
sqla-filter Documentation

Release 0.0.1

Marc-Aurele Coste

Nov 30, 2018

Contents:

1	Introduction	3
2	Guide	5
2.1	User Guide	5
2.2	Developer Guide	12
3	Indices and tables	15

Warning: This project is not ready for production so use it carefully because it's not stable.

CHAPTER 1

Introduction

The purpose of this library is to provide a simple way to filter queries generated with [Sqlalchemy](#). To filter a query a tree is generated. When the filter function is called the tree is traversed ([DFS](#)) and the filter function of each sub node is called until all nodes are scanned.

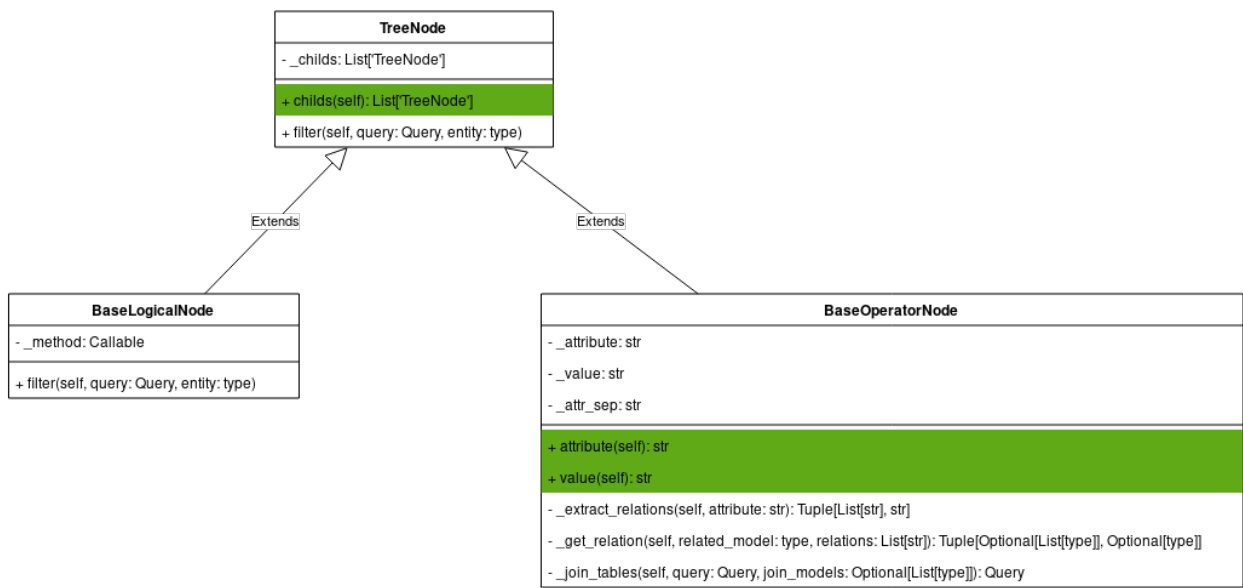
The default format for the tree generation is the `JSON` format. It's included in the package.

Other formats will be supported in the future. You can also create your own parser to generate the tree from the format you want ([here](#)).

2.1 User Guide

2.1.1 sqla-filters: Nodes

Base nodes



Tree

```
class sqla_filters.nodes.base.TreeNode
    Abstract base class for every nodes.
```

This class

childs

Property that return the node childs list.

Returns The node childs list.

Return type List[*TreeNode*]

filter (*query: sqlalchemy.orm.query.Query*, *entity: type*) → Tuple[sqlalchemy.orm.query.Query, Any]

Define the filter function that every node must to implement.

Parameters

- **query** (*Query*) – The sqlalchemy query.
- **entity** (*type*) – The entity model.

Returns The filtered query.

Return type Tuple[Query, Any]

Base logical

```
class sqala_filters.nodes.base.BaseLogicalNode (*args, method=<function de-
fault_method>, **kwargs)
```

filter (*query: sqlalchemy.orm.query.Query*, *entity: type*) → Tuple[sqlalchemy.orm.query.Query, Any]

Apply the *_method* to all childs of the node.

Parameters

- **query** (*Query*) – The sqlalchemy query.
- **entity** (*type*) – The entity model of the query.

Returns A tuple with in first place the updated query and in second place the list of filters to apply to the query.

Return type Tuple[Query, Any]

Base operational

```
class sqala_filters.nodes.base.BaseOperationalNode (attribute: str, value: Any, attr_sep: str = '.')
```

_extract_relations (*attribute: str*) → Tuple[List[str], str]

Split and return the list of relation(s) and the attribute.

Parameters **attribute** (*str*) –

Returns A tuple where the first element is the list of related entities and the second is the attribute.

Return type Tuple[List[str], str]

_get_relation (*related_model: type*, *relations: List[str]*) → Tuple[Optional[List[type]], Optional[type]]

Transform the list of relation to list of class.

Parameters

- **related_model** (*type*) – The model of the query.

- **relations** (*List[str]*) – The relation list get from the *_extract_relations*.

Returns Tuple with the list of relations (class) and the second element is the last relation class.

Return type Tuple[Optional[List[type]], Optional[type]]

_join_tables (*query: sqlalchemy.orm.query.Query, join_models: Optional[List[type]]*) → *sqlalchemy.orm.query.Query*

Method to make the join when relation is found.

Parameters

- **query** (*Query*) – The sqlalchemy query.
- **join_models** (*Optional[List[type]]*) – The list of joined models get from the method *_get_relation*.

Returns The new Query with the joined tables.

Return type Query

attribute

Property that return the model attribute.

Returns The model attribute.

Return type str

filter (*query: sqlalchemy.orm.query.Query, entity: type*) → Tuple[sqlalchemy.orm.query.Query, Any]

Add a filters to the list of filters to apply.

Warning: This method must be override in childs nodes.

Parameters

- **query** (*Query*) – The sqlalchemy query.
- **entity** (*type*) – The entity model of the query.

Returns A tuple with in first place the updated query and in second place the list of filters to apply to the query.

Return type Tuple[Query, Any]

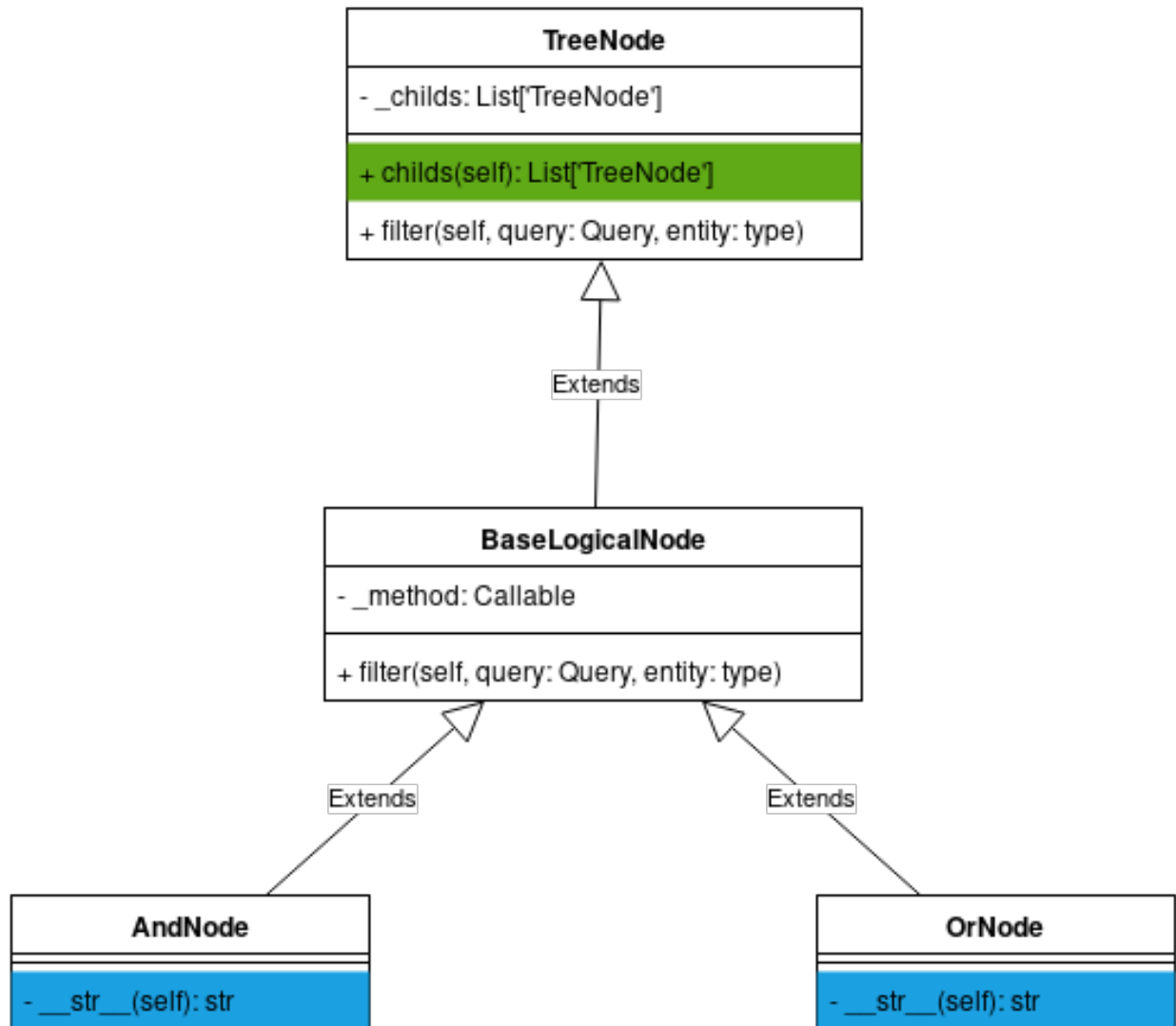
value

Property that return the value of the model attribute.

Returns The value of the model attribute.

Return type Any

Logical nodes



And

```
class sqa_filters.nodes.logical.AndNode
```

Represent the and operation from sqlalchemy.

When the filter method is called on this node it run on all of it's childs to create the filters list and apply the `and_` function to this list.

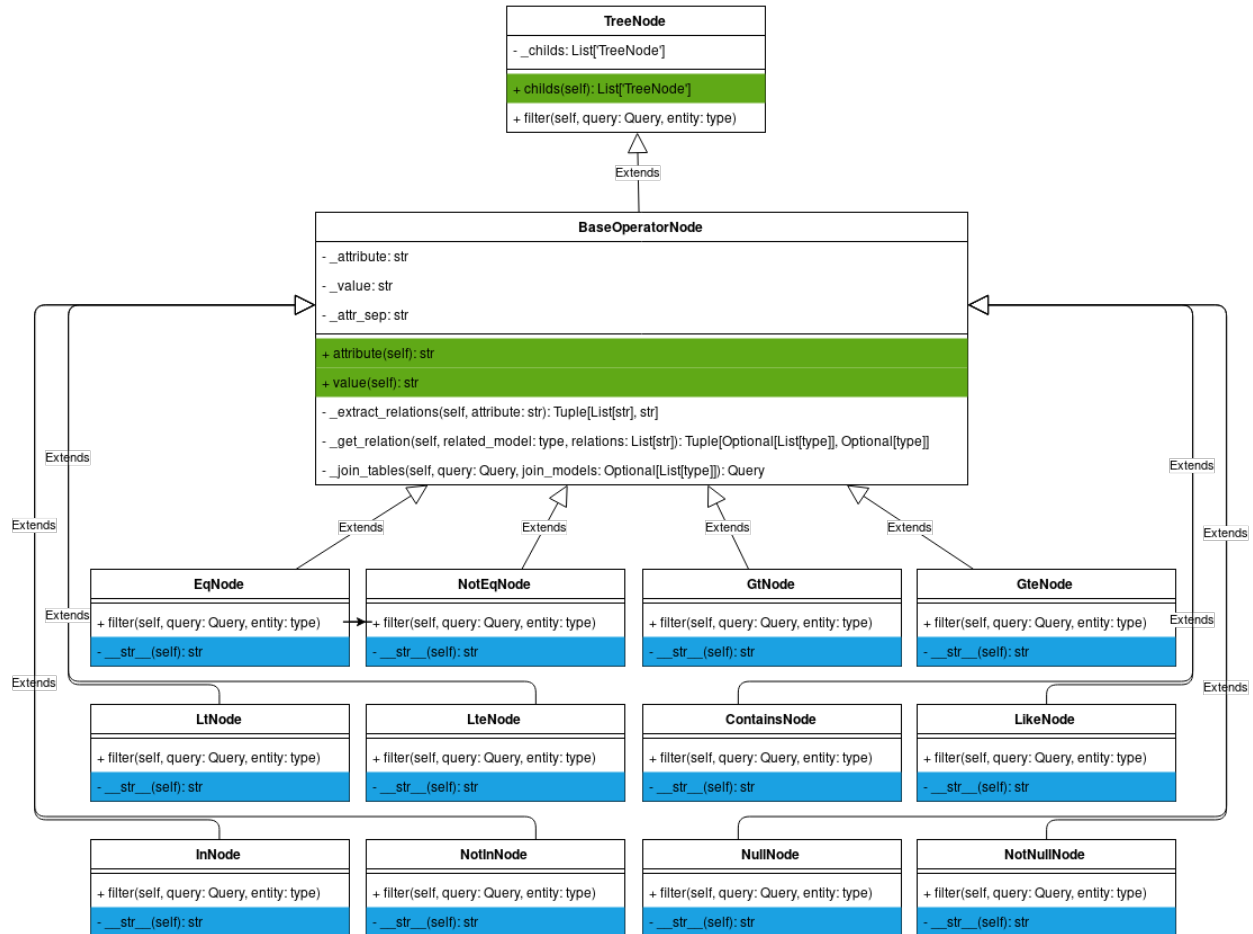
Or

```
class sqa_filters.nodes.logical.OrNode
```

Represent the or operation from sqlalchemy.

When the filter method is called on this node it run on all of it's childs to create the filters list and apply the `or_` function to this list.

Operational nodes



Equality

class `sqla_filters.nodes.operational.EqNode` (*attribute: str, value: Any, attr_sep: str = '.'*)
EqNode class.

This node test the equality between two values. Internally it use the *operators.eq* function available in *sqlalchemy.sql.operators*.

class `sqla_filters.nodes.operational.NotEqNode` (*attribute: str, value: Any, attr_sep: str = '.'*)
NotEqNode class.

This node test the non equality between two values. Internally it use the *operators.ne* function available in *sqlalchemy.sql.operators*.

Greater / Greater equal

class `sqla_filters.nodes.operational.GtNode` (*attribute: str, value: Any, attr_sep: str = '.'*)
GtNode class.

This node test if a value is greater than another one. Internally it use the *operators.gt* function available in *sqlalchemy.sql.operators*.

```
class sqala_filters.nodes.operational.GteNode (attribute: str, value: Any, attr_sep: str =  
                                                ``')  
GteNode class.
```

This node test if a value is greater or equal to another one. Internally it use the *operators.ge* function available in *sqlalchemy.sql.operators*.

Lower / Lower equal

```
class sqala_filters.nodes.operational.LtNode (attribute: str, value: Any, attr_sep: str = ``')  
LtNode class.
```

This node test if a value is lower than another one. Internally it use the *operators.lt* function available in *sqlalchemy.sql.operators*.

```
class sqala_filters.nodes.operational.LteNode (attribute: str, value: Any, attr_sep: str =  
                                                ``')  
LteNode class.
```

This node test if a value is lower or equal to another one. Internally it use the *operators.le* function available in *sqlalchemy.sql.operators*.

Contains

```
class sqala_filters.nodes.operational.ContainsNode (attribute: str, value: Any, attr_sep:  
                                                    str = ``')  
ContainsNode class.
```

This node test if an attribut contains the value. Internally it use the *operators.contains* function available in *sqlalchemy.sql.operators*.

Like

```
class sqala_filters.nodes.operational.LikeNode (attribute: str, value: Any, attr_sep: str =  
                                                ``')  
ContainsNode class.
```

This node test if an attribut is like the value. This function have the behavior of the *LIKE* in the sql language. This node use the *attr.like* function of a model attribute.

In

```
class sqala_filters.nodes.operational.InNode (attribute: str, value: Any, attr_sep: str = ``')  
InNode class.
```

This node test if an attribut is in a list of values. This function have the behavior of the *in* in the sql language. This node use the *attr.in* function of a model attribute.

```
class sqala_filters.nodes.operational.NotInNode (attribute: str, value: Any, attr_sep: str =  
                                                ``')  
NotInNode class.
```

This node test if an attribut is not in a list of values. This function have the behavior of the *not in* in the sql language. This node use the `~attr.in_` function of a model attribute.

Null

class `sqila_filters.nodes.operational.InNode` (*attribute: str, value: Any, attr_sep: str = '.'*)
 InNode class.

This node test if an attribut is in a list of values. This function have the behavior of the *in* in the sql language. This node use the `attr.in` function of a model attribute.

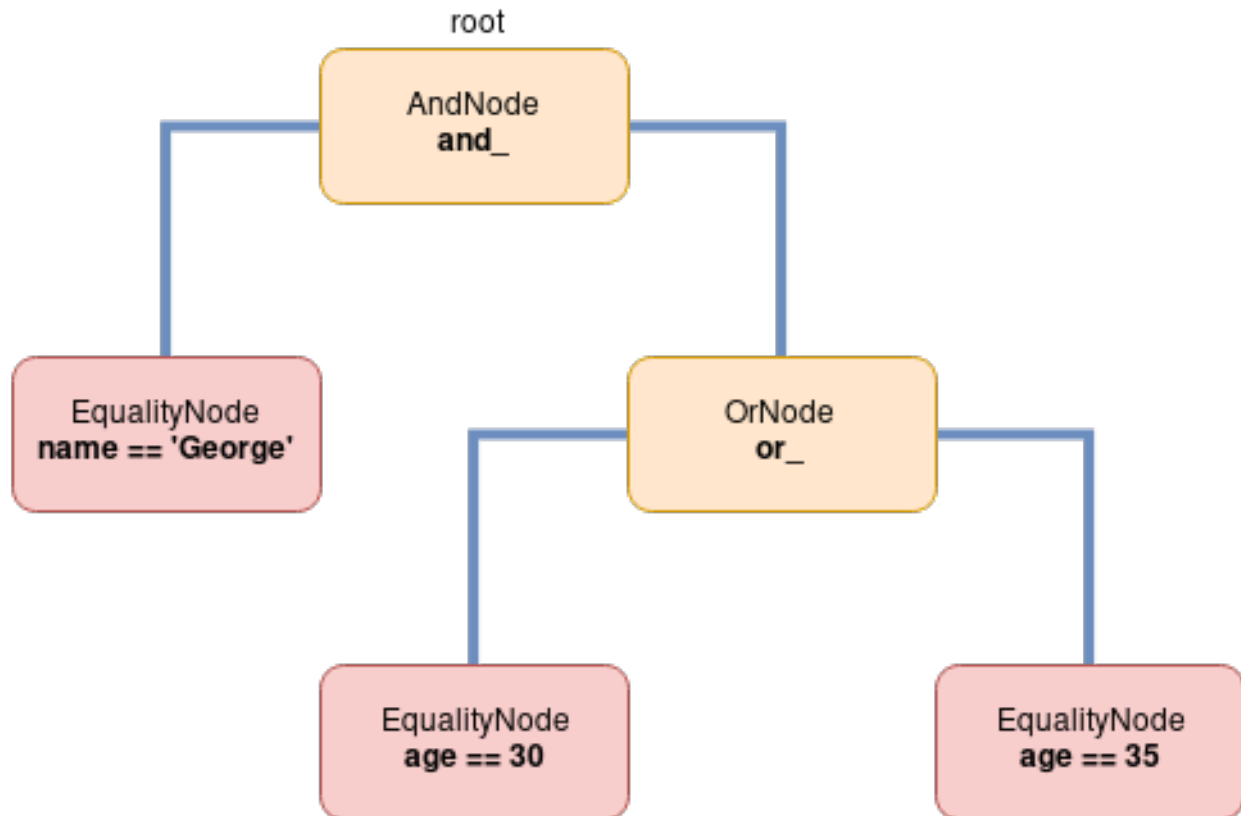
class `sqila_filters.nodes.operational.NotInNode` (*attribute: str, value: Any, attr_sep: str = '.'*)
 NotInNode class.

This node test if an attribut is not in a list of values. This function have the behavior of the *not in* in the sql language. This node use the `~attr.in_` function of a model attribute.

2.1.2 sqila-filters: Tree

Once the parser parses the data that was given it creates a tree. This tree is composed of the nodes found in the package `sqila_filter.filter.nodes`.

The generated tree is of the following form:



- Logical nodes
- Operator nodes

The class that contains this tree is the next class:

```
class sqa_filters.tree.SqaFilterTree (root: sqa_filters.nodes.base.base.TreeNode)
    Class SqaFilterTree.
```

When you access the `parser.tree` an instance of the class is returned. From the class you can access the root element and filter a sqlalchemy query.

Note: This is from this class you can call the filter function.

2.2 Developer Guide

2.2.1 sqa-filters: Plugins

Create your own plugin parser

You can of course create your own parser. Because you probably use a format that's specific to your usage you can create a parser to manage this new format.

Two possibilities are offered to you:

1. Namespace package
2. Standalone package

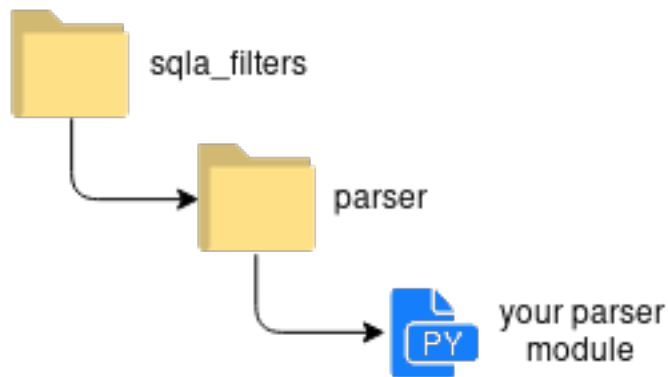
Namespace package

You can create a namespace package that injects the package code into the namespace “sqla_filter.parser “. This makes it possible to have a better logic in the e.g.

```
from sqla_filter.parser.<your module>import <your parser>
```

The creation of namespace package is as easy as a standalone package. You just need to respect some conventions.

You must to have the following directory structure:



If you want more information about the namespace packages you can read the documentation [packaging-namespaces](#).

You can also get more informations in the [pep420](#) that’s the chosen method for the sqla-filters project.

Standalone package

You can create a “standalone ” package as you normally do. The only thing that change compared to the namespace package will be the include.

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`

Symbols

`_extract_relations()` (sqla_filters.nodes.base.BaseOperationalNode method), 6
`_get_relation()` (sqla_filters.nodes.base.BaseOperationalNode method), 6
`_join_tables()` (sqla_filters.nodes.base.BaseOperationalNode method), 7

A

`AndNode` (class in sqla_filters.nodes.logical), 8
`attribute` (sqla_filters.nodes.base.BaseOperationalNode attribute), 7

B

`BaseLogicalNode` (class in sqla_filters.nodes.base), 6
`BaseOperationalNode` (class in sqla_filters.nodes.base), 6

C

`childs` (sqla_filters.nodes.base.TreeNode attribute), 6
`ContainsNode` (class in sqla_filters.nodes.operational), 10

E

`EqNode` (class in sqla_filters.nodes.operational), 9

F

`filter()` (sqla_filters.nodes.base.BaseLogicalNode method), 6
`filter()` (sqla_filters.nodes.base.BaseOperationalNode method), 7
`filter()` (sqla_filters.nodes.base.TreeNode method), 6

G

`GteNode` (class in sqla_filters.nodes.operational), 10
`GtNode` (class in sqla_filters.nodes.operational), 9

I

`InNode` (class in sqla_filters.nodes.operational), 10, 11

L

`LteNode` (class in sqla_filters.nodes.operational), 10
`LteNode` (class in sqla_filters.nodes.operational), 10
`LetNode` (class in sqla_filters.nodes.operational), 10

N

`NotEqNode` (class in sqla_filters.nodes.operational), 9
`NotInNode` (class in sqla_filters.nodes.operational), 10, 11

O

`OrNode` (class in sqla_filters.nodes.logical), 8

S

`SqlaFilterTree` (class in sqla_filters.tree), 12

T

`TreeNode` (class in sqla_filters.nodes.base), 5

V

`value` (sqla_filters.nodes.base.BaseOperationalNode attribute), 7