Activity 4: Flight Data

The data used for this assignment is provided by The OpenSky Network. We will use data from a 24hr-period corresponding to June 1, 2020 (dataset link). The raw data are provided in separate CSV documents for each hour of the day.

As in the previous assignment, we analyse the data quality and, if necessary, define and apply some rules to fix or discard data with problems (ETL). Again, we build trajectories from the raw data.

Exercise 1. Creating and populating the database

1.1 Create a mobilityDB-enabled database

```
CREATE DATABASE OpenSky;
```

Choose it and run the following sentence:

```
CREATE EXTENSION IF NOT EXISTS MobilityDB CASCADE;
```

1.1.1 Download the data from states-2020-06-01.zip

The raw data will be stored in a table called *flights*, with the following structure:

```
CREATE TABLE Flights(
et bigint,
icao24 varchar(20),
lat float,
lon float,
velocity float,
heading float,
vertrate float,
callsign varchar(10),
onground boolean,
alert boolean,
spi boolean,
```

```
squawk integer,
baroaltitude numeric(7,2),
geoaltitude numeric(7,2),
lastposupdate numeric(13,3),
lastcontact numeric(13,3)
);
```

1.1.2. Populate the *Flights* table.

Load the data into the database using the following command. Replace the <path_to_file> with the actual path of the CSV file. Do this for all files (change the path, if necessary)

```
COPY flights(et, icao24, lat, lon, velocity, heading,
vertrate, callsign, onground, alert, spi, squawk,
baroaltitude, geoaltitude, lastposupdate, lastcontact)
FROM '<path_to_file>' DELIMITER ',' CSV HEADER;
```

You can run the following script which iterates over the 23 files and executes dynamic SQL statements:

```
DO
$$DECLARE
prefixpath text= '/yourpath/states_2020-06-01-';
path text;

BEGIN
    FOR rec in 0..23 LOOP
       path:= prefixpath || trim(to_char(rec, '09')) ||
       '.csv'; -- fill with 0s

      EXECUTE format('COPY flights(et, icao24, lat, lon,
              velocity, heading, vertrate, callsign,
              onground, alert, spi, squawk, baroaltitude,
              geoaltitude, lastposupdate, lastcontact)
      FROM %L WITH DELIMITER '','' CSV HEADER', path);
     COMMIT;
     Raise Notice 'inserting %', path;
    END LOOP;
END
$$;
```

All the times in this dataset are in Unix timestamp (an integer) with timezone being UTC. Thus, we need to convert them to PostgreSQL timestamp type. This is done as follows:

```
ALTER TABLE Flights
ADD COLUMN et_ts timestamptz,
ADD COLUMN lastposupdate_ts timestamptz,
ADD COLUMN lastcontact_ts timestamptz,
ADD COLUMN Geom geometry(Point, 4326);

UPDATE Flights
SET et_ts = to_timestamp(et), lastposupdate_ts =
            to_timestamp(lastposupdate),
            lastcontact_ts = to_timestamp(lastcontact),

            Geom = ST_SetSRID(ST_MakePoint(Lon, Lat), 4326);
```

Check the size of the database with:

```
SELECT pg_size_pretty( pg_total_relation_size('flights') );
```

Exercise 2. Cleaning the database

2.1 Delete the NULL values for latitude.

```
-- icao24_with_null_lat is used to indicate the list of
   rows to be deleted

WITH icao24_with_null_lat AS (
SELECT icao24, COUNT(lat)
FROM flights
GROUP BY icao24
HAVING COUNT(lat) = 0
)

DELETE
FROM Flights
WHERE icao24 IN
-- this SELECT statement is needed for the IN statement to
   compare against a list
          (SELECT icao24 FROM icao24_with_null_lat);
```

2.2. Explore the database and propose and execute new cleaning tasks (in what follows we assume that only the cleaning tasks in 2.1. have been done).

Exercise 3. Raw data visualization using QGIS

We now visualize the raw data using different tools.

For example, we visualize the points of flights over Australia using QGIS as follows:

Load the Waze (world) map service and run

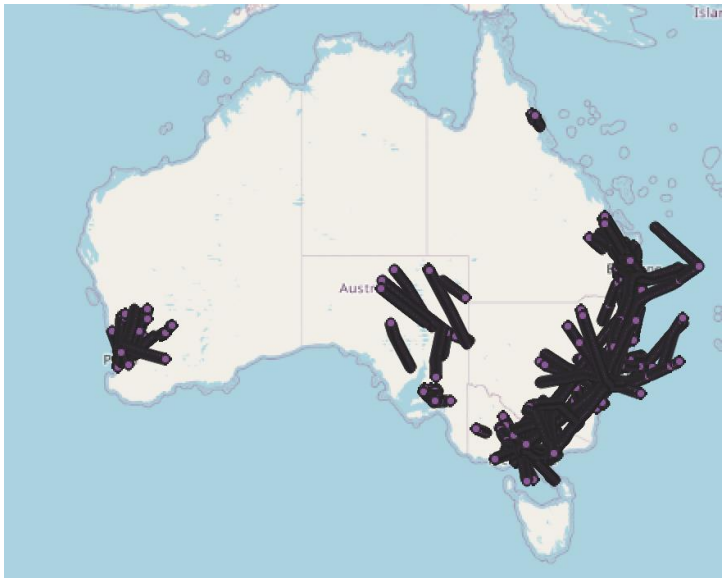
```
SELECT row_number() over() as ctid,
st_setsrid(geom, 4326) as geom
FROM(
WITH Australia(AustEnv) AS

   (SELECT ST_SETSRID(ST_makeEnvelope(113.338953078,-

    43.6345972634, 153.569469029, -10.6681857235), 4326))

 SELECT et_ts, icao24, geom
 FROM Flights, Australia A
 WHERE
       et_ts between '2020-06-01 2:30:00' and '2020-06-01
       4:30:00' AND ST_intersects(A.AustEnv,geom)
) as Subq1
```

The result looks like this:



Exercise 4. MobilityDB Data Visualization

We now create the MobilityDB database for flight trajectories. We first create a geometry point. We did this in the SELECT clause of the QGIS query in Exercise 3.1.

4.1. Airframe trajectories

Each "icao24" field in the dataset represents a single airplane. We create a composite index on icao24 (unique to each plane) and et_ts (timestamps of observations) to help improving the performance of the trajectory generation.

```
CREATE INDEX icao24_time_index ON Flights (icao24, et_ts);
```

We first create trajectories for a single airframe (the plane itself), later on we create another trajectory table for flights (that is, a single aircraft is used for different flights in the same day, and this will be identified by icao24, CallSign

```
CREATE TABLE airframe_traj(icao24, trip, velocity, heading,
vertrate, callsign, alert, geoaltitude) AS (
SELECT icao24, tgeompointseq(array_agg
(tgeompoint(geom,et_ts) ORDER BY et_ts)
FILTER (WHERE geom IS NOT NULL)),
tfloatseq(array_agg(tfloat(velocity, et_ts)
ORDER BY et_ts)
FILTER (WHERE velocity IS NOT NULL)),
tfloatseq(array_agg(tfloat(heading, et_ts) ORDER BY et_ts)
FILTER (WHERE heading IS NOT NULL)),
tfloatseq(array_agg(tfloat(vertrate, et_ts) ORDER BY et_ts)
FILTER (WHERE vertrate IS NOT NULL)),
ttextseq(array_agg(ttext(callsign, et_ts) ORDER BY et_ts)
FILTER (WHERE callsign IS NOT NULL)),
tboolseq(array_agg(tbool(alert, et_ts) ORDER BY et_ts)
FILTER (WHERE alert IS NOT NULL)),
tfloatseq(array_agg(tfloat(geoaltitude, et_ts) ORDER BY
et_ts) FILTER (WHERE geoaltitude IS NOT NULL))
FROM Flights
GROUP BY icao24);
```

This is similar to what we did with AIS data (Assignment 2), that is:

- `tgeompoint`: Combines each geometry point(lat, long) with the timestamp where that point existed
- `array_agg`: aggregates all the instants together into a single array for each item in the group by. In this case, it will create an array for each icao24
- `tgeompointseq`: Constructs the array as a sequence which can be manipulated with mobilityDB functionality. In general, *tbaseseq* constructs a temporal base type from an array of sequences of a temporal base type *tbase.* For example, we build a temporal integer type *tint* invoking the corresponding constructor *tint.* With this, we build the sequence.

4.2. Flight trajectories

Now we have, in a single row, an entire day's trip information for each physical airplane. We will now partition this information per flight (an airframe flying under a specific CallSign such as BRH720). The following query segments the airframe trajectories (in temporal columns) based on the time period of the CallSign. This is the table we will use for mobility analysis.

```
CREATE TABLE Flight(ICAO24, CallSign, FlightPeriod, Trip,
  Velocity, Heading, VertRate, Alert, GeoAltitude) AS
SELECT ICAO24,
  (rec).value AS CallSign,
  (rec).time AS
  FlightPeriod,
  atTime(Trip, (rec).time),
  atTime(Velocity, (rec).time),
  atTime(Heading, (rec).time),
  atTime(VertRate,(rec).time),
  atTime(Alert, (rec).time),
  atTime(GeoAltitude, (rec).time)
FROM airframe_traj f, unnest(f.CallSign) rec;
```

In the query above, for each row *f* in *airframe_traj*, we apply the unnest function to the temporal text value f.*CallSign*, which gives us a table of records *rec* composed of a first column *value* and a second column *time*.

We then apply the *atTime* function to the temporal values in the table *airframe_traj* to restrict the function to the times with the same *CallSign* value.

MobilityDB functions help us avoid the use of several hardcoded conditions that depend on user knowledge of the data. This approach is very generic and can be applied anytime we want to split a trajectory by the inflection points in time of some other trajectory.

4.2.1. Compute the average velocity per flight

The query which computes this is:

```
SELECT icao24, callsign, twavg(velocity) AS
average_velocity
FROM Flight
WHERE twavg(velocity)IS NOT NULL
   AND twavg(velocity) < 1500 -- removes erroneous data
ORDER BY twavg(velocity) desc;
```

`Twavg` computes the time-weighted average of a temporal value.

4.2.2. Flights taking-off in Australia in some (user-defined) time interval.

First, a `TimeInterval` CTE will be used to clip all the temporal columns to a user-specified time range. AscSpan will be used to obtain the ascending flights (vertrate between 1 and 50)

```
-- Bounding box around Australia
WITH Australia(AustEnv) AS (
  SELECT ST_MakeEnvelope(113.338953078, -43.6345972634,
         153.569469029, -10.6681857235, 4326)),
-- Span for determining ascending planes
AscSpan(Span) AS (SELECT floatspan '[1,50]' ),
-- Time period we are interested in
TimeInterval(Period) AS (
  SELECT tstzspan '[2020-06-01 08:00:00, 2020-06-01
         09:00:00)'),
-- Planes over Australia in the given time period
AUFlight(ICAO24, CallSign, RestFlight, RestGeoAlt,
         RestVertRate) AS (
  SELECT ICAO24, CallSign, atTime(Trip, Period),
         atTime(GeoAltitude, Period),
         atTime(VertRate, Period)
  FROM Flight, Australia, TimeInterval
  WHERE atTime(Trip, Period) IS NOT NULL AND
        ST_Intersects(AustEnv, trajectory(atTime(Trip,
        Period))) ),
-- Ascending planes
AUFlightAscent(ICAO24, CallSign, AscTrip, AscVertRate) AS (
  SELECT ICAO24, CallSign,
         atTime(RestFlight, timeSpan(sequenceN(atValues
```

```
          (RestVertRate, Span), 1))),
          atTime(RestVertRate, timeSpan(sequenceN(atValues
        (RestVertRate, Span), 1)))
  FROM AUFlight, AscSpan
  WHERE atValues(RestVertRate, Span) IS NOT NULL)
-- Result
SELECT ICAO24, CallSign, trajectory(AscTrip) AS Geom
FROM AUFlightAscent, Australia
WHERE atGeometry(AscTrip, AustEnv) IS NOT NULL;
```

In the CTE above:

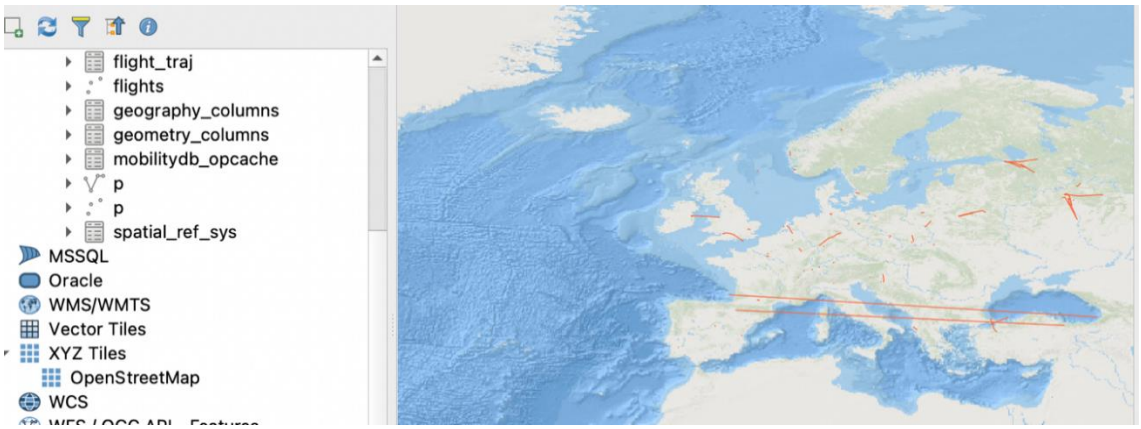
1. **atTime:** Clips the temporal data to create ranges where the vertrate was between '[1, 50]'. This vertrate means an aircraft was ascending.

2. **sequenceN:** Selects the first sequence (because N = 1) from the generated sequences.

This first sequence is the take off and eliminates mid-flight ascents.

3. **timeSpan:** Returns the part of the temporal values corresponding to the specified period.

4.2.3. Show in QGIS the result of the previous query



Exercise 5. Write the following queries and display their result in QGIS

5.1. *Trajectories of the flights of the aircraft '000001'.*

```
SELECT ROW_NUMBER()  OVER() as cid, icao24,callsign, traj
FROM (
SELECT icao24, callsign, trajectory(trip) AS traj
FROM Flight
WHERE icao24='000001' and callsign > 'A'   AND
      ST_GeometryType(trajectory(trip)) = 'ST_LineString'
) AS subq1
```

5.2. *Compute the flights over Australia between 8:00 a.m and 9:00 a.m. on June 1st, 2020.*

```
WITH Australia (AustEnv) AS (
  SELECT ST_MakeEnvelope(113.338953078, -43.6345972634,
153.569469029, -10.6681857235, 4326)),
TimeInterval(Period) AS (
  SELECT tstzspan '[2020-06-01 08:00:00, 2020-06-01
         09:00:00)')
SELECT ICAO24, CallSign, trajectory (atGeometryTime(Trip,
       AustEnv, Period)) AS Traj
FROM Flight, Australia, TimeInterval
WHERE eIntersects(Trip, AustEnv) IS NOT NULL AND
      atGeometryTime(Trip, AustEnv, Period) IS NOT NULL;
```

5.3. *Compute the trips between Auckland, Melbourne, and Sydney, and show the results in QGIS.*

```
WITH Cities (Sydney, Auckland, Melbourne) AS (
  SELECT ST_MakeEnvelope(150.3, -34.8, 151.8, -33.0, 4326),
    ST_MakeEnvelope(174.0, -37.67, 176.17, -36.12, 4326),
    ST_MakeEnvelope(143.9479, -38.8696, 146.3209, -37.0898,
                    4326)),
```

```
AuckSyd AS (
  SELECT ICAO24, CallSign, Trip, VertRate, Velocity,
         GeoAltitude, trajectory(Trip) AS Traj, Sydney,
         Auckland
  FROM Cities c, Flight f
  WHERE ST_Intersects(trajectory(f.Trip), c.Sydney) AND
        ST_Intersects(trajectory(f.Trip), c.Auckland) ),
AuckMel AS (
  SELECT ICAO24, CallSign, Trip, VertRate, Velocity,
         GeoAltitude, trajectory(Trip) AS Traj, Sydney,
         Auckland
  FROM Cities c, Flights f
  WHERE ST_Intersects(trajectory(f.Trip), c.Melbourne) AND
        ST_Intersects(trajectory(f.Trip), c.Auckland) )
SELECT *
FROM AuckMel
UNION
SELECT *
FROM AuckSyd;
```

5.4. Duration of all flights over Australia, showing the trajectory, the number of points in the trajectory, and the duration computed in two different ways: using `timespan` and using `duration`.

```
WITH

Australia AS (SELECT ST_Transform(ST_makeEnvelope(-1758447,
-4666808,2281883, -1535490,3112), 4326) AS Australia)

SELECT icao24, CallSign,

      atGeometry(trip,australia)::tstzspan,

      numtimestamps(atGeometry(trip,australia)),

      duration(atGeometry(trip,australia)),

      timespan(atGeometry(trip,australia)),

      trajectory(atGeometry(trip,australia))

FROM Flight, Australia

WHERE eintersects(trip,australia) IS NOT NULL AND

      atGeometry(trip,australia) IS NOT NULL
```

5.5. *Compute the distance between two planes at all instants and visualize the results in QGIS. Use planes with icao24='06a1bc' and icao24='040039'.*

```
WITH mindist AS(
      SELECT transform(s1.trip, 3112) <->
             transform(s2.trip,3112) AS distance
      FROM Flight s1, Flight s2
      WHERE s1.icao24 > s2.icao24 AND
            atMin(transform(s1.trip,3112) <->
                  transform(s2.trip,3112)) IS NOT NULL AND
            s1.icao24='06a1bc' AND s2.icao24='040039')
SELECT startTimestamp(unnest(instants(distance))) as time,
      getValue(unnest(instants(distance)))/1000 AS distance
FROM mindist
```

transform(s1.trip, 3112) transforms the geometries in the trip to the SRID of Australia.

<-> returns the distance between two moving points at all timestamps in the trajectory

atMin(s1.trip <-> s2.trip) returns the minimum distance between two moving objects and the moment when that occurred

instants(distance) converts the sequence into an array that can later be unnested to build the series.