Winnow Documentation

Release 0.0.1

Brian Schiller

May 29, 2017

Contents:

Winnow is a Python package for safely building SQL where clauses from untrusted user input. It's designed to be expressive, extensible, and fast. Winnow's inputs look something like this:

```
{
    "logical_op": "&",
    "filter_clauses": [
        {
            "data_source": "Created",
            "operator": "before",
            "value": "2015-03-01"
        },
        {
            "data_source": "Owner",
            "operator": "any of",
            "value": [
                {"name": "Steven", "id": 23},
                {"name": "Margaret", "id": 41},
                {"name": "Evan", "id": 90}
            ]
        }
    ]
}
```

And its outputs looks like this:

```
(
  "WHERE created_date < %s::timestamp AND owner_id = ANY(VALUES (%s),(%s),(%s))",
  ('2015-03-01', 23, 41, 90)
)
```

CHAPTER 1

Quickstart

Installation

Winnow is available from PyPI (I mean, it's not right now, but it will be).

TODO upload to PyPI and update the docs here.

Define your Sources

Your sources will usually be a list of the columns you want to make available for filtering. Each source needs a display name and a list of the *value_types* it supports. See valuetypes for a list of included value types.

```
sources = [
    {
        'display_name': 'Order Date',
        'column': 'order_date',
        'value_types': ['absolute_date', 'relative_date'] },
    {
        'display_name': 'Number Scoops',
        'column': 'num_scoops',
        'value_types': ['numeric', 'nullable'] },
    {
        'display_name': 'Flavor',
        'column': 'flavor',
        'value_types': ['collection'],
        'picklist_options': [
            'Mint Chocolate Chip',
            'Cherry Garcia',
            'Chocolate',
            'Cookie Dough',
            'Rocky Road',
            'Rainbow Sherbet',
            'Strawberry',
```

```
            'Vanilla',
            'Coffee',
        ]},
]
```

Create a Filter

Use the sources you defined to build a JSON filter. The value types specified on each source determine which operators are available.

```
ice_cream_filt = {
    'logical_op': '&',
    'filter_clauses': [
        {'data_source': 'Number Scoops', 'operator': '>=', 'value': '2'},
        {'data_source': 'Flavor', 'operator': 'any of', 'value': [
            'Strawberry',
            'Chocolate',
         ]}
    ]
}
```

Get a SQL Query

Now initialize a `Winnow()` instance using your sources, and the name of the table you're filtering against. Turn your filter into a query.

```
ice_cream_winnow = Winnow('ice_cream', sources)
query, params = ice_cream_filt.query(ice_cream_filt)
# query => SELECT * FROM ice_cream WHERE ((num_scoops >= %s) AND (flavor IN (%s,%s) ))
# params => (2, 'Strawberry', 'Chocolate')
```

CHAPTER 2

Value Types and Operators

The existing value types and operators can be easily extended by subclassing Winnow.

numeric

The numeric value type provides operators for >=, <=, >, <, `is`, and `is not`.

```
{
    'data_source': 'Number Scoops',
    'operator': '>=',
    'value': 3,
}
```

string

Data sources marked a supporting the string value type can use the `is`, `is not`, `contains`, `starts with`, `more than __ words`, and `fewer than __ words` operators.

```
{
    'data_source': "Scooper's Name",
    'operator': 'more than __ words',
    'value': 3,
},
{
    'data_source': "Scooper's Name",
    'operator': 'contains',
    'value': 'Heidi',
}
```

collection

To use the collection operators, a data source will usually need to provide a list of picklist_options to the client. It's fine to include those directly on the data source object:

```
{
    'display_name': 'Flavor',
    'column': 'flavor',
    'value_types': ['collection'],
    'picklist_options': [
        'Mint Chocolate Chip',
        'Cherry Garcia',
        'Chocolate',
        'Cookie Dough',
        'Rocky Road',
        'Rainbow Sherbet',
        'Strawberry',
        'Vanilla',
        'Coffee',
     ]
}
```

Collections have access to `any of` and `not any of` operators.

```
{
    'data_source': 'Flavor',
    'operator': 'any of',
    'value': ['Strawberry', 'Chocolate'],
}
```

Datetime Operators

Datetime operators are broken down into two sets, relative and absolute. Most timestamp sources will want to support both.

absolute_date

Absolute date values are ISO8601 strings, like `"2017-03-22T18:14:30"`. The supported operators are `before` and `after`.

```
{
    'data_source': 'Purchase Date',
    'operator': 'after',
    'value': '2017-03-22T18:14:30',
}
```

relative_date

Relative date values are also strings, but they're things like `"last_30_days"` and `"current_month"`. I'm not very happy with how these are designed, so they will likely change in a future version. Please let me know if you have any advice. Maybe there's already a standard way to refer to intervals of time that aren't anchored to a particular day?

```
{
    'data_source': 'Purchase Date',
    'operator': 'within',
    'value': 'last_7_days',
}
```

The list of available values is found in relative_dates.py.

CHAPTER 3

Extending Winnow

Adding Operators

Adding value types

relative_to_date_field

This will allow us to say “Expected close Date before <any other date field>”.

historical

Stage was ‘Prospecting’ as of <date>

Adding custom fields

User-specific fields, generated dynamically

Replacing relative date handling

“We want to start our year in February.”

CHAPTER 4

Indices and tables

- genindex
- modindex
- search