# MobilityDB SQLAlchemy

*Release 0.4.1*

**Bommakanti Krishna Chaitanya**

**Jun 11, 2022**

# CONTENTS:

CHAPTER

ONE

# QUICKSTART

## 1.1 Temporal data

mobilitydb-sqlalchemy lets you use pandas DataFrame (which are great for timeseries data) while you are in the Python world, and translates it back and for to temporal types defined in mobilitydb.

A point to note here is that we assume that the DataFrame's columns are named "value" (except in case of TGeomPoint where it is "geometry") and "t" for the data and the timestamp respectively.

Here we show how we can store numeric data which changes over time (i.e. tfloat), using the `mobilitydb_sqlalchemy.types.TFloat.TFloat` class.

Running the following code will create a new table with a tfloat column, and insert one row of hardcoded data into it.

### 1.1.1 Write data to MobilityDB

```
import datetime
import pandas as pd

from mobilitydb_sqlalchemy import TFloat
from sqlalchemy import Column, Integer, create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

# Setup the engine and session, make sure you set the right url to connect to your␣
↪mobilitydb instance
engine = create_engine("postgresql://docker:docker@localhost:25432/mobilitydb",␣
↪echo=True)
session = sessionmaker(bind=engine)()

# Setup and create the tables (only one in our case here)
Base = declarative_base()

class TemporalFloats(Base):
    __tablename__ = "tfloat_test_001"
    id = Column(Integer, primary_key=True)
    tdata = Column(TFloat(True, False))

Base.metadata.create_all(engine)

# Prepare and insert the data
df = pd.DataFrame(
    [
```

(continues on next page)

(continued from previous page)

```
        {"value": 0, "t": datetime.datetime(2018, 1, 1, 12, 0, 0)},
        {"value": 8.2, "t": datetime.datetime(2018, 1, 1, 12, 6, 0)},
        {"value": 6.6, "t": datetime.datetime(2018, 1, 1, 12, 10, 0)},
        {"value": 9.1, "t": datetime.datetime(2018, 1, 1, 12, 15, 0)},
    ]
).set_index("t")
row = TemporalFloats(tdata=df,)
session.add(row)
session.commit()
```

## 1.2 Geometric data

While creating the DataFrame, make sure the column is named "geometry" and not "value". This is to maintain compatibility with movingpandas. We can use Point objects from shapely for preparing the geometry data.

### 1.2.1 Writing

```
import datetime
import pandas as pd

from mobilitydb_sqlalchemy import TGeomPoint
from shapely.geometry import Point
from sqlalchemy import Column, Integer, create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

engine = create_engine("postgresql://docker:docker@db:25432/mobilitydb", echo=True)
session = sessionmaker(bind=engine)()

Base = declarative_base()

class Trips(Base):
    __tablename__ = "trips_test_001"
    car_id = Column(Integer, primary_key=True)
    trip_id = Column(Integer, primary_key=True)
    trip = Column(TGeomPoint)

Base.metadata.create_all(engine)

# Prepare and insert the data
df = pd.DataFrame(
    [
        {"geometry": Point(0, 0), "t": datetime.datetime(2012, 1, 1, 8, 0, 0),},
        {"geometry": Point(2, 0), "t": datetime.datetime(2012, 1, 1, 8, 10, 0),},
        {"geometry": Point(2, -1.9), "t": datetime.datetime(2012, 1, 1, 8, 15, 0),},
    ]
).set_index("t")

trip = Trips(car_id=1, trip_id=1, trip=df,)
session.add(trip)
session.commit()
```

### 1.2.2 Querying

Listing 1: Example usage of the **TGeomPoint** class as a column in a table defined using SQLAlchemy's declarative API

```
import datetime

from mobilitydb_sqlalchemy import TGeomPoint

from sqlalchemy import Column, Integer, create_engine, func
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

engine = create_engine("postgresql://docker:docker@db:25432/mobilitydb", echo=True)
session = sessionmaker(bind=engine)()

Base = declarative_base()

class Trips(Base):
    __tablename__ = "trips_test_001"
    car_id = Column(Integer, primary_key=True)
    trip_id = Column(Integer, primary_key=True)
    trip = Column(TGeomPoint)

trips = session.query(Trips).all()

# Querying using MobilityDB functions, for example - valueAtTimestamp
session.query(
    Trips.car_id,
    func.ST_asText(
        func.valueAtTimestamp(Trips.trip, datetime.datetime(2012, 1, 1, 8, 10, 0))
    ),
).all()
```

## 1.3 Inserting TGeomPoint data, using movingpandas

movingpandas is an optional dependency, but if installed, you can insert TGeomPoint data with Trajectory objects directly. Just be sure to enable the flag use_movingpandas on the column beforehand.

```
import datetime
import pandas as pd
from geopandas import GeoDataFrame
import movingpandas as mpd

from mobilitydb_sqlalchemy import TGeomPoint
from shapely.geometry import Point
from sqlalchemy import Column, Integer, create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

from fiona.crs import from_epsg
CRS_METRIC = from_epsg(31256)

engine = create_engine("postgresql://docker:docker@db:25432/mobilitydb", echo=True)
session = sessionmaker(bind=engine)()
```

(continues on next page)

(continued from previous page)

```
Base = declarative_base()

class Trips(Base):
    __tablename__ = "trips_test_002"
    car_id = Column(Integer, primary_key=True)
    trip_id = Column(Integer, primary_key=True)
    trip = Column(TGeomPoint(use_movingpandas=True))

Base.metadata.create_all(engine)

# Prepare and insert the data
df = pd.DataFrame(
    [
        {"geometry": Point(0, 0), "t": datetime.datetime(2012, 1, 1, 8, 0, 0),},
        {"geometry": Point(2, 0), "t": datetime.datetime(2012, 1, 1, 8, 10, 0),},
        {"geometry": Point(2, -1.9), "t": datetime.datetime(2012, 1, 1, 8, 15, 0),},
    ]
).set_index("t")
geo_df = GeoDataFrame(df, crs=CRS_METRIC)

traj = mpd.Trajectory(geo_df, 1)
# Note: In case you are depending on movingpandas 0.1 or lower,
# you might need to do mpd.Trajectory(1, geo_df) instead

trip = Trips(car_id=1, trip_id=1, trip=traj,)
session.add(trip)
session.commit()
```

## 1.4 Querying data from MobilityDB

SQLAlchemy's *func* is pretty generic and flexible, allowing us to use MobilityDB's functions without needing any new constructs.

Let's take few example queries from MobilityDB's documentation, and explain how we can achieve the same using this package.

```
from sqlalchemy import func
from shapely.wkt import loads
```

### 1.4.1 Value at a given timestamp

```
SELECT CarId, ST_AsText(valueAtTimestamp(Trip, timestamptz '2012-01-01 08:10:00'))␣
↪FROM Trips;
-- 10;"POINT(2 0)"
-- 20;"POINT(1 1)"
```

```
session.query(
    Trips.car_id,
    func.asText(
        func.valueAtTimestamp(Trips.trip, datetime.datetime(2012, 1, 1, 8, 10, 0))
```

(continues on next page)

(continued from previous page)

```
    ),
).all()
```

### 1.4.2 Restriction to a given value

```
SELECT CarId, asText(atValue(Trip, 'Point(2 0)'))
FROM Trips;
-- 10;"{"[POINT(2 0)@2012-01-01 08:10:00+00]"}"
-- 20; NULL
```

```
session.query(
    Trips.car_id,
    func.asText(func.atValue(Trips.trip, Point(2, 0).wkt)),
).all()
```

### 1.4.3 Restriction to a period

```
SELECT CarId, asText(atPeriod(Trip, '[2012-01-01 08:05:00,2012-01-01 08:10:00]'))
FROM Trips;
-- 10;"{[POINT(1 0)@2012-01-01 08:05:00+00, POINT(2 0)@2012-01-01 08:10:00+00]}"
-- 20;"{[POINT(0 0)@2012-01-01 08:05:00+00, POINT(1 1)@2012-01-01 08:10:00+00]}"
```

```
session.query(
    Trips.car_id,
    func.asText(
        func.atPeriod(Trips.trip, "[2012-01-01 08:05:00,2012-01-01 08:10:00]")
    ),
).all()
```

### 1.4.4 Temporal distance

```
-- Temporal distance
SELECT T1.CarId, T2.CarId, T1.Trip <-> T2.Trip
FROM Trips T1, Trips T2
WHERE T1.CarId < T2.CarId;
-- 10;20;"{[1@2012-01-01 08:05:00+00, 1.4142135623731@2012-01-01 08:10:00+00, 1@2012-
↪01-01 08:15:00+00)}"
```

```
session.query(
    T1.c.car_id,
    T2.c.car_id,
    T1.c.trip.distance(T2.c.trip),
) \
.filter(T1.c.car_id < T2.c.car_id,)
.all()
```

## 1.5 Using MobilityDB operators

Listing 2: Example usage of the distance operator ('<->')

```
session.query(
    T1.c.car_id,
    T2.c.car_id,
    T1.c.trip.distance(T2.c.trip),
) \
.filter(T1.c.car_id < T2.c.car_id,)
.all()
```

For exhaustive listing of operators, see *operators page*.

## 1.6 Using MobilityDB ranges

MobilityDB also allows you to store the temporal data in either open or closed intervals on either site. While this is supported by the package at the column level, because we use pandas DataFrame to hold the values once we load them into python runtime, this data is lost, and hence not of much use. In future, this can be avoided with a better suiting data structure to hold this data instead of relying on pandas.

However, to define a column which stores temporal data as a left closed, right open interval, ie. '[)', it can be done as shown below:

```
class Trips(Base):
    trip_id = Column(Integer, primary_key=True)
    trip = Column(TGeomPoint(True, False))
```

## 1.7 Making use of movingpandas Trajectory data structure

TGeomPoint objects can also be optionally mapped to movingpandas Trajectory objects.

For this the optional dependency "movingpandas" needs to be installed.

```
poetry install -E movingpandas
```

After this, movingpandas can be enabled with a flag on the TGeomPoint column

```
class Trips(Base):
    trip_id = Column(Integer, primary_key=True)
    trip = Column(TGeomPoint(use_movingpandas=True))
```

CHAPTER

# TWO

# OPERATORS

More details can be found in MobilityDB's documentation itself: https://docs.mobilitydb.com/nightly/index.html

**class** mobilitydb_sqlalchemy.comparator.**Comparator**(*expr*)
Bases: sqlalchemy.sql.type_api.Comparator

A custom comparator base class. It adds the ability to call spatial and temporal functions on columns that use this kind of comparator. It also defines functions that map to operators supported by `TBool`, `TInt`, `TFloat` and `TGeomPoint` columns.

**always_different_from**(*other*)
The "@<>" operator.

**always_equal_to**(*other*)
The "%=" operator.

Is lhs always equal to the rhs?

The function does not take into account whether the bounds are inclusive or not.

**always_greater_than**(*other*)
The "%>" operator.

**always_greater_than_or_equal_to**(*other*)
The "%>=" operator.

**always_less_than**(*other*)
The "%<" operator.

**always_less_than_or_equal_to**(*other*)
The "%<=" operator.

**bbox_always_after**(*other*)
The "<<#" operator.

**bbox_always_before**(*other*)
The "<<#" operator.

**bbox_always_strictly_greater_than**(*other*)
The ">>" operator.

**bbox_always_strictly_less_than**(*other*)
The "<<" operator.

**bbox_contained**(*other*)
The "<@" operator.

**bbox_contains**(*other*)
The "@>" operator.

**bbox_does_not_extend_above**(*other*)
The "|&>" operator.

**bbox_does_not_extend_below**(*other*)
The "&<|" operator.

**bbox_does_not_extend_in_back**(*other*)
The "/&>" operator.

**bbox_does_not_extend_in_front**(*other*)
The "&</" operator.

**bbox_does_not_extend_to_left**(*other*)
The "&>" operator.

**bbox_does_not_extend_to_right**(*other*)
The "&<" operator.

**bbox_never_after**(*other*)
The "&<#" operator.

**bbox_never_before**(*other*)
The "#&>" operator.

**bbox_never_greater_than**(*other*)
The "&<" operator.

**bbox_never_less_than**(*other*)
The "&>" operator.

**bbox_strictly_above**(*other*)
The "|>>" operator.

**bbox_strictly_below**(*other*)
The "<<|" operator.

**bbox_strictly_in_back**(*other*)
The "/>>" operator.

**bbox_strictly_in_front**(*other*)
The "<</" operator.

**bbox_strictly_to_left**(*other*)
The "<<" operator.

**bbox_strictly_to_right**(*other*)
The ">>" operator.

**bboxes_equal**(*other*)
The "~=" operator.

**bboxes_overlap**(*other*)
The "&&" operator.

**distance**(*other*)
The "<->" operator.

**ever_different_from**(*other*)
The "?<>" operator.

**ever_equal_to**(*other*)
The "?=" operator.

Is lhs ever equal to the rhs?

The function does not take into account whether the bounds are inclusive or not.

**ever_greater_than**(*other*)
The "?>" operator.

**ever_greater_than_or_equal_to**(*other*)
The "?>=" operator.

**ever_less_than**(*other*)
The "?<" operator.

**ever_less_than_or_equal_to**(*other*)
The "?<=" operator.

**expr**

**smallest_distance_ever_between**(*other*)
The "|=|" operator.

**temporal_equal**(*other*)
The "#=" operator.

**temporal_greater_than**(*other*)
The "#>" operator.

**temporal_greater_than_or_equal_to**(*other*)
The "#>=" operator.

**temporal_less_than**(*other*)
The "#<" operator.

**temporal_less_than_or_equal_to**(*other*)
The "#<=" operator.

**temporal_not_equal**(*other*)
The "#<>" operator.

**type**

MobilityDB extensions for SQLAlchemy.

Specifically, the following column types are provided, which can be used in SQLAlchemy models defined using the declarative API:

**class** mobilitydb_sqlalchemy.types.**TGeomPoint**

**class** mobilitydb_sqlalchemy.types.**TFloat**

**class** mobilitydb_sqlalchemy.types.**TInt**

**class** mobilitydb_sqlalchemy.types.**TBool**

CHAPTER

THREE

# INSTALLATION

```
pip install mobilitydb_sqlalchemy
```

For getting started with MobilityDB SQLAlchemy, read our *Quickstart*

CHAPTER

FOUR

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## m

# INDEX