

Filtering

- [Filtering](#)
- [Operators](#)
 - [HAS operator](#)
 - [Examples](#)
 - [IN operator](#)
 - [Examples](#)
 - [LT operator](#)
 - [Examples](#)
 - [LTE operator](#)
 - [Examples](#)
 - [GT operator](#)
 - [Examples](#)
 - [GTE operator](#)
 - [Examples](#)
 - [AND operator](#)
 - [Examples](#)
 - [NULL Operator](#)
 - [Examples](#)
 - [NOTNULL Operator](#)
 - [Examples](#)
- [Condition types](#)
 - [AND Examples](#)
 - [OR Examples](#)

Filtering

Filtering is done by passing query params in the api call, what can be filtered is decided per endpoint and will be documented in redoc for private and public api:

[Alpha - Private core API](#)

[Alpha - Public core API](#)

[Production - Public core API](#)

Query parameters are added at the end of the url.

Filtering will be added with the format:

```
?filter[{what to filter on}]= {operator}:{value}:{conditionType}
```

Skipping the condition type is supported then it will default to AND

```
?filter[{what to filter on}]= {operator}:{value}
```

Operators

The supported filter operations are

Type	Description
HAS	Checks if the column value contains the passed value – we can use this for search as well then we won't need a separate search parameter for the list endpoints.
IN	Checks if passed value/list of values contains the column value – can be used to filter using multiple Guids, enums etc.
LT	Lesser than check

LTE	Lesser than or equal check
GT	Greater than check
GTE	Greater than or equal check
AND	Bitwise AND operator check, can be used with enum flags
NULL	Checks if the value is null, can be used to get non deleted entities
NOTNULL	Checks if the value is not null, can be used to get deleted entities

HAS operator

The HASoperator works well when filtering for a single value, if the value is a string the has operator will perform a LIKE '%{value}%' in the database so it can be used to search on substrings in names or similar properties.

Will also work fine with intergers, boolean and guid values then a normal equal comparison will be made.

Examples

```
?filter[name]=has:beef
?filter[name]=has:sall&filter[name]=has:beef
?filter[name]=has:sall:or&filter[name]=has:beef
?filter[producttype]=has:2
?filter[productgroup.id]=has:ca68cd1e-2a4c-4e81-8b30-75f4227467e3
?filter[hasvariants]=has:true
```

1. Filters for names containing beef, items like: "Beefburger", "Steak beef 150g"
2. Filters for names containing sall **and** beef, items like: "Nice beef sallad"
3. Filters for names containing sall **or** beef, items like: "Ceasar sallad", "Beefburger"
4. Filters for product types = 2, for integers a exact match is needed.
5. Filters for product group id {Guid}, for guid values a exact match is needed.

Any number of has filters can be combined with ANDor OR conditions

IN operator

The IN operator is best used when filtering for exact matches on multiple values on the same property, this will perform a ANY({valueArray}) in the database.

Examples

```
?filter[productgroup.id]=in:ca68cd1e-2a4c-4e81-8b30-75f4227467e3,
51abeddc-bec7-4416-9aac-f791cbe0eb83
?filter[productgroup.id]=in:ca68cd1e-2a4c-4e81-8b30-75f4227467e3
?filter[producttype]=in:1,2,3
?filter[name]=in:Baconburger 200g,Ceasar sallad
?filter[productgroup.id]=in:ca68cd1e-2a4c-4e81-8b30-75f4227467e3:
or&filter[name]=in:Ceasar sallad
```

1. Filters for product group ids, returns all items with that product group id. Exact match is needed on guid columns, in this example we give two guids seperated by ,
2. Filters for product group ids, in this example only one guid is provided, if its always just one value its better to use the HAS filter instead.
3. Filters for product types, returns all GLassBottle=1, Volume=2 and Weight=3 products.

Note that there should be no empty space between separator if filter is passed as `'=in:one, two, three'` then the query will look for entities with 'one' 'two' and 'three' values

LT operator

The LT operator, lesser than, works on numeric properties and dates, this will perform a < comparison in the database.

Examples

```
?filter[created]=lt:2021-01-01 12:32  
?filter[producttype]=lt:2
```

1. Filters for created lesser than the datetime provided.
2. Filters for product types lesser than 2, so returns products with types 0 and 1

LTE operator

The LTE operator, lesser than or equal, works on numeric properties and dates, this will perform a <= comparison in the database.

Examples

```
?filter[created]=lte:2021-01-01 12:32  
?filter[producttype]=lte:2
```

1. Filters for created lesser than or equal the datetime provided.
2. Filters for product types lesser than or equal to 2, so returns products with types 0, 1 and 2

GT operator

The GT operator, greater than, works on numeric properties and dates, this will perform a > comparison in the database.

Examples

```
?filter[created]=gt:2022-10-26 00:00  
?filter[producttype]=gt:3
```

1. Filters for created greater than the datetime provided.
2. Filters for product types greater than 3, so returns products with types 4, 5, 6, 7, 8, 9, 10 and so on.

GTE operator

The GTE operator, greater than or equal, works on numeric properties and dates, this will perform a >= comparison in the database.

Examples

```
?filter[created]=gte:2022-10-26 00:00
?filter[producttype]=gte:3
```

1. Filters for created greater than or equal the datetime provided.
2. Filters for product types greater than or equal 3, so returns products with types 3, 4, 5, 6, 7, 8, 9, 10 and so on.

AND operator

The AND will perform a bitwise and operation on the database.

If a value is 15 then passing a 8 would then check if 15 contains an 8 which it does.

Works like this:

Value	64	32	16	8	4	2	1
19	No	No	Yes (19-16=3)	No	No	Yes (3-2=1)	Yes (1 left)
37	No	Yes (37-32=5)	No	No	Yes (5-4=1)	No	Yes (1 left)
15	No	No	No	Yes (15-8=7)	Yes (7-4=3)	Yes (3-2=1)	Yes (1 left)
66	Yes (66-64=2)	No	No	No	No	Yes (2-2=0)	No

So if you do an bitwise AND you pass in for example:

- 32, then items 37 is returned
- 4, then items 37 and 15 is returned
- 2, then items 19, 15 and 66 is returned.

Examples

```
?filter[orderformat]=and:8
```

1. Filters for order formats that has 8 bit set. in this case all items with orderformat delivery=8

NULL Operator

The NULL will check if the value is null, will perform a IS NULL in the database.

Examples

```
?filter[deleted]=null
```

NOTNULL Operator

The NOTNULL will check if the value is not null, will perform a IS NOT NULL in the database.

Examples

```
?filter[deleted]=notnull
```

Condition types

Type	Description
AND	Default condition type, it will be used when none is provided.
OR	Useful when we want for example want to find entities with a name and have multiple has filters, give me all with name containing "chicken" or "beef"

When multiple filters are passed you can decide to put a AND condition or a OR condition between them, default is AND

The condition between filters are passes like this `?filter[{what to filter on}={operator}:{value}:{conditionType}]` with condition type as last.

AND Examples

```
?Filter[tagid]=in:044ebf50-3436-4bb3-af6e-2f34df5afcbe&filter  
[productgroup.id]=in:ca68cd1e-2a4c-4e81-8b30-75f4227467e3
```

Here it will be filtered by tagId AND product group id, both must pass for a single item to be returned.

Will be useful in filters where you want for example all orders with payment amount > 500 SEK and is a online order

OR Examples

```
?Filter[tagid]=in:044ebf50-3436-4bb3-af6e-2f34df5afcbe:or&filter  
[productgroup.id]=in:ca68cd1e-2a4c-4e81-8b30-75f4227467e3
```

Here it will be filtered by tagId OR product group id, if any filter passes for a single item it will be returned.

Will be useful in filters where you for example want all orders with payment > 500 SEK or is older than 2022-05-01 12:00