Python Client for Pilosa Documentation

Release 0.3.30

Pilosa Engineering

Jun 12, 2017

Contents:

Python client for Pilosa high performance distributed bitmap index.

CHAPTER 1

pilosa package

Submodules

pilosa.client module

class pilosa.client.**Client**(*cluster_or_uri=None*, *connect_timeout=30000*, *socket_timeout=300000*, *pool_size_per_route=10*, *pool_size_total=100*, *retry_count=3*)

Bases: object

Pilosa HTTP client

This client uses Pilosa's http+protobuf API.

Usage:

```
import pilosa

# Create a Client instance
client = pilosa.Client()

# Create an Index instance
index = pilosa.Index("repository")

stargazer = index.frame("stargazer")
response = client.query(stargazer.bitmap(5))

# Act on the result
print(response.result)
```

- See Pilosa API Reference.
- See Query Language.

create_frame(*frame*)
Creates a frame on the server using the given Frame object.

Parameters **frame** (*pilosa.Frame*) –

Raises **pilosa.FrameExistsError** – if there already is a frame with the given name

create_index(*index*)
Creates an index on the server using the given Index object.

Parameters **index** (*pilosa.Index*) –

Raises **pilosa.IndexExistsError** – if there already is a index with the given name

delete_frame(*frame*)
Deletes the given frame on the server.

Parameters **frame** (*pilosa.Frame*) –

Raises **pilosa.PilosaError** – if the frame does not exist

delete_index(*index*)
Deletes the given index on the server.

Parameters **index** (*pilosa.Index*) –

Raises **pilosa.PilosaError** – if the index does not exist

ensure_frame(*frame*)
Creates a frame on the server if it does not exist.

Parameters **frame** (*pilosa.Frame*) –

ensure_index(*index*)
Creates an index on the server if it does not exist.

Parameters **index** (*pilosa.Index*) –

query(*query*, *columns=False*, *time_quantum=<pilosa.orm.TimeQuantum instance>*)
Runs the given query against the server with the given options.

Parameters

- **query** (*pilosa.PqlQuery*) – a PqlQuery object with a non-null index
- **columns** (*bool*) – Enables returning column data from bitmap queries
- **time_quantum** (*pilosa.TimeQuantum*) – Sets the time quantum for this query

Returns Pilosa response

Return type pilosa.Response

class pilosa.client.**Cluster**(**hosts*)
Contains hosts in a Pilosa cluster.

Parameters **hosts** – URIs of hosts. Leaving out hosts creates the default cluster

add_host(*uri*)
Adds a host to the cluster.

Parameters **uri** (*pilosa.URI*) –

get_host()
Returns the next host in the cluster.

Returns next host

Return type pilosa.URI

remove_host(*uri*)
Removes the host with the given URI from the cluster.

Parameters **uri** (*pilosa.URI*) –

class pilosa.client.**URI**(*scheme='http'*, *host='localhost'*, *port=10101*)
Represents a Pilosa URI

A Pilosa URI consists of three parts:

•Scheme: Protocol of the URI. Default: `http`

•Host: Hostname or IP URI. Default: `localhost`

•Port: Port of the URI. Default `10101`

All parts of the URI are optional. The following are equivalent:

•`http://localhost:10101`

•`http://localhost`

•`http://:10101`

•`localhost:10101`

•`localhost`

•`:10101`

Parameters

- **scheme** (*str*) – is the scheme of the Pilosa Server. Currently only `http` is supported
- **host** (*str*) – is the hostname or IP address of the Pilosa server
- **port** (*int*) – is the port of the Pilosa server

classmethod address(*address*)
Creates a URI from an address.

Parameters **address** (*str*) – of the form `${SCHEME}://${HOST}:{$PORT}`

Returns a Pilosa URI

Type pilosa.URI

pilosa.exceptions module

exception pilosa.exceptions.**PilosaError**
Bases: exceptions.Exception

exception pilosa.exceptions.**ValidationError**
Bases: *pilosa.exceptions.PilosaError*

exception pilosa.exceptions.**PilosaURIError**
Bases: *pilosa.exceptions.PilosaError*

exception pilosa.exceptions.**IndexExistsError**
Bases: *pilosa.exceptions.PilosaError*

exception pilosa.exceptions.**FrameExistsError**
Bases: *pilosa.exceptions.PilosaError*

pilosa.orm module

class pilosa.orm.**TimeQuantum**(*value*)
Valid time quantum values for frames having support for that.

•See: Data Model

DAY = <pilosa.orm.TimeQuantum instance>

DAY_HOUR = <pilosa.orm.TimeQuantum instance>

HOUR = <pilosa.orm.TimeQuantum instance>

MONTH = <pilosa.orm.TimeQuantum instance>

MONTH_DAY = <pilosa.orm.TimeQuantum instance>

MONTH_DAY_HOUR = <pilosa.orm.TimeQuantum instance>

NONE = <pilosa.orm.TimeQuantum instance>

YEAR = <pilosa.orm.TimeQuantum instance>

YEAR_MONTH = <pilosa.orm.TimeQuantum instance>

YEAR_MONTH_DAY = <pilosa.orm.TimeQuantum instance>

YEAR_MONTH_DAY_HOUR = <pilosa.orm.TimeQuantum instance>

class pilosa.orm.**CacheType**(*value*)

DEFAULT = <pilosa.orm.CacheType instance>

LRU = <pilosa.orm.CacheType instance>

RANKED = <pilosa.orm.CacheType instance>

class pilosa.orm.**Index**(*name*, *column_label='columnID'*, *time_quantum=<pilosa.orm.TimeQuantum instance>*)
The purpose of the Index is to represent a data namespace.

You cannot perform cross-index queries. Column-level attributes are global to the Index.

Parameters

- **name** (*str*) – index name
- **column_label** (*str*) – a valid column label
- **time_quantum** (*pilosa.TimeQuantum*) – Sets the time quantum

•See Data Model

•See Query Language

batch_query(**queries*)
Creates a batch query.

Parameters **queries** (*pilosa.PQLQuery*) – the queries in the batch

Returns Pilosa batch query

Return type pilosa.PQLBatchQuery

count(*bitmap*)
Creates a Count query.

`Count` returns the number of set bits in the BITMAP_CALL passed in.

Parameters **bitmap** (*pilosa.PQLQuery*) – the bitmap query

Returns Pilosa query

Return type pilosa.PQLQuery

difference(**bitmaps*)
Creates a `Difference` query.

`Difference` returns all of the bits from the first BITMAP_CALL argument passed to it, without the bits from each subsequent BITMAP_CALL.

Parameters **bitmaps** (*pilosa.PQLBitmapQuery*) – 0 or more bitmap queries to differentiate

Returns Pilosa bitmap query

Return type pilosa.PQLBitmapQuery

Raises ***PilosaError*** – if the number of bitmaps is less than 1

frame(*name*, *row_label='rowID'*, *time_quantum=<pilosa.orm.TimeQuantum instance>*, *inverse_enabled=False*, *cache_type=<pilosa.orm.CacheType instance>*, *cache_size=0*)
Creates a frame object with the specified name and defaults.

Parameters

- **name** (*str*) – frame name
- **row_label** (*str*) – a valid row label
- **time_quantum** (*pilosa.TimeQuantum*) – Sets the time quantum for the frame. If a Frame has a time quantum, then Views are generated for each of the defined time segments.
- **inverse_enabled** (*bool*) –
- **cache_type** (*pilosa.CacheType*) – `CacheType.DEFAULT`, `CacheType.LRU` or `CacheType.RANKED`
- **cache_size** (*int*) – Values greater than 0 sets the cache size. Otherwise uses the default cache size

Returns Pilosa frame

Return type pilosa.Frame

intersect(**bitmaps*)
Creates an `Intersect` query.

`Intersect` performs a logical AND on the results of each BITMAP_CALL query passed to it.

Parameters **bitmaps** (*pilosa.PQLBitmapQuery*) – 1 or more bitmap queries to intersect

Returns Pilosa bitmap query

Return type pilosa.PQLBitmapQuery

Raises ***PilosaError*** – if the number of bitmaps is less than 1

raw_query(*query*)
Creates a raw query.

Note that the query is not validated before sending to the server.

Parameters **query** (*str*) –

Returns Pilosa query

Return type pilosa.PQLQuery

set_column_attrs(*column_id*, *attrs*)
Creates a SetColumnAttrs query.

`SetColumnAttrs` associates arbitrary key/value pairs with a column in an index.

Following object types are accepted:

•int

•str

•bool

•float

Parameters

- **column_id** (*int*) –
- **attrs** (*dict*) – column attributes

Returns Pilosa query

Return type pilosa.PQLQuery

union(**bitmaps*)
Creates a `Union` query.

`Union` performs a logical OR on the results of each BITMAP_CALL query passed to it.

Parameters **bitmaps** (*pilosa.PQLBitmapQuery*) – 0 or more bitmap queries to union

Returns Pilosa bitmap query

Return type pilosa.PQLBitmapQuery

class `pilosa.orm.`**Frame**(*index*, *name*, *row_label*, *time_quantum*, *inverse_enabled*, *cache_type*, *cache_size*)
Frames are used to segment and define different functional characteristics within your entire index.

You can think of a Frame as a table-like data partition within your Index. Row-level attributes are namespaced at the Frame level.

Do not create a Frame object directly. Instead, use `pilosa.Index.frame` method.

•See Data Model

•See Query Language

bitmap(*row_id*)
Creates a Bitmap query.

Bitmap retrieves the indices of all the set bits in a row or column based on whether the row label or column label is given in the query. It also retrieves any attributes set on that row or column.

This variant of Bitmap query uses the row label.

Parameters **row_id** (*int*) –

Returns Pilosa bitmap query

Return type pilosa.PQLBitmapQuery

clearbit(*row_id*, *column_id*)
Creates a ClearBit query.

`ClearBit` assigns a value of 0 to a bit in the binary matrix, thus disassociating the given row in the given frame from the given column.

Parameters

- **row_id** (*int*) –
- **column_id** (*int*) –

Returns Pilosa query

Return type pilosa.PQLQuery

inverse_bitmap(*column_id*)
Creates a Bitmap query.

`Bitmap` retrieves the indices of all the set bits in a row or column based on whether the row label or column label is given in the query. It also retrieves any attributes set on that row or column.

This variant of Bitmap query uses the column label.

Parameters **column_id** (*int*) –

Returns Pilosa bitmap query

Return type pilosa.PQLBitmapQuery

inverse_range(*column_id*, *start*, *end*)
Creates a Range query.

Similar to `Bitmap`, but only returns bits which were set with timestamps between the given start and end timestamps.

Parameters

- **column_id** (*int*) –
- **start** (*datetime.datetime*) – start timestamp
- **end** (*datetime.datetime*) – end timestamp

inverse_topn(*n*, *bitmap=None*, *field=''*, **values*)
Creates a TopN query.

`TopN` returns the id and count of the top n bitmaps (by count of bits) in the frame.

This version sets *inverse=true*.

•see: TopN Query

Parameters

- **n** (*int*) – number of items to return
- **bitmap** (*pilosa.PQLBitmapQuery*) – a PQL Bitmap query
- **str field** (*field*) – field name
- **values** (*object*) – filter values to be matched against the field

range(*row_id*, *start*, *end*)
Creates a Range query.

Similar to `Bitmap`, but only returns bits which were set with timestamps between the given start and end timestamps.

•see: Range Query

Parameters

- **row_id** (*int*) –
- **start** (*datetime.datetime*) – start timestamp
- **end** (*datetime.datetime*) – end timestamp

set_row_attrs(*row_id*, *attrs*)
Creates a SetRowAttrs query.

`SetRowAttrs` associates arbitrary key/value pairs with a row in a frame.

Following object types are accepted:

•int

•str

•bool

•float

Parameters

- **row_id** (*int*) –
- **attrs** (*dict*) – row attributes

Returns Pilosa query

Return type pilosa.PQLQuery

setbit(*row_id*, *column_id*, *timestamp=None*)
Creates a SetBit query.

`SetBit` assigns a value of 1 to a bit in the binary matrix, thus associating the given row in the given frame with the given column.

Parameters

- **row_id** (*int*) –
- **column_id** (*int*) –
- **timestamp** (*pilosa.TimeStamp*) –

Returns Pilosa query

Return type pilosa.PQLQuery

topn(*n*, *bitmap=None*, *field=''*, **values*)
Creates a TopN query.

`TopN` returns the id and count of the top n bitmaps (by count of bits) in the frame.

•see: TopN Query

Parameters

- **n** (*int*) – number of items to return
- **bitmap** (*pilosa.PQLBitmapQuery*) – a PQL Bitmap query
- **str field** (*field*) – field name
- **values** (*object*) – filter values to be matched against the field

class pilosa.orm.**PQLQuery**(*pql*, *index*)

serialize()

class pilosa.orm.**PQLBatchQuery**(*index*)

add(**queries*)

serialize()

pilosa.response module

class pilosa.response.**BitmapResult**(*bits=None*, *attributes=None*)
Represents a result from Bitmap, Union, Intersect, Difference and Range PQL calls.

•See Query Language

classmethod from_internal(*obj*)

class pilosa.response.**CountResultItem**(*id*, *count*)
Represents a result from TopN call.

•See Query Language

class pilosa.response.**QueryResult**(*bitmap=None*, *count_items=None*, *count=0*)
Represent one of the results in the response.

•See Query Language

classmethod from_internal(*obj*)

class pilosa.response.**ColumnItem**(*id*, *attributes*)
Contains data about a column.

Column data is returned from QueryResponse.getColumns() method. They are only returned if Client.query was called with columns=True.

class pilosa.response.**QueryResponse**(*results=None*, *columns=None*, *error_message=''*)
Bases: object

Represents the response from a Pilosa query.

•See Query Language

column

result

pilosa.validator module

pilosa.validator.**valid_index_name**(*index_name*)

pilosa.validator.**validate_index_name**(*index_name*)

pilosa.validator.**valid_frame_name**(*frame_name*)

pilosa.validator.**validate_frame_name**(*frame_name*)

pilosa.validator.**valid_label**(*label*)

pilosa.validator.**validate_label**(*label*)

pilosa.version module

pilosa.version.**get_version**()
Returns the version being used

Module contents

CHAPTER 2

Requirements

- Python 2.6 and higher or Python 3.3 and higher

CHAPTER 3

Install

Pilosa client is on PyPI. You can install the library using `pip`:

```
pip install pilosa
```

CHAPTER 4

Quick overview

Assuming Pilosa server is running at `localhost:10101` (the default):

```
import pilosa

# Create the default client
client = pilosa.Client()

# Create an Index object
myindex = pilosa.Index("myindex")

# Make sure the index exists on the server
client.ensure_index(myindex)

# Create a Frame object
myframe = myindex.frame("myframe")

# Make sure the frame exists on the server
client.ensure_frame(myframe)

# Send a SetBit query. PilosaError is thrown if execution of the query fails.
client.query(myframe.setbit(5, 42))

# Send a Bitmap query. PilosaError is thrown if execution of the query fails.
response = client.query(myframe.bitmap(5))

# Get the result
result = response.result

# Act on the result
if result:
    bits = result.bitmap.bits
    print("Got bits: ", bits)

# You can batch queries to improve throughput
response = client.query(
```

```
    myindex.batch_query(
        myframe.bitmap(5),
        myframe.bitmap(10),
    )
)
for result in response.results:
    # Act on the result
    print(result)
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Q

R

S

T

U

V

Y