Aircraft Trajectories in PostGIS

Benjamin Trigona-Harany

Personal: bosth@sfu.ca | Professional: benjamin@planet.com

FOSS4G
Prizren, 2023

Flight Info

Search Flight Number, Airline or City

ARRIVALS **DEPARTURES**

	FLIGHT	AIRLINE	FROM	DATE	SCHEDULED	ETA	STATUS
chair	GM602	CHAIR AIRLINES	Zurich	29 June	18:55		SCHEDULED
TURKISH AIRLINES	TK1019	TURKISH AIRLINES	Istanbul	29 June	19:30		SCHEDULED
easyJet	EZS1495	EASYJET	Geneve	29 June	19:50	20:07	EXPECTED 20:07
enter air	E46061	ENTER AIR	Stuttgart	29 June	20:05		SCHEDULED
easyJet	EZS1209	EASYJET	Basel - Mulhouse	29 June	20:20		SCHEDULED
Eurowings	EW5704	EUROWINGS	Hamburg	29 June	20:40		SCHEDULED

News

"GLOBAL ENGIN
GIRLS" IN KOSO

PRISTINA, 22 Decem
Limak Kosovo launc
Engineer Girls (GEG

HOW CAN WE MODEL THESE FLIGHTS IN A POSTGIS DATABASE?

Proposal 1: Data model

```
CREATE TABLE flight (
  callsign TEXT,
  airport_depart TEXT,
  airport_arrive TEXT,
  time_depart TIMESTAMP,
  time_arrive TIMESTAMP,
  geom GEOMETRY(LINESTRING, 4326)
);
```

Proposal 1: Data

```
INSERT INTO flight VALUES (
   'TK1019',
   'LTFM', -- ICAO code for Istanbul (IATA code: IST)
   'BKPR', -- ICAO code for Pristina (IATA code: PRN)
   '2023-06-29T1900+3',
   '2023-06-29T1930+2',
   'LINESTRING(41.2768 28.7300, ...,
               42.5746 21.0295)'
);
```

Proposal 1: Queries

```
SELECT
    ST_StartPoint(geom) AS origin,
    ST_EndPoint(geom) AS destination
FROM flight;
```

Get start and end points of each flight

```
SELECT
  ST_Length(geom::geography) / 1000 AS distance
FROM flight;
```

Get the distance travelled by each flight

```
SELECT
  f.callsign, string_agg(c.name, ', ')
FROM
  flight f, country c
WHERE
  ST_Intersects(f.geom, c.geom)
GROUP BY f.callsign;
```

Get a list of countries traversed by each flight

WHAT HAPPENS WHEN YOU TRAVEL BY PLANE IN TWO DIMENSIONS?


Ouest
Montréal

Proposal 2: Data model

```
CREATE TABLE flight (
  callsign TEXT,
  airport_depart TEXT,
  airport_arrive TEXT,
  time_depart TIMESTAMP,
  time_arrive TIMESTAMP,
  geom GEOMETRY(LINESTRINGZ, 4326) -- 3D vertices
);
```

Proposal 2: Data

```
INSERT INTO flight VALUES (
   'TK1019',
   'LTFM',
   'BKPR',
   '2023-06-29T1900+3',
   '2023-06-29T1930+2',
   'LINESTRING(41.2768 28.7300  99, ...,
               42.5746 21.0295 545)'
);
```

Proposal 2: Queries

```
SELECT
    ST_ZMax(geom) AS max_altitude,
FROM flight;
```

Get the highest altitude that each flight reached

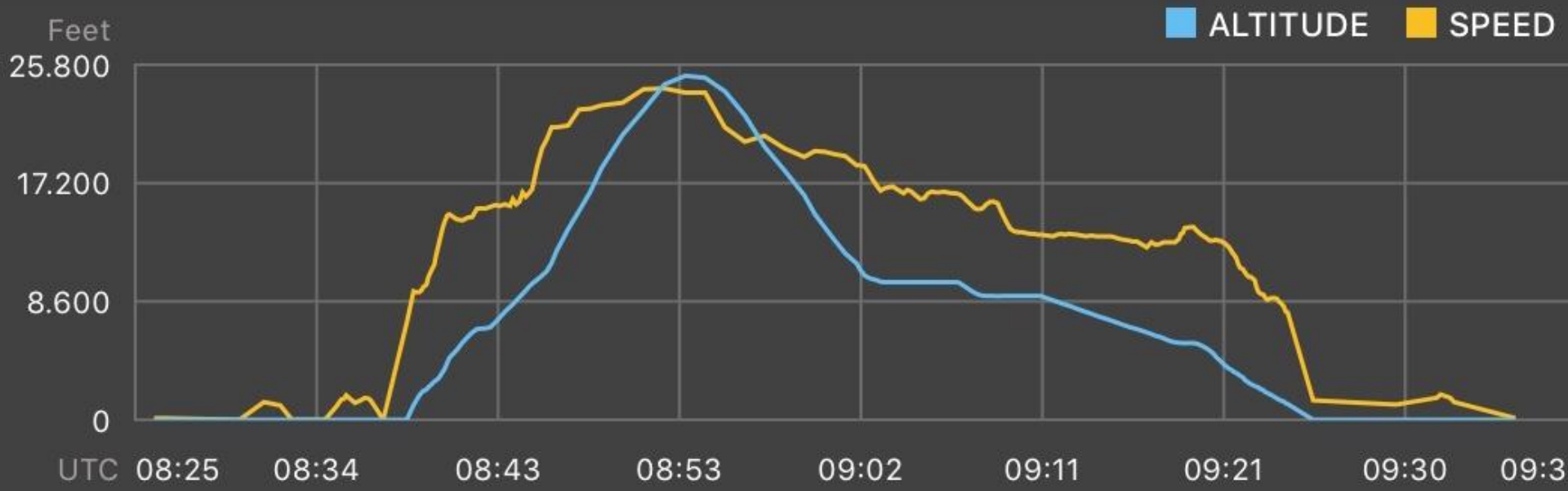
```
SELECT
  ST_3DClosestPoint(
    geom,
    ST_MakePoint(42.689, 23.414, 531)
  ) AS closest_point
FROM flight;
```

Get the point at which each flight was closest to Sofia airport*

* This comparison is using mixed units so will produce improper results.

CAN WE DO BETTER?

flightradar24



TIME	SPEED	TRACK	ALTITUDE
08:25 UTC	2 KTS	176°	0 FT

Proposal 3: Data model

```
CREATE TABLE flight (
  callsign TEXT,
  airport_depart TEXT,
  airport_arrive TEXT,
  time_depart TIMESTAMP,
  time_arrive TIMESTAMP,
  geom GEOMETRY(LINESTRINGZM, 4326) -- 4D vertices
);
```

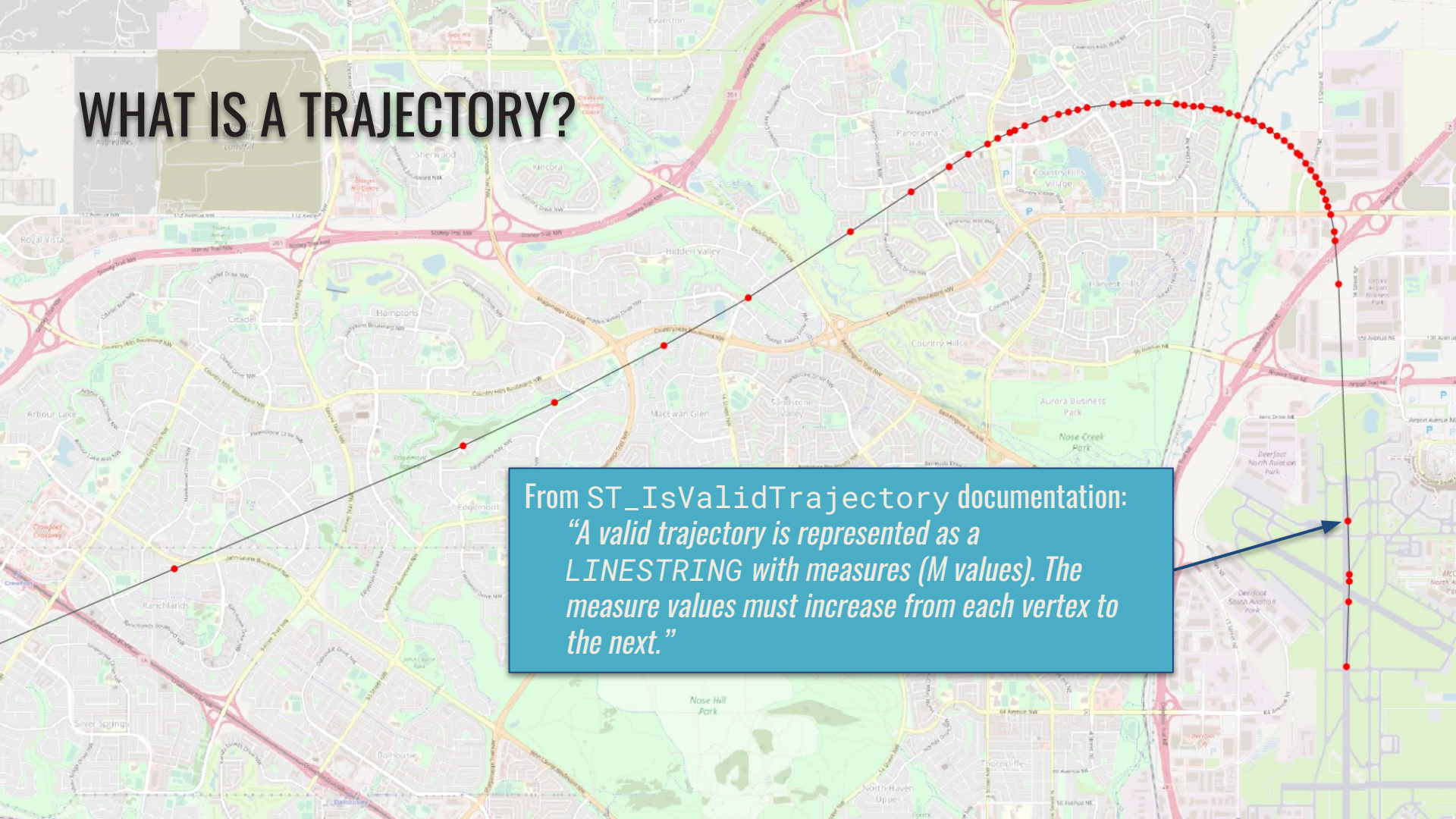
Proposal 3: Data

```
INSERT INTO flight VALUES (
   'TK1019',
   'LTFM',
   'BKPR',
   '2023-06-29T1900+3',
   '2023-06-29T1930+2',
   'LINESTRING(41.2768 28.7300  99 1688054400, ...'
               42.5746 21.0295 545 1688059800)'
);
```

The “m value” in this example is stored using Unix Time, but this is not mandatory.

THESE ARE TRAJECTORIES

WHAT IS A TRAJECTORY?



From `ST_IsValidTrajectory` documentation:

"A valid trajectory is represented as a `LINESTRING` with measures (M values). The measure values must increase from each vertex to the next."

WHERE CAN WE GET DATA?

ADS-B

Most aircraft are continuously broadcasting packets of data using the **Automatic Dependent Surveillance–Broadcast** protocol. Packets are unencrypted and a cheap receiver can pick up broadcasts from local aircraft.

Packets contain data such as callsign (e.g. TK1019), speed, **longitude**, **latitude**, **altitude** and **a time stamp**.

Each aircraft has a 6-digit hexadecimal number as a unique identifier, such as `3c6444`. These are known as ICAO 24-bit addresses and are also transmitted in ADS-B packets.

This is a registration number, **NOT** an `icao24` address!

WHAT IF I DON’T HAVE AN ADS-B RECEIVER?

Bringing up OpenSky: A large-scale ADS-B sensor network for research
Matthias Schäfer, Martin Strohmeier, Vincent Lenders, Ivan Martinovic, Matthias Wilhelm
ACM/IEEE International Conference on Information Processing in Sensor Networks, April 2014

https://www.opensky-network.org

RESTful API: https://opensky-network.org/api/...

OpenSky API: `/states/all`

Parameters

`icao24`	`optional`	One or more ICAO24 transponder addresses represented by a hex string (e.g. abc9f3). If omitted, the state vectors of all aircraft are returned.
`lamin`	`optional`	Lower bound for the latitude in decimal degrees
`lomin`	`optional`	Lower bound for the longitude in decimal degrees
`lamax`	`optional`	Upper bound for the latitude in decimal degrees
`lomax`	`optional`	Upper bound for the longitude in decimal degrees
`time`	`optional`	The time in seconds since epoch (Unix time stamp). Current time will be used if omitted.

Sample API calls

```
/states/all
/states/all?icao24=3c6444
/states/all?lamin=45.8389&lomin=5.9962&lamax=47.8229&lomax=10.5226
```

OpenSky API: `/states/all`

Response

JSON

State vector

```
{
    "time":1685812876,
    "states":[
        ["4b1814", "EDW58T", "Switzerland", 1685812818, 1685812818, 2.7859, 39.5669,
        137.16, false, 68.11, 238.59, -3.9, null, 198.12, "3030", false, 0, 0],
        ["880441", "AIQ394", "Thailand", 1685812866, 1685812869, 104.0786, 1.8492,
        10668, false, 234.88, 159.62, -0.33, null, 11369.04, null, false,0, 1],
        ...
    ]
}
```

REAL-TIME DATA IN POSTGIS

Create a (Foreign) Table

```
CREATE FOREIGN TABLE live_aircraft (
  icao24 TEXT,
  callsign TEXT,
  time TIMESTAMP,
  geom GEOMETRY,
  origin_country TEXT,
  true_track FLOAT,
  velocity FLOAT,
  category_text TEXT) SERVER opensky_api_states;
```

See https://github.com/bosth/geofdw

```
CREATE SERVER
   opensky_api_states
FOREIGN DATA WRAPPER
   multicorn OPTIONS (WRAPPER 'geofdw.fdw.opensky.StateVector');
```

Query a Foreign Table

```
SELECT
   callsign, origin_country, ST_AsText(geom)
FROM
   live_aircraft
WHERE
   icao24 = 'ab1644';
```

API call: `api/states/all?icao24=ab1644`

```
 callsign | origin_country |             st_astext
----------+----------------+-----------------------------------
 UAL2619  | United States  | POINT Z (-90.22 35.0806 10759.44)
(1 row)
```

Query a Foreign Table

```
SELECT
   callsign, icao24, origin_country, ST_AsText(geom)
FROM
   live_aircraft
WHERE
   geom && 'POLYGON((20.0 41.8, 20.0 43.2, 21.7 43.2, 21.8 41.8, 20.0 41.8))'::geometry;
```

```
 callsign | icao24 | origin_country |             st_astext
----------+--------+----------------+-------------------------------------
 EWG79EB  | 3c5ee1 | Germany        | POINT Z (21.1693 42.5809 12131.04)
 THY9GD   | 4baa86 | Turkey         | POINT Z (21.5869 42.7698 10660.38)
 TOM1KM   | 406ca3 | United Kingdom | POINT Z (20.0416 42.7306 10972.8)
 RYR1843  | 4d2224 | Malta          | POINT Z (21.7624 41.9092 11254.74)
 RYR63ZR  | 48c2a5 | Poland         | POINT Z (20.4638 42.9922 10957.56)
 EDW46Y   | 4b18b8 | Switzerland    | POINT Z (21.5209 41.8931 3177.54)
(6 rows)
```

API call: `api/states/all?lomin=20.0&lomax=41.8&lamin=21.8&lamax=43.2`

WHAT ABOUT TRAJECTORIES?

OpenSky API: `/flights/aircraft`

Parameters

`icao24`	`required`	ICAO24 transponder address represented by a hex string (e.g. abc9f3).
`begin`	`required`	Start of time interval to retrieve flights for as Unix time (seconds since epoch)
`end`	`required`	End of time interval to retrieve flights for as Unix time (seconds since epoch)

Sample API calls `/flights/aircraft?icao24=3c675a&begin=1517200000&end=1518000000`

OpenSky API: `/flights/aircraft`

Response

JSON


```
[
    {
        "icao24": "3c675a",
        "firstSeen": 1517258040,
        "estDepartureAirport": "EDDF",
        "lastSeen": 1517263900,
        "estArrivalAirport": "ESSA",
        "callsign": "DLH2VC  ",
        "estDepartureAirportHorizDistance": 1462,
        "estDepartureAirportVertDistance": 49,
        "estArrivalAirportHorizDistance": 7194,
        "estArrivalAirportVertDistance": 423,
        "departureAirportCandidatesCount": 1,
        "arrivalAirportCandidatesCount": 3
    }, {
        ...
    }
]
```

Proposal 4: Data model

```
CREATE TABLE flight (
  icao24 TEXT,
  callsign TEXT,
  airport_depart TEXT,
  airport_arrive TEXT,
  time_depart TIMESTAMP,
  time_arrive TIMESTAMP,
  geom GEOMETRY(LINESTRINGZM, 4326) -- 4D vertices
);

CREATE INDEX ON
  flight
USING
  gist (geom gist_geometry_ops_nd);
```

PL/PYTHON FUNCTION

```
CREATE OR REPLACE FUNCTION
    opensky_get_aircraft_flights(icao24 TEXT, datebegin DATE, dateend DATE)
RETURNS
    TABLE (LIKE flight)
AS $$
    # CODE TO CALL OPENSKY /flights/aircraft API HERE

    if response.status_code == 200:
        flights = [
                    (icao24,
                    f["callsign"],
                    f["estDepartureAirport"],
                    f["estArrivalAirport"],
                    datetime.fromtimestamp(f["firstSeen"]),
                    datetime.fromtimestamp(f["lastSeen"]),
                    None)
                   for f in response.json()]
        return flights
$$ LANGUAGE plpython3u;
```

POPULATE TABLE

```
INSERT INTO
  flight
SELECT
  *
FROM
  opensky_get_aircraft_flights('c05f01', '2023-06-01', '2023-06-07');
```

QUERY TABLE

```
SELECT callsign, geom FROM flight;

 callsign | geom
----------+------
 JZA70    |
 JZA69    |
 JZA21    |
 JZA664   |
 JZA663   |
 JZA660   |
 JZA660   |
 JZA659   |
 JZA7714  |
(9 rows)
```

OpenSky API: `/tracks`

Parameters

`icao24`	`required`	ICAO24 transponder address represented by a hex string (e.g. abc9f3).
`time`	`optional`	Any time between the start and end of a flight. Time is represented by seconds since epoch (Unix time stamp). Current time will be used if omitted. Data only available for the past 30 days.

Sample API calls

`/tracks?icao24=3c6444`

`/tracks?icao24=3c6444&time=1649693000`

OpenSky API: `/tracks`

Response

JSON

```
{
  "icao24": "3c6444",
  "callsign": "D-AIBD",
  "startTime": 1649692883,
  "endTime": 1649696435,
  "path": [
      [1649693075, 45.129, 2.631, 3352, 23, false],
      [1649693095, 45.191, 2.681, 3352, 23, false],
      ...
   ]
}
```

timestamp (m), latitude (y), longitude (x), altitude (z), heading, ground flag

PL/PYTHON FUNCTION

```
CREATE OR REPLACE FUNCTION
    opensky_get_track(icao24 TEXT, in_date TIMESTAMP WITH TIME ZONE)
RETURNS
    GEOMETRY(LINESTRINGZM, 4326)
AS $$
    from plpygis import LineString

    # CODE TO CALL OPENSKY /tracks API HERE
    if response.status_code == 200:
            track = response.json()
            return LineString([[v[2], v[1], v[3], v[0]] for v in track["path"]])
```

See https://github.com/bosth/plpygis

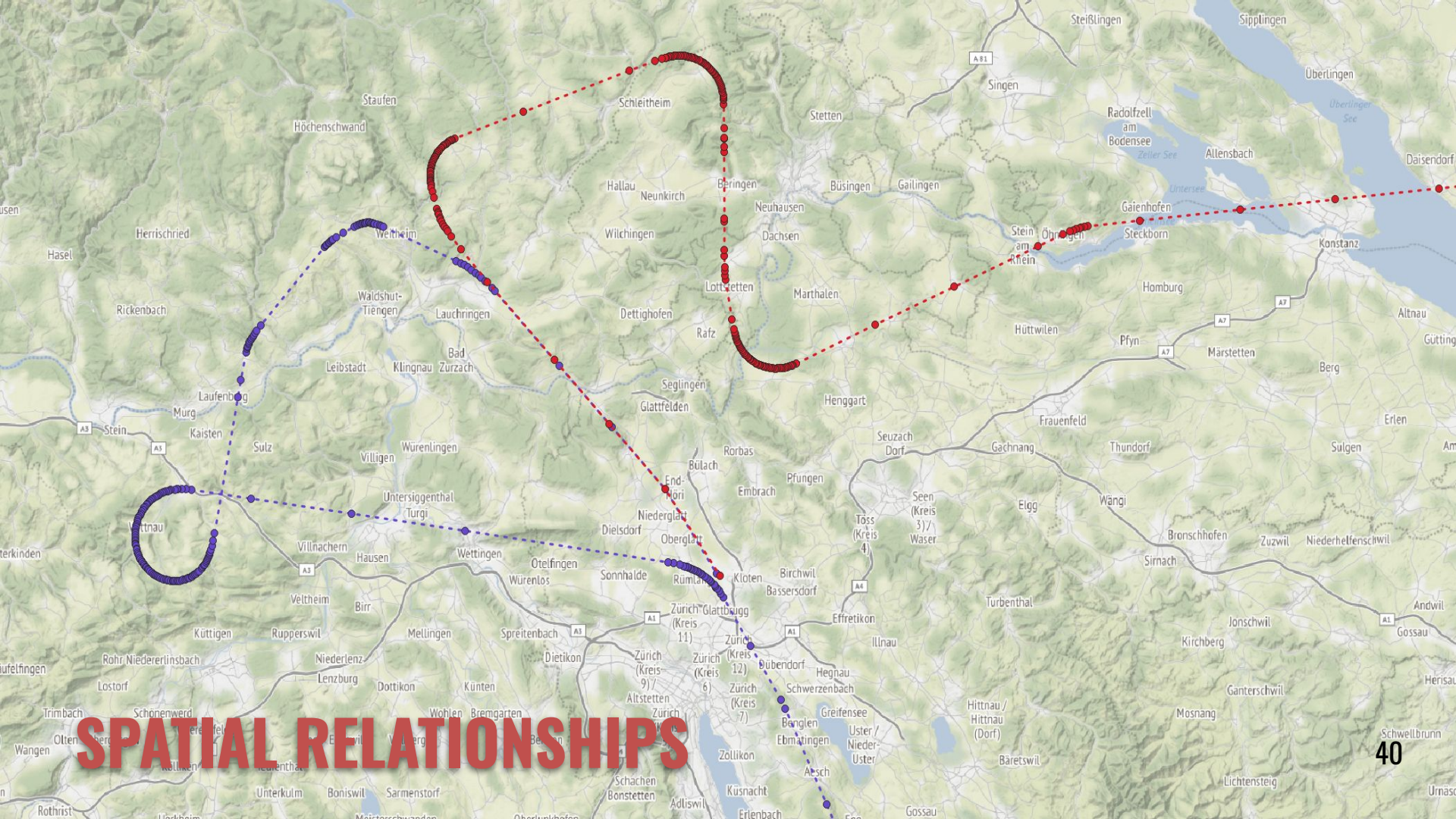
UPDATE TABLE WITH TRAJECTORIES

```
UPDATE
  flight
SET
  geom = opensky_get_track(icao24, time_depart)
WHERE
  geom IS NULL;
```

QUERY TABLE

```
SELECT
  callsign,
  ST_IsValidTrajectory(geom) AS valid,
  ST_AsText(ST_StartPoint(geom)) AS origin
FROM flight;

 callsign | valid |                    origin
----------+-------+-----------------------------------------------
 JZA70    | t     | POINT ZM (-75.6529 45.3092 0 1685832837)
 JZA69    | t     | POINT ZM (-69.9938 45.5988 7315 1685826302)
 JZA21    | t     | POINT ZM (-61.041 46.5555 6400 1685794547)
 JZA659   | t     | POINT ZM (-68.1527 44.235 7315 1685633460)
 JZA664   | t     | POINT ZM (-71.0031 42.3814 0 1685745859)
 JZA663   | t     | POINT ZM (-67.7245 44.1866 7315 1685738980)
 JZA660   | t     | POINT ZM (-74.1716 40.6878 0 1685725473)
 JZA660   | t     | POINT ZM (-74.1767 40.6798 0 1685642094)
 JZA7714  | t     | POINT ZM (-79.3967 43.6291 0 1685578999)
```



SPATIAL RELATIONSHIPS

INTERSECTIONS

```
SELECT
  a.callsign AS a,
  b.callsign AS b
FROM
  flight a, flight b
WHERE
  a.icao24 != b.icao24 AND
  a.callsign = 'CAI6KA' AND
  ST_Intersects(a.geom, b.geom);
```

Two-dimensional intersection

```
   a     |    b
--------+---------
 CAI6KA | CAI6KA
 CAI6KA | THY8MR
 CAI6KA | VJT929
 CAI6KA | ENT6062
 CAI6KA | JAV3821
 CAI6KA | NSZ2827
 CAI6KA | NSZ2827
 ...
(289 rows)
```

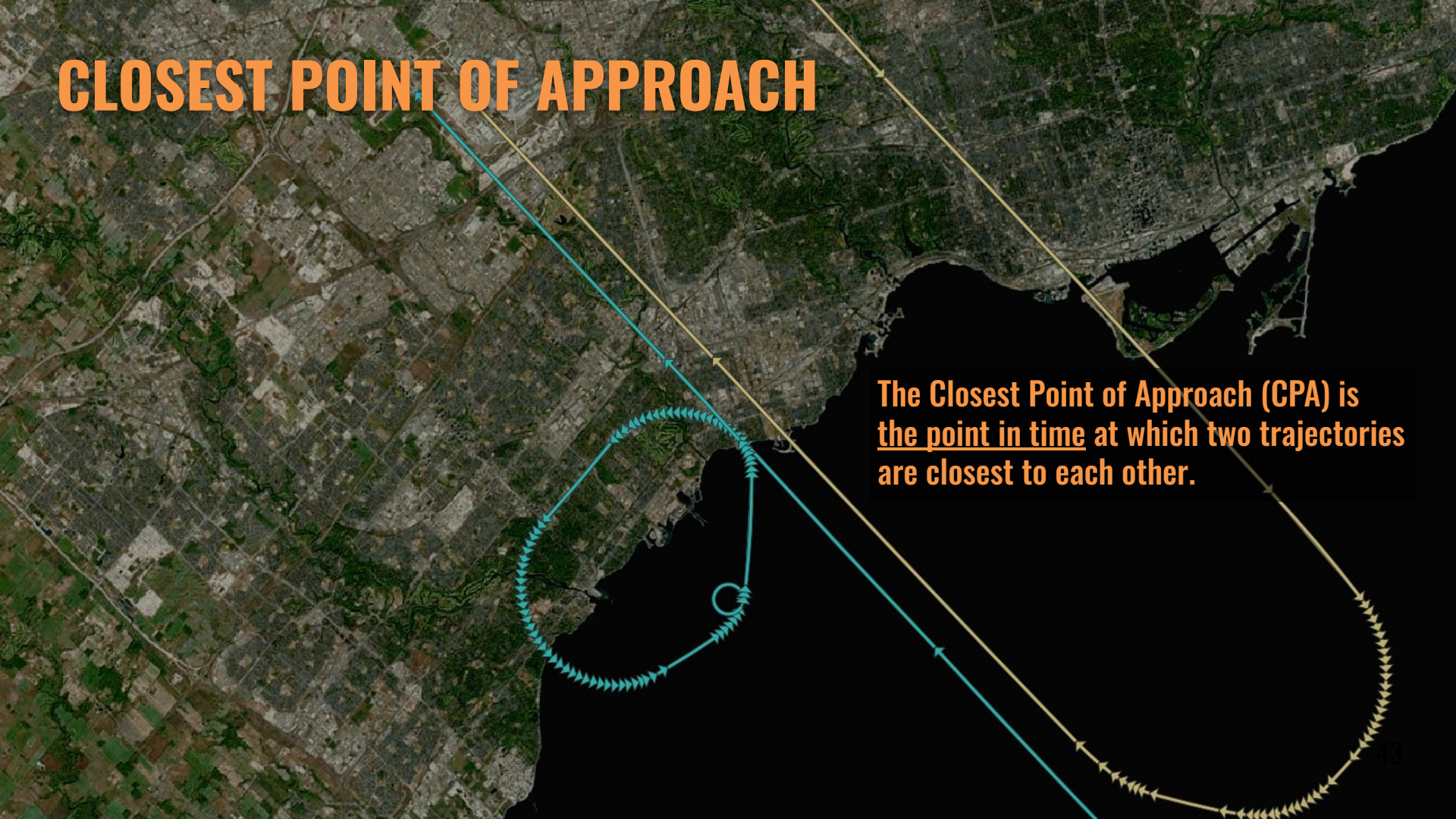
INTERSECTIONS (3D VERSION)

```
SELECT
  a.callsign AS a,
  b.callsign AS b
FROM
  flight a, flight b
WHERE
  a.icao24 != b.icao24 AND
  a.callsign = 'CAI6KA' AND
  ST_3dIntersects(a.geom, b.geom);
```

Three-dimensional intersection

```
   a    |   b
--------+--------
 CAI6KA | SXS4PH
(1 row)
```

CLOSEST POINT OF APPROACH



The Closest Point of Approach (CPA) is <u>the point in time</u> at which two trajectories are closest to each other.

CLOSEST POINT OF APPROACH

```
SELECT
  a.callsign AS a,
  b.callsign AS b,
  to_timestamp(ST_ClosestPointOfApproach(a.geom, b.geom)) AS cpa_time
FROM
  flight a, flight b
WHERE
  a.icao24 != b.icao24 AND
  a.callsign = 'CAI6KA' AND
  ST_ClosestPointOfApproach(a.geom, b.geom) IS NOT NULL;
```

```
   a    |    b    |           cpa_time
--------+---------+-------------------------------
 CAI6KA | SXS4PH  | 2023-06-17 01:40:44.999999+00
 CAI6KA | SXS4HP  | 2023-06-03 06:08:38.886364+00
 CAI6KA | GPX680  | 2023-06-03 05:37:15.311652+00
 CAI6KA | TDR2000 | 2023-06-03 06:23:20.568472+00
 CAI6KA | THY50C  | 2023-06-03 05:31:59+00
(5 rows)
```

INTERSECTIONS (TRAJECTORY VERSION)

`ST_DistanceCPA` calculates the distance between the trajectories at the CPA. If the distance at 0, then the two trajectories intersect.

```
SELECT
  a.callsign AS a,
  b.callsign AS b
FROM
  flight a, flight b
WHERE
  a.icao24 != b.icao24 AND
  a.callsign = 'CAI6KA' AND
  a.geom |=| b.geom = 0;
```

`|=|` operator is equivalent to `ST_DistanceCPA`

```
   a    |   b
--------+--------
(0 rows)
```

CPA ANALYSIS

`ST_LocateAlong(geom, m)` finds the `POINTZ` along the trajectory at time `m`.

In our dataset, only two flights were ever within 1,000 metres of one another:

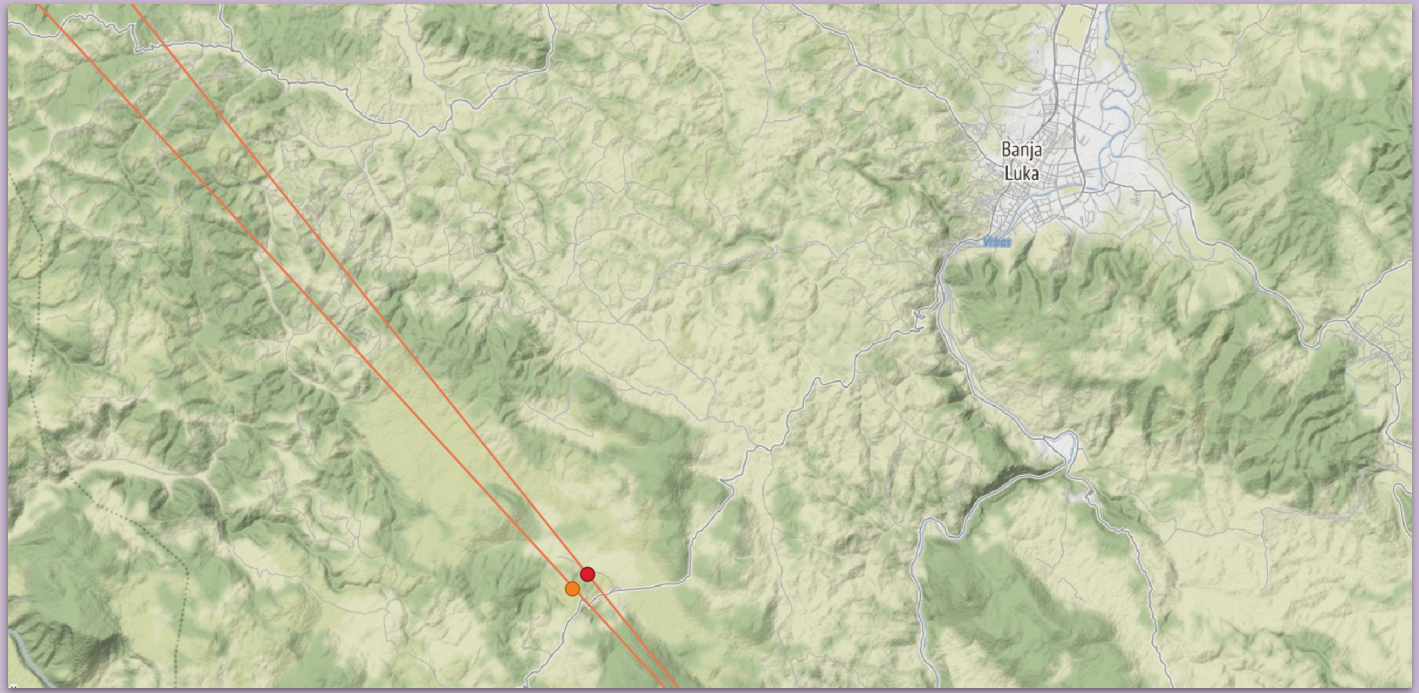
```
            time      | sep | h_sep | v_sep |    a     |    b
----------------------+-----+-------+-------+----------+---------
 2023-06-05 10:35:26 | 142 |  142  |     0 | EWG6624 | EDW403Y
 2023-06-05 12:13:34 | 780 |  624  |   468 | EWG3BH  | WZZ8004
(2 rows)
```

`sep` (separation) is the 3d distance between the two planes at their CPA.
`h_sep` is the horizontal separation.
`v_sep` is the vertical separation.

FINDING NEAR COLLISIONS

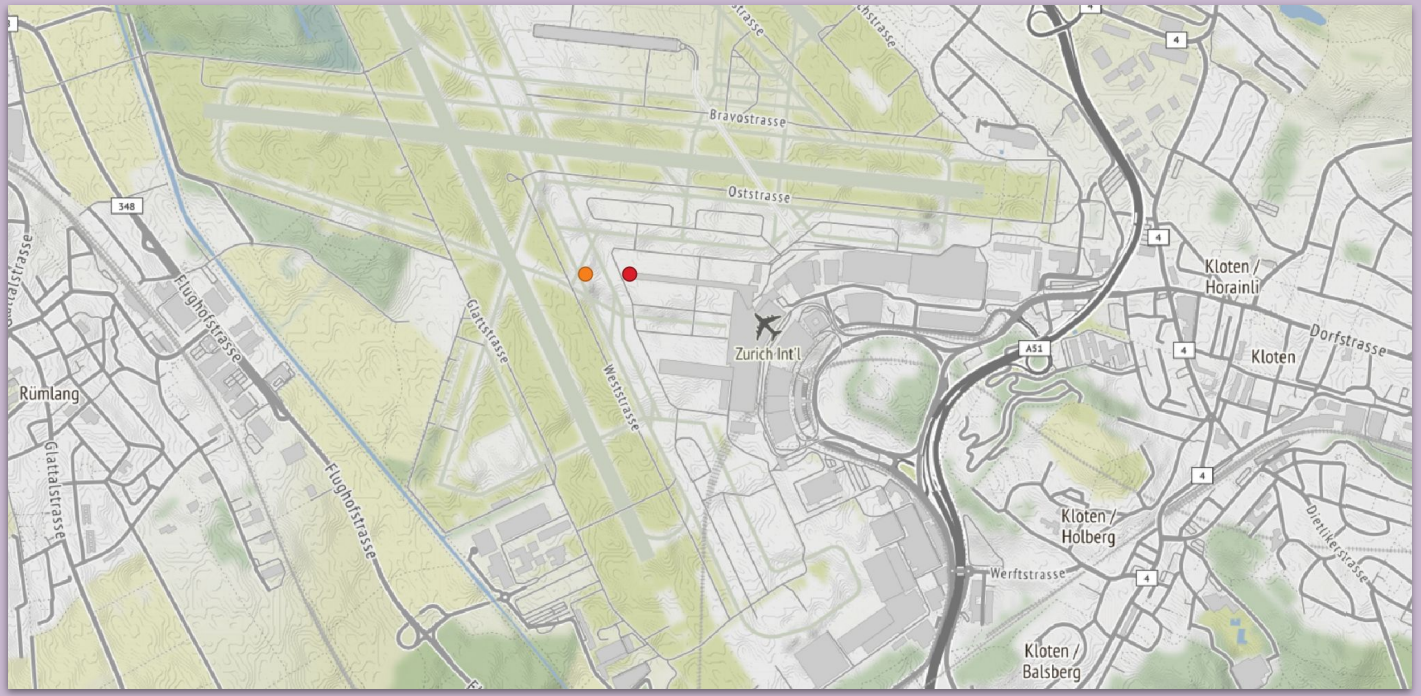
time	sep	h_sep	v_sep	a	b	a_position	b_position
2023-06-05 10:35:26	142	142	0	EWG6624	EDW403Y	POINT ZM (8.556 47.454 304)	POINT ZM (8.554 47.454 304)
2023-06-05 12:13:34	780	624	468	EWG3BH	WZZ8004	POINT ZM (17.015 44.66 10922)	POINT ZM (17.009 44.656 10454)

(2 rows)



FINDING NEAR COLLISIONS

```
            time      | sep | h_sep | v_sep |    a    |    b    |            a_position            |            b_position
----------------------+-----+-------+-------+---------+---------+----------------------------------+----------------------------------
 2023-06-05 10:35:26 | 142 |  142  |     0 | EWG6624 | EDW403Y | POINT ZM (8.556 47.454 304 )     | POINT ZM (8.554 47.454 304 )
 2023-06-05 12:13:34 | 780 |  624  |   468 | EWG3BH  | WZZ8004 | POINT ZM (17.015 44.66 10922 ) | POINT ZM (17.009 44.656 10454 )
(2 rows)
```



RELATED PROJECTS

MobilityDB

movingpandas

www.opensky-network.org

www.jaxartes.net/pages/ads-b

github.com/bosth/plpygis

github.com/bosth/geofdw

github.com/bosth/foss4g-2023-docker

bosth@sfu.ca

SOFTWARE USED

PostgreSQL

PostGIS
Spatial PostgreSQL

python

Multicorn

QGIS

APPENDIX

FLIGHT ANALYSIS

WHAT WERE SPEED AND ALTITUDE DURING FLIGHT?

```
SELECT
  to_timestamp(ST_M(endp)) AS time,
  ST_Z(endp) AS altitude,
  round(ST_Length(segment::geography) / (ST_M(endp) - ST_M(startp)) * 3600 / 1000) AS velocity
FROM (
  SELECT
    ST_Force2D(geom) AS segment,
    ST_StartPoint(geom) AS startp,
    ST_EndPoint(geom) AS endp
  FROM (
    SELECT
      (ST_DumpSegments(track.geom)).geom
    FROM (
      SELECT icao24, geom from flight AS f WHERE callsign = 'EZS1210' LIMIT 1
    ) AS track
  ) AS segment
) AS details;
```

Table view | Fill | Actual | 2023-06-17 19:08:44 to 2023-06-17 21:03:09 UTC | Time series

Aircraft Altitude and Velocity

Query 3 | Transform 0 | Alert 0

C (PostgreSQL)

Format: Table | Run query | Builder | Code

```
SELECT
  to_timestamp(ST_M(endp)),
  ST_Z(endp) AS altitude,
  round(ST_Length(segment::geography) / (ST_M(endp) - ST_M(startp)) * 3600 / 1000) AS velocity
FROM (
  SELECT
    ST_Force2D(geom) AS segment,
    ST_StartPoint(geom) AS startp,
    ST_EndPoint(geom) AS endp
  FROM (
    SELECT
      (ST_DumpSegments(track.geom)).geom
```

+ Query | + Expression


Aircraft Altitude and Velocity
800
750
700
650
600
550
500
450
400
350
300
250
200
150
100
5500
5250
5000
4750
4500
4250
4000
3750
3500
3250
3000
2750
2500
2250
2000
1750
1500
1250
1000
750
500
250
0
20:48:00
20:49:00
20:50:00
20:51:00
20:52:00
20:53:00
20:54:00
20:55:00
20:56:00
20:57:00
20:58:00
20:59:00
21:00:00
21:01:00
velocity
altitude

WHAT WERE SPEED AND ALTITUDE DURING FLIGHT?

```
SELECT
  to_timestamp(ST_M(endp)) AS time,
  ST_Z(endp) AS altitude,
  round(ST_Length(segment::geography) / (ST_M(endp) - ST_M(startp)) * 3600 / 1000) AS velocity
FROM (
  SELECT
    ST_Force2D(geom) AS segment,
    ST_StartPoint(geom) AS startp,
    ST_EndPoint(geom) AS endp
  FROM (
    SELECT
      (ST_DumpSegments(track.geom)).geom
    FROM (
      SELECT icao24, geom from flight AS f WHERE callsign = 'EZS1210' LIMIT 1
    ) AS track
  ) AS segment
) AS details WHERE ST_M(endp) - ST_M(startp) > 5;
```

WHAT WERE SPEED AND ALTITUDE DURING FLIGHT?

time	altitude	velocity
...		
2023-06-17 20:56:19+00	1219	361
2023-06-17 20:56:20+00	1219	563
2023-06-17 20:56:21+00	1219	448
2023-06-17 20:56:22+00	1219	162
2023-06-17 20:56:23+00	1219	576
2023-06-17 20:56:24+00	1219	369
2023-06-17 20:56:25+00	1219	415
2023-06-17 20:56:26+00	1219	422
2023-06-17 20:56:28+00	1219	380
2023-06-17 20:56:30+00	1219	277
2023-06-17 20:56:33+00	1219	472
2023-06-17 20:56:37+00	1219	326
2023-06-17 20:57:29+00	914	367
2023-06-17 20:58:30+00	609	352
2023-06-17 20:59:40+00	304	313
2023-06-17 21:01:01+00	0	269
2023-06-17 21:01:27+00	0	262

Multiples of ~304.5?

SMOOTHING THE ELEVATIONS

```
CREATE OR REPLACE FUNCTION
    interpolate_track_elevation(geom_in GEOMETRY(LINESTRINGZM))
RETURNS
    GEOMETRY(LINESTRINGZM)
AS $$
    from plpygis import Geometry, LineString
    from itertools import groupby
    geom = Geometry(geom_in)

    # smooth ascent/descent to new elevations

    return LineString(vertices, srid=4326)
$$ LANGUAGE plpython3u;
```

SMOOTHING THE ELEVATIONS

```
CREATE TABLE flight_smooth (LIKE flight);

INSERT INTO flight_smooth SELECT * FROM flight;

UPDATE
  flight_smooth
SET
  geom = interpolate_track_elevation(geom);
```


Aircraft Altitude and Velocity
680
660
640
620
600
580
560
540
520
500
480
460
440
420
400
380
360
340
320
300
280
260
240
5500
5250
5000
4750
4500
4250
4000
3750
3500
3250
3000
2750
2500
2250
2000
1750
1500
1250
1000
750
500
250
0
20:48:00
20:49:00
20:50:00
20:51:00
20:52:00
20:53:00
20:54:00
20:55:00
20:56:00
20:57:00
20:58:00
20:59:00
21:00:00
21:01:00
velocity
altitude