**Measure types**

* [Usage](https://cloud.google.com/looker/docs/reference/param-measure-types#usage)
* [Measure type categories](https://cloud.google.com/looker/docs/reference/param-measure-types#measure_type_categories)
* [List of type definitions](https://cloud.google.com/looker/docs/reference/param-measure-types#list_of_type_definitions)
* [average](https://cloud.google.com/looker/docs/reference/param-measure-types#average)
* [average\_distinct](https://cloud.google.com/looker/docs/reference/param-measure-types#average_distinct)

This page refers to the type parameter that is part of a [measure](https://cloud.google.com/looker/docs/reference/param-field-measure).

type can also be used as part of a [dimension](https://cloud.google.com/looker/docs/reference/param-field-dimension) or [filter](https://cloud.google.com/looker/docs/reference/param-field-filter), described on the [Dimension, filter, and parameter types](https://cloud.google.com/looker/docs/reference/param-dimension-filter-parameter-types) documentation page.

type can also be used as part of a dimension group, described on the [dimension\_group](https://cloud.google.com/looker/docs/reference/param-field-dimension-group) parameter documentation page.

**Usage**

view: view\_name {  
  measure: field\_name {  
    type: measure\_field\_type  
  }  
}

|  |  |
| --- | --- |
| **Hierarchy**  [View File](https://cloud.google.com/looker/docs/reference/param-view)  [view](https://cloud.google.com/looker/docs/reference/param-view-view)  [Field Type](https://cloud.google.com/looker/docs/reference/param-field)  type | **Possible Field Types**  Measure  **Accepts**  [A measure type](https://cloud.google.com/looker/docs/reference/param-measure-types#type_definitions) |

This page includes details about the various types that can be assigned to a measure. A measure can only have one type, and it defaults to string if no type is specified.

Some measure types have supporting parameters, which are described within the appropriate section.

**Measure type categories**

Each measure type falls into one of the following categories. These categories determine whether the measure type performs aggregations, the type of fields that the measure type can reference, and whether you can filter the measure type using the [filters](https://cloud.google.com/looker/docs/reference/param-field-filters) parameter:

* **Aggregate measures**: Aggregate measure types perform aggregations, such as sum and average. Aggregate measures can reference only dimensions, not other measures. This is the only measure type that works with the [filters](https://cloud.google.com/looker/docs/reference/param-field-filters) parameter.
* **Non-aggregate measures**: Non-aggregate measures are, as the name suggests, measure types that do not perform aggregations, such as number and yesno. These measure types perform simple transformations, and since they do not perform aggregations, can reference only aggregate measures or previously-aggregated dimensions. You cannot use the [filters](https://cloud.google.com/looker/docs/reference/param-field-filters) parameter with these measure types.
* **Post-SQL measures**: Post-SQL measures are special measure types that perform specific calculations after Looker has generated query SQL. They can reference only numeric measures or numeric dimensions. You cannot use the [filters](https://cloud.google.com/looker/docs/reference/param-field-filters) parameter with these measure types.

**List of type definitions**

| **Type** | **Category** | **Description** |
| --- | --- | --- |
| [average](https://cloud.google.com/looker/docs/reference/param-measure-types#average) | Aggregate | Generates an average (mean) of values within a column |
| [average\_distinct](https://cloud.google.com/looker/docs/reference/param-measure-types#average_distinct) | Aggregate | Properly generates an average (mean) of values when using denormalized data. *See the definition below for a complete description.* |
| [count](https://cloud.google.com/looker/docs/reference/param-measure-types#count) | Aggregate | Generates a count of rows |
| [count\_distinct](https://cloud.google.com/looker/docs/reference/param-measure-types#count_distinct) | Aggregate | Generates a count of unique values within a column |
| [date](https://cloud.google.com/looker/docs/reference/param-measure-types#date) | Non-aggregate | For measures that contain dates |
| [list](https://cloud.google.com/looker/docs/reference/param-measure-types#list) | Aggregate | Generates a list of the unique values within a column |
| [max](https://cloud.google.com/looker/docs/reference/param-measure-types#max) | Aggregate | Generates the maximum value within a column |
| [median](https://cloud.google.com/looker/docs/reference/param-measure-types#median) | Aggregate | Generates the median (midpoint value) of values within a column |
| [median\_distinct](https://cloud.google.com/looker/docs/reference/param-measure-types#median_distinct) | Aggregate | Properly generates a median (midpoint value) of the values when a join causes a fanout. *See the definition below for a complete description.* |
| [min](https://cloud.google.com/looker/docs/reference/param-measure-types#min) | Aggregate | Generates the minimum value within a column |
| [number](https://cloud.google.com/looker/docs/reference/param-measure-types#number) | Non-aggregate | For measures that contain numbers |
| [percent\_of\_previous](https://cloud.google.com/looker/docs/reference/param-measure-types#percent_of_previous) | Post-SQL | Generates the percent difference between displayed rows |
| [percent\_of\_total](https://cloud.google.com/looker/docs/reference/param-measure-types#percent_of_total) | Post-SQL | Generates the percent of total for each displayed row |
| [percentile](https://cloud.google.com/looker/docs/reference/param-measure-types#percentile) | Aggregate | Generates the value at the specified percentile within a column |
| [percentile\_distinct](https://cloud.google.com/looker/docs/reference/param-measure-types#percentile_distinct) | Aggregate | Properly generates the value at the specified percentile when a join causes a fanout. *See the definition below for a complete description.* |
| [running\_total](https://cloud.google.com/looker/docs/reference/param-measure-types#running_total) | Post-SQL | Generates the running total for each displayed row |
| [string](https://cloud.google.com/looker/docs/reference/param-measure-types#string) | Non-aggregate | For measures that contain letters or special characters (as with MySQL's GROUP\_CONCAT function) |
| [sum](https://cloud.google.com/looker/docs/reference/param-measure-types#sum) | Aggregate | Generates a sum of values within a column |
| [sum\_distinct](https://cloud.google.com/looker/docs/reference/param-measure-types#sum_distinct) | Aggregate | Properly generates a sum of values when using denormalized data. *See the definition below for a complete description.* |
| [yesno](https://cloud.google.com/looker/docs/reference/param-measure-types#yesno) | Non-aggregate | For fields that will show if something is true or false |
| int | Non-aggregate | REMOVED 5.4 Replaced by [type: number](https://cloud.google.com/looker/docs/reference/param-measure-types#number) |

average

type: average averages the values in a given field. It is similar to SQL's AVG function. However, unlike writing raw SQL, Looker will properly calculate averages even if your query's joins contain fanouts.

The [sql](https://cloud.google.com/looker/docs/reference/param-field-sql) parameter for type: average measures can take any valid SQL expression that results in a numeric table column, LookML dimension, or combination of LookML dimensions.

type: average fields can be formatted by using the [value\_format](https://cloud.google.com/looker/docs/reference/param-field-value-format) or [value\_format\_name](https://cloud.google.com/looker/docs/reference/param-field-value-format-name) parameters.

For example, the following LookML creates a field called avg\_order by averaging the sales\_price dimension, then displays it in a money format ($1,234.56):

measure: avg\_order {

type: average

sql: ${sales\_price} ;;

value\_format\_name: usd

}

average\_distinct

type: average\_distinct is for use with denormalized datasets. It averages the nonrepeated values in a given field, based on the unique values defined by the [sql\_distinct\_key](https://cloud.google.com/looker/docs/reference/param-field-sql-distinct-key) parameter.

This is an advanced concept which may be more clearly explained with an example. Consider a denormalized table like this:

| **Order Item ID** | **Order ID** | **Order Shipping** |
| --- | --- | --- |
| 1 | 1 | 10.00 |
| 2 | 1 | 10.00 |
| 3 | 2 | 20.00 |
| 4 | 2 | 20.00 |
| 5 | 2 | 20.00 |

In this situation you can see that there are multiple rows for each order. Consequently, if you added a simple type: average measure for the order\_shipping column, you would get a value of 16.00, even though the actual average is 15.00.

# Will NOT calculate the correct average

measure: avg\_shipping {

type: average

sql: ${order\_shipping} ;;

}

To get an accurate result, you can explain to Looker how it should identify each unique entity (in this case, each unique order) by using the sql\_distinct\_key parameter. This *will* calculate the correct 15.00 amount:

# Will calculate the correct average

measure: avg\_shipping {

type: average\_distinct

sql\_distinct\_key: ${order\_id} ;;

sql: ${order\_shipping} ;;

}

Please note that every unique value of sql\_distinct\_key must have just one corresponding value in [sql](https://cloud.google.com/looker/docs/reference/param-field-sql). In other words, the above example works because *every* row with an order\_id of 1 has the same order\_shipping of 10.00, *every* row with an order\_id of 2 has the same order\_shipping of 20.00, and so on.

type: average\_distinct fields can be formatted by using the [value\_format](https://cloud.google.com/looker/docs/reference/param-field-value-format) or [value\_format\_name](https://cloud.google.com/looker/docs/reference/param-field-value-format-name) parameters.

count

type: count performs a table count, similar to SQL's COUNT function. However, unlike writing raw SQL, Looker will properly calculate counts even if your query's joins contain fanouts.

type: count measures do not support the [sql](https://cloud.google.com/looker/docs/reference/param-field-sql) parameter, as a type: count measure performs table counts based on the table's primary key. If you want to perform a table count on a field other than the table's primary key, use a type: count\_distinct measure.

For example, the following LookML creates a field number\_of\_products:

view: products {

measure: number\_of\_products {

type: count

drill\_fields: [product\_details\*] # optional

}

}

It is very common to provide a [drill\_fields (for fields)](https://cloud.google.com/looker/docs/reference/param-field-drill-fields) parameter when defining a type: count measure, so that users can see the individual records that make up a count when they click on it.

When you use a [measure](https://cloud.google.com/looker/docs/reference/param-measure-types#count) of type: count in an Explore, the visualization labels the resulting values with the view name rather than the word "Count." To avoid confusion, we recommend pluralizing your view name, selecting **Show Full Field Name** under **Series** in the visualization settings, or using a view\_label with a pluralized version of your view name.

If you want to perform a COUNT (not a COUNT\_DISTINCT) on a field that is not the primary key, you can do so using a measure of [type: number](https://cloud.google.com/looker/docs/reference/param-measure-types#number). See the Community post [The Difference Between count and count\_distinct Measure Types](https://community.looker.com/technical-tips-tricks-1021/how-to-count-a-non-primary-key-30009) for more information.

You can add a filter to a measure of type: count using the [filters](https://cloud.google.com/looker/docs/reference/param-field-filters) parameter.

count\_distinct

type: count\_distinct calculates the number of distinct values in a given field. It makes use of SQL's COUNT DISTINCT function.

The [sql](https://cloud.google.com/looker/docs/reference/param-field-sql) parameter for type: count\_distinct measures can take any valid SQL expression that results in a table column, LookML dimension, or combination of LookML dimensions.

For example, the following LookML creates a field number\_of\_unique\_customers, which counts the number of unique customer IDs:

measure: number\_of\_unique\_customers {

type: count\_distinct

sql: ${customer\_id} ;;

}

You can add a filter to a measure of type: count\_distinct using the [filters](https://cloud.google.com/looker/docs/reference/param-field-filters) parameter.

date

type: date is used with fields that contain dates.

The [sql](https://cloud.google.com/looker/docs/reference/param-field-sql) parameter for type: date measures can take any valid SQL expression that results in a date. In practice, this type is rarely used, because most SQL aggregate functions do not return dates. One common exception is a MIN or MAX of a date dimension.

**Creating a max or min date measure with**type: date

If you want to create a measure of a maximum or minimum date, you might initially think it would work to use a measure of [type: max](https://cloud.google.com/looker/docs/reference/param-measure-types#max) or of [type: min](https://cloud.google.com/looker/docs/reference/param-measure-types#min). However, these measure types are compatible only with numerical fields. Instead, you can capture a maximum or minimum date by defining a measure of type: date and wrapping the date field that is referenced in the sql parameter in a MIN() or MAX() function.

Suppose you have a [dimension group](https://cloud.google.com/looker/docs/reference/param-field-dimension-group) of type: time, called updated:

dimension\_group: updated {

type: time

timeframes: [time, date, week, month, raw]

sql: ${TABLE}.updated\_at ;;

}

You can create a measure of type: date to capture the maximum date of this dimension group as follows:

measure: last\_updated\_date {

type: date

sql: MAX(${updated\_raw}) ;;

convert\_tz: no

}

In this example, instead of using a measure of type: max to create the last\_updated\_date measure, the MAX() function is applied in the sql parameter. The last\_updated\_date measure also has the convert\_tz parameter set to **no** to prevent double time zone conversion in the measure, since time zone conversion has already occurred in the definition of the dimension group updated. For more information, see the documentation on the [convert\_tz](https://cloud.google.com/looker/docs/reference/param-field-convert-tz) parameter.

In the example LookML for the last\_updated\_date measure, type: date could be omitted, and the value would be treated as a string, because string is the default value for type. However, you will get better filtering capability for users if you use type: date.

You may also notice that the last\_updated\_date measure definition references the ${updated\_raw} timeframe instead of the ${updated\_date} timeframe. Because the value returned from ${updated\_date} is a string, it is necessary to use ${updated\_raw} to reference the actual date value instead.

You can also use the [datatype](https://cloud.google.com/looker/docs/reference/param-field-datatype) parameter with type: date to enhance query performance by specifying the type of date data your database table uses.

**Creating a max or min measure for a datetime column**

Computing the maximum for a type: datetime column is a little different. In this case, you want to create a measure without declaring the type, like this:

measure: last\_updated\_datetime {

sql: MAX(${TABLE}.datetime\_string\_field) ;;

}

list

type: list creates a list of the distinct values in a given field. It is similar to MySQL's GROUP\_CONCAT function.

You do not need to include a [sql](https://cloud.google.com/looker/docs/reference/param-field-sql) parameter for type: list measures. Instead, you'll use the list\_field parameter to specify the dimension from which you want to create lists.

The usage is:

view: view\_name {  
  measure: field\_name {  
    type: list  
    list\_field: my\_field\_name  
  }  
}

For example, the following LookML creates a measure name\_list based on the name dimension:

measure: name\_list {

type: list

list\_field: name

}

Note the following for list:

* The list measure type does not support filtering. You cannot use the [filters](https://cloud.google.com/looker/docs/reference/param-field-filters) parameter on a type: list measure.
* The list measure type cannot be referenced using the [substitution operator ($)](https://cloud.google.com/looker/docs/sql-and-referring-to-lookml#substitution_operator_($)). You cannot use the ${} syntax to refer to a type: list measure.

**Supported database dialects for**list

For Looker to support type: list in your Looker project, your database dialect must also support it. The following table shows which dialects support type: list in the latest release of Looker:

max

type: max finds the largest value in a given field. It makes use of SQL's MAX function.

The [sql](https://cloud.google.com/looker/docs/reference/param-field-sql) parameter for measures of type: max can take any valid SQL expression that results in a numeric table column, LookML dimension, or combination of LookML dimensions.

Since measures of type: max are compatible only with numerical fields, you cannot use a measure of type: max to find a maximum date. Instead, you can use the MAX() function in the sql parameter of a measure of type: date to capture a maximum date, as shown previously in the examples in the [date](https://cloud.google.com/looker/docs/reference/param-measure-types#date) section.

type: max fields can be formatted by using the [value\_format](https://cloud.google.com/looker/docs/reference/param-field-value-format) or [value\_format\_name](https://cloud.google.com/looker/docs/reference/param-field-value-format-name) parameters.

For example, the following LookML creates a field called largest\_order by looking at the sales\_price dimension, then displays it in a money format ($1,234.56):

measure: largest\_order {

type: max

sql: ${sales\_price} ;;

value\_format\_name: usd

}

You cannot currently use type: max measures for strings or dates, but you can manually add the MAX function to create such a field, like this:

measure: latest\_name\_in\_alphabet {

type: string

sql: MAX(${name}) ;;

}

median

type: median returns the midpoint value for the values in a given field. This is especially useful when the data has a few very large or small outlier values that would skew a simple average (mean) of the data.

Consider a table like this:

| **Order Item ID** | **Cost** | **Midpoint?** |
| --- | --- | --- |
| 2 | 10.00 |  |
| 4 | 10.00 |  |
| 3 | 20.00 | Midpoint value |
| 1 | 80.00 |  |
| 5 | 90.00 |  |

For easy viewing, the table is sorted by cost but that does not affect the result. While the average type would return 42 (adding all the values and dividing by 5), the median type would return the midpoint value: 20.00.

If there is an even number of values, then the median value is calculated by taking the mean of the two values closest to the midpoint. Consider a table like this with an even number of rows:

| **Order Item ID** | **Cost** | **Midpoint?** |
| --- | --- | --- |
| 2 | 10 |  |
| 3 | 20 | Closest before midpoint |
| 1 | 80 | Closest after midpoint |
| 4 | 90 |  |

The median, the middle value, is (20 + 80)/2 = 50.

The median is also equal to the value at the 50th [percentile](https://cloud.google.com/looker/docs/reference/param-measure-types#percentile).

The [sql](https://cloud.google.com/looker/docs/reference/param-field-sql) parameter for type: median measures can take any valid SQL expression that results in a numeric table column, LookML dimension, or combination of LookML dimensions.

type: median fields can be formatted by using the [value\_format](https://cloud.google.com/looker/docs/reference/param-field-value-format) or [value\_format\_name](https://cloud.google.com/looker/docs/reference/param-field-value-format-name) parameters.

**Example**

For example, the following LookML creates a field called median\_order by averaging the sales\_price dimension, then displays it in a money format ($1,234.56):

measure: median\_order {

type: median

sql: ${sales\_price} ;;

value\_format\_name: usd

}

**Things to consider for**median

If you are using median for a field involved in a [fanout](https://cloud.google.com/looker/docs/glossary#fanout), Looker will attempt to use [median\_distinct](https://cloud.google.com/looker/docs/reference/param-measure-types" \l "median_distinct) instead. However, medium\_distinct is supported only for [certain dialects](https://cloud.google.com/looker/docs/reference/param-measure-types#supported_database_dialects_for_median_distinct). If median\_distinct is not available for your dialect, Looker returns an error. Since the median can be considered the 50th percentile, the error states that the dialect does not support distinct percentiles.

**Supported database dialects for**median

For Looker to support the median type in your Looker project, your database dialect must also support it. The following table shows which dialects support the median type in the latest release of Looker:

When there is a fanout involved in a query, Looker tries to convert the median into median\_distinct. This is only successful in [dialects that support median\_distinct](https://cloud.google.com/looker/docs/reference/param-measure-types#supported_database_dialects_for_median_distinct).

median\_distinct

Use type: median\_distinct when your join involves a fanout. It averages the nonrepeated values in a given field, based on the unique values defined by the [sql\_distinct\_key](https://cloud.google.com/looker/docs/reference/param-field-sql-distinct-key) parameter. If the measure does not have a sql\_distinct\_key parameter, then Looker tries to use the [primary\_key](https://cloud.google.com/looker/docs/reference/param-field-primary-key) field.

Consider the result of a query joining the Order Item and Order tables:

|  |  |  |
| --- | --- | --- |
| **Order Item ID** | **Order ID** | **Order Shipping** |
| 1 | 1 | 10 |
| 2 | 1 | 10 |
| 3 | 2 | 20 |
| 4 | 3 | 50 |
| 5 | 3 | 50 |
| 6 | 3 | 50 |

In this situation you can see that there are multiple rows for each order. This query involved a fanout because each order maps to several order items. The median\_distinct takes this into consideration and finds the median between the distinct values 10, 20, and 50 so you would get a value of 20.

To get an accurate result, you can explain to Looker how it should identify each unique entity (in this case, each unique order) by using the sql\_distinct\_key parameter. This will calculate the correct amount:

measure: median\_shipping {

type: median\_distinct

sql\_distinct\_key: ${order\_id} ;;

sql: ${order\_shipping} ;;

}

Please note that every unique value of sql\_distinct\_key must have just one corresponding value in the measure's [sql](https://cloud.google.com/looker/docs/reference/param-field-sql) parameter. In other words, the above example works because *every* row with an order\_id of 1 has the same order\_shipping of 10, *every* row with an order\_id of 2 has the same order\_shipping of 20, and so on.

type: median\_distinct fields can be formatted by using the [value\_format](https://cloud.google.com/looker/docs/reference/param-field-value-format) or [value\_format\_name](https://cloud.google.com/looker/docs/reference/param-field-value-format-name) parameters.

**Things to consider for**median\_distinct

The medium\_distinct measure type is supported only for [certain dialects](https://cloud.google.com/looker/docs/reference/param-measure-types#supported_database_dialects_for_median_distinct). If median\_distinct is not available for the dialect, Looker returns an error. Since the median can be considered the 50th percentile, the error states that the dialect does not support distinct percentiles.

**Supported database dialects for**median\_distinct

For Looker to support the median\_distinct type in your Looker project, your database dialect must also support it. The following table shows which dialects support the median\_distinct type in the latest release of Looker:

min

type: min finds the smallest value in a given field. It makes use of SQL's MIN function.

The [sql](https://cloud.google.com/looker/docs/reference/param-field-sql) parameter for measures of type: min can take any valid SQL expression that results in a numeric table column, LookML dimension, or combination of LookML dimensions.

Since measures of type: min are compatible only with numerical fields, you cannot use a measure of type: min to find a minimum date. Instead, you can use the MIN() function in the sql parameter of a measure of type: date to capture a minimum, just as you can use the MAX() function with a measure of type: date to capture a maximum date. This is shown previously on this page in the [date](https://cloud.google.com/looker/docs/reference/param-measure-types#date) section, which includes examples of using the MAX() function in the sql parameter to find a maximum date.

type: min fields can be formatted by using the [value\_format](https://cloud.google.com/looker/docs/reference/param-field-value-format) or [value\_format\_name](https://cloud.google.com/looker/docs/reference/param-field-value-format-name) parameters.

For example, the following LookML creates a field called smallest\_order by looking at the sales\_price dimension, then displays it in a money format ($1,234.56):

measure: smallest\_order {

type: min

sql: ${sales\_price} ;;

value\_format\_name: usd

}

You cannot currently use type: min measures for strings or dates, but you can manually add the MIN function to create such a field, like this:

measure: earliest\_name\_in\_alphabet {

type: string

sql: MIN(${name}) ;;

}

number

type: number is used with numbers or integers. A measure of type: number does not perform any aggregation, and is meant to perform simple transformations on other measures. If you are defining a measure based on another measure, the new measure must be of type: number to avoid nested-aggregation errors.

The [sql](https://cloud.google.com/looker/docs/reference/param-field-sql) parameter for type: number measures can take any valid SQL expression that results in a number or an integer.

type: number fields can be formatted by using the [value\_format](https://cloud.google.com/looker/docs/reference/param-field-value-format) or [value\_format\_name](https://cloud.google.com/looker/docs/reference/param-field-value-format-name) parameters.

For example, the following LookML creates a measure called total\_gross\_margin\_percentage based on the total\_sale\_price and total\_gross\_margin aggregate measures, then displays it in a percentage format with two decimals (12.34%):

measure: total\_sale\_price {

type: sum

value\_format\_name: usd

sql: ${sale\_price} ;;

}

measure: total\_gross\_margin {

type: sum

value\_format\_name: usd

sql: ${gross\_margin} ;;

}

measure: total\_gross\_margin\_percentage {

type: number

value\_format\_name: percent\_2

sql: ${total\_gross\_margin}/ NULLIF(${total\_sale\_price},0) ;;

}

The example above also uses the NULLIF() SQL function to remove the possibility of division-by-zero errors.

**Things to consider for**type: number

There are several important things to keep in mind when using type: number measures:

* A measure of type: number can perform arithmetic only on other measures, not on other dimensions.
* Looker's [symmetric aggregates](https://cloud.google.com/looker/docs/best-practices/understanding-symmetric-aggregates) will not protect aggregate functions in the SQL of a measure type: number when computed across a join.
* The [filters](https://cloud.google.com/looker/docs/reference/param-field-filters) parameter cannot be used with type: number measures, but the [filters documentation](https://cloud.google.com/looker/docs/reference/param-field-filters) explains a workaround.
* type: number measures will not provide suggestions to users.

percent\_of\_previous

type: percent\_of\_previous calculates the percent difference between a cell and the previous cell in its column.

The [sql](https://cloud.google.com/looker/docs/reference/param-field-sql) parameter for type: percent\_of\_previous measures must reference another numeric measure.

type: percent\_of\_previous fields can be formatted by using the [value\_format](https://cloud.google.com/looker/docs/reference/param-field-value-format) or [value\_format\_name](https://cloud.google.com/looker/docs/reference/param-field-value-format-name) parameters. However, the [percentage formats](https://cloud.google.com/looker/docs/reference/param-field-value-format-name#percentage_formats) of the value\_format\_name parameter do not work with type: percent\_of\_previous measures. These percentage formats multiply values by 100, which skews results of a percent of previous calculation.

In the following example, this LookML creates a measure count\_growth based on the count measure:

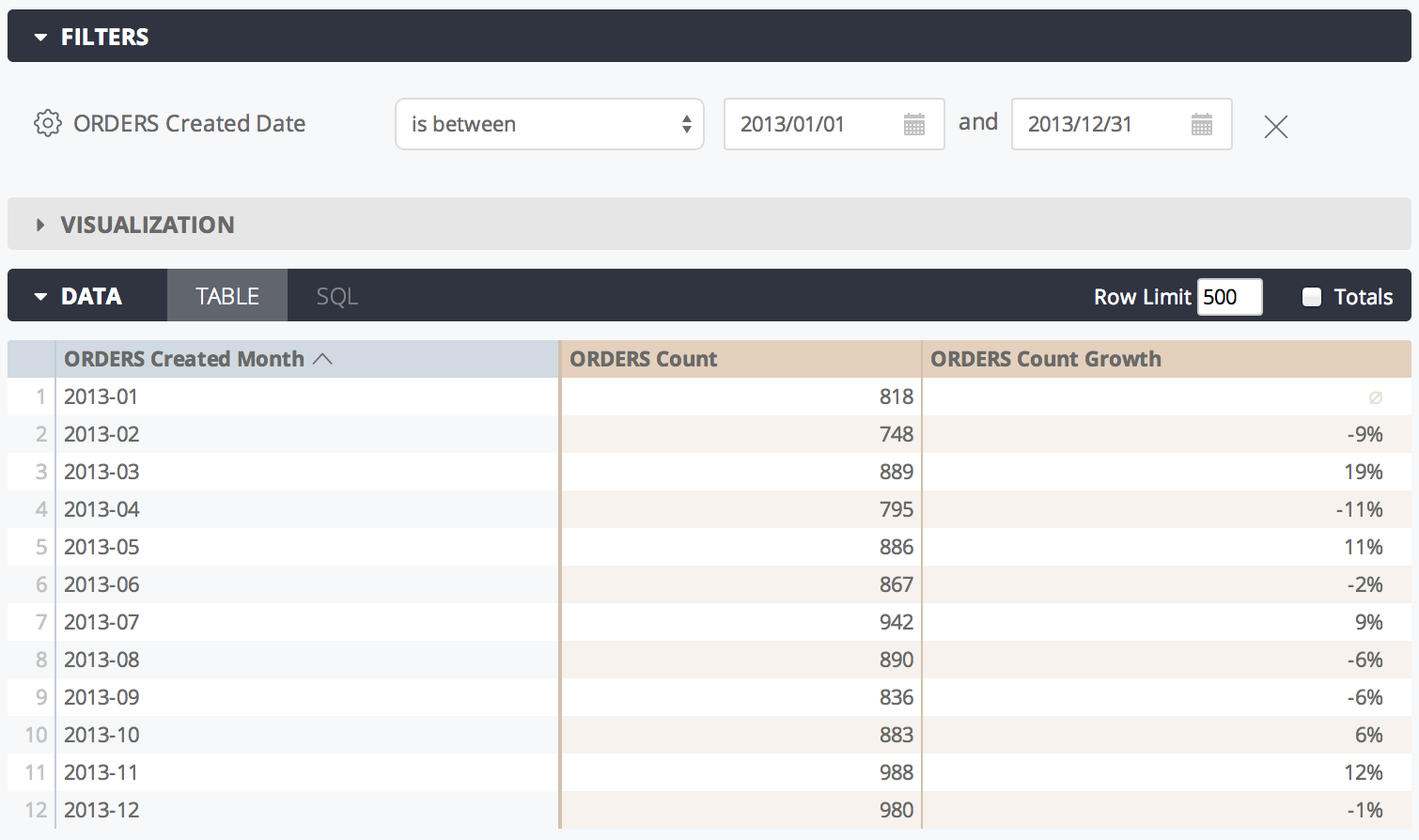
measure: count\_growth {

type: percent\_of\_previous

sql: ${count} ;;

}

In the Looker UI, this would look like:



Note that percent\_of\_previous values depend on sort order. If you change the sort, you must rerun the query to recalculate the percent\_of\_previous values. In cases where a query is pivoted, percent\_of\_previous runs across the row instead of down the column. You cannot currently change this behavior.

Additionally, percent\_of\_previous measures are calculated *after* data is returned from your database. This means that you should not reference a percent\_of\_previous measure within another measure; since they might be calculated at different times, you may not get accurate results. It also means that percent\_of\_previous measures cannot be filtered on.

percent\_of\_total

type: percent\_of\_total calculates a cell's portion of the column total. The percentage is calculated against the total of the rows returned by your query, and *not* the total of all possible rows. However, if the data returned by your query exceeds a row limit, the field's values will appear as nulls, since it needs the full results to calculate the percent of total.

The [sql](https://cloud.google.com/looker/docs/reference/param-field-sql) parameter for type: percent\_of\_total measures must reference another numeric measure.

type: percent\_of\_total fields can be formatted by using the [value\_format](https://cloud.google.com/looker/docs/reference/param-field-value-format) or [value\_format\_name](https://cloud.google.com/looker/docs/reference/param-field-value-format-name) parameters. However, the [percentage formats](https://cloud.google.com/looker/docs/reference/param-field-value-format-name#percentage_formats) of the value\_format\_name parameter do not work with type: percent\_of\_total measures. These percentage formats multiply values by 100, which skews results of a percent\_of\_total calculation.

In the following example, this LookML creates a measure percent\_of\_total\_gross\_margin based on the total\_gross\_margin measure:

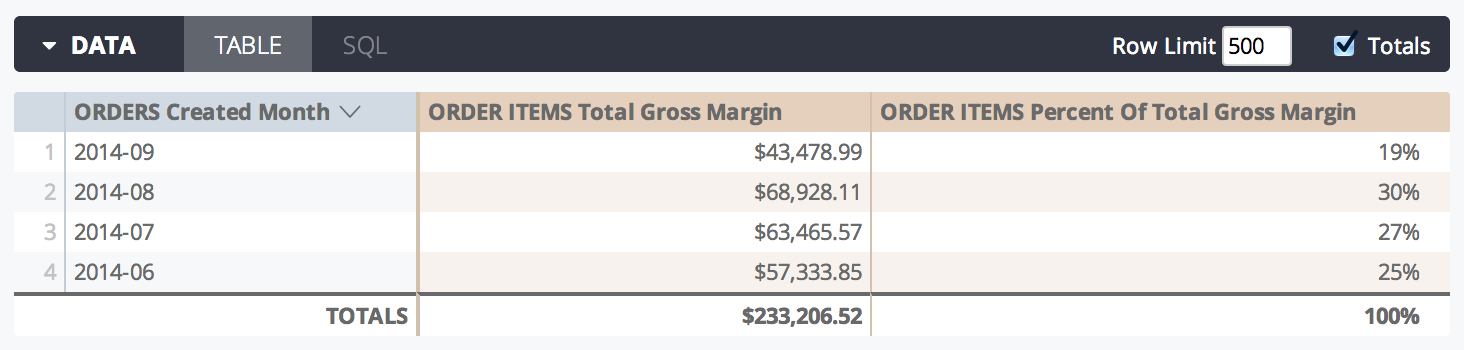
measure: percent\_of\_total\_gross\_margin {

type: percent\_of\_total

sql: ${total\_gross\_margin} ;;

}

In the Looker UI, this would look like:



In cases where a query is pivoted, percent\_of\_total runs across the row instead of down the column. If this is not desired, add [direction: "column"](https://cloud.google.com/looker/docs/reference/param-field-direction) to the measure definition.

Additionally, percent\_of\_total measures are calculated *after* data is returned from your database. This means that you should not reference a percent\_of\_total measure within another measure; since they might be calculated at different times, you may not get accurate results. It also means that percent\_of\_total measures cannot be filtered on.

percentile

type: percentile returns the value at the specified [percentile](https://en.wikipedia.org/wiki/Percentile) of values in a given field. For example, specifying the 75th percentile will return the value that is greater than 75% of the other values in the dataset.

To identify the value to return, Looker calculates the total number of data values and multiplies the specified percentile times the total number of data values. Regardless of how the data is actually sorted, Looker identifies the data values' relative order in increasing value. The data value that Looker returns depends on whether the calculation results in an integer or not, as discussed below.

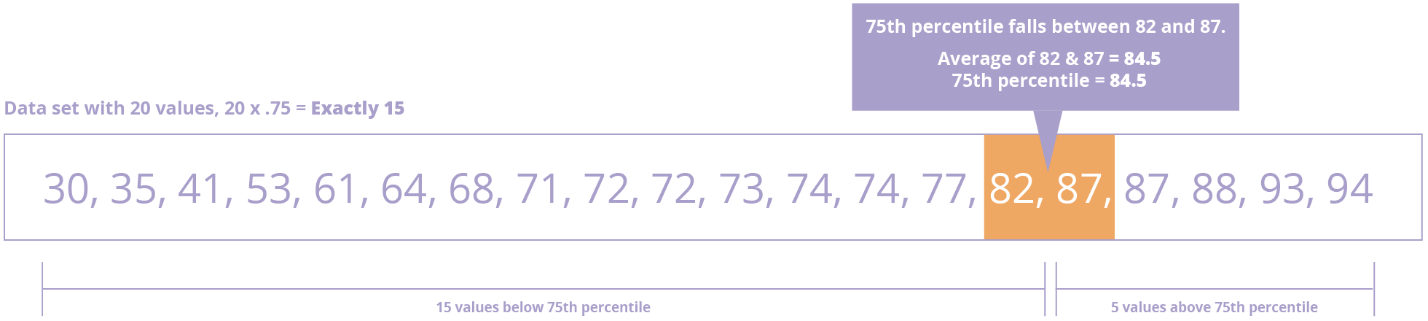
**If the calculated value is not an integer**

Looker rounds the calculated value up and uses it to identify the data value to return. In this example set of 19 test scores, the 75th percentile would be identified by 19 \* .75 = 14.25, which means that 75% of the values are in the first 14 data values -- below the 15th position. Thus, Looker returns the 15th data value (87) as being larger than 75% of the data values.



**If the calculated value is an integer**

In this slightly more complex case, Looker returns an average of the data value at that position and the following data value. To understand this, consider a set of 20 test scores, the 75th percentile would be identified by 20 \* .75 = 15, which means that the data value at the 15th position is part of the 75th percentile and we need to return a value that is *above* 75% of the data values. By returning the average of the values at the 15th position (82) and the 16th position (87), Looker ensures that 75%. That average (84.5) does not exist in the set of data values but would be larger than 75% of the data values.



**Required and optional parameters**

Use the percentile: keyword to specify the fractional value, meaning the percent of the data that should be below the returned value. For example, use percentile: 75 to specify the value at the 75th percentile in the order of data, or percentile: 10 to return the value at the 10th percentile. If you want to find the value at the 50th percentile, you can specify percentile: 50 or simply use the [median](https://cloud.google.com/looker/docs/reference/param-measure-types#median) type.

The [sql](https://cloud.google.com/looker/docs/reference/param-field-sql) parameter for type: percentile measures can take any valid SQL expression that results in a numeric table column, LookML dimension, or combination of LookML dimensions.

type: percentile fields can be formatted by using the [value\_format](https://cloud.google.com/looker/docs/reference/param-field-value-format) or [value\_format\_name](https://cloud.google.com/looker/docs/reference/param-field-value-format-name) parameters.

**Example**

For example, the following LookML creates a field called test\_scores\_75th\_percentile which returns the value at the 75th percentile in the test\_scores dimension:

measure: test\_scores\_75th\_percentile {

type: percentile

percentile: 75

sql: ${TABLE}.test\_scores ;;

}

**Things to consider for**percentile

If you are using percentile for a field involved in a fanout, Looker will attempt to use [percentile\_distinct](https://cloud.google.com/looker/docs/reference/param-measure-types" \l "percentile_distinct) instead. If percentile\_distinct is not available for the dialect, Looker returns an error. For more information, see the [supported dialects for percentile\_distinct](https://cloud.google.com/looker/docs/reference/param-measure-types#supported_database_dialects_for_percentile_distinct).

**Supported database dialects for**percentile

For Looker to support the percentile type in your Looker project, your database dialect must also support it. The following table shows which dialects support the percentile type in the latest release of Looker:

percentile\_distinct

The type: percentile\_distinct is a specialized form of [percentile](https://cloud.google.com/looker/docs/reference/param-measure-types#percentile) and should be used when your join involves a fanout. It uses the nonrepeated values in a given field, based on the unique values defined by the [sql\_distinct\_key](https://cloud.google.com/looker/docs/reference/param-field-sql-distinct-key) parameter. If the measure does not have a sql\_distinct\_key parameter, then Looker tries to use the [primary\_key](https://cloud.google.com/looker/docs/reference/param-field-primary-key) field.

Consider the result of a query joining the Order Item and Order tables:

|  |  |  |
| --- | --- | --- |
| **Order Item ID** | **Order ID** | **Order Shipping** |
| 1 | 1 | 10 |
| 2 | 1 | 10 |
| 3 | 2 | 20 |
| 4 | 3 | 50 |
| 5 | 3 | 50 |
| 6 | 3 | 50 |
| 7 | 4 | 70 |
| 8 | 4 | 70 |
| 9 | 5 | 110 |
| 10 | 5 | 110 |

In this situation you can see that there are multiple rows for each order. This query involved a fanout because each order maps to several order items. The percentile\_distinct takes this into consideration and finds the percentile value using the distinct values 10, 20, 50, 70, and 110. The 25th percentile will return the second distinct value, or 20, while the 80th percentile will return the average of the fourth and fifth distinct values, or 90.

**Required and optional parameters**

Use the percentile: keyword to specify the fractional value. For example, use percentile: 75 to specify the value at the 75th percentile in the order of data, or percentile: 10 to return the value at the 10th percentile. If you are trying to find the value at the 50th percentile, you can use the [median\_distinct](https://cloud.google.com/looker/docs/reference/param-measure-types" \l "median_distinct) type instead.

To get an accurate result, specify how Looker should identify each unique entity (in this case, each unique order) by using the sql\_distinct\_key parameter.

Here's an example of using percentile\_distinct to return the value at the 90th percentile:

measure: order\_shipping\_90th\_percentile {

type: percentile\_distinct

percentile: 90

sql\_distinct\_key: ${order\_id} ;;

sql: ${order\_shipping} ;;

}

Please note that every unique value of sql\_distinct\_key must have just one corresponding value in the measure's [sql](https://cloud.google.com/looker/docs/reference/param-field-sql) parameter. In other words, the above example works because *every* row with order\_id of 1 has the same order\_shipping of 10, *every* row with an order\_id of 2 has the same order\_shipping of 20, and so on.

type: percentile\_distinct fields can be formatted by using the [value\_format](https://cloud.google.com/looker/docs/reference/param-field-value-format) or [value\_format\_name](https://cloud.google.com/looker/docs/reference/param-field-value-format-name) parameters.

**Things to consider for**percentile\_distinct

If percentile\_distinct is not available for the dialect, Looker returns an error. For more information, see the [supported dialects for percentile\_distinct](https://cloud.google.com/looker/docs/reference/param-measure-types#supported_database_dialects_for_percentile_distinct).

**Supported database dialects for**percentile\_distinct

For Looker to support the percentile\_distinct type in your Looker project, your database dialect must also support it. The following table shows which dialects support the percentile\_distinct type in the latest release of Looker:

running\_total

type: running\_total calculates a cumulative sum of the cells along a column. It cannot be used to calculate sums along a row, unless the row has resulted from a pivot.

The [sql](https://cloud.google.com/looker/docs/reference/param-field-sql) parameter for type: running\_total measures must reference another numeric measure.

type: running\_total fields can be formatted by using the [value\_format](https://cloud.google.com/looker/docs/reference/param-field-value-format) or [value\_format\_name](https://cloud.google.com/looker/docs/reference/param-field-value-format-name) parameters.

For example, the following LookML creates a measure cumulative\_total\_revenue based on the total\_sale\_price measure:

measure: cumulative\_total\_revenue {

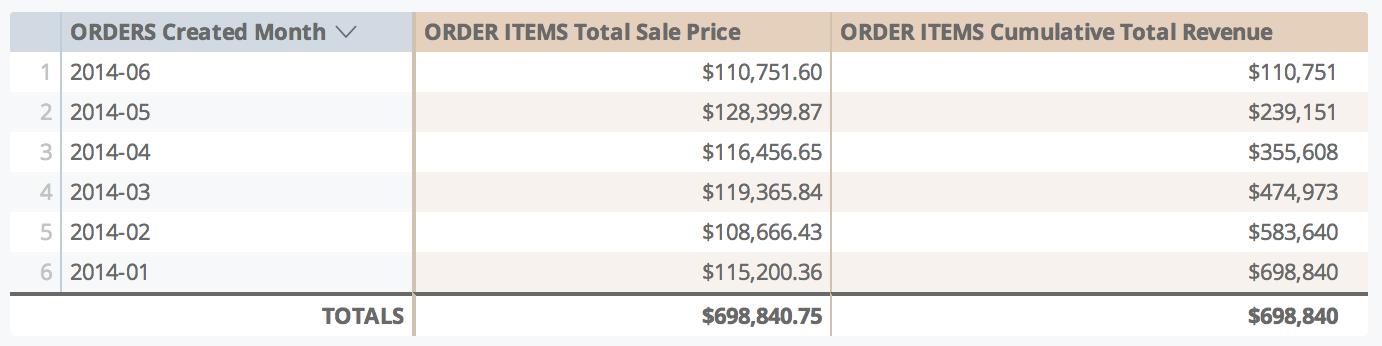
type: running\_total

sql: ${total\_sale\_price} ;;

value\_format\_name: usd

}

In the Looker UI, this would look like:



Note that running\_total values depend on sort order. If you change the sort, you must re-run the query to re-calculate the running\_total values. In cases where a query is pivoted, running\_total runs across the row instead of down the column. If this is not desired, add [direction: "column"](https://cloud.google.com/looker/docs/reference/param-field-direction) to the measure definition.

Additionally, running\_total measures are calculated *after* data is returned from your database. This means that you should not reference a running\_total measure within another measure; since they might be calculated at different times, you may not get accurate results. It also means that running\_total measures cannot be filtered on.

string

type: string is used with fields that contain letters or special characters.

The [sql](https://cloud.google.com/looker/docs/reference/param-field-sql) parameter for type: string measures can take any valid SQL expression that results in a string. In practice, this type is rarely used, because most SQL aggregate functions do not return strings. One common exception is MySQL's GROUP\_CONCAT function, although Looker provides [type: list](https://cloud.google.com/looker/docs/reference/param-measure-types#list) for that use case.

For example, the following LookML creates a field category\_list by combining the unique values of a field called category:

measure: category\_list {

type: string

sql: GROUP\_CONCAT(${category}) ;;

}

In this example type: string could be omitted, because string is the default value for type.

sum

type: sum adds up the values in a given field. It is similar to SQL's SUM function. However, unlike writing raw SQL, Looker will properly calculate sums even if your query's joins contain fanouts.

The [sql](https://cloud.google.com/looker/docs/reference/param-field-sql) parameter for type: sum measures can take any valid SQL expression that results in a numeric table column, LookML dimension, or combination of LookML dimensions.

type: sum fields can be formatted by using the [value\_format](https://cloud.google.com/looker/docs/reference/param-field-value-format) or [value\_format\_name](https://cloud.google.com/looker/docs/reference/param-field-value-format-name) parameters.

For example, the following LookML creates a field called total\_revenue by adding up the sales\_price dimension, then displays it in a money format ($1,234.56):

measure: total\_revenue {

type: sum

sql: ${sales\_price} ;;

value\_format\_name: usd

}

sum\_distinct

type: sum\_distinct is for use with denormalized datasets. It adds up the nonrepeated values in a given field, based on the unique values defined by the [sql\_distinct\_key](https://cloud.google.com/looker/docs/reference/param-field-sql-distinct-key) parameter.

This is an advanced concept which may be more clearly explained with an example. Consider a denormalized table like this:

| **Order Item ID** | **Order ID** | **Order Shipping** |
| --- | --- | --- |
| 1 | 1 | 10.00 |
| 2 | 1 | 10.00 |
| 3 | 2 | 20.00 |
| 4 | 2 | 20.00 |
| 5 | 2 | 20.00 |

In this situation you can see that there are multiple rows for each order. Consequently, if you added a simple type: sum measure for the order\_shipping column, you would get a total of 80.00, even though the total shipping collected is actually 30.00.

# Will NOT calculate the correct shipping amount

measure: total\_shipping {

type: sum

sql: ${order\_shipping} ;;

}

To get an accurate result, you can explain to Looker how it should identify each unique entity (in this case, each unique order) by using the sql\_distinct\_key parameter. This *will* calculate the correct 30.00 amount:

# Will calculate the correct shipping amount

measure: total\_shipping {

type: sum\_distinct

sql\_distinct\_key: ${order\_id} ;;

sql: ${order\_shipping} ;;

}

Please note that every unique value of sql\_distinct\_key must have just one corresponding value in [sql](https://cloud.google.com/looker/docs/reference/param-field-sql). In other words, the above example works because *every* row with an order\_id of 1 has the same order\_shipping of 10.00, *every* row with an order\_id of 2 has the same order\_shipping of 20.00, and so on.

type: sum\_distinct fields can be formatted by using the [value\_format](https://cloud.google.com/looker/docs/reference/param-field-value-format) or [value\_format\_name](https://cloud.google.com/looker/docs/reference/param-field-value-format-name) parameters.

yesno

type: yesno creates a field that indicates if something is true or false. The values appear as **Yes** and **No** in the Explore UI.

The [sql](https://cloud.google.com/looker/docs/reference/param-field-sql) parameter for a type: yesno measure takes a valid SQL expression that evaluates to TRUE or FALSE. If the condition evaluates to TRUE, **Yes** is displayed to the user; otherwise, **No** is displayed.

The SQL expression for type: yesno measures must include only aggregations, which means SQL aggregations or references to LookML measures. If you want to create a yesno field that includes a reference to a LookML dimension or a SQL expression that is not an aggregation, use a *dimension* with [type: yesno](https://cloud.google.com/looker/docs/reference/param-dimension-filter-parameter-types#yesno), not a measure.

Similar to measures with [type: number](https://cloud.google.com/looker/docs/reference/param-measure-types#number), a measure with type: yesno doesn't do any aggregations; it just references other aggregations.

For example, the total\_sale\_price measure below is a sum of the total sale price of order items in an order. A second measure called is\_large\_total is type: yesno. The is\_large\_total measure has a sql parameter that evaluates whether the total\_sale\_price value is greater than $1,000.

measure: total\_sale\_price {

type: sum

value\_format\_name: usd

sql: ${sale\_price} ;;

drill\_fields: [detail\*]

}

measure: is\_large\_total {

description: "Is order total over $1000?"

type: yesno

sql: ${total\_sale\_price} > 1000 ;;

}

If you want to reference a type: yesno field in another field, you should treat the type: yesno field as a boolean (in other words, as if it contains a true or false value already). For example:

measure: is\_large\_total {

description: "Is order total over $1000?"

type: yesno

sql: ${total\_sale\_price} > 1000 ;;

}

}

# This is correct

measure: reward\_points {

type: number

sql: CASE WHEN ${is\_large\_total} THEN 200 ELSE 100 END ;;

}

# This is NOT correct

measure: reward\_points {

type: number

sql: CASE WHEN ${is\_large\_total} = 'Yes' THEN 200 ELSE 100 END ;;

}