer displays an error message naming any objects dependent on the item you're trying to delete. You can delete the item after you've removed the association from the dependent object. See Fields Usage.