Aircraft Trajectories in PostGIS

Benjamin Trigona-Harany

Personal: bosth@sfu.ca | Professional: benjamin@planet.com
<table>
<thead>
<tr>
<th>FLIGHT</th>
<th>AIRLINE</th>
<th>FROM</th>
<th>DATE</th>
<th>SCHEDULED</th>
<th>ETA</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GM602</td>
<td>CHAIR AIRLINES</td>
<td>Zurich</td>
<td>29 June</td>
<td>18:55</td>
<td></td>
<td>SCHEDULED</td>
</tr>
<tr>
<td>TK1019</td>
<td>TURKISH AIRLINES</td>
<td>Istanbul</td>
<td>29 June</td>
<td>19:30</td>
<td></td>
<td>SCHEDULED</td>
</tr>
<tr>
<td>EZS1495</td>
<td>EASYJET</td>
<td>Geneve</td>
<td>29 June</td>
<td>19:50</td>
<td>20:07</td>
<td>EXPECTED 20:07</td>
</tr>
<tr>
<td>E46061</td>
<td>ENTER AIR</td>
<td>Stuttgart</td>
<td>29 June</td>
<td>20:05</td>
<td></td>
<td>SCHEDULED</td>
</tr>
<tr>
<td>EZS1209</td>
<td>EASYJET</td>
<td>Basel - Mulhouse</td>
<td>29 June</td>
<td>20:20</td>
<td></td>
<td>SCHEDULED</td>
</tr>
<tr>
<td>EW5704</td>
<td>EUROWINGS</td>
<td>Hamburg</td>
<td>29 June</td>
<td>20:40</td>
<td></td>
<td>SCHEDULED</td>
</tr>
</tbody>
</table>
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;

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

SELECT
  f.callsign, string_agg(c.name, ',', '')
FROM
  flight f, country c
WHERE
  ST_Intersects(f.geom, c.geom)
GROUP BY f.callsign;
WHAT HAPPENS WHEN YOU TRAVEL BY PLANE IN TWO DIMENSIONS?
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;

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

Get the highest altitude that each flight reached

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?
08:25 UTC
SPEED 2 KTS
TRACK 176°
ALTITUDE 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
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?
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.
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

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

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

```json
{
    "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;

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

See https://github.com/bosth/geofdw
Query a Foreign Table

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

<table>
<thead>
<tr>
<th>callsign</th>
<th>origin_country</th>
<th>ST_Astext</th>
</tr>
</thead>
<tbody>
<tr>
<td>UAL2619</td>
<td>United States</td>
<td>POINT Z (-90.22 35.0806 10759.44)</td>
</tr>
</tbody>
</table>

API call: `api/states/all?icao24=ab1644`
Query a Foreign Table

```sql
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;
```

<table>
<thead>
<tr>
<th>callsign</th>
<th>icao24</th>
<th>origin_country</th>
<th>st_astext</th>
</tr>
</thead>
<tbody>
<tr>
<td>EWG79EB</td>
<td>3c5ee1</td>
<td>Germany</td>
<td>POINT Z (21.1693 42.5809 12131.04)</td>
</tr>
<tr>
<td>THY9GD</td>
<td>4baa86</td>
<td>Turkey</td>
<td>POINT Z (21.5869 42.7698 10660.38)</td>
</tr>
<tr>
<td>TOM1KM</td>
<td>406ca3</td>
<td>United Kingdom</td>
<td>POINT Z (20.0416 42.7306 10972.8)</td>
</tr>
<tr>
<td>RYR1843</td>
<td>4d2224</td>
<td>Malta</td>
<td>POINT Z (21.7624 41.9092 11254.74)</td>
</tr>
<tr>
<td>RYR63ZR</td>
<td>48c2a5</td>
<td>Poland</td>
<td>POINT Z (20.4638 42.9922 10957.56)</td>
</tr>
<tr>
<td>EDW46Y</td>
<td>4b18b8</td>
<td>Switzerland</td>
<td>POINT Z (21.5209 41.8931 3177.54)</td>
</tr>
</tbody>
</table>

(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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>icao24</td>
<td>required</td>
<td>ICAO24 transponder address represented by a hex string (e.g. abc9f3).</td>
</tr>
<tr>
<td>begin</td>
<td>required</td>
<td>Start of time interval to retrieve flights for as Unix time (seconds since epoch)</td>
</tr>
<tr>
<td>end</td>
<td>required</td>
<td>End of time interval to retrieve flights for as Unix time (seconds since epoch)</td>
</tr>
</tbody>
</table>

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);
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>icao24</td>
<td>required</td>
<td>ICAO24 transponder address represented by a hex string (e.g. abc9f3).</td>
</tr>
<tr>
<td>time</td>
<td>optional</td>
<td>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.</td>
</tr>
</tbody>
</table>

## Sample API calls

- `/tracks?icao24=3c6444`
- `/tracks?icao24=3c6444&time=1649693000`
OpenSky API: /tracks

Response

```
{
  "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
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"]])
$$
UPDATE TABLE WITH TRAJECTORIES

UPDATE
    flight
SET
    geom = opensky_get_track(icao24, time_depart)
WHERE
    geom IS NULL;
### QUERY TABLE

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

<table>
<thead>
<tr>
<th>callsign</th>
<th>valid</th>
<th>origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>JZA70</td>
<td>t</td>
<td>POINT ZM (-75.6529 45.3092 0 1685832837)</td>
</tr>
<tr>
<td>JZA69</td>
<td>t</td>
<td>POINT ZM (-69.9938 45.5988 7315 1685826302)</td>
</tr>
<tr>
<td>JZA21</td>
<td>t</td>
<td>POINT ZM (-61.041 46.5555 6400 1685794547)</td>
</tr>
<tr>
<td>JZA659</td>
<td>t</td>
<td>POINT ZM (-68.1527 44.235 7315 1685633460)</td>
</tr>
<tr>
<td>JZA664</td>
<td>t</td>
<td>POINT ZM (-71.0031 42.3814 0 1685745859)</td>
</tr>
<tr>
<td>JZA663</td>
<td>t</td>
<td>POINT ZM (-67.7245 44.1866 7315 1685738980)</td>
</tr>
<tr>
<td>JZA660</td>
<td>t</td>
<td>POINT ZM (-74.1716 40.6878 0 1685725473)</td>
</tr>
<tr>
<td>JZA660</td>
<td>t</td>
<td>POINT ZM (-74.1767 40.6798 0 1685642094)</td>
</tr>
<tr>
<td>JZA7714</td>
<td>t</td>
<td>POINT ZM (-79.3967 43.6291 0 1685578999)</td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAI6KA</td>
<td>CAI6KA</td>
</tr>
<tr>
<td>CAI6KA</td>
<td>THY8MR</td>
</tr>
<tr>
<td>CAI6KA</td>
<td>VJT929</td>
</tr>
<tr>
<td>CAI6KA</td>
<td>ENT6062</td>
</tr>
<tr>
<td>CAI6KA</td>
<td>JAV3821</td>
</tr>
<tr>
<td>CAI6KA</td>
<td>NSZ2827</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(289 rows)</td>
<td></td>
</tr>
</tbody>
</table>
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);
```

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAI6KA</td>
<td>SXS4PH</td>
</tr>
</tbody>
</table>

(1 row)
The Closest Point of Approach (CPA) is the point in time at which two trajectories are closest to each other.
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;

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>cpa_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAI6KA</td>
<td>SXS4PH</td>
<td>2023-06-17 01:40:44.999999+00</td>
</tr>
<tr>
<td>CAI6KA</td>
<td>SXS4HP</td>
<td>2023-06-03 06:08:38.886364+00</td>
</tr>
<tr>
<td>CAI6KA</td>
<td>GPX680</td>
<td>2023-06-03 05:37:15.311652+00</td>
</tr>
<tr>
<td>CAI6KA</td>
<td>TDR2000</td>
<td>2023-06-03 06:23:20.568472+00</td>
</tr>
<tr>
<td>CAI6KA</td>
<td>THY50C</td>
<td>2023-06-03 05:31:59+00</td>
</tr>
</tbody>
</table>
(5 rows)
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;
```

| a    |   b |
|-------+-----|
|--------+--------|

(0 rows)

|=| operator is equivalent to ST_DistanceCPA
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:

<table>
<thead>
<tr>
<th>time</th>
<th>sep</th>
<th>h_sep</th>
<th>v_sep</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>2023-06-05 10:35:26</td>
<td>142</td>
<td>142</td>
<td>0</td>
<td>EWG6624</td>
<td>EDW403Y</td>
</tr>
<tr>
<td>2023-06-05 12:13:34</td>
<td>780</td>
<td>624</td>
<td>468</td>
<td>EWG3BH</td>
<td>WZZ8004</td>
</tr>
</tbody>
</table>

(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

<table>
<thead>
<tr>
<th>time</th>
<th>sep</th>
<th>h_sep</th>
<th>v_sep</th>
<th>a</th>
<th>b</th>
<th>a_position</th>
<th>b_position</th>
</tr>
</thead>
<tbody>
<tr>
<td>2023-06-05 10:35:26</td>
<td>142</td>
<td>142</td>
<td>0</td>
<td>EWG6624</td>
<td>EDW493Y</td>
<td>POINT ZM (8.556 47.454 304)</td>
<td>POINT ZM (8.554 47.454 304)</td>
</tr>
<tr>
<td>2023-06-05 12:13:34</td>
<td>780</td>
<td>624</td>
<td>468</td>
<td>EWG3BH</td>
<td>WZZ8004</td>
<td>POINT ZM (17.015 44.66 10922)</td>
<td>POINT ZM (17.009 44.656 10454)</td>
</tr>
</tbody>
</table>

(2 rows)
### FINDING NEAR COLLISIONS

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

(2 rows)
RELATED PROJECTS

MobilityDB

moving pandas
SOFTWARE USED

- PostgreSQL
- PostGIS
- Python
- Multicorn
- QGIS
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;
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?

<table>
<thead>
<tr>
<th>Time</th>
<th>Altitude</th>
<th>Velocity</th>
</tr>
</thead>
<tbody>
<tr>
<td>2023-06-17 20:56:19+00</td>
<td>1219</td>
<td>361</td>
</tr>
<tr>
<td>2023-06-17 20:56:20+00</td>
<td>1219</td>
<td>563</td>
</tr>
<tr>
<td>2023-06-17 20:56:21+00</td>
<td>1219</td>
<td>448</td>
</tr>
<tr>
<td>2023-06-17 20:56:22+00</td>
<td>1219</td>
<td>162</td>
</tr>
<tr>
<td>2023-06-17 20:56:23+00</td>
<td>1219</td>
<td>576</td>
</tr>
<tr>
<td>2023-06-17 20:56:24+00</td>
<td>1219</td>
<td>369</td>
</tr>
<tr>
<td>2023-06-17 20:56:25+00</td>
<td>1219</td>
<td>415</td>
</tr>
<tr>
<td>2023-06-17 20:56:26+00</td>
<td>1219</td>
<td>422</td>
</tr>
<tr>
<td>2023-06-17 20:56:28+00</td>
<td>1219</td>
<td>380</td>
</tr>
<tr>
<td>2023-06-17 20:56:30+00</td>
<td>1219</td>
<td>277</td>
</tr>
<tr>
<td>2023-06-17 20:56:33+00</td>
<td>1219</td>
<td>472</td>
</tr>
<tr>
<td>2023-06-17 20:56:37+00</td>
<td>1219</td>
<td>326</td>
</tr>
<tr>
<td>2023-06-17 20:57:29+00</td>
<td>914</td>
<td>367</td>
</tr>
<tr>
<td>2023-06-17 20:58:30+00</td>
<td>609</td>
<td>352</td>
</tr>
<tr>
<td>2023-06-17 20:59:40+00</td>
<td>304</td>
<td>313</td>
</tr>
<tr>
<td>2023-06-17 21:01:01+00</td>
<td>0</td>
<td>269</td>
</tr>
<tr>
<td>2023-06-17 21:01:27+00</td>
<td>0</td>
<td>262</td>
</tr>
</tbody>
</table>

Multiples of ~304.5?
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);