MobilityDB tutorial pgconf.ru 2021

In this tutorial you will walk through the MobilityDB API, and learn how to use it for analyzing geospatial trajectory data.

Connection to the database

We have prepared a MobilityDB server for this tutorial.

```
psql -h 135.181.248.131 -U mobdbguest -d pgconf
Password:  pgconf2021mobdb
```

This database is only accessible during the tutorial. But it is easy afterwards to build it yourself by following our BerlinMOD benchmark manual.

Data

During the tutorial you will work with this schema.

```
CREATE TABLE trips(
vehicle integer NOT NULL,
day date NOT NULL,
seq integer NOT NULL,
source bigint,
target bigint,
trip tgeompoint,
trajectory geometry,
CONSTRAINT trips_pkey
  PRIMARY KEY (vehicle, day, seq)
)
```

```
CREATE TABLE municipalities(
id integer,
name text,
population integer,
percpop numeric,
popdensitykm2 integer,
noenterp integer,
percenterp numeric,
geom geometry
)
```

```
CREATE TABLE planet_osm_point(
...
Name text
way geometry(Point, 3857),
...
)
```

Curious about the source of the data ? It has been generated using the BerlinMOD data generator.

Tutorial queries

Explore the database

1) Display the geospatial trips in a readable text format:

```
SELECT astext(trip)::varchar(100) AS trip
FROM trips
LIMIT 5;
```

```
                                                 trip
-------------------------------------------------------------------------------------------------
 [POINT(493688.895023847 6580329.05100153)@2020-06-01 09:00:47.886+02, POINT(493683.990711951 6580328
 [POINT(490872.022101018 6604317.29566003)@2020-06-01 16:42:53.192+02, POINT(490876.083834722 6604314
 [POINT(493688.895023847 6580329.05100153)@2020-06-01 21:25:10.562+02, POINT(493683.990711951 6580328
 [POINT(500260.040037578 6581288.76162654)@2020-06-01 08:10:59.002+02, POINT(500263.776324533 6581292
 [POINT(496464.245776611 6580346.59913265)@2020-06-01 22:15:40.545128+02, POINT(496466.790775785 6580
(5 rows)
```

The function astext(.) creates a human readable output from the binary representation of the temporal object.

Notice the structure of the trip values. They are in the form of an array of pairs point@timestamp. These pairs come from the input observations (for example GPS). Typically some observations will be redundant, and MobilityDB will not store them. MobilityDB will also interpolate “linearly” between observations. So, one can query the position of the car at any timestamp in the continuous time range of the trip.

2) Display the individual observations composing the trip:

```
SELECT unnest(instants(trip)) AS instant
FROM trips
WHERE Vehicle = 1 AND day='2020-06-01' AND seq=1
LIMIT 10;

                                        instant
-------------------------------------------------------------------------------
 0101000020110F0000A5218194E3211E41EB9B43431A1A5941@2020-06-01 09:00:47.886+02
 0101000020110F000095317DF6CF211E415898F5041A1A5941@2020-06-01 09:00:49.386+02
 0101000020110F000084417958BC211E41C694A7C6191A5941@2020-06-01 09:00:50.286+02
 0101000020110F0000ADF1B781B0211E41C0D80DA1191A5941@2020-06-01 09:00:58.217104+02
 0101000020110F0000934C4E2C9E211E4123372D21191A5941@2020-06-01 09:00:59.563436+02
 0101000020110F00007A34C480D4201E41624586A2131A5941@2020-06-01 09:01:09.463436+02
 0101000020110F00007A34C480D4201E41624586A2131A5941@2020-06-01 09:01:09.50292+02
 0101000020110F0000C3F0E6BCD3201E413B1D309D131A5941@2020-06-01 09:01:09.565517+02
 0101000020110F0000978D98FF41201E4102D180E40E1A5941@2020-06-01 09:01:16.95247+02
 0101000020110F0000D779762130201E41D2E1BA540E1A5941@2020-06-01 09:01:17.85247+02
(10 rows)
```

The function instants(.) returns a postgresql ARRAY of all the instants that compose the temporal object. Every instant has the format point@timestamp. Note that the points here are displayed in well know binary format WKB, which is the default display of PostGIS geometries.

3) Display the start and end timestamp of a trip:

```
SELECT startTimestamp(trip) AS start, endTimestamp(trip) AS end
FROM trips
LIMIT 5;
```

```
             start               |            end
---------------------------------+---------------------------------
 2020-06-01 09:00:47.886+02      | 2020-06-01 09:46:42.031692+02
 2020-06-01 16:42:53.192+02      | 2020-06-01 17:23:28.350813+02
 2020-06-01 21:25:10.562+02      | 2020-06-01 21:37:14.920128+02
 2020-06-01 08:10:59.002+02      | 2020-06-01 08:12:31.179416+02
 2020-06-01 22:15:40.545128+02 | 2020-06-01 22:25:18.204499+02
(5 rows)
```

4) What is the time span of the whole dataset ?

```
SELECT MIN(startTimestamp(trip)) AS begin, MAX(endTimestamp(trip)) AS end
FROM trips;
```

```
             begin         |            end
---------------------------+-----------------------------
 2020-06-01 08:00:09.334+02 | 2020-06-03 00:25:53.92958+02
(1 row)
```

5) What is the driven distance of every trip ?

```
SELECT length(trip)/1000 AS tripKms
FROM trips
LIMIT 5;

       tripkms
---------------------
  32.819675689126214
  31.340453572998296
  4.62088788850962
  0.46188900092268487
  3.4244842695518236
(5 rows)
```

The function length(.) returns the total length of a temporal point. Since the coordinates of the points are stored in meters, this length will also be returned in meters.

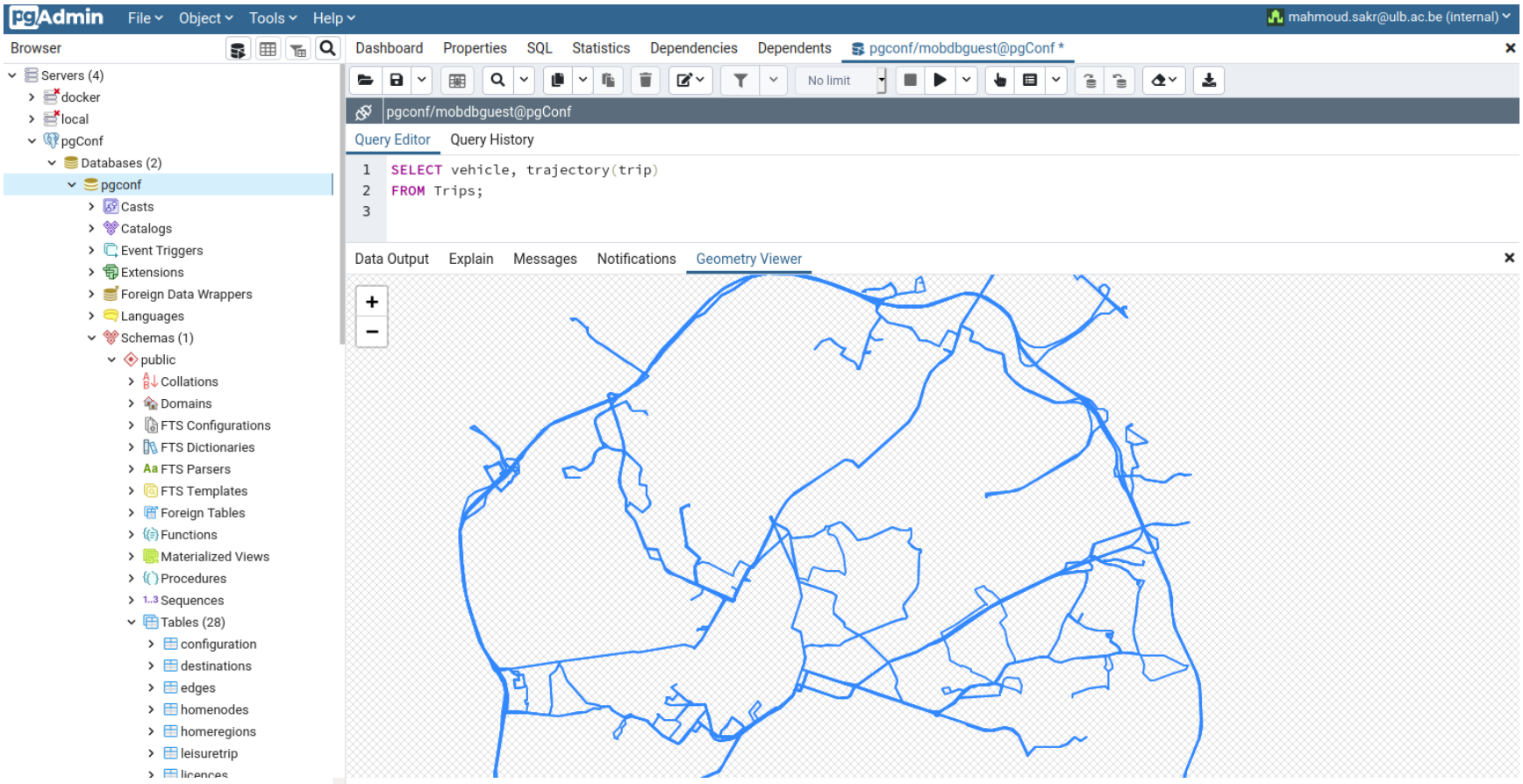
6) Dataset summaries:

```
SELECT
       MIN(length(trip)) AS minLength,
       MAX(length(trip)) AS maxLength,
       AVG(length(trip)) AS avgLength,
       MIN(duration(trip)) AS minDuration,
       MAX(duration(trip)) AS maxDuration,
       AVG(duration(trip)) AS avgDuration,
       MIN(numInstants(trip)) AS minPoints,
       MAX(numInstants(trip)) AS maxPoints,
       AVG(numInstants(trip)) AS avgPoints,
       AVG(numInstants(trip) * 60 / extract (epoch from timespan(trip))) AS avgPointsPerMinute
FROM trips;

       minlength     |      maxlength     |      avglength     |   minduration   |   maxduration   |   avgduration   |
---------------------+--------------------+--------------------+-----------------+-----------------+-----------------+
 461.88900092268483 | 37543.95951933945 | 14914.107113980963 | 00:01:15.052723 | 01:10:16.893662 | 00:25:25.772065 |

minpoints | maxpoints |   avgpoints     | avgpointsperminute
-----------+-----------+----------------------+--------------------
25 |   3093 | 1244.1916666666666667 |  45.63301083678095

(1 row)
```

7) Visualize the trajectories in pgAdmin:

```
SELECT vehicle, trajectory(trip)
FROM trips;
```

This query selects both the vehicle id, as well as the trajectory of the trip (returned as a PostGIS linestring).



8) Cast the trip to a PostGIS trajectory object:

```
SELECT st_astext(trip::geometry)::varchar(100) tripGeom
FROM trips
LIMIT 5;

                                                tripgeom
-----------------------------------------------------------------------------------------------
 LINESTRING M (493688.895023847 6580329.05100153 1590994847.886,493683.990711951 6580328.07748993 159
 LINESTRING M (490872.022101018 6604317.29566003 1591022573.192,490876.083834722 6604314.37978635 159
 LINESTRING M (493688.895023847 6580329.05100153 1591039510.562,493683.990711951 6580328.07748993 159
 LINESTRING M (500260.040037578 6581288.76162654 1590991859.002,500263.776324533 6581292.08430013 159
 LINESTRING M (496464.245776611 6580346.59913265 1591042540.54513,496466.790775785 6580342.29529957 1
(5 rows)
```

A MobilityDB tgeompoint object can be cast to a PostGIS LinestringM object, with the M dimension of the LinestringM storing the the timestamps as epoch (seconds since 1/1/1970).

9) Retrieve the SRID of the temporal points:

```
SELECT DISTINCT SRID(trip) AS srid
FROM trips;

 srid
------
 3857
(1 row)
```

The function SRID(.) returns the Spatial Reference System Identifier (the spatial coordinate system) of a temporal point. The value 0 corresponds to having no SRID.

10) Project the temporal point to another SRID:

```
SELECT asText(transform(trip, 4326))::varchar(100) AS trip4326
FROM trips
LIMIT 5;
                                                trip4326
-------------------------------------------------------------------------------------------------------
 [POINT(4.4348828 50.7678369)@2020-06-01 09:00:47.886+02, POINT(4.43483874381665 50.7678313689714)@20
 [POINT(4.4095784 50.9039282)@2020-06-01 16:42:53.192+02, POINT(4.40961488717467 50.9039116816333)@20
 [POINT(4.4348828 50.7678369)@2020-06-01 21:25:10.562+02, POINT(4.43483874381665 50.7678313689714)@20
 [POINT(4.4939124 50.7732892)@2020-06-01 08:10:59.002+02, POINT(4.49394596363678 50.7733080756416)@20
 [POINT(4.4598142 50.7679366)@2020-06-01 22:15:40.545128+02, POINT(4.45983706211656 50.7679121477201)
(5 rows)
```

11) Compute the time-varying speed of the vehicles:

```
SELECT speed(trip)::varchar(100)
FROM trips
LIMIT 5;
                                                speed
-------------------------------------------------------------------------------------------------------
 Interp=Stepwise;[3.333333333369115@2020-06-01 09:00:47.886+02, 5.555555555477151@2020-06-01 09:00:49
 Interp=Stepwise;[3.3333333332276402@2020-06-01 16:42:53.192+02, 0@2020-06-01 16:42:54.692+02, 3.3333
 Interp=Stepwise;[3.333333333369115@2020-06-01 21:25:10.562+02, 5.555555555477151@2020-06-01 21:25:12
 Interp=Stepwise;[3.333333333523908@2020-06-01 08:10:59.002+02, 0.5938307772393736@2020-06-01 08:11:0
 Interp=Stepwise;[3.3333333335554904@2020-06-01 22:15:40.545128+02, 5.555555555480455@2020-06-01 22:1
(5 rows)
```

The function speed(.) returns the speed of a temporal point as a temporal float. Notice that the speed between two instants is assumed to be constant, which means that there is a jump in speed at each instant of the temporal point. The returned temporal float thus uses a stepwise interpolation method.

12) What is the average speed of the vehicles in Km/h:

```
SELECT twAvg(speed(trip)) * 3.6 AS averageSpeedKmH
FROM trips
LIMIT 5;

  averagespeedkmh
--------------------
  42.89926739316961
 46.331940348398845
 22.965430711139373
 18.039130141396736
  21.34154484336507
(5 rows)
```

The function twAvg(.) returns the time-weighted average of a temporal float. Since the speed is given in meters and the time in seconds, we multiply the average by 3.6 to have the result in Km/h.

13) Compute the cumulative distance traveled by a vehicle as a temporal float:

```
SELECT
      cumulativeLength(trip)::varchar(100) AS cumulativeLength,
      endValue(cumulativeLength(trip)) AS totalLength1,
      length(trip) AS totalLength2
FROM trips
LIMIT 5;
```

```
                                        cumulativelength                                       |      totallength1      |      totallength2
-----------------------------------------------------------------------------------------------+--------------------+--------------------
 [0@2020-06-01 09:00:47.886+02, 5.000000000053673@2020-06-01 09:00:49.386+02, 9.999999999983109@2020- |  32819.67568912621 |  32819.67568912621
 [0@2020-06-01 16:42:53.192+02, 4.99999999984146@2020-06-01 16:42:54.692+02, 4.99999999984146@2020-06 | 31340.453572998296 | 31340.453572998296
 [0@2020-06-01 21:25:10.562+02, 5.000000000053673@2020-06-01 21:25:12.062+02, 9.999999999983109@2020- |   4620.88788850962 |   4620.88788850962
 [0@2020-06-01 08:10:59.002+02, 5.000000000285862@2020-06-01 08:11:00.502+02, 7.999729045541231@2020- |  461.8890009226849 |  461.8890009226849
 [0@2020-06-01 22:15:40.545128+02, 5.000000000333236@2020-06-01 22:15:42.045128+02, 15.00000000019805 | 3424.4842695518237 | 3424.4842695518237
 (5 rows)
```

The function cumulativeLength(.) returns the cumulative length traveled by a temporal point as a temporal float. This temporal float has linear interpolation and its last value corresponds to the total length traveled by the temporal point, which is also the result of the length(.) function.

14) Which vehicle trips passed in the municipality of Evere

```
SELECT t.vehicle, t.day, t.seq
FROM trips t, municipalities c
WHERE c.name like '%Evere%' and intersects(t.trip, c.geom);
```

```
 vehicle |    day     | seq
---------+------------+-----
      18 | 2020-06-01 |   2
      18 | 2020-06-01 |   1
      17 | 2020-06-01 |   4
      17 | 2020-06-01 |   3
      18 | 2020-06-02 |   2
      18 | 2020-06-02 |   1
(6 rows)
```

This query uses the GiST index on the trip attribute.

Analysis

15) What was the closest distance between any vehicle and 'Grand Place - Grote Markt'

```
SELECT MIN(trip |=| way) AS distance
FROM trips, planet_osm_point
WHERE name = 'Grand Place - Grote Markt';

       distance
--------------------
 1233.9475496127047
(1 row)
```

16) How many trips start and finish in different municipalities ?

```
SELECT COUNT(*)
FROM trips T, municipalities S, municipalities E
WHERE S.id <> E.id AND ST_Intersects(startValue(T.trip), S.geom) AND ST_Intersects(endValue(T.trip), E.geom);

 count
-------
    38
(1 row)
```

17) Are there car accidents in this database ?

```
SELECT a.vehicle, a.day, a.seq, b.vehicle, b.day, b.seq
FROM trips a, trips b
WHERE a.vehicle < b.vehicle AND
    a.day < b.day AND
    a.seq < b.seq AND
    tdwithin(a.trip, b.trip, 10) %= TRUE;

 vehicle | day | seq | vehicle | day | seq
---------+-----+-----+---------+-----+-----
(0 rows)
```

18) Average duration of trips starting in the municipality of ‘Uccle’.

```
SELECT avg(duration(t.trip))
FROM trips t, municipalities m
WHERE m.name like '%Uccle%' AND
       ST_Intersects(startValue(t.trip), m.geom);
        avg
-----------------
 00:42:05.263094
(1 row)
```

Distributed MobilityDB (Citus)

19) What is the total driven distance by all vehicles in the municipality of `Anderlecht` ?

```
SELECT SUM(length(atGeometry(t.trip, m.geom))) / 1e3 AS lengthKm
FROM trips_d t, municipalities_r m
WHERE m.name like '%Anderlecht%' AND t.trip && m.Geom;

       lengthkm
--------------------
 102.06300673547392
(1 row)
```

20) Give the number of trips such that their route intersects the municipality of Jette for more than 10 minutes.

```
SELECT COUNT(*) AS numTrips
FROM trips_d t, municipalities_r m
WHERE m.name like '%Jette%' AND
       t.trip && m.geom AND
       duration(atGeometry(t.trip, m.geom)) >= interval '10 min';

 numtrips
----------
       2
(1 row)
```

21) For each speed range of 20 Km/h, give the total distance traveled by all vehicles within that range.

```
WITH Ranges AS (
  SELECT I AS RangeKey, floatrange(((I-1)*20), I*20) AS Range
  FROM generate_series(1,10) I )
SELECT R.Range AS speedRange,
  SUM(
    length(
      atPeriodSet(T.Trip,
        getTime(
          atRange(
            speed(T.Trip) * 3.6, R.Range)))) / 1000) AS km
FROM trips_d t, ranges r
WHERE atRange(speed(t.trip) * 3.6, r.range) IS NOT NULL
GROUP BY R.RangeKey, R.Range
ORDER BY R.RangeKey;
```

```
   range   |        sum
-----------+--------------------
 [0,20)    | 120.03251053881047
 [20,40)   | 266.00301309677815
 [40,60)   |  571.6992484428724
 [60,80)   | 313.72340400160704
 [80,100)  | 136.59922745332932
 [100,120) | 179.63843877331487
 [120,140) | 201.99701137100493
(7 rows)
```

Contacts

- User questions: http://lists.osgeo.org/mailman/listinfo/mobilitydb-users
- Developer questions: http://lists.osgeo.org/mailman/listinfo/mobilitydb-dev

Credits

This tutorial has been prepared by Esteban Zimanyi, Mahmoud SAKR, Mohammed Bakli, Maxime Schoemans, Robin Choquet, and Dimitris Tsesmelis.