Activity 3: 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 the link above https://opensky-network.org/datasets/states/2020-06-01/

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

```
CREATE TABLE FlightsInput(

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 FlightsInput (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 also run the following script which iterates over the 23 files and executes dynamic SQL statements:

If you are using Docker:

```
docker cp "/your path/."   mobilitydb:/tmp
```

```
DO
$$DECLARE
prefixpath text= 'your-path/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 FlightsInput
ADD COLUMN EtTs TIMESTAMP,
ADD COLUMN LastPosUpdateTs TIMESTAMP,
ADD COLUMN LastContactTs TIMESTAMP;
UPDATE FlightsInput SET
  EtTs = to_timestamp(Et),
  LastPosUpdateTs = to_timestamp(LastPosUpdate),
  LastContactTs = to_timestamp(LastContact);
```

Check the size of the database with:

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

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 FlightsInput
GROUP BY icao24
HAVING COUNT(lat) = 0
)

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


CREATE INDEX icao24_time_index ON

             FlightsInput (ICAO24,  EtTs);


DELETE FROM FlightsInput WHERE Squawk IS NULL;
```

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).

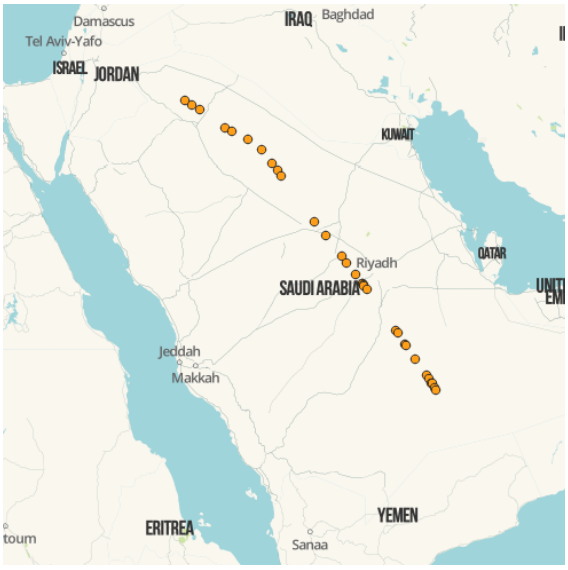
Exercise 3. Raw data visualization using QGIS

We now visualize the raw data using different tools. For example, we visualize the points of a single flight 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 (
  SELECT etts, icao24, ST_MAKEPOINT(lat, lon) AS geom
-- TABLESAMPLE SYSTEM (n) returns only n% of the data from
-- the table.
FROM FlightsInput TABLESAMPLE SYSTEM (5)
WHERE icao24 IN ('738286') AND
      etts between '2020-06-01 2:30:00' and '2020-06-01
      4:30:00') as Subq1
```

The results looks like this:



Exercise 4. Mobility Database Creation

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. This treats each latitude and longitude as a point in space. 4326 is the SRID.

```
ALTER TABLE FlightsInput
ADD COLUMN geom geometry(Point, 4326);
UPDATE FlightsInput SET
       geom = ST_SetSRID( ST_MakePoint( lon, lat ), 4326);
```

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 FlightsInput(icao24,
                                                    etts);
```

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  FlightDay(ICAO24,  Trip,  Velocity,  Heading,
VertRate, CallSign, Alert, GeoAltitude) AS (

SELECT ICAO24,

tgeompointSeq(array_agg(tgeompoint(Geom,EtTs)

  ORDER BY  EtTs)

  FILTER (WHERE Geom IS NOT NULL)),

tfloatSeq(array_agg(tfloat(Velocity, EtTs) ORDER BY EtTs)

  FILTER (WHERE Velocity IS NOT NULL)),

tfloatSeq(array_agg(tfloat(Heading, EtTs) ORDER BY EtTs)

  FILTER (WHERE Heading IS NOT NULL)),

tfloatSeq(array_agg(tfloat(VertRate, EtTs) ORDER BY EtTs)

  FILTER (WHERE VertRate IS NOT NULL)),

ttextSeq(array_agg(ttext(CallSign, EtTs) ORDER BY EtTs)
```

```
  FILTER (WHERE CallSign IS NOT NULL)),
tboolSeq(array_agg(tbool(Alert, EtTs) ORDER BY EtTs)
  FILTER (WHERE Alert IS NOT NULL)),
tfloatSeq(array_agg(tfloat(GeoAltitude, EtTs) ORDER BY EtTs)
  FILTER (WHERE GeoAltitude IS NOT NULL))
FROM FlightsInput
GROUP BY ICAO24);
```

This is 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. With this, we build the sequence.

4.2. Flight trajectories

In table `FlightDay` we have, in a single row, an airframe's (where an airframe is a single physical airplane) entire day's trip information.

For example, if we write

```
SELECT icao24,
    startValue(unnest(segments(callsign))) AS
    start_value_callsign,
    unnest(segments(callsign))::tstzspan AS
    callsign_segment_period
FROM FlightDay
WHERE icao24='a6e0dc'
```

We obtain

	icao24 character varying (20)	start_value_callsign text	callsign_segment_period tstzspan
1	a6e0dc	SCX3001	[2020-06-01 00:03:40+00, 2020-06-01 03:10:00+00)
2	a6e0dc	SCX4001	[2020-06-01 03:10:00+00, 2020-06-01 16:16:00+00)
3	a6e0dc	SCX3005	[2020-06-01 16:16:00+00, 2020-06-01 22:37:00+00)
4	a6e0dc	SCX3001	[2020-06-01 22:37:00+00, 2020-06-01 23:55:30+00]

That is, the **icao24='a6e0dc'** had the same callsign 'SCX3001' in two different time instants

```
[2020-06-01 00:03:40+00, 2020-06-01 03:10:00+00)
```

and

```
[2020-06-01 22:37:00+00, 2020-06-01 23:55:30+00]
```

In the same day, one flight took 3hours 6min 20 secs to complete, and the other one took 1hour, 18min 30secs.

```
SELECT duration('[2020-06-01 00:03:40+00, 2020-06-01 03:10:00+00)'::tstzspan)

SELECT duration('[2020-06-01 22:37:00+00, 2020-06-01 23:55:30+00]'::tstzspan)
```

These are two different flights, so we must split them.

Note: We could have tried to create the table "flight_traj" by simply including "callsign" in the GROUP BY statement in the query used to create the previous airframe_traj table (GROUP BY icao24, callsign).

The problem with this solution is that it would put the trajectory data of two distinct flights where the airplane and flight number are the same in a single row, which is not correct.

We will now partition this information per flight (**an airframe flying under a specific callsign**). The following query segments the airframe trajectories (in temporal columns) based on the time period of the callsign. Below we show and explain the query.

```sql
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 FlightDay f, unnest(f.CallSign) AS Rec;
```

Exercise 5. Querying the Mobility Database

5.1. Compute the average velocity per flight

The query which computes this is:

```
SELECT callsign,twavg(velocity) AS average_velocity
FROM Flight
WHERE twavg(velocity)IS NOT NULL AND /*drops rows
                                     without velocity data*/
      twavg(velocity) < 1500  -- removes erroneous data
ORDER BY twavg(velocity) desc;
```

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

5.2. Compute the flights taking-off in Australia between 8 a.m. and 9 a.m.

```
-- 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;
```

The first CTE builds a MBB for Australia. The `AscSpan` and `TimeInterval` CTEs are used to state the value and time spans, respectively, that are used in the query to extract the portions of interest of the flight. The CTE `AUFlight` restricts the spatiotemporal `Trip`, the `GeoAltitude` and the vertical speed `VertRate` to the selected time and vertrate intervals respectively; the conditions in the WHERE clause guarantee that only trips in the required zone and time intervals are considered in the computation. Finally, the `AUFlightAscent` CTE selects the take-off portions of the flight and the `GeoAltitude` and `VertRate` during such segments. In the expression

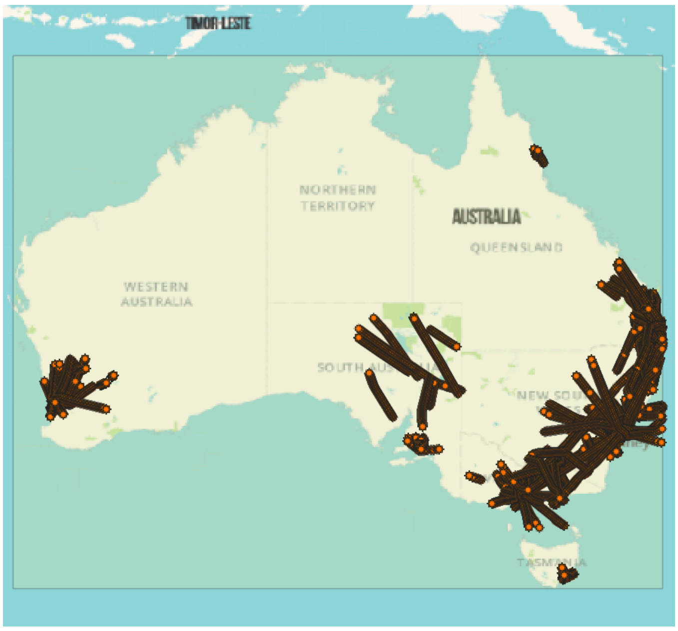
```
atTime(RestFlight,
       timeSpan(sequenceN(atValues(RestVertRate,Span),1)))
```

the `atValues` function restricts the function to the instants where the `VertRate` parameter is between 1 and 50. Since this returns a sequence set, `sequenceN` obtains the first sequence, and `tstzspan` retrieves the time interval of this element. Finally, `atTime` obtains the take-off part of the trip. The final query clips the result to the Australian territory using the `atGeometry` function and finally we compute the spatial projection `(trajectory(ascendingTrip))`.

5.3. Compute the flights over Australia between 8:00 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, 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;
```

In the query above, we start defining in the first CTE a MBB that corresponds to Australia and in the second CTE the time period in which we are interested. In the SELECT clause, `atGeometryTime` restricts the trip to Australia's MBB and to the period indicated in the query. This function is a shorthand for the combination of the functions `atGeometry` and `atTime`. Finally, the spatial projection is computed with the function `trajectory`. In the WHERE clause, we ensure that trip actually intersected Australia's MBB and the specified time period. The result looks in QGIS:



Exercise 6. Write the following queries and display their result in QGIS .

6.1. 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
```

6.2. Altitude of aircraft icao24='06a0af' during take-off in the interval 2020-06-01 03:00:00, 2020-06-01 04:00:00.

This query allows reviewing important functions

```
WITH
flight_traj_time_slice (icao24, callsign, time_slice_trip, time_slice_geoaltitude,
time_slice_vertrate) AS(
SELECT icao24,callsign,
   atTime(trip, '[2020-06-01 03:00:00, 2020-06-01
              04:00:00)'::tstzspan),
   atTime(geoaltitude, '[2020-06-01 03:00:00, 2020-06-01 04:00:00)'::tstzspan),
  atTime(vertrate, '[2020-06-01 03:00:00, 2020-06-01 04:00:00)'::tstzspan)
FROM Flight
WHERE  icao24='0100a3'  AND atTime(trip, '[2020-06-01 03:00:00, 2020-06-01
                                   04:00:00)'::tstzspan)  IS NOT NULL
 ),
-- ASCENDING PLANES
flight_traj_time_slice_ascent(icao24, callsign, ascending_trip,
ascending_geoaltitude, ascending_vertrate) AS
(SELECT icao24, callsign,
  atTime(time_slice_trip, sequenceN(atValues(time_slice_vertrate,
```

```
                                    '[1,20]'::floatspan), 1)::tstzspan),
 atTime(time_slice_geoaltitude, sequenceN(atValues(time_slice_vertrate,
                                           '[1,20]'::floatspan),1)::tstzspan),
 atTime(time_slice_vertrate, sequenceN(atValues(time_slice_vertrate,
                                         '[1,20]'::floatspan), 1)::tstzspan)
FROM flight_traj_time_slice
WHERE
atTime(time_slice_trip, sequenceN(atValues(time_slice_vertrate,
                                   '[1,20]'::floatspan), 1)::tstzspan) IS NOT NULL),
StatsTable AS(
  SELECT icao24, callsign, unnest(instants(ascending_geoaltitude)),
         startTimestamp(unnest(instants(ascending_geoaltitude))) AS timeasc,
         getValue(unnest(instants(ascending_geoaltitude))) AS altitude
-- instants return the (value,time) pairs as an array
-- unnest them as single value@time
-- getValue gets the value
-- startTimestamp obtains the initial timestamp (in this case, the only one)
-- startValue obtains the initial value (=getValue for a simple element)
FROM flight_traj_time_slice_ascent)
SELECT timeasc,altitude
FROM StatsTable
```

6.3. 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, 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
```

6.4. Duration of flight icao24= 'c0175a' over Australia, together with the distance travelled within the country.

```
WITH
Australia AS (SELECT ST_Transform(ST_makeEnvelope
         (-1758447, -4666808,2281883,-1535490,3112),4326) AS Australia)
SELECT icao24,length(transform((atGeometry(trip,australia)), 3112)),
      st_length(st_transform(trajectory(atGeometry(trip,australia))::geometry,
      3112)),
      atGeometry(trip,australia)::tstzspan,
      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 AND  icao24='c0175a'
```

6.5. 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.

6.6. Compute the aircraft flying at altitudes between three and eight thousand meters

```
WITH TimeAltitude(Period,AltSpan) AS (
 SELECT tstzspan '[2020-06-01 08:00:00, 2020-06-01 09:00:00)',
  floatspan '[3000,8000)' ),

FlightTimeSlice(ICAO24, CallSign, TripTimeSlice, AltitudeTimeSlice) AS (
 SELECT ICAO24, CallSign, atTime(Trip, Period), atTime(GeoAltitude, Period)
 FROM Flight, TimeAltitude
 WHERE atTime(Trip, Period) IS NOT NULL),

FlightTimeSliceCruising(ICAO24, CallSign, CruisingTrip,
  CruisingGeoAltitude) AS (
SELECT ICAO24, CallSign,
   atTime(TripTimeSlice, timeSpan(sequenceN(atValues(AltitudeTimeSlice,
   AltSpan), 1))), atTime(AltitudeTimeSlice, timeSpan(sequenceN(
   atValues(AltitudeTimeSlice, AltSpan), 1)))
 FROM FlightTimeSlice, TimeAltitude
 WHERE atTime(TripTimeSlice, timeSpan(sequenceN(atValues(AltitudeTimeSlice,
           AltSpan), 1))) IS NOT NULL ),

FinalOutput AS (
SELECT ICAO24, CallSign,
getValue(unnest(instants(CruisingGeoAltitude))) AS GeoAltitude,
ST_X(getValue(unnest(instants(CruisingTrip)))) AS Lon,
ST_Y(getValue(unnest(instants(CruisingTrip)))) AS Lat
FROM FlightTimeSliceCruising)

SELECT *
FROM FinalOutput
WHERE GeoAltitude IS NOT NULL;
```

6.7. Compute the flight landing in the world between 8 a.m. and 10 a.m.

```
-- Span for determining descending planes
WITH DescSpan(Span) AS ( SELECT floatspan '[-20,0]' ),

-- Time period we are interested in
TimeInterval(Period) AS (
 SELECT tstzspan '[2020-06-01 08:00:00, 2020-06-01 10:00:00)' ),

-- Planes in the given time period
WorldFlight(ICAO24, CallSign, RestFlight, RestGeoAlt, RestVertRate) AS (
 SELECT ICAO24, CallSign, atTime(Trip, Period), atTime(GeoAltitude, Period),
```

```
          atTime(VertRate, Period)
 FROM Flight, TimeInterval
 WHERE atTime(Trip, Period) IS NOT NULL ),

-- Descending planes
WorldFlightDescent(ICAO24, CallSign, RestGeoAlt, DescTrip, RestVertRate) AS (
  SELECT ICAO24, CallSign, RestGeoAlt, atTime(RestFlight, timeSpan(
            sequenceN(atValues(RestVertRate, Span), 1))),
            atTime(RestVertRate, timeSpan(sequenceN(atValues(RestVertRate,
            Span),  1)))
  FROM WorldFlight, DescSpan
  WHERE atValues(RestVertRate, Span) IS NOT NULL ),

FinalOutput AS (
  SELECT ICAO24, CallSign, getValue(unnest(instants(RestGeoAlt))) AS
            GeoAltitude, getValue(unnest(instants(RestVertRate))) AS VertRate,
   ST_X(getValue(unnest(instants(DescTrip)))) AS Lon,
   ST_Y(getValue(unnest(instants(DescTrip)))) AS Lat
 FROM WorldFlightDescent)

SELECT ICAO24, CallSign, GeoAltitude, VertRate, Lon, Lat
FROM FinalOutput
WHERE VertRate IS NOT NULL AND GeoAltitude IS NOT NULL AND
         GeoAltitude < 1000;
```