


Filters in Azure Cognitive Search

11/04/2019 • 9 minutes to read •  +3

In this article

[When to use a filter](#)

[How filters are executed](#)

[Defining filters](#)

[Filter usage patterns](#)

[Field requirements for filtering](#)

[Text filter fundamentals](#)

[Numeric filter fundamentals](#)

[Next steps](#)

[See also](#)

A *filter* provides criteria for selecting documents used in an Azure Cognitive Search query. Unfiltered search includes all documents in the index. A filter scopes a search query to a subset of documents. For example, a filter could restrict full text search to just those products having a specific brand or color, at price points above a certain threshold.

Some search experiences impose filter requirements as part of the implementation, but you can use filters anytime you want to constrain search using *value-based* criteria (scoping search to product type "books" for category "non-fiction" published by "Simon & Schuster").

If instead your goal is targeted search on specific data *structures* (scoping search to a customer-reviews field), there are alternative methods, described below.

When to use a filter

Filters are foundational to several search experiences, including "find near me", faceted navigation, and security filters that show only those documents a user is allowed to see. If you implement any one of these experiences, a filter is required. It's the filter attached to the search query that provides the geolocation coordinates, the facet category selected by the user, or the security ID of the requestor.

Example scenarios include the following:

1. Use a filter to slice your index based on data values in the index. Given a schema with city, housing type, and amenities, you might create a filter to explicitly select documents that satisfy your criteria (in Seattle, condos, waterfront).

Full text search with the same inputs often produces similar results, but a filter is more precise in that it requires an exact match of the filter term against content in your index.

2. Use a filter if the search experience comes with a filter requirement:

- [Faceted navigation](#) uses a filter to pass back the facet category selected by the user.
- Geo-search uses a filter to pass coordinates of the current location in "find near me" apps.
- Security filters pass security identifiers as filter criteria, where a match in the index serves as a proxy for access rights to the document.

3. Use a filter if you want search criteria on a numeric field.

Numeric fields are retrievable in the document and can appear in search results, but they are not searchable (subject to full text search) individually. If you need selection criteria based on numeric data, use a filter.

Alternative methods for reducing scope

If you want a narrowing effect in your search results, filters are not your only choice. These alternatives could be a better fit, depending on your objective:

- `searchFields` query parameter pegs search to specific fields. For example, if your index provides separate fields for English and Spanish descriptions, you can use `searchFields` to target which fields to use for full text search.

- `$select` parameter is used to specify which fields to include in a result set, effectively trimming the response before sending it to the calling application. This parameter does not refine the query or reduce the document collection, but if a smaller response is your goal, this parameter is an option to consider.

For more information about either parameter, see [Search Documents > Request > Query parameters](#).

How filters are executed

At query time, a filter parser accepts criteria as input, converts the expression into atomic Boolean expressions represented as a tree, and then evaluates the filter tree over filterable fields in an index.

Filtering occurs in tandem with search, qualifying which documents to include in downstream processing for document retrieval and relevance scoring. When paired with a search string, the filter effectively reduces the recall set of the subsequent search operation. When used alone (for example, when the query string is empty where `search=*`), the filter criteria is the sole input.

Defining filters

Filters are OData expressions, articulated using a [subset of OData V4 syntax supported in Azure Cognitive Search](#).

You can specify one filter for each **search** operation, but the filter itself can include multiple fields, multiple criteria, and if you use an **ismatch** function, multiple full-text search expressions. In a multi-part filter expression, you can specify predicates in any order (subject to the rules of operator precedence). There is no appreciable gain in performance if you try to rearrange predicates in a particular sequence.

One of the limits on a filter expression is the maximum size limit of the request. The entire request, inclusive of the filter, can be a maximum of 16 MB for POST, or 8 KB for GET. There is also a limit on the number of clauses in your filter expression. A good rule of thumb is that if you have hundreds of clauses, you are at risk of running into the limit. We recommend designing your application in such a way that it does not generate filters of unbounded size.

The following examples represent prototypical filter definitions in several APIs.

HTTP

 Copy

```
# Option 1: Use $filter for GET
GET https://[service name].search.windows.net/indexes/hotels/docs?api-version=2020-06-30&search=*&$filter=Rooms/any(room: room/BaseRate lt 150.0)&$select=HotelId, HotelName, Rooms/Description, Rooms/BaseRate

# Option 2: Use filter for POST and pass it in the request body
POST https://[service name].search.windows.net/indexes/hotels/docs/search?api-version=2020-06-30
{
  "search": "*",
  "filter": "Rooms/any(room: room/BaseRate lt 150.0)",
  "select": "HotelId, HotelName, Rooms/Description, Rooms/BaseRate"
}
```

C#

 Copy

```
parameters =
    new SearchParameters()
    {
        Filter = "Rooms/any(room: room/BaseRate lt 150.0)",
        Select = new[] { "HotelId", "HotelName", "Rooms/Description", "Rooms/BaseRate" }
    };

var results = searchIndexClient.Documents.Search("*", parameters);
```

Filter usage patterns

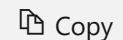
The following examples illustrate several usage patterns for filter scenarios. For more ideas, see [OData expression syntax > Examples](#).

- Standalone **\$filter**, without a query string, useful when the filter expression is able to fully qualify documents of interest. Without a query string, there is no lexical or linguistic analysis, no scoring, and no ranking. Notice the search string is just an asterisk, which means "match all documents".



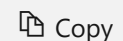
```
search=*&$filter=Rooms/any(room: room/BaseRate ge 60 and room/BaseRate lt 300) and Address/City eq 'Honolulu'
```

- Combination of query string and **\$filter**, where the filter creates the subset, and the query string provides the term inputs for full text search over the filtered subset. The addition of terms (walking distance theaters) introduces search scores in the results, where documents that best match the terms are ranked higher. Using a filter with a query string is the most common usage pattern.



```
search=walking distance theaters&$filter=Rooms/any(room: room/BaseRate ge 60 and room/BaseRate lt 300) and Address/City eq 'Seattle'&$count=true
```

- Compound queries, separated by "or", each with its own filter criteria (for example, 'beagles' in 'dog' or 'siamese' in 'cat'). Expressions combined with **or** are evaluated individually, with the union of documents matching each expression sent back in the response. This usage pattern is achieved through the `search.ismatchscoring` function. You can also use the non-scoring version, `search.ismatch`.



```
# Match on hostels rated higher than 4 OR 5-star motels.
$filter=search.ismatchscoring('hostel') and Rating ge 4 or search.ismatchscoring('motel') and Rating eq 5

# Match on 'luxury' or 'high-end' in the description field OR on category exactly equal to 'Luxury'.
$filter=search.ismatchscoring('luxury | high-end', 'Description') or Category eq 'Luxury'&$count=true
```

It is also possible to combine full-text search via `search.ismatchscoring` with filters using `and` instead of `or`, but this is functionally equivalent to using the `search` and `$filter` parameters in a search request. For example, the following two queries produce the same result:

 Copy

```
$filter=search.ismatchscoring('pool') and Rating ge 4  
  
search=pool&$filter=Rating ge 4
```

Follow up with these articles for comprehensive guidance on specific use cases:

- [Facet filters](#)
- [Language filters](#)
- [Security trimming](#)

Field requirements for filtering

In the REST API, filterable is *on* by default for simple fields. Filterable fields increase index size; be sure to set `"filterable": false` for fields that you don't plan to actually use in a filter. For more information about settings for field definitions, see [Create Index](#).

In the .NET SDK, the filterable is *off* by default. You can make a field filterable by setting the [IsFilterable property](#) of the corresponding [SearchField](#) object to `true`. In the example below, the attribute is set on the `BaseRate` property of a model class that maps to the index definition.

C#

 Copy

```
[IsFilterable, IsSortable, IsFacetable]  
public double? BaseRate { get; set; }
```

Making an existing field filterable

You can't modify existing fields to make them filterable. Instead, you need to add a new field, or rebuild the index. For more information about rebuilding an index or repopulating fields, see [How to rebuild an Azure Cognitive Search index](#).

Text filter fundamentals

Text filters match string fields against literal strings that you provide in the filter. Unlike full-text search, there is no lexical analysis or word-breaking for text filters, so comparisons are for exact matches only. For example, assume a field *f* contains "sunny day", `$filter=f eq 'Sunny'` does not match, but `$filter=f eq 'sunny day'` will.

Text strings are case-sensitive. There is no lower-casing of upper-cased words: `$filter=f eq 'Sunny day'` will not find "sunny day".

Approaches for filtering on text

Approach	Description	When to use
<code>search.in</code>	A function that matches a field against a delimited list of strings.	Recommended for security filters and for any filters where many raw text values need to be matched with a string field. The search.in function is designed for speed and is much faster than explicitly comparing the field against each string using <code>eq</code> and <code>or</code> .
<code>search.ismatch</code>	A function that allows you to mix full-text search operations with strictly Boolean filter operations in the same filter expression.	Use search.ismatch (or its scoring equivalent, search.ismatchscoring) when you want multiple search-filter combinations in one request. You can also use it for a <i>contains</i> filter to filter on a partial string within a larger string.

Approach	Description	When to use
\$filter=field operator string	A user-defined expression composed of fields, operators, and values.	Use this when you want to find exact matches between a string field and a string value.


Numeric filter fundamentals

Numeric fields are not `searchable` in the context of full text search. Only strings are subject to full text search. For example, if you enter 99.99 as a search term, you won't get back items priced at \$99.99. Instead, you would see items that have the number 99 in string fields of the document. Thus, if you have numeric data, the assumption is that you will use them for filters, including ranges, facets, groups, and so forth.

Documents that contain numeric fields (price, size, SKU, ID) provide those values in search results if the field is marked `retrievable`. The point here is that full text search itself is not applicable to numeric field types.

Next steps

First, try **Search explorer** in the portal to submit queries with **\$filter** parameters. The [real-estate-sample index](#) provides interesting results for the following filtered queries when you paste them into the search bar:

 Copy

```

# Geo-filter returning documents within 5 kilometers of Redmond, Washington state
# Use $count=true to get a number of hits returned by the query
# Use $select to trim results, showing values for named fields only
# Use search=* for an empty query string. The filter is the sole input

search=*&$count=true&$select=description,city,postCode&$filter=geo.distance(location,geography'POINT(-122.1215
13 47.673988)') le 5

```



```
# Numeric filters use comparison like greater than (gt), less than (lt), not equal (ne)
# Include "and" to filter on multiple fields (baths and bed)
# Full text search is on John Leclerc, matching on John or Leclerc

search=John Leclerc&$count=true&$select=source,city,postCode,baths,beds&$filter=baths gt 3 and beds gt 4

# Text filters can also use comparison operators
# Wrap text in single or double quotes and use the correct case
# Full text search is on John Leclerc, matching on John or Leclerc

search=John Leclerc&$count=true&$select=source,city,postCode,baths,beds&$filter=city gt 'Seattle'
```

To work with more examples, see [OData Filter Expression Syntax > Examples](#).

See also

- [How full text search works in Azure Cognitive Search](#)
- [Search Documents REST API](#)
- [Simple query syntax](#)
- [Lucene query syntax](#)
- [Supported data types](#)

Is this page helpful?

 Yes  No
