pypika Documentation

Release 2.1.2

Timothy Heys

May 10, 2019

Contents

fireant is a a data analysis tool used for quickly building charts, tables, reports, and dashboards. It defines a schema for configuring metrics and dimensions which removes most of the leg work of writing queries and formatting charts. *fireant* even works great with Jupyter notebooks and in the Python shell providing quick and easy access to your data.

CHAPTER 1

Contents

1.1 Installation and Setup

To install *fireant*, run the following command in the terminal:

```
pip install fireant
```

1.1.1 Database Connector add-ons

By default, *fireant* does not include any database drivers. You can optionally include one or provide your own. The following extra installations will set up *fireant* with the recommended drivers. *fireant* may work with additional drivrers, but is untested and requires a custom extension of the `fireant.database.Database` class (see below).

```
# Vertica
pip install fireant[vertica]

# MySQL
pip install fireant[mysql]

# PostgreSQL
pip install fireant[postgresql]

# Amazon Redshift
pip install fireant[redshift]
```

1.1.2 Transformer add-ons

Some transformers have additional dependencies that are not included by default. Include the following extra installations in your *requirements.txt* file if you intend to use those transformers.

```
# matplotlib
pip install fireant[matplotlib]
```

1.2 Connecting to the database

In order for *fireant* to connect to your database, a database connectors must be used. This takes the form of an instance of a concrete subclass of *fireant*'s `Database` class. Database connectors are shipped with *fireant* for all of the supported databases, but it is also possible to write your own. See below on how to extend *fireant* to support additional databases.

To configure a database, instantiate a subclass of `fireant.database.Database`. You will use this instance to create a *Slicer*. It is possible to use multiple databases simultaneous, but *Slicer* can only use a single database, since they inherently model the structure of a table in the database.

Vertica

```
import fireant.settings
from fireant.database import VerticaDatabase

database = VerticaDatabase(
    host='example.com',
    port=5433,
    database='example',
    user='user',
    password='password123',
)
```

MySQL

```
import fireant.settings
from fireant.database import MySQLDatabase

database = MySQLDatabase(
    database='testdb',
    host='mysql.example.com',
    port=3308,
    user='user',
    password='password123',
    charset='utf8mb4',
)
```

MySQL additionally requires a custom function that *fireant* uses to rollup date values to specific intervals, equivalent to the `TRUNC_DATE` function available in other database platforms. To install the `TRUNC_DATE` function in your MySQL database, run the script found in `fireant/scripts/mysql_functions.sql`. Further information is provided in this script on how to grant permissions on this function to your MySQL users.

PostgreSQL

```
import fireant.settings
from fireant.database import PostgreSQLDatabase

database = PostgreSQLDatabase(
    database='testdb',
    host='example.com',
    port=5432,
```

(continues on next page)

(continued from previous page)

```
    user='user',
    password='password123',
)
```

Amazon Redshift

```
import fireant.settings
from fireant.database import RedshiftDatabase

fireant.settings = RedshiftDatabase(
    database='testdb',
    host='example.com',
    port=5439,
    user='user',
    password='password123',
)
```

1.2.1 Using a different Database

Instead of using one of the built in database connectors, you can provide your own by extending `fireant.database.Database`.

```
import vertica_python

class MyVertica(Database):
    # Vertica client that uses the vertica_python driver.

    def __init__(self, host='localhost', port=5433, database='vertica',
                 user='vertica', password=None,
                 read_timeout=None):
        self.host = host
        self.port = port
        self.database = database
        self.user = user
        self.password = password
        self.read_timeout = read_timeout

    def connect(self):
        return vertica_python.connect(
            host=self.host, port=self.port, database=self.database,
            user=self.user, password=self.password,
            read_timeout=self.read_timeout,
        )

    def trunc_date(self, field, interval):
        return Trunc(...)  # custom Trunc function

    def date_add(self, date_part, interval, field):
        return DateAdd(...)  # custom DateAdd function

hostage.settings = MyVertica(
    host='example.com',
    port=5433,
    database='example',
    user='user',
```

(continues on next page)

(continued from previous page)

```
    password='password123',
)
```

In a custom database connector, the `connect` function must be overridden to provide a `connection` to the database. The `trunc_date` and `date_add` functions must also be overridden since are no common ways to truncate/add dates in SQL databases.

1.3 Creating a *Slicer*

A *Slicer* consists of five main parts: A database connector, a primary database table, join tables, dimensions, and metrics. The database connector is a connection to a database where the primary and join tables exist. Both dimensions and metrics are given definitions using PyPika expressions which are effectively converted into SQL when a query is executed. Metrics are quantifiers that are aggregated across Dimensions. Dimensions and metrics can be thought of as `SELECT` and `GROUP BY` clauses in a SQL query, concretely that is how they are used.

Once a *Slicer* is configured, it can be used to write queries using a builder pattern. A query selects which dimensions and metrics to use and how to transform that data into a visualization such as a chart or table. Queries also provide a mechanism to filter on dimension and metric values, which is equivalent to the `WHERE` and `HAVING` clauses of the SQL query.

The key assumption in using a *Slicer* is that when analyzing data, definitions for dimensions or metrics do not change often, but which combinations of them are used and how the data is visualized does. Therefore, the *Slicer* removes much of the boilerplate in building visualizations and dashboards.

1.3.1 The slicer class

Creating a *Slicer* involves instantiating a `fireant.Slicer` with at a minimum a database, primary table, and one metric.

```
from fireant.slicer import *
from fireant.database.vertica import VerticaDatabase
from pypika import Tables, functions as fn

vertica_database = VerticaDatabase(user='jane_doe', password='strongpassword123')
analytics, customers = Tables('analytics', 'customers')

slicer = Slicer(
    database=vertica_database,
    table=analytics,

    joins=[
        Join(customers, analytics.customer_id == customers.id),
    ],

    dimensions=[
        UniqueDimension('customer',
                        definition=customers.id,
                        display_field=fn.Concat(customers.fname, ' ', customers.
↪lname))
    ],

    metrics=[
        Metric('clicks', 'Clicks'),
```

(continues on next page)

(continued from previous page)

```
    ],
)
```

1.3.2 Metrics

A *Metric* is a quantifier which is aggregated across dimensions when used in a *Slicer* query. A *Slicer* requires at a minimum one metric. Metrics are the values used to draw lines or bars in charts or fill cells in tables, the measurements in the data.

A *Metric* is represented in code by the class `fireant.slicer.metrics.Metric`. When instantiating a *Slicer*, at least one instance of `fireant.slicer.metrics.Metric` must be given in the `metrics` argument.

```
from pypika import Table
from fireant import Metric

analytics = Table('analytics')

roi = Metric('roi',
             label='Return on Investment',
             definition=fn.Sum(analytics.revenue) / fn.Sum(analytics.cost),
             precision=3,
             suffix='%'),
```

Note: When defining a `fireant.slicer.metrics.Metric`, it is important to note that all queries executed by fireant are aggregated over the dimensions (via a `GROUP BY` clause in the SQL query) and therefore are required to use aggregation functions. By default, a `fireant.slicer.metrics.Metric` will use the `SUM` function and it's `key`. A custom definition is commonly required and must use a SQL aggregate function over any columns.

1.3.3 Dimensions

There are different types of dimensions and choosing one depends on some properties of the column.

Continuous Dimensions

For dimensions that do not have a finite set of values, a continuous dimension can be used. This is especially useful for numeric values that can be viewed with varied precision, such as a decimal value. Continuous Dimensions require an additional parameter for an `Interval` which groups values into discrete segments. In a numerical example, values could be grouped by increments of 5.

Date/Time Dimensions

Date/Time Dimensions are a special type of continuous dimension which contain some predefined intervals: hours, days, weeks, months, quarters, and years. In any widget that displays time series data, a date/time dimension.

```
from fireant import DatetimeDimension

DatetimeDimension('timestamp',
                  label='Timestamp',
                  definition=customers.ts)
```

Boolean Dimensions

A boolean dimension is a dimension that represents a boolean field.

```
from fireant import BooleanDimension

BooleanDimension('is_active',
                 label='Active',
                 definition=customers.active)
```

Categorical Dimensions

A categorical dimension represents a column that contains one value from a finite set, such as a member of an enumeration. This is the most common type of dimension. For example, a *color* column could be used that contains values such as *red*, *green*, and *blue*. Aggregating metrics with this dimension will give a data set grouped by each color.

```
from fireant import CategoricalDimension

CategoricalDimension('device',
                     label='Device Type',
                     definition=customers.device
                     display_values=(
                        ('d', 'Desktop'),
                        ('m', 'Mobile'),
                        ('t', 'Tablet'),
                     ))
```

Unique Dimensions

A unique dimension represents a column that has one or more identifier columns and optionally a display field column. This is useful when your data contains a significant number of values that cannot be represented by a small list of categories and is akin to using a foreign key in a SQL table. In conjunction with a join on a foreign key, a display value can be selected from a second table and used when rendering your widgets.

```
from fireant import UniqueDimension

UniqueDimension('customer',
                label='Customer',
                definition=customers.id,
                display_field=fn.Concat(customers.fname, ' ', customers.lname))
```

1.3.4 Joins

A *Slicer* can be configured with additional join tables that will automatically be when using a metric or dimension that requires it. *fireant* determines that the join is required if the joined table defined in the join is used in the definition of a dimension or metric. Joins can also join from other joins.

A join is defined with two arguments: the join table and a conditional expression of how to join the table. The join expression should be used as a condition in the `ON` clause of a join in a SQL query.

```
from fireant import Join

```

(continues on next page)

(continued from previous page)

```
Join(customers, analytics.customer_id == customers.id)
Join(orders, (orders.customer_id == customers.id) & (orders.store_id == store.id))
```

1.4 Querying with a *Slicer*

A *Slicer* query uses a Slicer's configuration to execute a SQL query and to transform the result into a data visualization, or a widget. A single *fireant* query translates into a single SQL query, but it is possible to transform the a single query into multiple widgets. Queries are constructed with a builder pattern API.

A query object is created by calling `slicer.data`. Subsequent function call can be chained in order to build up a query.

When fetch is called, the SQL query is executed and the resulting data set is transformed for each widget in the query. The *fetch()* method returns an array containing the data for each widget in the order that the widget was added to the query.

Example

```
from fireant.slicer import *
from fireant.database.vertica import VerticaDatabase
from pypika import Tables, functions as fn

query = slicer.data \
    .widget( ... ) \
    .dimension( ... ) \
    .filter( ... ) \
    .reference( ... ) \
    .orderby( ... )

query.fetch()
```

All builder methods can be called multiple times and in any order.

1.4.1 Builder Functions

widget Add a widget to a query. A widget is what is returned from the call to `fetch` containing data for a visualization. At least one widget should be used, otherwise the raw data will be returned in a pandas Data Frame. See *Visualizing the data with Widgets* below for more details.

The function takes one or more arguments that should be instances of subclasses of `fireant.slicer.widgets.base.Widget`. Passing two arguments is synonymous with calling this function twice with each widget.

```
from fireant import Pandas

slicer.data \
    ...
   .widget( Pandas(slicer.metrics.clicks, slicer.metrics.cost, slicer.metrics.
↪revenue) )
```

dimension Add a dimension to the query. This adds a grouping level to the query that will be used to break the data down. See *Grouping data with Dimensions* below for more details.

The function takes one or more arguments that must be references to dimensions of the slicer that the query is being built from.

```
slicer.data \
    ...
   .dimension( slicer.dimensions.device, slicer.dimensions.account )
```

filter Add a filter to the query. This constrains the results of the data by certain criteria. There are many types of filters, some apply to metrics and others apply to dimenions. See *Filtering the query* below for more details.

The function takes one or more arguments of filter expressions using elements of the slicer that the query is being built from.

```
slicer.data \
    ...
   .filter( slicer.dimensions.date.between(date(2000, 1, 1), date(2009, 12, 31)) ) \
   .filter( slicer.dimensions.device.isin(['m', 't']) ) \
   .filter( slicer.metrics.clicks > 10 )
```

reference A reference is a way of comparing the data to itself over an interval of time using a Date/Time dimension, such as Week-over-Week. See *Comparing Data to Previous Values using References* below for more details.

```
from fireant import WeekOverWeek

slicer.data \
    ...
   .reference( WeekOverWeek() )
```

orderby This function allows the results of the SQL query to be ordered by dimensions and/or metrics. Please note that this will *only* order the results of the SQL query and that the order may be affected by the Widget. Ordering is entirely optional. The default order will be by all of the dimensions used in the query in the order that they were added.

```
from pypika import Order

slicer.data \
    ...
   .orderby( slicer.metrics.clicks, Order.asc )
```

fetch A call to fetch exits the build function chain and returns the results of the query. An optional hint parameter is accepted which will used in the query if monitoring the queries triggered from *fireant* fireant is needed.

```
from pypika import Order

slicer.data \
    ...
   .fetch()
```

1.4.2 Grouping data with Dimensions

Dimensions are referenced using the alias defined for them when instantiating the slicer via the `dimensions` attribute on the slicer instance.

```
slicer = Slicer(
    ...
    dimensions=[
```

(continues on next page)

(continued from previous page)

```
        UniqueDimension('customer', ... ),
        ...
    ],
    ...
)

# Reference to customer dimension
slicer.dimensions.customer
```

A dimension can be used in a slicer query by calling the *.dimension(...)* method when building a query. A reference to one or more dimensions must be passed as an argument.

The order of the dimensions is important. The dimensions are grouped in the order that they are added and displayed in the widgets in that order.

```
slicer.data \
    ...
   .dimension( slicer.dimensions.device, slicer.dimensions.account )
```

Using intervals with Date/Time dimensions

All continuous dimensions require an interval to group into. For a Date/Time dimension, these intervals are common temporal intervals, such as hours, days, quarters, etc. These dimensions have a default interval and can be used without explicitly setting one. To set the interval, use the reference to the dimension as a function and pass the interval as an arguement

```
from fireant import monthly

slicer.data \
    ...
   .dimension( slicer.dimensions.date(monthly) )
```

The following intervals are available for Date/Time dimensions and can be imported directly from the `fireant` package.

- `hourly`
- `daily`
- `weekly`
- `monthly`
- `quarterly`
- `annually`

It is also possible to define a custom interval as an instance of `fireant.DatetimeInterval`.

Roll Up (Totals)

Rolling up a dimension allows the totals across a dimension to be displayed in addition to the breakdown for each dimension value. To enable rollup for a dimension, call the rollup method on the dimension reference. Rollup is available for all dimension types.

```
slicer.data \
    ...
   .dimension( slicer.dimensions.date(hourly).rollup() ) \
   .dimension( slicer.dimensions.device.rollup() )
```

1.4.3 Filtering the query

A query can be filtered using several different filters. Some filter types are used with metrics while others work with dimensions. A metric filter is synonomous with the `HAVING` clause in a SQL query whereas a dimension filter corresponds to the `WHERE` clause. Dimension filters can also be applied to the display definition of Unique Dimensions.

When more than one filter is applied to a query, the results will be filtered to all rows/groups matching **all** of the conditions like a boolean `AND`. Some filters accept multiple values which create multiple conditions and filter data to rows/groups matching **any** of the conditions like a boolean `OR`.

Comparator (Metrics)

Comparator filters are created using standard operators:

- ==
- *!=*
- >
- >=
- <
- <=

```
slicer.data \
    ...
   .filter( slicer.metrics.clicks >= 100 ) \
   .filter( slicer.metrics.conversions == 1 )
```

Boolean (Boolean Dimensions)

Boolean filters only apply to boolean dimensions and filter whether the value of that boolean dimension is *True* or False‘ using the *.is_(True/False)* method on a `fireant.slicer.dimensions.BooleanDimension`.

```
slicer.data \
    ...
   .filter( slicer.dimensions.is_member.is_(True) )
```

Range (Date/Time dimensions)

Range filters apply to `fireant.slicer.dimensions.DatetimeDimension` dimensions using the *.between(start, end)* method. This is equivalent to a *BETWEEN* expression in the SQL query.

```
slicer.data \
    ...
   .filter( slicer.dimensions.date.between( datetime(2018, 8, 21), datetime(2019, 8,␣
↪20) ) )
```

(continues on next page)

(continued from previous page)

```
```

Includes (Category and Unique dimensions)

Includes filters apply to `fireant.slicer.dimensions.CategoricalDimension` and `fireant.slicer.dimensions.UniqueDimension` dimensions using the *.isin(list)* method. Results will be included if they are equal to any of the values in the argument supplied.

Combining multiple include filters makes it possible to use both `AND` and `OR` filter logic.

```
slicer.data \
    ...
   .filter( slicer.dimensions.accounts.isin([1, 2, 3]) )
```

Excludes (Category and Unique dimensions)

Excludes filters are equivalent to Includes filters with negative logic. The same conditions apply using the *.notin(list)* method.

```
slicer.data \
    ...
   .filter( slicer.dimensions.accounts.notin([1, 2, 3]) )
```

Pattern (Category and Unique dimensions)

Pattern filters apply to `fireant.slicer.dimensions.CategoricalDimension` and `fireant.slicer.dimensions.UniqueDimension` dimensions using the *.like(*patterns)* method. They are the equivalent of a SQL `ILIKE` expression. The method accepts one or more pattern arguments which should be formatted for *SQL LIKE https://www.w3schools.com/sql/sql_like.asp*. With multiple arguments, results are returned that match **any** of the patterns.

Combining multiple pattern filters makes it possible to use both `AND` and `OR` filter logic.

```
slicer.data \
    ...
   .filter( slicer.dimensions.device.like('desk%', 'mob%') )
```

Anti-Pattern

Anti-Pattern filters are equivalent to Pattern filters with negative logic. The same conditions apply using the *.not_like(*patterns)* method.

```
slicer = Slicer(
    ...
    dimensions=[
        UniqueDimension('customer',
                        definition=customers.id,
                        display_definition=fn.Concat(customers.fname, ' ', customers.
↪lname))
    ],
    ...
```

(continues on next page)

(continued from previous page)

```
)

slicer.data \
    ...
   .filter( slicer.dimensions.device.not_like('desk%', 'mob%') )
```

Filtering on Display Definitions

When using a `fireant.slicer.dimensions.UniqueDimension` with the `display_defintion` attribute, it is also possible to filter based on display values instead of the definition.

The `display` attribute on an instance of `fireant.slicer.dimensions.UniqueDimension` returns a `fireant.slicer.dimensions.DisplayDimension` which works like a normal slicer dimension. It works with the following filters: '

1.4.4 Visualizing the data with Widgets

At least one widget must be added to every query before calling the *fetch()* builder chain method. Each Slicer query can return multiple widgets of different types, but because a slicer query resolves to a single SQL query, other parts of the query must be shared across all widgets, such as filters and dimensions.

Metrics are selected for each widget. Widgets can use the same metrics or different metrics in a query. The instance API for each widget is different since each widget uses metrics in different ways.

```
slicer.data \
    ...
   .widget( ... )
```

matplotlib

Coming soon!

pandas

The Pandas widget will return a Pandas data frame which is useful when displaying results in a Jupyter notebook. The Pandas widget is used by instantiating a `fireant.slicer.widgets.pandas.Pandas` class and passing one or more instances of `fireant.slicer.metrics.Metric` as arguments.

The data frame will be structured with an index level for each dimension.

The Pandas widget takes additional arguments.

pivot [list[dimension]] A list of dimensions which should be pivoted as columns. If all dimensions are pivoted, the result will be identical to setting the `transpose` argument to `True`.

transpose [bool] When `True`, the data frame will be transposed.

sort [list[int]] A list of column indices to sort by. This sorts the data frame after it's been pivoted and transposed. Which columns are present depends on the selected dimensions and metrics as well as the `pivot` and `transponse` arguments.

```
from fireant import Pandas

Pandas( *metrics )
```

```
slicer.data \
    ...
   .dimension( slicer.dimension.date, slicer.dimension.device )
   .widget( Pandas(slicer.metrics.clicks, slicer.metrics.cost, slicer.metrics.revenue,
                   pivot=(slicer.dimension.device, )
                   transpose=True) )
```

HighCharts

A HighCharts widget transforms the results into a HighCharts JSON config object. The widget is used by instantiating `fireant.slicer.widgets.highcharts.HighCharts` and calling the `axis` method with instances of `fireant.slicer.widgets.highcharts.Series` arguments. The axis method can be chained to create multiple axes.

Each `fireant.slicer.widgets.highcharts.Series` instance is constructed with one or more metrics.

```
from fireant import HighCharts

HighCharts( title ) \
    .axis ( HighCharts.LineChart( *metrics ), HighCharts.LineChart( *metrics ), ... )␣
↪\
    .axis ( HighCharts.BarChart( *metrics ) )
    ...
```

Datatables

React-Table

The React Table widget's instance API is identical to the Pandas widget, although it transforms results into a JSON config object meant for React-Table. See the section above on pandas for more information on the instance API.

```
from fireant import ReactTable

slicer.data \
    ...
   .dimension( slicer.dimension.date, slicer.dimension.device )
   .widget( ReactTable(slicer.metrics.clicks, slicer.metrics.cost, slicer.metrics.
↪revenue,
                       pivot=(slicer.dimension.device, )
                       transpose=True) )
```

1.4.5 Comparing Data to Previous Values using References

In some cases it is useful to compare the selected metrics over a period time such as in a Week-over-Week report. A *Reference* can be used to achieve this. *Reference* is a built-in function which can be chosen from the subclasses of `fireant.slicer.references.Reference`.

A *Reference* can be used as a fixed comparison, a change in value (delta), or a change in value as a percentage.

The *Reference* compares the currently selected data with itself shifted by the amount of the *Reference*.

The following options are available

- Day Over Day - Shifts by 1 day.
- Week Over Week - Shifts by 1 week.
- Month over Month - Shifts by 1 month.
- Quarter over Quarter - Shifts by 1 quarter or 3 months depending on whether the database backend supports quarter intervals.
- Year over Year - Shifts by 1 year.

For each *Reference*, there are the following variations:

- Delta - Difference in value
- Delta Percentage - Difference in value as a percentage of the previous value

A Date/Time dimension is required.

```python
from fireant.slicer.references import WeekOverWeek

# Use a Week-over-Week reference
slicer.data \
    ...
   .reference( WeekOverWeek(slicer.dimensions.date) )

# Compare Week-over-Week change (delta)
slicer.data \
    ...
   .reference( WeekOverWeek(slicer.dimensions.date, delta=True) )

# Compare Week-over-Week change as a percentage (delta percentage)
slicer.data \
    ...
   .reference( WeekOverWeek(slicer.dimensions.date, delta=True, percent=True) )
```

Note: For any reference, the comparison is made for the same days of the week.

1.4.6 Post-Processing Operations

Operations include extra computations applied in python to the result of the SQL query to modify the result.

More on this later!

1.5 Extending *fireant*

1.6 fireant

1.6.1 API Reference

Subpackages

fireant.database package

Submodules

fireant.database.base module

class `fireant.database.base.`**`Database`**(*host=None*, *port=None*, *database=None*, *max_processes=2*, *max_result_set_size=200000*, *cache_middleware=None*)

Bases: `object`

This is a abstract base class used for interfacing with a database platform.

`connect`()
This function must establish a connection to the database platform and return it.

`date_add`(*field: pypika.terms.Term*, *date_part: str*, *interval: int*)
This function must add/subtract a Date or Date/Time object.

`query_cls`
alias of `pypika.queries.Query`

`slow_query_log_min_seconds = 15`

`to_char`(*definition*)

`trunc_date`(*field*, *interval*)
This function must create a Pypika function which truncates a Date or DateTime object to a specific interval.

fireant.database.mysql module

class `fireant.database.mysql.`**`DateAdd`**(*field*, *interval_term*, *alias=None*)
Bases: `pypika.terms.Function`

Override for the MySQL specific DateAdd function which expects an interval instead of the date part and interval unit e.g. DATE_ADD("date", INTERVAL 1 YEAR)

class `fireant.database.mysql.`**`MySQLDatabase`**(*host='localhost'*, *port=3306*, *database=None*, *user=None*, *password=None*, *charset='utf8mb4'*, *max_processes=1*, *cache_middleware=None*)

Bases: *`fireant.database.base.Database`*

MySQL client that uses the PyMySQL module.

`connect`()
Returns a MySQL connection

Returns pymysql Connection class

`date_add`(*field*, *date_part*, *interval*)
This function must add/subtract a Date or Date/Time object.

`query_cls`
alias of `pypika.dialects.MySQLQuery`

`to_char`(*definition*)

trunc_date(*field*, *interval*)
This function must create a Pypika function which truncates a Date or DateTime object to a specific interval.

class fireant.database.mysql.**Trunc**(*field*, *date_format*, *alias=None*)
Bases: `pypika.terms.Function`

Wrapper for a custom MySQL TRUNC function (installed via a custom FireAnt MySQL script)

fireant.database.postgresql module

class fireant.database.postgresql.**DateTrunc**(*field*, *date_format*, *alias=None*)
Bases: `pypika.terms.Function`

Wrapper for the PostgreSQL date_trunc function

class fireant.database.postgresql.**PostgreSQLDatabase**(*host='localhost'*, *port=5432*, *database=None*, *user=None*, *password=None*, *max_processes=1*, *cache_middleware=None*)
Bases: *fireant.database.base.Database*

PostgreSQL client that uses the psycopg module.

connect()
This function must establish a connection to the database platform and return it.

date_add(*field*, *date_part*, *interval*)
This function must add/subtract a Date or Date/Time object.

query_cls
alias of `pypika.dialects.PostgreSQLQuery`

trunc_date(*field*, *interval*)
This function must create a Pypika function which truncates a Date or DateTime object to a specific interval.

fireant.database.redshift module

class fireant.database.redshift.**RedshiftDatabase**(*host='localhost'*, *port=5439*, *database=None*, *user=None*, *password=None*, *max_processes=1*, *cache_middleware=None*)
Bases: *fireant.database.postgresql.PostgreSQLDatabase*

Redshift client that uses the psycopg module.

query_cls
alias of `pypika.dialects.RedshiftQuery`

fireant.database.vertica module

class fireant.database.vertica.**Trunc**(*field*, *date_format*, *alias=None*)
Bases: `pypika.terms.Function`

Wrapper for Vertica TRUNC function for truncating dates.

class `fireant.database.vertica.`**`VerticaDatabase`**(*host='localhost'*, *port=5433*, *database='vertica'*, *user='vertica'*, *password=None*, *read_timeout=None*, *max_processes=1*, *cache_middleware=None*)

Bases: *`fireant.database.base.Database`*

Vertica client that uses the vertica_python driver.

`DATETIME_INTERVALS = {'day': 'DD', 'hour': 'HH', 'month': 'MM', 'quarter': 'Q', 'w`

`connect`()

This function must establish a connection to the database platform and return it.

`date_add`(*field*, *date_part*, *interval*)

This function must add/subtract a Date or Date/Time object.

`query_cls`

alias of `pypika.dialects.VerticaQuery`

`trunc_date`(*field*, *interval*)

This function must create a Pypika function which truncates a Date or DateTime object to a specific interval.

Module contents

fireant.slicer package

Subpackages

fireant.slicer.queries package

Submodules

fireant.slicer.queries.builder module

class `fireant.slicer.queries.builder.`**`DimensionChoicesQueryBuilder`**(*slicer*, *dimension*)

Bases: *`fireant.slicer.queries.builder.QueryBuilder`*

This builder is used for building slicer queries for fetching the choices for a dimension given a set of filters.

`fetch`(*hint=None*, *force_include=()*) → pandas.core.series.Series

Fetch the data for this query and transform it into the widgets.

Parameters

- **`hint`** – For database vendors that support it, add a query hint to collect analytics on the queries triggerd by fireant.
- **`force_include`** – A list of dimension values to include in the result set. This can be used to avoid having necessary results cut off due to the pagination. These results will be returned at the head of the results.

Returns A list of dict (JSON) objects containing the widget configurations.

queries
Serializes this query builder as a set of SQL queries. This method will always return a list of one query since only one query is required to retrieve dimension choices.

This function only handles dimensions (select+group by) and filtering (where/having), which is everything needed for the query to fetch choices for dimensions.

The slicer query extends this with metrics, references, and totals.

class `fireant.slicer.queries.builder.`**`DimensionLatestQueryBuilder`**(*slicer*)
Bases: *`fireant.slicer.queries.builder.QueryBuilder`*

`fetch`(*hint=None*)
Fetches the data for this query instance and returns it in an instance of *pd.DataFrame*

Parameters **`hint`** – For database vendors that support it, add a query hint to collect analytics on the queries triggerd by fireant.

queries
Serializes this query builder as a set of SQL queries. This method will always return a list of one query since only one query is required to retrieve dimension choices.

This function only handles dimensions (select+group by) and filtering (where/having), which is everything needed for the query to fetch choices for dimensions.

The slicer query extends this with metrics, references, and totals.

class `fireant.slicer.queries.builder.`**`QueryBuilder`**(*slicer*, *table*)
Bases: `object`

This is the base class for building slicer queries. This class provides an interface for building slicer queries via a set of functions which can be chained together.

`fetch`(*hint=None*)
Fetches the data for this query instance and returns it in an instance of *pd.DataFrame*

Parameters **`hint`** – For database vendors that support it, add a query hint to collect analytics on the queries triggerd by fireant.

`filter`(**args*, *mutate=False*, ***kwargs*)

Parameters **`mutate`** – When True, overrides the immutable behavior of this decorator.

`limit`(**args*, *mutate=False*, ***kwargs*)

Parameters **`mutate`** – When True, overrides the immutable behavior of this decorator.

`offset`(**args*, *mutate=False*, ***kwargs*)

Parameters **`mutate`** – When True, overrides the immutable behavior of this decorator.

queries
Serialize this query builder object to a set of Pypika/SQL queries.

This is the base implementation shared by two implementations: the query to fetch data for a slicer request and the query to fetch choices for dimensions.

This function only handles dimensions (select+group by) and filtering (where/having), which is everything needed for the query to fetch choices for dimensions.

The slicer query extends this with metrics, references, and totals.

class `fireant.slicer.queries.builder.`**`SlicerQueryBuilder`**(*slicer*)
Bases: *`fireant.slicer.queries.builder.QueryBuilder`*

Slicer queries consist of widgets, dimensions, filters, and references. At least one or more widgets is required. All others are optional.

`dimension` (**args*, *mutate=False*, ***kwargs*)

Parameters **`mutate`** – When True, overrides the immutable behavior of this decorator.

`fetch` (*hint=None*) → Iterable[Dict]

Fetch the data for this query and transform it into the widgets.

Parameters **`hint`** – A query hint label used with database vendors which support it. Adds a label comment to the query.

Returns A list of dict (JSON) objects containing the widget configurations.

`orderby` (**args*, *mutate=False*, ***kwargs*)

Parameters **`mutate`** – When True, overrides the immutable behavior of this decorator.

`queries`

Serialize this query builder to a list of Pypika/SQL queries. This function will return one query for every combination of reference and rolled up dimension (including null options).

This collects all of the metrics in each widget, dimensions, and filters and builds a corresponding pypika query to fetch the data. When references are used, the base query normally produced is wrapped in an outer query and a query for each reference is joined based on the referenced dimension shifted.

`reference` (**args*, *mutate=False*, ***kwargs*)

Parameters **`mutate`** – When True, overrides the immutable behavior of this decorator.

`reference_groups`

`widget` (**args*, *mutate=False*, ***kwargs*)

Parameters **`mutate`** – When True, overrides the immutable behavior of this decorator.

`fireant.slicer.queries.builder.`**`add_hints`** (*queries*, *hint=None*)

fireant.slicer.queries.database module

fireant.slicer.queries.finders module

`class` `fireant.slicer.queries.finders.`**`ReferenceGroup`** (*dimension*, *time_unit*, *intervals*)

Bases: `tuple`

`dimension`

Alias for field number 0

`intervals`

Alias for field number 2

`time_unit`

Alias for field number 1

`fireant.slicer.queries.finders.`**`find_and_group_references_for_dimensions`** (*references*)

Finds all of the references for dimensions and groups them by dimension, interval unit, number of intervals.

This structure reflects how the references need to be joined to the slicer query. References of the same type (WoW, WoW.delta, WoW.delta_percent) can share a join query.

Parameters **`references`** –

Returns

An *OrderedDict* where the keys are 3-item tuples consisting of "Dimension, interval unit, # of intervals.

```
Example
{
    (Dimension(date_1), 'weeks', 1): [WoW, WoW.delta],
    (Dimension(date_1), 'years', 1): [YoY],
    (Dimension(date_7), 'days', 1): [DoD, DoD.delta_percent],
}
```

`fireant.slicer.queries.finders.`**`find_and_replace_reference_dimensions`**`(`*references*, *dimensions*`)`

Finds the dimension for a reference in the query if there is one and replaces it. This is to force the reference to use the same modifiers with a dimension if it is selected in the query.

Parameters

- **`references`** –
- **`dimensions`** –

Returns

`fireant.slicer.queries.finders.`**`find_joins_for_tables`**`(`*joins*, *base_table*, *required_tables*`)`

Given a set of tables required for a slicer query, this function finds the joins required for the query and sorts them topologically.

Returns A list of joins in the order that they must be joined to the query.

Raises MissingTableJoinException - If a table is required but there is no join for that table CircularJoinsException - If there is a circular dependency between two or more joins

`fireant.slicer.queries.finders.`**`find_metrics_for_widgets`**`(`*widgets*`)`

Returns an ordered, distinct list of metrics used in all widgets as part of this query.

`fireant.slicer.queries.finders.`**`find_operations_for_widgets`**`(`*widgets*`)`

Returns an ordered, distinct list of metrics used in all widgets as part of this query.

`fireant.slicer.queries.finders.`**`find_required_tables_to_join`**`(`*elements*, *base_table*`)`

Collect all the tables required for a given list of slicer elements. This looks through the definition and display_definition attributes of all elements and

This looks through the metrics, dimensions, and filter included in this slicer query. It also checks both the definition field of each element as well as the display definition for Unique Dimensions.

Returns A collection of tables required to execute a query,

`fireant.slicer.queries.finders.`**`find_share_dimensions`**`(`*dimensions*, *operations*`)`

Returns a subset list of dimensions from the list of dimensions that are used as the over-dimension in share operations.

Parameters

- **`dimensions`** –
- **`operations`** –

Returns

`fireant.slicer.queries.finders.`**`find_totals_dimensions`**(*dimensions*, *share_dimensions*)

WRITEME

Parameters

- **`dimensions`** –
- **`share_dimensions`** –

Returns

fireant.slicer.queries.logger module

fireant.slicer.queries.makers module

fireant.slicer.queries.references module

fireant.slicer.queries.special_cases module

`fireant.slicer.queries.special_cases.`**`adjust_dataframe_for_rolling_window`**(*operations*, *data_frame*)

This function adjusts the resulting data frame after executing a slicer query with a rolling operation. If there is a date dimension in the first level of the data frame's index and a rolling operation is applied, it will slice the dates following the max window to remove it. This way, the adjustment of date filters applied in #adjust_daterange_filter_for_rolling_window are removed from the data frame but also in case there are no filters, the first few date data points will be removed where the rolling window cannot be calculated.

Parameters

- **`operations`** –
- **`data_frame`** –

Returns

`fireant.slicer.queries.special_cases.`**`adjust_daterange_filter_for_rolling_window`**(*dimensions*, *operations*, *filters*)

This function adjusts date filters for a rolling operation in order to select enough date to compute the values for within the original range.

It only applies when using a date dimension in the first position and a RangeFilter is used on that dimension. It is meant to be applied to a slicer query.

Parameters

- **`dimensions`** – The dimensions applied to a slicer query
- **`operations`** – The dimensions used in widgets in a slicer query
- **`filters`** – The filters applied to a slicer query

Returns

`fireant.slicer.queries.special_cases.`**`apply_operations_to_data_frame`**(*operations*, *data_frame*) fireant.slicer.queries.special_cases.**apply_special_cases**(*f*)

fireant.slicer.queries.special_cases.**apply_to_query_args**(*database*, *table*, *joins*, *dimensions*, *metrics*, *operations*, *filters*, *references*, *orders*)

Module contents

fireant.slicer.widgets package

Submodules

fireant.slicer.widgets.base module

class fireant.slicer.widgets.base.**TransformableWidget**(**items*)

Bases: *fireant.slicer.widgets.base.Widget*

group_pagination = False

transform(*data_frame*, *slicer*, *dimensions*, *references*)

- Main entry point -

Transformers the result set *pd.DataFrame* from a slicer query into the output format for this specific widget type.

Parameters

- **data_frame** – The data frame containing the data. Index must match the dimensions parameter.
- **slicer** – The slicer that is in use.
- **dimensions** – A list of dimensions that are being rendered.
- **references** – A list of references that are being rendered.

Returns A dict meant to be dumped as JSON.

class fireant.slicer.widgets.base.**Widget**(**items*)

Bases: object

item(**args*, *mutate=False*, ***kwargs*)

Parameters **mutate** – When True, overrides the immutable behavior of this decorator.

metrics

operations

fireant.slicer.widgets.csv module

class fireant.slicer.widgets.csv.**CSV**(*metric: fireant.slicer.metrics.Metric*, **metrics*, *group_pagination=False*, ***kwargs*)

Bases: *fireant.slicer.widgets.pandas.Pandas*

transform(*data_frame*, *slicer*, *dimensions*, *references*)

WRITEME

Parameters

- **data_frame** –
- **slicer** –
- **dimensions** –
- **references** –

Returns

fireant.slicer.widgets.datatables module

class fireant.slicer.widgets.datatables.**DataTablesJS**(*metric*, **metrics*, *pivot=False*, *max_columns=None*)

Bases: *fireant.slicer.widgets.base.TransformableWidget*

transform(*data_frame*, *slicer*, *dimensions*, *references*)

WRITEME

Parameters

- **data_frame** –
- **slicer** –
- **dimensions** –

Returns

fireant.slicer.widgets.helpers module

fireant.slicer.widgets.helpers.**dimensional_metric_label**(*dimensions*, *dimension_display_values*)

Creates a callback function for rendering series labels.

Parameters

- **dimensions** – A list of fireant.Dimension which is being rendered.
- **dimension_display_values** – A dictionary containing key-value pairs for each dimension.

Returns a callback function which renders a label for a metric, reference, and list of dimension values.

fireant.slicer.widgets.helpers.**extract_display_values**(*dimensions*, *data_frame*)

Retrieves the display values for each dimension.

For UniqueDimension, this will retrieve the display values from the data frame containing the data from the slicer query. For CategoricalDimension, the values are retrieved from the set of display values configured in the slicer.

Parameters

- **dimensions** – A list of dimensions present in a slicer query.
- **data_frame** – The data frame containing the data result of the slicer query.

Returns A dict containing keys for dimensions with display values (If there are no display values then the dimension's key will not be present). The value of the dict will be either a dict or a data frame where the display value can be accessed using the display value as the key.

fireant.slicer.widgets.highcharts module

class `fireant.slicer.widgets.highcharts.`**`HighCharts`**(*title=None*, *colors=None*, *x_axis_visible=True*, *tooltip_visible=True*)

Bases: `fireant.slicer.widgets.chart_base.ChartWidget`, *`fireant.slicer.widgets.base.TransformableWidget`*

`group_pagination = True`

`transform`(*data_frame*, *slicer*, *dimensions*, *references*)

- Main entry point -

Transforms a data frame into HighCharts JSON format.

See https://api.highcharts.com/highcharts/

Parameters

- **`data_frame`** – The data frame containing the data. Index must match the dimensions parameter.
- **`slicer`** – The slicer that is in use.
- **`dimensions`** – A list of dimensions that are being rendered.
- **`references`** – A list of references that are being rendered.

Returns A dict meant to be dumped as JSON.

fireant.slicer.widgets.matplotlib module

class `fireant.slicer.widgets.matplotlib.`**`Matplotlib`**(*title=None*)

Bases: `fireant.slicer.widgets.chart_base.ChartWidget`, *`fireant.slicer.widgets.base.TransformableWidget`*

static `get_plot_func_for_series_type`(*pd_series*, *label*, *chart_series*)

`transform`(*data_frame*, *slicer*, *dimensions*, *references*)

- Main entry point -

Transformers the result set *pd.DataFrame* from a slicer query into the output format for this specific widget type.

Parameters

- **`data_frame`** – The data frame containing the data. Index must match the dimensions parameter.
- **`slicer`** – The slicer that is in use.
- **`dimensions`** – A list of dimensions that are being rendered.
- **`references`** – A list of references that are being rendered.

Returns A dict meant to be dumped as JSON.

fireant.slicer.widgets.pandas module

class fireant.slicer.widgets.pandas.**Pandas**(*metric: fireant.slicer.metrics.Metric*, **metrics*, *pivot=()*, *transpose=False*, *sort=None*, *ascending=None*, *max_columns=None*)

Bases: `fireant.slicer.widgets.base.TransformableWidget`

pivot_data_frame(*data_frame*, *pivot=()*, *transpose=False*)

Pivot and transpose the data frame. Dimensions including in the *pivot* arg will be unshifted to columns. If *transpose* is True the data frame will be transposed. If there is only index level in the data frame (ie. one dimension), and that dimension is pivoted, then the data frame will just be transposed. If there is a single metric in the data frame and at least one dimension pivoted, the metrics column level will be dropped for simplicity.

Parameters

- **data_frame** – The result set data frame
- **pivot** – A list of index keys for *data_frame* of levels to shift
- **transpose** – A boolean true or false whether to transpose the data frame.

Returns The shifted/transposed data frame

sort_data_frame(*data_frame*)

transform(*data_frame*, *slicer*, *dimensions*, *references*)

WRITEME

Parameters

- **data_frame** –
- **slicer** –
- **dimensions** –
- **references** –

Returns

fireant.slicer.widgets.reacttable module

class fireant.slicer.widgets.reacttable.**ReactTable**(*metric*, **metrics*, *pivot=()*, *transpose=False*, *sort=None*, *ascending=None*, *max_columns=None*)

Bases: `fireant.slicer.widgets.pandas.Pandas`

This component does not work with react-table out of the box, some customization is needed in order to work with the transformed data.

```
// A Custom TdComponent implementation is required by Fireant in order to render␣
↪display values
const TdComponent = ({
                        toggleSort,
                        className,
                        children,
                        ...rest
                      }) =>
    <div className={classNames('rt-td', className)} role="gridcell" {...rest}>
        {_.get(children, 'display', children.raw) || <span>&nbsp;</span>}
```

(continues on next page)

(continued from previous page)

```
    </div>;

const FireantReactTable = ({
                        config, // The payload from fireant
                      }) =>
    <ReactTable columns={config.columns}
                data={config.data}
                minRows={0}

                TdComponent={ DashmoreTdComponent}
                defaultSortMethod={(a, b, desc) => ReactTableDefaults.
↪defaultSortMethod(a.raw, b.raw, desc)}>
    </ReactTable>;
```

static format_data_frame(*data_frame*, *dimensions*)

This function prepares the raw data frame for transformation by formatting dates in the index and removing any remaining NaN/NaT values. It also names the column as metrics so that it can be treated like a dimension level.

Parameters

- **data_frame** – The result set data frame
- **dimensions** –

Returns

static map_display_values(*df*, *dimensions*)

Creates a mapping for dimension values to their display values.

Parameters

- **df** – The result data set that is being transformed.
- **dimensions** – The list of dimensions included in the query that created the result data set df.

Returns A tree-structure dict with two levels of depth. The top level dict has keys for each dimension's display key. The lower level dict has keys for each raw dimension value and values which are the display value.

static map_hyperlink_templates(*df*, *dimensions*)

Creates a mapping for each dimension to it's hyperlink template if it is possible to create the hyperlink template for it.

The hyperlink template is a URL-like string containing curley braces enclosing dimension keys: *{dimension}*. While rendering this widget, the dimension key placeholders need to be replaced with the dimension values for that row.

Parameters

- **df** – The result data set that is being transformed. The data frame SHOULD be pivoted/transposed if that step is required, before calling this function, in order to prevent the template from being included for the dimension if one of the required dimensions is pivoted.
- **dimensions** – The list of dimensions included in the query that created the result data set df.

Returns A dict with the dimension key as the key and the hyperlink template as the value. Templates will only be included if it will be possible to fill in the required parameters.

transform(*data_frame*, *slicer*, *dimensions*, *references*)
Transforms a data frame into a format for ReactTable. This is an object containing attributes *columns* and *data* which align with the props in ReactTable with the same name.

Parameters

- **data_frame** – The result set data frame
- **slicer** – The slicer that generated the data query
- **dimensions** – A list of dimensions that were selected in the data query
- **references** – A list of references that were selected in the data query

Returns An dict containing attributes *columns* and *data* which align with the props in ReactTable with the same names.

classmethod transform_data(*data_frame*, *item_map*, *dimension_display_values*, *dimension_hyperlink_templates*)
Builds a list of dicts containing the data for ReactTable. This aligns with the accessors set by #transform_dimension_column_headers and #transform_metric_column_headers

Parameters

- **data_frame** – The result set data frame
- **item_map** – A map to find metrics/operations based on their keys found in the data frame.
- **dimension_display_values** – A map for finding display values for dimensions based on their key and value.
- **dimension_hyperlink_templates** –

classmethod transform_data_row_index(*index_values*, *dimension_display_values*, *dimension_hyperlink_templates*)

classmethod transform_data_row_values(*series*, *item_map*)

static transform_dimension_column_headers(*data_frame*, *dimensions*)
Convert the un-pivoted dimensions into ReactTable column header definitions.

Parameters

- **data_frame** – The result set data frame
- **dimensions** – A list of dimensions in the data frame that are part of the index

Returns A list of column header definitions with the following structure.

```
columns = [{
  Header: 'Column A',
  accessor: 'a',
}, {
  Header: 'Column B',
  accessor: 'b',
}]
```

static transform_metric_column_headers(*data_frame*, *item_map*, *dimension_display_values*)
Convert the metrics into ReactTable column header definitions. This includes any pivoted dimensions, which will result in multiple rows of headers.

Parameters

- **data_frame** – The result set data frame

- **item_map** – A map to find metrics/operations based on their keys found in the data frame.
- **dimension_display_values** – A map for finding display values for dimensions based on their key and value.

Returns A list of column header definitions with the following structure.

```
columns = [{
  Header: 'Column A',
  columns: [{
    Header: 'SubColumn A.0',
    accessor: 'a.0',
  }, {
    Header: 'SubColumn A.1',
    accessor: 'a.1',
  }]
}, {
  Header: 'Column B',
  columns: [
    ...
  ]
}]
```

class `fireant.slicer.widgets.reacttable.`**ReferenceItem**(*item*, *reference*)
Bases: `object`

class `fireant.slicer.widgets.reacttable.`**TotalsItem**
Bases: `object`

key = 'totals'

label = 'Totals'

precision = None

prefix = None

suffix = None

`fireant.slicer.widgets.reacttable.`**map_index_level**(*index*, *level*, *func*)

Module contents

Submodules

fireant.slicer.base module

class `fireant.slicer.base.`**SlicerElement**(*key*, *label=None*, *definition=None*, *display_definition=None*)
Bases: `object`

The *SlicerElement* class represents an element of the slicer, either a metric or dimension, which contains information about such as how to query it from the database.

has_display_field

fireant.slicer.dimensions module

class `fireant.slicer.dimensions.`**`BooleanDimension`**(*key*, *label=None*, *definition=None*, *hyperlink_template=None*)

Bases: `fireant.slicer.dimensions.Dimension`

This is a dimension that represents a boolean true/false value. The expression should always result in a boolean value.

`is_`(*value: bool*)

Creates a filter to filter a slicer query.

Parameters **`value`** – True or False

Returns A slicer query filter used to filter a slicer query to results where this dimension is True or False.

class `fireant.slicer.dimensions.`**`CategoricalDimension`**(*key*, *label=None*, *definition=None*, *hyperlink_template=None*, *display_values=()*)

Bases: `fireant.slicer.dimensions.PatternFilterableMixin`, `fireant.slicer.dimensions.Dimension`

This is a dimension that represents an enum-like database field, with a finite list of options to chose from. It provides support for configuring a display value for each of the possible values.

`isin`(*values: Iterable*)

Creates a filter to filter a slicer query.

Parameters **`values`** – An iterable of value to constrain the slicer query results by.

Returns A slicer query filter used to filter a slicer query to results where this dimension is one of a set of values. Opposite of #notin.

`notin`(*values*)

Creates a filter to filter a slicer query.

Parameters **`values`** – An iterable of value to constrain the slicer query results by.

Returns A slicer query filter used to filter a slicer query to results where this dimension is *not* one of a set of values. Opposite of #isin.

class `fireant.slicer.dimensions.`**`ContinuousDimension`**(*key*, *label=None*, *definition=None*, *hyperlink_template=None*, *default_interval=NumericInterval(size=1, offset=0)*)

Bases: `fireant.slicer.dimensions.Dimension`

This is a dimension that represents a field in the database which is a continuous value, such as a decimal, integer, or date/time. It requires the use of an interval which is the window over the values.

class `fireant.slicer.dimensions.`**`DatetimeDimension`**(*key*, *label=None*, *definition=None*, *hyperlink_template=None*, *default_interval=DatetimeInterval('day')*)

Bases: `fireant.slicer.dimensions.ContinuousDimension`

A subclass of ContinuousDimension which reflects a date/time data type. Intervals are replaced with time intervals such as daily, weekly, annually, etc. A reference can be used to show a comparison over time such as week-over-week or month-over-month.

between (*start*, *stop*)
Creates a filter to filter a slicer query.

Parameters

- **start** – The start time of the filter. This is the beginning of the window for which results should be included.
- **stop** – The stop time of the filter. This is the end of the window for which results should be included.

Returns A slicer query filter used to filter a slicer query to results where this dimension is between the values start and stop.

class fireant.slicer.dimensions.**Dimension** (*key*, *label=None*, *definition=None*, *display_definition=None*, *hyperlink_template=None*)
Bases: *fireant.slicer.base.SlicerElement*

The *Dimension* class represents a dimension in the *Slicer* object.

Parameters

- **alias** – A unique identifier used to identify the metric when writing slicer queries. This value must be unique over the metrics in the slicer.
- **definition** – A pypika expression which is used to select the value when building SQL queries.
- **display_definition** – A pypika expression which is used to select the display value for this dimension.
- **hyperlink_template** – A hyperlink template for constructing a URL that can link a value for a dimension to a web page. This is used by some transformers such as the ReactTable transformer for displaying hyperlinks.

rollup (**args*, *mutate=False*, ***kwargs*)

Parameters **mutate** – When True, overrides the immutable behavior of this decorator.

class fireant.slicer.dimensions.**DisplayDimension** (*dimension*)
Bases: fireant.slicer.dimensions._UniqueDimensionBase

WRITEME

class fireant.slicer.dimensions.**PatternFilterableMixin**
Bases: object

definition = None

key = None

like (*pattern*, **patterns*)
Creates a filter to filter a slicer query.

Parameters

- **pattern** – A pattern to match against the dimension's display definition. This pattern is used in the SQL query as the *LIKE* expression.
- **patterns** – Additional patterns. This is the same as the pattern argument. The function signature is intended to syntactically require at least one pattern.

Returns A slicer query filter used to filter a slicer query to results where this dimension's display definition matches the pattern.

`not_like`(*pattern*, **patterns*)
Creates a filter to filter a slicer query.

Parameters

- **`pattern`** – A pattern to match against the dimension's display definition. This pattern is used in the SQL query as the *NOT LIKE* expression.
- **`patterns`** – Additional patterns. This is the same as the pattern argument. The function signature is intended to syntactically require at least one pattern.

Returns A slicer query filter used to filter a slicer query to results where this dimension's display definition matches the pattern.

`pattern_definition_attribute = 'definition'`

`class` `fireant.slicer.dimensions.`**`TotalsDimension`**(*dimension*)
Bases: *`fireant.slicer.dimensions.Dimension`*

`class` `fireant.slicer.dimensions.`**`UniqueDimension`**(*key*, *label=None*, *definition=None*, *display_definition=None*, *hyperlink_template=None*)
Bases: `fireant.slicer.dimensions._UniqueDimensionBase`

This is a dimension that represents a field in a database which is a unique identifier, such as a primary/foreign key. It provides support for a display value field which is selected and used in the results.

`has_display_field`

`like`(*pattern*, **patterns*)
Creates a filter to filter a slicer query.

Parameters

- **`pattern`** – A pattern to match against the dimension's display definition. This pattern is used in the SQL query as the *LIKE* expression.
- **`patterns`** – Additional patterns. This is the same as the pattern argument. The function signature is intended to syntactically require at least one pattern.

Returns A slicer query filter used to filter a slicer query to results where this dimension's display definition matches the pattern.

`not_like`(*pattern*, **patterns*)
Creates a filter to filter a slicer query.

Parameters

- **`pattern`** – A pattern to match against the dimension's display definition. This pattern is used in the SQL query as the *NOT LIKE* expression.
- **`patterns`** – Additional patterns. This is the same as the pattern argument. The function signature is intended to syntactically require at least one pattern.

Returns A slicer query filter used to filter a slicer query to results where this dimension's display definition matches the pattern.

fireant.slicer.exceptions module

`exception` `fireant.slicer.exceptions.`**`CircularJoinsException`**
Bases: *`fireant.slicer.exceptions.SlicerException`*

exception `fireant.slicer.exceptions.`**`ContinuousDimensionRequiredException`**
    Bases: *`fireant.slicer.exceptions.SlicerException`*

exception `fireant.slicer.exceptions.`**`MetricRequiredException`**
    Bases: *`fireant.slicer.exceptions.SlicerException`*

exception `fireant.slicer.exceptions.`**`MissingTableJoinException`**
    Bases: *`fireant.slicer.exceptions.SlicerException`*

exception `fireant.slicer.exceptions.`**`MissingTotalsForShareException`**
    Bases: `Exception`

exception `fireant.slicer.exceptions.`**`QueryException`**
    Bases: *`fireant.slicer.exceptions.SlicerException`*

exception `fireant.slicer.exceptions.`**`RollupException`**
    Bases: *`fireant.slicer.exceptions.SlicerException`*

exception `fireant.slicer.exceptions.`**`SlicerException`**
    Bases: `Exception`

fireant.slicer.filters module

class `fireant.slicer.filters.`**`AntiPatternFilter`**(*dimension_key*, *dimension_definition*, *pattern*, **patterns*)
    Bases: *`fireant.slicer.filters.PatternFilter`*

class `fireant.slicer.filters.`**`BooleanFilter`**(*dimension_key*, *dimension_definition*, *value*)
    Bases: *`fireant.slicer.filters.DimensionFilter`*

class `fireant.slicer.filters.`**`ComparatorFilter`**(*metric_key*, *metric_definition*, *operator*, *value*)
    Bases: *`fireant.slicer.filters.MetricFilter`*

    class Operator
        Bases: `object`

        eq = 'eq'

        gt = 'gt'

        gte = 'gte'

        lt = 'lt'

        lte = 'lte'

        ne = 'ne'

class `fireant.slicer.filters.`**`ContainsFilter`**(*dimension_key*, *dimension_definition*, *values*)
    Bases: *`fireant.slicer.filters.DimensionFilter`*

class `fireant.slicer.filters.`**`DimensionFilter`**(*dimension_key*, *definition*)
    Bases: *`fireant.slicer.filters.Filter`*

class `fireant.slicer.filters.`**`ExcludesFilter`**(*dimension_key*, *dimension_definition*, *values*)
    Bases: *`fireant.slicer.filters.DimensionFilter`*

class `fireant.slicer.filters.`**`Filter`**(*definition*)
    Bases: `object`

class fireant.slicer.filters.**MetricFilter**(*metric_key*, *definition*)

Bases: *fireant.slicer.filters.Filter*

class fireant.slicer.filters.**PatternFilter**(*dimension_key*, *dimension_definition*, *pattern*, **patterns*)

Bases: *fireant.slicer.filters.DimensionFilter*

class fireant.slicer.filters.**RangeFilter**(*dimension_key*, *dimension_definition*, *start*, *stop*)

Bases: *fireant.slicer.filters.DimensionFilter*

fireant.slicer.intervals module

class fireant.slicer.intervals.**DatetimeInterval**(*key*)

Bases: object

class fireant.slicer.intervals.**NumericInterval**(*size=1*, *offset=0*)

Bases: object

fireant.slicer.joins module

class fireant.slicer.joins.**Join**(*table*, *criterion*, *join_type=<JoinType.inner: ''>*)

Bases: object

WRITEME

fireant.slicer.metrics module

class fireant.slicer.metrics.**Metric**(*key*, *definition*, *label=None*, *precision=None*, *prefix=None*, *suffix=None*)

Bases: *fireant.slicer.base.SlicerElement*

The *Metric* class represents a metric in the *Slicer* object.

Parameters

- **alias** – A unique identifier used to identify the metric when writing slicer queries. This value must be unique over the metrics in the slicer.
- **definition** – A pypika expression which is used to select the value when building SQL queries. For metrics, this query **must** be aggregated, since queries always use a `GROUP BY` clause an metrics are not used as a group.
- **label** – (optional) A display value used for the metric. This is used for rendering the labels within the visualizations. If not set, the alias will be used as the default.
- **precision** – (optional) A precision value for rounding decimals. By default, no rounding will be applied.
- **prefix** – (optional) A prefix for rendering labels in visualizations such as '$'
- **suffix** – A suffix for rendering labels in visualizations such as '€'

share

Parameters mutate – When True, overrides the immutable behavior of this decorator.

fireant.slicer.operations module

class fireant.slicer.operations.**CumMean**(*arg*)
Bases: fireant.slicer.operations._Cumulative

apply(*data_frame*, *reference*)

static cummean(*x*)

class fireant.slicer.operations.**CumProd**(*arg*)
Bases: fireant.slicer.operations._Cumulative

apply(*data_frame*, *reference*)

class fireant.slicer.operations.**CumSum**(*arg*)
Bases: fireant.slicer.operations._Cumulative

apply(*data_frame*, *reference*)

class fireant.slicer.operations.**Operation**
Bases: object

The *Operation* class represents an operation in the *Slicer* API.

apply(*data_frame*, *reference*)

metrics

operations

class fireant.slicer.operations.**RollingMean**(*arg*, *window*, *min_periods=None*)
Bases: *fireant.slicer.operations.RollingOperation*

apply(*data_frame*, *reference*)

rolling_mean(*x*)

class fireant.slicer.operations.**RollingOperation**(*arg*, *window*, *min_periods=None*)
Bases: fireant.slicer.operations._BaseOperation

apply(*data_frame*, *reference*)

metrics

operations

class fireant.slicer.operations.**Share**(*metric: fireant.slicer.metrics.Metric*, *over: fireant.slicer.dimensions.Dimension = None*, *precision=2*)
Bases: fireant.slicer.operations._BaseOperation

apply(*data_frame*, *reference*)

metrics

operations

fireant.slicer.references module

class fireant.slicer.references.**Reference**(*dimension*, *reference_type*, *delta=False*, *delta_percent=False*)
Bases: object

class fireant.slicer.references.**ReferenceType**(*key*, *label*, *time_unit: str*, *interval: int*)
Bases: object

`fireant.slicer.references.`**`reference_key`**`(`*metric*, *reference*`)`

Format a metric key for a reference.

Returns A string that is used as the key for a reference metric.

`fireant.slicer.references.`**`reference_label`**`(`*metric*, *reference*`)`

Format a metric label for a reference.

Returns A string that is used as the display value for a reference metric.

`fireant.slicer.references.`**`reference_prefix`**`(`*metric*, *reference*`)`

Return the prefix for a metric displayed for a reference (or no Reference)

Returns A string that is used as the prefix for a reference metric.

`fireant.slicer.references.`**`reference_suffix`**`(`*metric*, *reference*`)`

Return the suffix for a metric displayed for a reference (or no Reference)

Returns A string that is used as the suffix for a reference metric.

`fireant.slicer.references.`**`reference_term`**`(`*reference: fireant.slicer.references.Reference*, *original_query: pypika.queries.QueryBuilder*, *ref_query: pypika.queries.QueryBuilder*`)`

Part of query building. Given a reference, the original slicer query, and the ref query, creates the pypika for the reference that should be selected in the reference container query.

Parameters

- **`reference`** –
- **`original_query`** –
- **`ref_query`** –

Returns

fireant.slicer.slicers module

`class` `fireant.slicer.slicers.`**`Slicer`**`(`*table*, *database*, *joins=()*, *dimensions=()*, *metrics=()*, *hint_table=None*, *always_query_all_metrics=False*`)`

Bases: `object`

WRITEME

`class Dimensions``(`*items*`)`

Bases: `fireant.slicer.slicers._Container`

`class Fields``(`*items*`)`

Bases: `fireant.slicer.slicers._Container`

`class Metrics``(`*items*`)`

Bases: `fireant.slicer.slicers._Container`

Module contents

fireant.tests package

Subpackages

fireant.tests.database package

Submodules

fireant.tests.database.mock_database module

class `fireant.tests.database.mock_database.`**`TestDatabase`**(*host='localhost', port=5433, database='vertica', user='vertica', password=None, read_timeout=None, max_processes=1, cache_middleware=None*)

Bases: `fireant.database.vertica.VerticaDatabase`

`connect`()

This function must establish a connection to the database platform and return it.

fireant.tests.database.test_databases module

fireant.tests.database.test_mysql module

class `fireant.tests.database.test_mysql.`**`TestMySQLDatabase`**(*methodName='runTest'*)

Bases: `unittest.case.TestCase`

classmethod setUpClass()

Hook method for setting up class fixture before running tests in the class.

`test_connect`(*mock_connection_class*)

`test_date_add_day`()

`test_date_add_hour`()

`test_date_add_month`()

`test_date_add_quarter`()

`test_date_add_week`()

`test_date_add_year`()

`test_defaults`()

`test_to_char`()

`test_trunc_day`()

`test_trunc_hour`()

`test_trunc_month`()

`test_trunc_quarter`()

`test_trunc_week`()

`test_trunc_year`()

fireant.tests.database.test_postgresql module

class fireant.tests.database.test_postgresql.**TestPostgreSQL**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

classmethod setUpClass()
Hook method for setting up class fixture before running tests in the class.

test_connect()

test_date_add_day()

test_date_add_hour()

test_date_add_month()

test_date_add_quarter()

test_date_add_week()

test_date_add_year()

test_defaults()

test_trunc_day()

test_trunc_hour()

test_trunc_quarter()

test_trunc_week()

test_trunc_year()

fireant.tests.database.test_redshift module

class fireant.tests.database.test_redshift.**TestRedshift**(*methodName='runTest'*)
Bases: *fireant.tests.database.test_postgresql.TestPostgreSQL*

Inherits from TestPostgreSQL as Redshift is almost identical to PostgreSQL so the tests are similar

classmethod setUpClass()
Hook method for setting up class fixture before running tests in the class.

test_connect()

test_defaults()

fireant.tests.database.test_vertica module

class fireant.tests.database.test_vertica.**TestVertica**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

test_connect()

test_date_add_day()

test_date_add_hour()

test_date_add_month()

test_date_add_quarter()

test_date_add_week()

test_date_add_year()

test_defaults()

test_trunc_day()

test_trunc_hour()

test_trunc_quarter()

test_trunc_week()

test_trunc_year()

Module contents

fireant.tests.slicer package

Subpackages

fireant.tests.slicer.queries package

Submodules

fireant.tests.slicer.queries.test_build_dimension_filters module

fireant.tests.slicer.queries.test_build_dimensions module

fireant.tests.slicer.queries.test_build_joins module

fireant.tests.slicer.queries.test_build_metric_filters module

fireant.tests.slicer.queries.test_build_metrics module

fireant.tests.slicer.queries.test_build_operations module

fireant.tests.slicer.queries.test_build_orderbys module

fireant.tests.slicer.queries.test_build_pagination module

fireant.tests.slicer.queries.test_build_references module

fireant.tests.slicer.queries.test_build_render module

fireant.tests.slicer.queries.test_builder module

fireant.tests.slicer.queries.test_database module

fireant.tests.slicer.queries.test_dimension_choices module

Module contents

fireant.tests.slicer.widgets package

Submodules

fireant.tests.slicer.widgets.test_csv module

class fireant.tests.slicer.widgets.test_csv.**CSVWidgetTests**(*methodName='runTest'*)
Bases: unittest.case.TestCase

maxDiff = None

test_cat_dim()

test_multi_dims_time_series_and_uni()

test_multiple_metrics()

test_multiple_metrics_reversed()

test_pivoted_multi_dims_time_series_and_cat()

test_pivoted_multi_dims_time_series_and_uni()

test_pivoted_single_dimension_transposes_data_frame()

test_single_metric()

test_time_series_dim()

test_time_series_dim_with_operation()

test_time_series_ref()

test_uni_dim()

test_uni_dim_no_display_definition()

fireant.tests.slicer.widgets.test_datatables module

class fireant.tests.slicer.widgets.test_datatables.**DataTablesTransformerTests**(*methodName='rur*
Bases: unittest.case.TestCase

maxDiff = None

test_cat_dim()

test_multi_dims_time_series_and_uni()

test_multi_dims_with_all_levels_totals()

test_multi_dims_with_one_level_totals()

test_multiple_metrics()

test_multiple_metrics_reversed()

test_pivoted_multi_dims_time_series_and_cat()

test_pivoted_multi_dims_time_series_and_uni()

test_pivoted_single_dimension_no_effect()

test_single_metric()

test_time_series_dim()

test_time_series_dim_with_operation()

test_time_series_ref()

test_uni_dim()

test_uni_dim_no_display_definition()

class fireant.tests.slicer.widgets.test_datatables.**MetricCellFormatTests**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_does_not_prettify_none_string()

test_does_prettify_int_values()

test_does_prettify_non_none_strings()

test_does_prettify_pandas_date_objects()

fireant.tests.slicer.widgets.test_highcharts module

class fireant.tests.slicer.widgets.test_highcharts.**HighChartsAreaChartTransformerTests**(*meth*
Bases: *fireant.tests.slicer.widgets.test_highcharts.HighChartsLineChartTransformerTests*

chart_class
alias of fireant.slicer.widgets.chart_base.ChartWidget.AreaSeries

chart_type = 'area'

class fireant.tests.slicer.widgets.test_highcharts.**HighChartsAreaPercentChartTransformerTes**
Bases: *fireant.tests.slicer.widgets.test_highcharts.HighChartsAreaChartTransformerTests*

chart_class
alias of fireant.slicer.widgets.chart_base.ChartWidget.AreaPercentageSeries

stacking = 'percent'

class fireant.tests.slicer.widgets.test_highcharts.**HighChartsAreaStackedChartTransformerTes**
Bases: *fireant.tests.slicer.widgets.test_highcharts.HighChartsAreaChartTransformerTests*

chart_class
alias of fireant.slicer.widgets.chart_base.ChartWidget.AreaStackedSeries

stacking = 'normal'

class fireant.tests.slicer.widgets.test_highcharts.**HighChartsBarChartTransformerTests**(*metho*
Bases: unittest.case.TestCase

chart_class
alias of fireant.slicer.widgets.chart_base.ChartWidget.BarSeries

chart_type = 'bar'

maxDiff = None

stacking = None

test_cat_dim_multi_metric_bar_chart()

test_cat_dim_single_metric_bar_chart()

test_cat_dim_with_totals_chart()

test_cat_uni_dim_with_missing_categories()

test_cont_uni_dims_multi_metric_multi_axis_bar_chart()

test_cont_uni_dims_multi_metric_single_axis_bar_chart()

test_cont_uni_dims_single_metric_bar_chart()

test_invisible_y_axis()

test_multi_metric_bar_chart()

test_single_metric_bar_chart()

class fireant.tests.slicer.widgets.test_highcharts.**HighChartsColumnChartTransformerTests**(*m*

Bases: *fireant.tests.slicer.widgets.test_highcharts.HighChartsBarChartTransformerTests*

chart_class

alias of fireant.slicer.widgets.chart_base.ChartWidget.ColumnSeries

chart_type = 'column'

class fireant.tests.slicer.widgets.test_highcharts.**HighChartsLineChartTransformerTests**(*meth*

Bases: unittest.case.TestCase

chart_class

alias of fireant.slicer.widgets.chart_base.ChartWidget.LineSeries

chart_type = 'line'

maxDiff = None

stacking = None

test_invisible_y_axis()

test_metric_precision_line_chart()

test_metric_prefix_line_chart()

test_metric_suffix_line_chart()

test_multi_dim_with_totals_line_chart()

test_multi_dim_with_totals_on_first_dim_line_chart()

test_multi_metrics_multi_axis_line_chart()

test_multi_metrics_single_axis_line_chart()

test_ref_axes_set_to_same_visibility_as_parent_axis()

test_single_metric_line_chart()

test_single_metric_with_uni_dim_line_chart()

test_single_operation_line_chart()

test_uni_dim_with_ref_delta_line_chart()

test_uni_dim_with_ref_line_chart()

class fireant.tests.slicer.widgets.test_highcharts.**HighChartsPieChartTransformerTests**(*methoc*

Bases: unittest.case.TestCase

chart_class

alias of fireant.slicer.widgets.chart_base.ChartWidget.PieSeries

chart_type = 'pie'

maxDiff = None

test_cat_dim_multi_metric_bar_chart()

test_cat_dim_single_metric_chart()

test_cont_uni_dims_multi_metric_multi_axis_bar_chart()

test_cont_uni_dims_multi_metric_single_axis_bar_chart()

test_cont_uni_dims_single_metric_bar_chart()

test_multi_metric_chart()

test_single_metric_chart()

class fireant.tests.slicer.widgets.test_highcharts.**HighChartsStackedBarChartTransformerTest**

Bases: *fireant.tests.slicer.widgets.test_highcharts.HighChartsBarChartTransformerTests*

chart_class

alias of fireant.slicer.widgets.chart_base.ChartWidget.StackedBarSeries

chart_type = 'bar'

maxDiff = None

stacking = 'normal'

class fireant.tests.slicer.widgets.test_highcharts.**HighChartsStackedColumnChartTransformer**

Bases: *fireant.tests.slicer.widgets.test_highcharts.HighChartsBarChartTransformerTests*

chart_class

alias of fireant.slicer.widgets.chart_base.ChartWidget.StackedColumnSeries

chart_type = 'column'

stacking = 'normal'

fireant.tests.slicer.widgets.test_pandas module

class fireant.tests.slicer.widgets.test_pandas.**PandasTransformerSortTests**(*methodName='runTest'*

Bases: unittest.case.TestCase

test_empty_sort_array_is_ignored()

test_multi_dims_time_series_and_cat_sort_index_level_0_asc()

test_multiple_metrics_sort_index_and_value()

test_multiple_metrics_sort_index_asc()

test_multiple_metrics_sort_index_desc()

test_multiple_metrics_sort_value_asc()

test_multiple_metrics_sort_value_desc()

test_pivoted_multi_dims_time_series_and_cat_sort_index_and_values()

test_pivoted_multi_dims_time_series_and_cat_sort_index_level_1_desc()

test_pivoted_multi_dims_time_series_and_uni_with_sort_first_metric_asc()

test_pivoted_multi_dims_time_series_and_uni_with_sort_first_metric_desc()

test_pivoted_multi_dims_time_series_and_uni_with_sort_index_and_columns()

test_pivoted_multi_dims_time_series_and_uni_with_sort_index_asc()

test_pivoted_multi_dims_time_series_and_uni_with_sort_index_desc()

test_pivoted_multi_dims_time_series_and_uni_with_sort_second_metric_asc()

test_pivoted_multi_dims_time_series_and_uni_with_sort_second_metric_desc()

test_sort_value_greater_than_number_of_columns_is_ignored()

test_sort_with_no_index()

class fireant.tests.slicer.widgets.test_pandas.**PandasTransformerTests**(*methodName='runTest'*)
Bases: unittest.case.TestCase

maxDiff = None

test_cat_dim()

test_inf_in_metrics()

test_inf_in_metrics_with_precision_zero()

test_metric_format()

test_multi_dims_time_series_and_uni()

test_multiple_metrics()

test_multiple_metrics_reversed()

test_nan_in_metrics()

test_neginf_in_metrics()

test_pivoted_multi_dims_time_series_and_cat()

test_pivoted_multi_dims_time_series_and_uni()

test_pivoted_single_dimension_transposes_data_frame()

test_single_metric()

test_time_series_dim()

test_time_series_dim_with_operation()

test_time_series_ref()

test_transpose_single_dimension()

test_uni_dim()

test_uni_dim_no_display_definition()

fireant.tests.slicer.widgets.test_reacttable module

class fireant.tests.slicer.widgets.test_reacttable.**ReactTableHyperlinkTransformerTests**(*meth*

Bases: unittest.case.TestCase

maxDiff = None

classmethod setUpClass()

Hook method for setting up class fixture before running tests in the class.

test_dim_with_hyperlink_depending_on_another_dim_included_if_other_dim_is_selected()

test_dim_with_hyperlink_depending_on_another_dim_not_included_if_other_dim_is_not_sele

test_dim_with_hyperlink_hyperlink_is_always_included()

class fireant.tests.slicer.widgets.test_reacttable.**ReactTableReferenceItemFormatTests**(*metho*

Bases: unittest.case.TestCase

assert_object_dict(*obj*, *exp*, *attributes=[]*)

classmethod setUpClass()

Hook method for setting up class fixture before running tests in the class.

test_base_ref_item()

test_ref_item_with_delta_percentage_formats_prefix_suffix()

class fireant.tests.slicer.widgets.test_reacttable.**ReactTableTransformerTests**(*methodName='run*

Bases: unittest.case.TestCase

maxDiff = None

test_cat_dim()

test_multi_dims_time_series_and_uni()

test_multi_dims_with_all_levels_totals()

test_multi_dims_with_one_level_totals()

test_multiple_metrics()

test_multiple_metrics_reversed()

test_pivot_first_dimension_and_transpose_with_all_levels_totals()

test_pivot_multi_dims_with_all_levels_totals()

test_pivot_second_dimension_and_transpose_with_all_levels_totals()

test_pivot_second_dimension_with_multiple_metrics()

test_pivot_second_dimension_with_multiple_metrics_and_references()

test_pivot_second_dimension_with_one_metric()

test_pivot_single_dimension_as_rows_multiple_metrics()

test_pivot_single_dimension_as_rows_single_metric_and_transpose_set_to_true()

test_pivot_single_dimension_as_rows_single_metric_metrics_automatically_pivoted()

test_pivot_single_metric_time_series_dim()

test_single_metric()

test_time_series_dim()

test_time_series_dim_with_operation()

test_time_series_ref()

test_time_series_ref_multiple_metrics()

test_transpose()

test_uni_dim()

test_uni_dim_no_display_definition()

fireant.tests.slicer.widgets.test_widgets module

class fireant.tests.slicer.widgets.test_widgets.**BaseWidgetTests**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_add_widget_to_items()

test_create_widget_with_items()

test_item_func_immuatable()

test_transformable_widget_has_transform_function()

Module contents

Submodules

fireant.tests.slicer.matchers module

class fireant.tests.slicer.matchers.**DimensionMatcher**(**elements*)
Bases: fireant.tests.slicer.matchers._ElementsMatcher

expected_class
alias of *fireant.slicer.dimensions.Dimension*

class fireant.tests.slicer.matchers.**MetricMatcher**(**elements*)
Bases: fireant.tests.slicer.matchers._ElementsMatcher

expected_class
alias of *fireant.slicer.metrics.Metric*

class fireant.tests.slicer.matchers.**PypikaQueryMatcher**(*query_str*)
Bases: object

fireant.tests.slicer.mocks module

fireant.tests.slicer.mocks.**PoliticsRow**(*timestamp, candidate, candidate_display, political_party, election, election_display, state, state_display, winner, votes, wins*)

class fireant.tests.slicer.mocks.**TestDatabase**(*host='localhost', port=5433, database='vertica', user='vertica', password=None, read_timeout=None, max_processes=1, cache_middleware=None*)
Bases: *fireant.database.vertica.VerticaDatabase*

connect = <Mock id='139625906476816'>

fireant.tests.slicer.mocks.**ref**(*data_frame*, *columns*)

fireant.tests.slicer.mocks.**ref_delta**(*ref_data_frame*, *columns*)

fireant.tests.slicer.mocks.**split**(*list*, *i*)

fireant.tests.slicer.mocks.**totals**(*data_frame*, *dimensions*, *columns*)
Computes the totals across a dimension and adds the total as an extra row.

fireant.tests.slicer.test_dimensions module

class fireant.tests.slicer.test_dimensions.**UniqueDimensionTests**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_display_attribute_does_not_exist_on_dimension_with_no_display_definition()

test_display_attribute_exists_on_dimension_with_display_definition()

test_display_key_is_set_when_display_definition_is_used()

fireant.tests.slicer.test_operations module

fireant.tests.slicer.test_slicer module

class fireant.tests.slicer.test_slicer.**SlicerContainerTests**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_access_dimension_display_via_attr()

test_access_dimension_display_via_item()

test_access_dimension_via_attr()

test_access_dimension_via_item()

test_access_metric_via_attr()

test_access_metric_via_item()

test_iter_dimensions()

test_iter_metrics()

Module contents

Submodules

fireant.tests.test_formats module

class fireant.tests.test_formats.**CoerceTypeTests**(*methodName='runTest'*)
Bases: unittest.case.TestCase

allow_literal_inf()

allow_literal_inf_upper()

allow_literal_nan()

allow_literal_nan_upper()

class fireant.tests.test_formats.**DisplayValueTests**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_bool_false_value_no_formats()

test_bool_false_value_with_precision()

test_bool_false_value_with_prefix()

test_bool_false_value_with_suffix()

test_bool_true_value_no_formats()

test_bool_true_value_with_precision()

test_bool_true_value_with_prefix()

test_bool_true_value_with_suffix()

test_decimal_value_no_formats()

test_decimal_value_with_precision_0()

test_decimal_value_with_precision_2()

test_decimal_value_with_precision_9()

test_decimal_value_with_precision_trim_trailing_zeros()

test_decimal_value_with_prefix()

test_decimal_value_with_suffix()

test_inf_format_no_formatting()

test_int_value_no_formats()

test_int_value_with_precision()

test_int_value_with_prefix()

test_int_value_with_suffix()

test_nan_format_no_formatting()

test_negative_usd_float_value()

test_negative_usd_int_value()

test_null_format_no_formatting()

test_str_value_no_formats()

test_str_value_with_precision()

test_str_value_with_prefix()

test_str_value_with_suffix()

class fireant.tests.test_formats.**FormatMetricValueTests**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_data_data_point_is_returned_as_string_iso_no_time()

test_datatime_data_point_is_returned_as_string_iso_with_time()

test_int64_data_point_is_returned_as_py_int()

test_str_data_point_is_returned_unchanged()

test_that_inf_data_point_is_converted_to_none()

test_that_literal_strings_are_not_converted()

test_that_nan_data_point_is_converted_to_none()

test_that_neg_inf_data_point_is_converted_to_none()

test_timestamp_data_point_is_returned_as_string_iso_no_time()

test_timestamp_no_time_data_point_is_returned_as_string_iso_no_time()

fireant.tests.utils module

class fireant.tests.utils.**MergeDictTests**(*methodName='runTest'*)

Bases: unittest.case.TestCase

setUp()

Hook method for setting up the test fixture before exercising it.

test_dict_on_right_side_overwrites_left_side_dict()

test_merge_dict_does_not_modify_dicts()

test_merge_dict_merges_dicts_successfully()

Module contents

Submodules

fireant.formats module

fireant.formats.**coerce_type**(*value*)

fireant.formats.**date_as_millis**(*value*)

fireant.formats.**dimension_value**(*value*)

Format a dimension value. This will coerce the raw string or date values into a proper primitive value like a string, float, or int.

Parameters

- **value** – The raw str or datetime value
- **str_date** – When True, dates and datetimes will be converted to ISO strings. The time is omitted for dates. When False, the datetime will be converted to a POSIX timestamp (millis-since-epoch).

fireant.formats.**metric_display**(*value*, *prefix=None*, *suffix=None*, *precision=None*)

Converts a metric value into the display value by applying formatting.

Parameters

- **value** – The raw metric value.
- **prefix** – An optional prefix.
- **suffix** – An optional suffix.
- **precision** – The decimal precision, the number of decimal places to round to.

Returns A formatted string containing the display value for the metric.

`fireant.formats.`**`metric_value`**(*value*)

Converts a raw metric value into a safe type. This will change dates into strings, NaNs into Nones, and np types into their corresponding python types.

Parameters **`value`** – The raw metric value.

fireant.settings module

fireant.utils module

`fireant.utils.`**`chunks`**(*l*, *n*)

Yield successive n-sized chunks from l.

`fireant.utils.`**`filter_duplicates`**(*iterable*)

`fireant.utils.`**`flatten`**(*items*)

`fireant.utils.`**`format_dimension_key`**(*key*)

`fireant.utils.`**`format_key`**(*key*, *prefix=None*)

`fireant.utils.`**`format_metric_key`**(*key*)

`fireant.utils.`**`getdeepattr`**(*d*, *keys*, *default_value=None*)

Similar to the built-in *getattr*, this function accepts a list/tuple of keys to get a value deep in a *dict*

Given the following dict structure

```
d = {
  'A': {
    '0': {
      'a': 1,
      'b': 2,
    }
  },
}
```

Calling *getdeepattr* with a key path to a value deep in the structure will return that value. If the value or any of the objects in the key path do not exist, then the default value is returned.

```
assert 1 == getdeepattr(d, ('A', '0', 'a'))
assert 2 == getdeepattr(d, ('A', '0', 'b'))
assert 0 == getdeepattr(d, ('A', '0', 'c'), default_value=0)
assert 0 == getdeepattr(d, ('X', '0', 'a'), default_value=0)
```

Parameters

- **`d`** – A dict value with nested dict attributes.
- **`keys`** – A list/tuple path of keys in *d* to the desired value
- **`default_value`** – A default value that will be returned if the path *keys* does not yield a value.

Returns The value following the path *keys* or *default_value*

`fireant.utils.`**`groupby`**(*items*, *by*)

Group items using a function to derive a key.

Parameters

- **items** – The items to group
- **by** – A lambda function to create a key based on the item

Returns an Ordered dict

`fireant.utils.`**`groupby_first_level`**(*index*)

`fireant.utils.`**`immutable`**(*func*)

Decorator for wrapper "builder" functions. These are functions on the Query class or other classes used for building queries which mutate the query and return self. To make the build functions immutable, this decorator is used which will deepcopy the current instance. This decorator will return the return value of the inner function or the new copy of the instance. The inner function does not need to return self.

`fireant.utils.`**`merge_dicts`**(**dict_args*)

Given any number of dicts, shallow copy and merge into a new dict, precedence goes to key value pairs in latter dicts.

https://stackoverflow.com/questions/38987/how-to-merge-two-python-dictionaries-in-a-single-expression

`fireant.utils.`**`ordered_distinct_list`**(*l*)

`fireant.utils.`**`ordered_distinct_list_by_attr`**(*l*, *attr='key'*)

`fireant.utils.`**`reduce_data_frame_levels`**(*data_frame*, *level*)

`fireant.utils.`**`repr_field_key`**(*key*)

`fireant.utils.`**`setdeepattr`**(*d*, *keys*, *value*)

Similar to the built-in *setattr*, this function accepts a list/tuple of keys to set a value deep in a *dict*

Given the following dict structure

```
d = {
  'A': {
    '0': {
      'a': 1,
      'b': 2,
    }
  },
}
```

Calling *setdeepattr* with a key path to a value deep in the structure will set that value. If the value or any of the objects in the key path do not exist, then a dict will be created.

```
# Overwrites the value in `d` at A.0.a, which was 1, to 3
setdeepattr(d, ('A', '0', 'a'), 3)

# Adds an entry in `d` to A.0 with the key 'c' and the value 3
setdeepattr(d, ('A', '0', 'c'), 3)

# Adds an entry in `d` with the key 'X' and the value a new dict
# Adds an entry in `d` to `X` with the key '0' and the value a new dict
# Adds an entry in `d` to `X.0` with the key 'a' and the value 0
setdeepattr(d, ('X', '0', 'a'), 0)
```

Parameters

- **d** – A dict value with nested dict attributes.
- **keys** – A list/tuple path of keys in *d* to the desired value
- **value** – The value to set at the given path *keys*.

fireant.utils.**slice_first**(*item*)

fireant.utils.**wrap_list**(*value*)

Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex

Python Module Index

Index