VariableRangeGaps

/!\ NOTE (!) NOTE ⚠ NOTE ⚠ NOTE ⚠ NOTE ⚠ NOTE ⚠ NOTE ⚠

This doc was written to help flesh out a user API. The features described here are all hypothetical and do not actually exist yet, don't assume anything you see on this page works in any version of Solr

⚠ NOTE ⚠ NOTE ⚠ NOTE ⚠ NOTE ⚠ NOTE ⚠ NOTE ⚠ NOTE ⚠

Brief

The Jira issue SOLR-2366 discusses how to implement variable gap sizes into range faceting. This is useful for things like prices, date ranges etc. This page is an sandbox for preparing the WIKI documentation for Range Facet, until the feature is released.

I (janhoy) propose to leave the old facet.range.start/end/gap/ syntax as is for backward compat, and introduce a new simpler param facet.range.spec for the new advanced range spec feature. See below how this would look in WIKI:

Proposed Facet by Range - second take by janhoy

As a generalization of the Date faceting described above, one can use the Range Faceting feature on any date field or any numeric field that supports range queries. This is particularly useful for the cases in the past where one might stitch together a series of range queries (as facet by query) for things like prices, etc.

facet.range

This param indicates what field to create range facets for. This param allows you to specify names of fields which should be treated as range facets.

Example: `facet.range=price&facet.range=age`

facet.range.start

The lower bound of the ranges.

This parameter can be specified on a per field basis.

Example: `f.price.facet.range.start=0.0&f.age.facet.range.start=10`

facet.range.end

The upper bound of the ranges.

This parameter can be specified on a per field basis.

Example: `f.price.facet.range.end=1000.0&f.age.facet.range.start=99`

facet.range.gap

The size of each range expressed as a value to be added to the lower bound. For date fields, this should be expressed using the DateMathParser syntax. (ie: `facet.range.gap=%2B1DAY` ... '+1DAY')

This parameter can be specified on a per field basis.

Example: `f.price.facet.range.gap=100&f.age.facet.range.gap=10`

facet.range.hardend

A Boolean parameter instructing Solr what to do in the event that `facet.range.gap` does not divide evenly between `facet.range.start` and `facet.range.end`. If this is true, the last range constraint will have an upper bound of `facet.range.end`; if false, the last range will have the smallest possible upper bound greater then `facet.range.end` such that the range is exactly `facet.range.gap` wide.

The default is false.

This parameter can be specified on a per field basis.

facet.range.other

This param indicates that in addition to the counts for each range constraint between `facet.range.start` and `facet.range.end`, counts should also be computed for...

- `before` all records with field values lower then lower bound of the first range
- `after` all records with field values greater then the upper bound of the last range
- `between` all records with field values between the start and end bounds of all ranges
- `none` compute none of this information
- `all` shortcut for `before`, `between`, and `after`

This parameter can be specified on a per field basis.

In addition to the `all` option, this parameter can be specified multiple times to indicate multiple choices – but `none` will override all other options.

facet.range.include

By default, the ranges used to compute range faceting between facet.range.start and facet.range.end are inclusive of their lower bounds and exclusive of the upper bounds. The "before" range is exclusive and the "after" range is inclusive. This default, equivalent to `lower` below, will *not* result in double counting at the boundaries. This behavior can be modified by the `facet.range.include` param, which can be any combination of the following options...

- `lower` = all gap based ranges include their lower bound
- `upper` = all gap based ranges include their upper bound
- `edge` = the first and last gap ranges include their edge bounds (ie: lower for the first one, upper for the last one) even if the corresponding upper /lower option is not specified
- `outer` = the "before" and "after" ranges will be inclusive of their bounds, even if the first or last ranges already include those boundaries.
- `all` = shorthand for lower, upper, edge, outer

This parameter can be specified on a per field basis.

This parameter can be specified multiple times to indicate multiple choices.

If you want to ensure you don't double-count, don't choose both lower & upper, don't choose outer, and don't choose all.

facet.range.spec

Often times you want un-equally spaced facet gaps. A typical example is prices, e.g. **0-10$**, **10-50$**, **50-100$**, **100-250$**, **250-500$**. Another common example is date ranges, where we also need overlapping ranges such as **last day**, **last week**, **last month**. The facet.range.spec parameter lets you configure such complex ranges with one parameter. The syntax of this parameter is as follows:

`facet.range.spec=[<start>,]<gap>[,<gap>[,<gap>[,<gap>]]][,<end>]`

where

- `<start>` is of syntax `N..` where `N` is either a number or a date or `*` (asterisk) for MIN
- `<end>` is of syntax `..M` where `M` is either a number or a date or `*` (asterisk) for MAX
- `<gap>` is either:
  - an absolute number or date, e.g. `50` or `2011-01-01T00:00:00Z`
  - a relative gap size, e.g. `+40` or `+1DAY`
  - an explicit range, e.g. `0..10` or `100..*` or `2011-01-01..NOW`. **NOTE** that these ranges may be **overlapping**

Some constraints:

- Relative `<gap>`s repeat until next absolute number/date. There **must** be an absolute gap value following a relative one
- If you want to specify explicit ranges, then **all** <gap>s must be in this format, and <start>/<end> are not allowed. Any global start/end values will be disregarded in this mode.

Examples:

`facet.range.spec=*..,10,50,100,250,..*` - gives 5 ranges: MIN-10, 10-50, 50-100, 100-250, 250->MAX

`facet.range.spec=*..,10,+40,+50,250,..*` - gives exactly the same ranges, using relative gap size

`facet.range.spec=0..,+10,50,250,..500` - gives ranges: 0-10, 10-20, 20-30, 30-40, 40-50, 20-250, 250-MAX

`facet.range.spec=0..,+10,50,250,500&facet.range.end=*` - gives the exact same result as previous example, but the end is specified as a separate query parameter

`facet.range.spec=0..,10,50,+50,+100,500,..*` - gives ranges: 0-10, 10-50, 50-100, 100-200, 200-300, 300-400, 400-500, 500-MAX

`facet.range.spec=NOW-1DAY..NOW, NOW-1WEEK..NOW, NOW-1MONTH..NOW` - gives three overlapping date ranges for last day, week and month. **Any start/end will be disregarded**

This parameter can be specified on a per field basis.

Example: `f.price.facet.range.spec=*..,10,50,100,250,..*` to configure the range spec for the price field

This parameter is mutually exclusive to facet.range.gap

Proposed Facet by Range - first take by janhoy

As a generalization of the Date faceting described above, one can use the Range Faceting feature on any date field or any numeric field that supports range queries. This is particularly useful for the cases in the past where one might stitch together a series of range queries (as facet by query) for things like prices, etc.

facet.range

This param indicates what field to create range facets for. This param allows you to specify names of fields which should be treated as range facets.

Example: `facet.range=price&facet.range=age`

facet.range.start

The lower bound of the ranges.

This parameter can be specified on a per field basis.

Example: `f.price.facet.range.start=0.0&f.age.facet.range.start=10`

facet.range.end

The upper bound of the ranges.

This parameter can be specified on a per field basis.

Example: `f.price.facet.range.end=1000.0&f.age.facet.range.start=99`

facet.range.gap

The size of each range expressed as a value to be added to the lower bound. For date fields, this should be expressed using the DateMathParser syntax. (ie: `facet.range.gap=%2B1DAY` ... '+1DAY')

This parameter can be specified on a per field basis.

Example: `f.price.facet.range.gap=100&f.age.facet.range.gap=10`

facet.range.hardend

A Boolean parameter instructing Solr what to do in the event that `facet.range.gap` does not divide evenly between `facet.range.start` and `facet.range.end`. If this is true, the last range constraint will have an upper bound of `facet.range.end`; if false, the last range will have the smallest possible upper bound greater then `facet.range.end` such that the range is exactly `facet.range.gap` wide.

The default is false.

This parameter can be specified on a per field basis.

facet.range.other

This param indicates that in addition to the counts for each range constraint between `facet.range.start` and `facet.range.end`, counts should also be computed for...

- `before` all records with field values lower then lower bound of the first range
- `after` all records with field values greater then the upper bound of the last range

- `between` all records with field values between the start and end bounds of all ranges
- `none` compute none of this information
- `all` shortcut for `before`, `between`, and `after`

This parameter can be specified on a per field basis.

In addition to the `all` option, this parameter can be specified multiple times to indicate multiple choices – but `none` will override all other options.

facet.range.include

By default, the ranges used to compute range faceting between facet.range.start and facet.range.end are inclusive of their lower bounds and exclusive of the upper bounds. The "before" range is exclusive and the "after" range is inclusive. This default, equivalent to `lower` below, will *not* result in double counting at the boundaries. This behavior can be modified by the `facet.range.include` param, which can be any combination of the following options...

- `lower` = all gap based ranges include their lower bound
- `upper` = all gap based ranges include their upper bound
- `edge` = the first and last gap ranges include their edge bounds (ie: lower for the first one, upper for the last one) even if the corresponding upper /lower option is not specified
- `outer` = the "before" and "after" ranges will be inclusive of their bounds, even if the first or last ranges already include those boundaries.
- `all` = shorthand for lower, upper, edge, outer

This parameter can be specified on a per field basis.

This parameter can be specified multiple times to indicate multiple choices.

If you want to ensure you don't double-count, don't choose both lower & upper, don't choose outer, and don't choose all.

facet.range.spec

Often times you want un-equally spaced facet gaps. A typical example is prices, e.g. **0-10$**, **10-50$**, **50-100$**, **100-250$**, **250-500$**. Another common example is date ranges, where we also need overlapping ranges such as **last day**, **last week**, **last month**. The facet.range.spec parameter lets you configure such complex ranges with one parameter. The syntax of this parameter is as follows:

`facet.range.spec=<lower-bound>,<gap>[,<gap>...,<gap>n],<upper-bound>`

where

`<lower-bound>/<upper-bound>` is either a number or a date or `*` (asterisk) for MIN/MAX

`<gap>` is either an absolute number or date, e.g. `50` or `2011-01-01T00:00:00Z`, a relative gap size, e.g. `+40` or `+1DAY` or even a complete range such as `10..50` or `2011-01-01..NOW`.

Relative `<gap>`s repeat until next absolute number/date.

Examples:

`facet.range.spec=*,10,50,100,250,*` - gives 5 ranges: MIN-10, 10-50, 50-100, 100-250, 250->MAX

`facet.range.spec=*,10,+40,+50,250,*` - gives exactly the same ranges, using relative gap size

`facet.range.spec=0,+10,50,250,*` - gives ranges: 0-10, 10-20, 20-30, 30-40, 40-50, 20-250, 250-MAX

`facet.range.spec=0,10,50,+50,+100,*` - gives ranges: 0-10, 10-50, 50-100, 100-200, 200-300 repeating until max

`facet.range.spec=NOW-1DAY..NOW, NOW-1WEEK..NOW, NOW-1MONTH..NOW` - gives three overlapping date ranges for last day, week and month

This parameter can be specified on a per field basis.

Example: `f.price.facet.range.spec=*,10,50,100,250,*` to configure the range spec for the price field

This parameter can be combined with facet.range.include, but is mutually exclusive to facet.range.gap, facet.range.begin, facet.range.end and facet.range. other, resulting in an exception if uncompatible mix is attempted.