Mobility Data Analysis

Assignment 3: Vessels

AIS, standing for Automatic Identification System, is the location-tracking system for sea vessels. AIS equipment on board of ships produces data that are collected by a shore-based AIS system operated by the Danish Maritime Authority. AIS data are published in CSV file format. Check the site https://dma.dk/safety-at-sea/navigational-information/ais-data for more details.

You must be aware that this information is produced by the ships' own instruments and contains errors, such as erroneous ship's speeds and/or positions. We should analyze data quality and, if necessary, define and apply some rules to fix or discard data with problems (ETL). Trajectories will be built from raw AIS Data.

In this assignment we propose to analyze trajectory patterns to learn some basic MobilityDB functionality. Whenever possible we also visualize data in some GIS tool such as QGIS.

Exercise 1. Creating and populating the database

1.1 Create a mobilityDB-enabled database

```
CREATE DATABASE  vessels;
```

Choose it and run the following sentence:

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

1.1.1 Download the file aisdk_20180401.zip, which contains temporal data occurred during April-2018. More precisely, we will use only information generated during one day, 01/04/2018.

You must create the table that stores the information in the csv.

For this, execute the following sentence:

```
CREATE TABLE AISInput(
T timestamp,
TypeOfMobile varchar(50),
MMSI integer,
Latitude float,
Longitude float,
navigationalStatus varchar(60),
ROT float,
SOG float,
COG float,
Heading integer,
IMO varchar(50),
Callsign varchar(50),
Name varchar(100),
ShipType varchar(50),
CargoType varchar(100),
Width float, Length float,
TypeOfPositionFixingDevice varchar(50),
Draught float,
Destination varchar(50),
ETA varchar(50),
DataSourceType varchar(50),
SizeA float,
SizeB float,
SizeC float,
SizeD float
);
```

1.1.2. Populate the table.

Note that the COPY sentence is server oriented. Thus, the path needs to be solved on the server side. Assure that the timestamp format is the correct one according to the data in the csv file. If the file is in the server we can use the following sentence to import data in the target table. Run it. After some minutes, we will obtain more than one million tuples (**10619231** rows). (change the path accordingly)

```
SET datestyle TO postgres, dmy;

COPY AISInput(T, TypeOfMobile, MMSI, Latitude, Longitude, NavigationalStatus, ROT, SOG, COG, Heading, IMO, CallSign, Name, ShipType, CargoType, Width, Length, TypeOfPositionFixingDevice, Draught, Destination, ETA, DataSourceType)

FROM '/software/data/ais/aisdk_20180401.csv' DELIMITER ',' CSV HEADER;
```

When the size of the file is huge, it is convenient to follow these steps: put the file in the server and use the server-oriented COPY command.

If the file were small we could use the client-side oriented \COPY command. In this case, the file path should be found on the client side. The problem with this is that, to load the data into the table, the file should be transferred from the client to the server during the execution of this command.

Exercise 2. Cleaning the database

The csv file has at least two major problems:

- Presence of null values: The 'Unknown' text is used for representing null values. We need to convert this text into the null database value.
- Points as latitude-longitude values: The csv does not contain points. Latitude and longitude values are stored in two table columns. For spatial queries we must transform the lat/lon pairs into point geometries. In order to fix this problem, we add an extra column in the table that the csv does not contain: **geom**. Two PostGIS functions allow constructing a Point data from lat/lon: ST_MakePoint and ST_Point.

At this point, it is important to remark that PostGIS offers **geography** and **geometry** data types. The former is based on a spherical model and uses a geodetic coordinates (lat/lon) system (unit of measurement: degrees). The latter uses the Cartesian coordinate system (unit of measurement: meters/feets). The execution performance of functions such as distance, area, intersects, among others, are less complicated when working on the plane than on the spherical mode because the underlying mathematics used is simpler. Although PostGIS offers conversion between these both types, fewer functions are defined on geography than on geometry types.

Both data types above are associated with a Spatial Reference System (SRS) via a Spatial Reference System Identifier (SRID) which defines how the object is referenced to locations on the Earth's surface. On the one hand, **Geodic SRS** uses angular coordinates (lat/lon) which map directly to the surface of the earth. On the other hand, **Projected SRS** uses a mathematical transformation to flatten the spherical surface onto the plane. In general, the last one needs to be limited to a **bounded area** to avoid distortions.

For our extra column we may use geography or geometry type. *SRID 4326 is used for worldwide* geodesic shapes and *SRID 25832 is used for geometries that belong to Denmark zone.*

Fig 1- (https://epsg.io/25832) bounded zone for SRID 25832. The measures in this projection are expressed in meters.

We define the attribute

geom geometry(Point, 25832)

2.1. Considering the above discussion, run the following query.

It takes five minutes, approximately, on standard hardware.

```
ALTER TABLE AISInput
ADD COLUMN geom geometry(Point, 25832)
```

We can check with:

SELECT Find_SRID('public', 'aisinput', 'geom');

2.2. Now, we fix NULL values and generate points from the lat/lon values. When we populate the geom column using the statement:

Geom = ST_Transform(

ST_SetSRID(ST_MakePoint(Longitude, Latitude), 4326), 25832),

we would obtain an error because the ST_Transform function checks that latitude and longitude ranges from -90 to 90 and from -180 to 180, respectively. Run the following query and check that there are values outside the expected range.

```
SELECT min(latitude) as minlatitude, max(latitude) as maxlatitude,
 min(longitude) as minlongitude, max(longitude) as maxlongitude
FROM aisInput
```

	minlatitude double precision	maxlatitude double precision	minlongitude double precision	maxlongitude double precision
1	-108.482315	91	-217.806473	181

Thus, we need to exclude those values.

But, in fact, there exist other points that are outside the minimum bounding box corresponding to Denmark. According to https://epsg.io/25832 the limits should be

Latitude: 40.18 and 84.73

Longitude: -16.1 and 32.88

For this, we execute the following statement:

```
UPDATE AISInput
SET
    NavigationalStatus = CASE NavigationalStatus
                             WHEN 'Unknown value' THEN NULL END,
   IMO = CASE IMO WHEN 'Unknown' THEN NULL END,
   ShipType = CASE ShipType WHEN 'Undefined' THEN NULL END,
   TypeOfPositionFixingDevice = CASE TypeOfPositionFixingDevice
                                WHEN 'Undefined' THEN NULL END,
   Geom = ST_Transform(ST_SetSRID(ST_MakePoint(Longitude,
                Latitude), 4326), 25832)
WHERE latitude between 40.18 and 84.73 AND
          longitude between -16.1 AND 32.88
```

The execution takes approximately 10 minutes on standard hardware.

When the process finishes the points outside the boundaries will contain ***null*** values (97340 in total). Check with

```
SELECT
   NavigationalStatus,
   IMO,
   ShipType,
   TypeOfPositionFixingDevice,
   geom
FROM AISInput
WHERE latitude between 40.18 and 84.73 AND
      longitude between -16.1  and 32.88
```

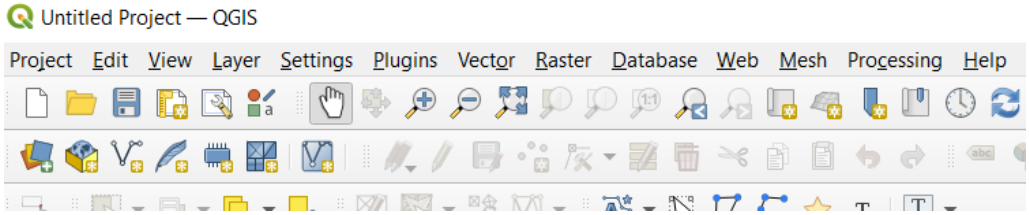
Exercise 3 Visualization and data exploration

3.1. Download QGIS V 3.22 from

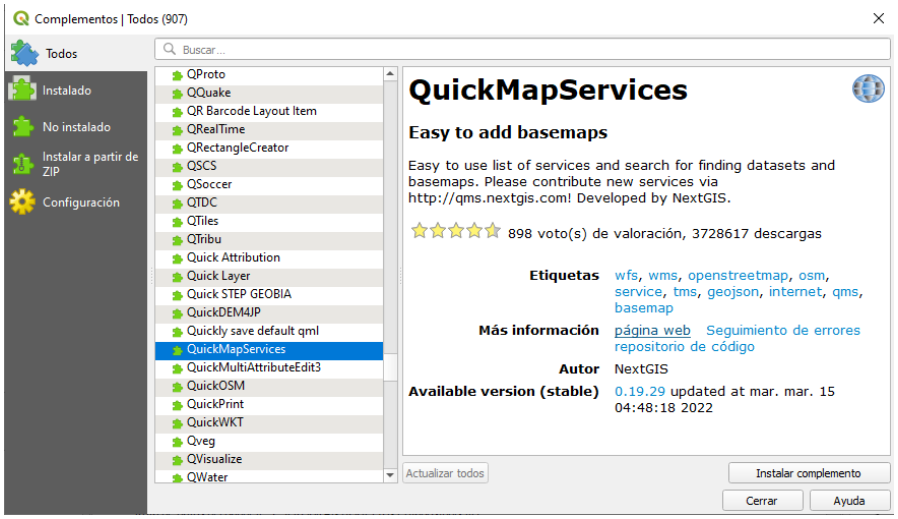
https://www.qgis.org/en/site/forusers/download.html

Open the QGIS Desktop application.

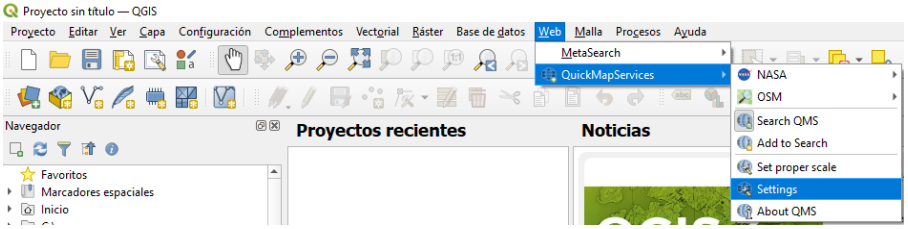
To add a map layer we first need to add the QuickMapServices QGIS plugins.



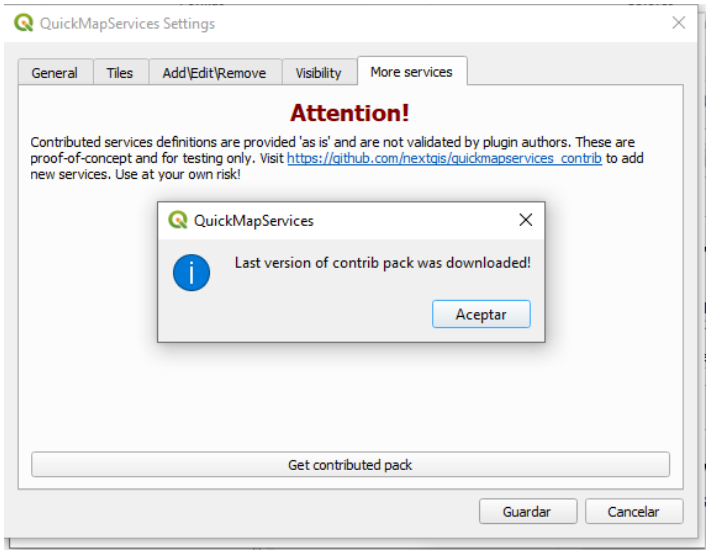
Choose the option Install Plugin:



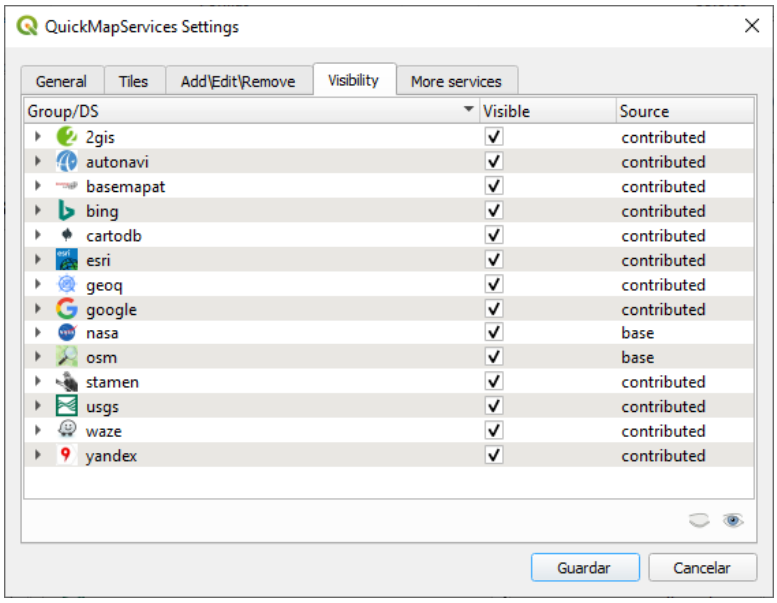
To visualize services such as Google ESRI, among others, choose the "Web-QuickMapServices-Settings" sub-menu.



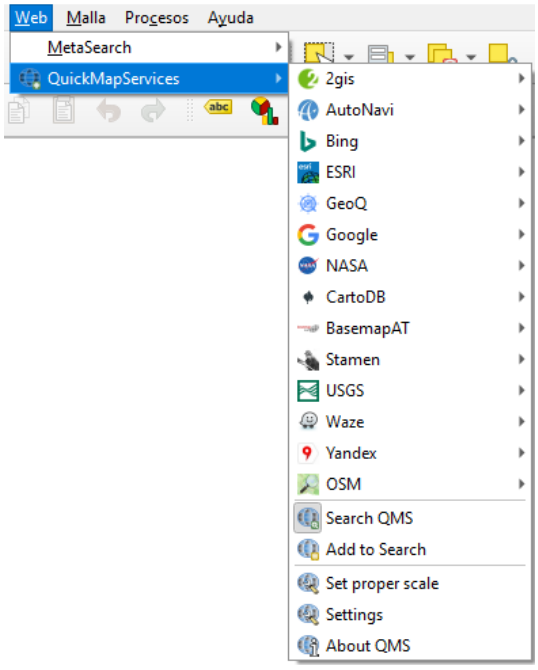
In the pop-up dialog choose the “Get Contributed Pack” option.



In the “Visibility” panel you can see the new enabled services. You can choose which ones to use. Close the dialog box.

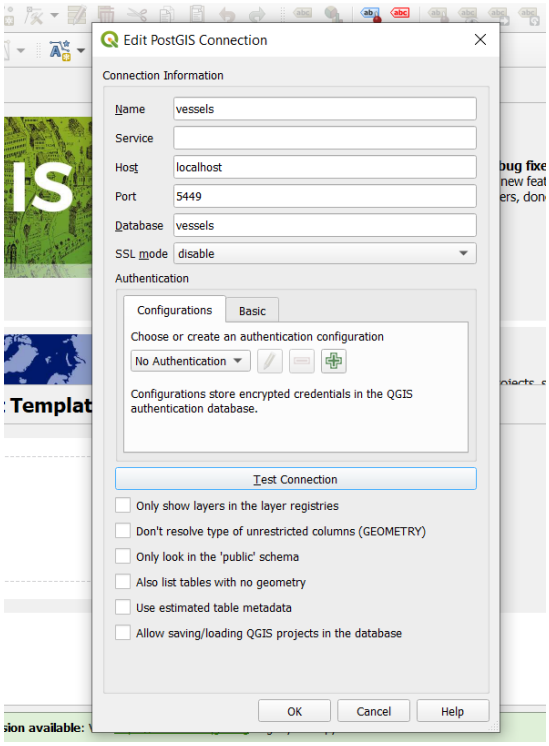


Now, the sub menu “Web-QuickMapServices” will display the enabled services.



3.2. Add the Waze World map layer or the ESRI Topo map layer, or the OSM standard or topo maps.

3.3. Create a PostGIS connection and test it.



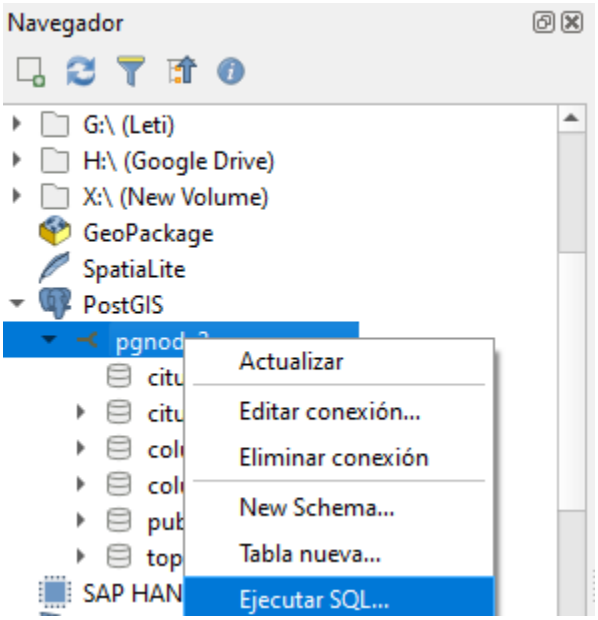
3.4. Add the geographic layers

To create a layer from an SQL query, we need to make sure that there exist at least two special columns: one that identifies each feature to be drawn and one to be displayed (geometry).

If a table contains a Primary Key/Unique and a geometry column the process is straightforward.

However, in our case study, we do not have a primary key. Thus, we need to generate an ID ad hoc.

Right-click on the connection, and choose “Execute SQL Query”. The Editor screen will appear.

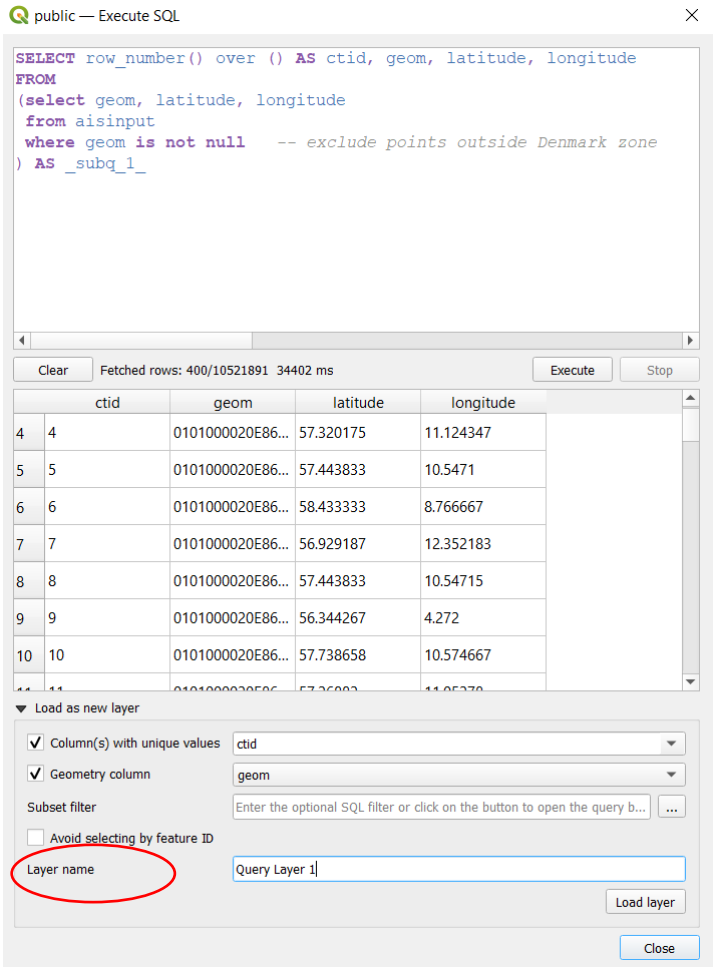


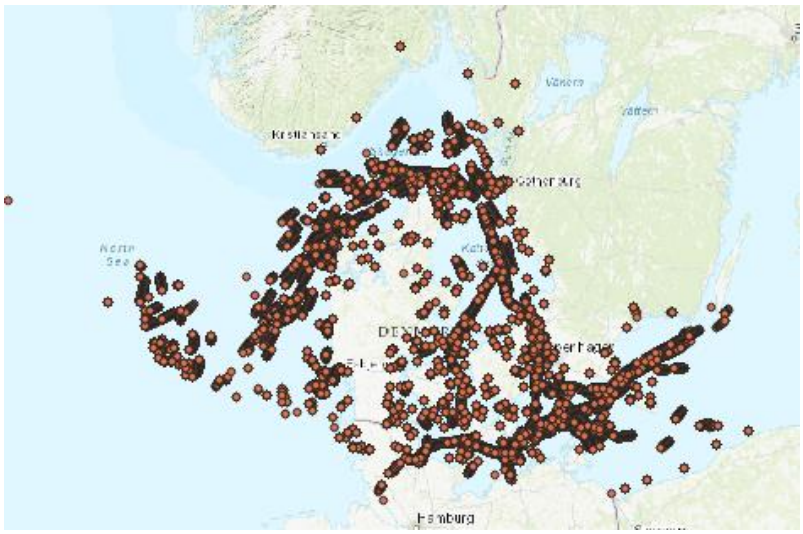
First, we run the query.

We need at least two columns: ctid (the ID) and the geometry (geom).

```
SELECT row_number() over () AS ctid, geom, latitude, longitude
FROM
(select geom, latitude, longitude
 from aisinput
 where geom is not null   -- exclude points outside Denmark zone
) AS _subq_1_
```

Once the execution is finished, we need to load the result in a layer. Check the "Column with unique values" and "Geometry Column" boxes and choose *ctid* and *geom* values. Then click the "Load Layer" option. After several minutes (depending on the size of the dataset) we will obtain the result set over the map. This can also take a while.





3.5. Due to the lack of a data quality analysis, there are other problems that have not been detected and should be fixed before generating the trajectories.

The MMSI is the vessel identifier. Thus, it is not possible that the same vessel can report more than one position at the same instant. To find out the vessels with more than one location at the same time instant, we run the following query:

```
SELECT mmsi, t, count(geom)
FROM aisinput
WHERE geom is not null
GROUP BY mmsi, t
HAVING count(geom) > 1
```

We found that 161117 tuples have problems.

3.6. Calculate the percentage of vessels with this problem

```
select count(distinct mmsi)
from aisinput
where geom is not null
-- There are 2995 vessels.
SELECT  count(distinct mmsi)
FROM (
```

```
    SELECT mmsi, t, count(geom)
    FROM aisinput
    WHERE geom is not null
    GROUP BY mmsi, t
    HAVING count(geom) > 1
    ) as a
-- Returns  1873 vessels
-- 62,53% of vessels have problems.
```

3.7. The analysis scenario could be divided in two cases:

- If a vessel (MMSI) at the same time reports the same location severañ times, the solution is straightforward: keep only one of them.

Check that:

```
SELECT mmsi, t, count(geom)
FROM aisinput
WHERE geom is not null
GROUP BY mmsi, t
HAVING count(distinct geom) = 1  AND count(geom) > 1
```

There are 121859 tuples that have this problem, which correspond to 1749 vessels.

- If a vessel (MMSI) at the same time reports more than one location, not necessarily the same one, the solution is more involved.

We have 161117 tuples with problems and only 121859 with a simple solution. Therefore, 39258 tuples have a bigger problem.

We now analyze some possible solutions. In each option, we generate a new table, named AISInputFilteredX, which contains the fixed tuples

- Option A: Assuming that the values are close to each other, we can consider a representative lat/lon pair as the average of latitudes and the average of longitudes, respectively. The query for this, reads:

```
CREATE TABLE AISInputFiltered1 AS
select t, max(typeofmobile) as typeofmobile, mmsi,
avg(latitude) as latitude, avg(longitude) as longitude,
max(navigationalstatus) as navigationalstatus,
max(rot) as rot, max(sog) as sog, max(cog) as cog,
max(heading) as heading, max(imo) as imo, max(callsign) as callsign,
max(name) as name, max(shiptype) as shiptype,
max(width) as width, max(length) as length,
max(typeofpositionfixingdevice) as typeofpositionfixingdevice,
max(draught) as draught, max(destination) as destination,
max(eta) as eta, max(datasourcetype) as datasourcetype,
ST_Transform(ST_SetSRID(ST_MakePoint(avg(Longitude), avg(Latitude)
), 4326), 25832) as geom
FROM aisinput
WHERE geom is not null
GROUP  BY mmsi, t
```

After several minutes, the AISInputFilter1 table contains 10357785 representative tuples.

- Option B: Ignore those points. We assume that the vessels' instruments had problems during the transmission. The decision is to ignore those tuples where for the same MMSI there is more than one location reported (either the same or not), considering that instruments were not reliable at those instants. We run the following query:

```
CREATE TABLE AISInputFiltered2 AS
SELECT *
FROM aisinput
WHERE geom IS NOT NULL AND
```

```
(mmsi, t) IN
(
        SELECT mmsi, t
        FROM aisinput
        WHERE geom IS NOT NULL
        GROUP BY mmsi, t
        HAVING count(geom) = 1
)
```

As a result, the AISInputFilter2 table contains 10196668 tuples.

That is, from the total of 10357785 (MMSI, t) pairs, we have excluded the 161117 pairs that have some problem.

- Option C: Choose one representative element. For instance, for those (MMSI, t) pairs with more than one location, choose the first one. The Postgres SQL: "SELECT DISTINCT ON (expression) LISTAATTR from TABLE" returns the first tuple in each group. If we want to control which tuple we want to keep, an ORDER BY clause should be used. Usually, we sort the table with respect to the time dimension, and keep the first occurrence. But in our case, that is not useful because the temporal attribute "t" is part of the grouping attributes. Thus, we cannot write "SELECT DISTINCT ON(MMSI,T) * FROM AISInput ORDER BY t"

  In our case, there is no information that can help us to decide which tuple is preferable to keep. Thus, we do not use ORDER BY clause. The query for this solution is as follows:

```
CREATE TABLE AISInputFiltered3 AS
SELECT DISTINCT ON(MMSI,T) *
FROM AISInput
WHERE geom IS NOT NULL
```

After several minutes, AISInputFilter3 table contains 10357785 representative tuples.

Exercise 4 Trajectory generation

For a specific MMSI, the list of 2D-points, sorted by their timestamp values, defines a trip. We assume that the vessel located at position pi at instant ti moved linearly to position pi+1 at instant ti+1.

Any of the previously AISInputFilteredX could be used, and we choose the last one, namely AISInputFiltered3.

To represent ships and their spatiotemporal trajectory (trip) we select the following attributes of interest from **AISInputFiltered3**:

- MMSI: integer invariant
- geom: geom changes over time. We use the mobilityDB temporal data type **tgeompoint**. From tgeompoints we can build trajectories.
- sog: float that changes over time. We use the mobilityDB **tfloat**.
- cog: float that changes over time. We use the mobilityDB **tfloat**

As we discussed in "Mobilitydb data model", the table Ships should be defined as follows:

```
CREATE TABLE Ships(

MMSI integer,

Trip tgeompoint,

SOG tfloat,

COG tfloat

)
```

The three attributes Trip, SOG and COG represent the evolution of a value during a sequence of time instants where the values between these instants are obtained using **linear interpolation**. Thus, we use the MobilityDB **sequence subtype** when populating these attributes.

4.1. Run the following query which builds temporal sequences for each MMSI value:

```
CREATE TABLE Ships(MMSI, Trip, SOG, COG) AS
SELECT MMSI,
      tgeompointseq(array_agg(
          tgeompoint(geom , t) ORDER BY t )),
     tfloatseq(array_agg(
          tfloat(SOG , t) ORDER BY t
      ) FILTER (WHERE SOG IS NOT NULL ) ),
      tfloatseq(array_agg(
           tfloat (COG, t) ORDER BY t
      ) FILTER (WHERE COG IS NOT NULL ) )
FROM AISInputFiltered3
GROUP BY MMSI;
```

We obtain 2995 tuples.

Finally, to display the trajectories in QGIS, we need to generate a geom column (QGIS does not interpret temporal data types). Let us name this column **traj** and populate it with the geometry of each trip.

4.2. Generate a column to store the trajectory of the ships, that is, the spatial projection of the spatiotemporal trip. Run the following query:

```
ALTER TABLE Ships ADD COLUMN Traj geometry;
UPDATE Ships SET Traj= trajectory(Trip);
```

4.3. We can compute, for example, the length or speed of trips.

Next, we list the vessels whose trips have the maximal length among all trips.

```
SELECT mmsi, length(trip)
FROM ships
WHERE length(trip) = (SELECT max(length (trip)) FROM ships)
```

4.4. We now classify vessels by their trip length. Show a histogram where for different ranges of trip lengths (measured in Km), and display number of vessels in each bucket. We want to obtain the following output:

bucketno integer	rangekm floatspan	freq bigint	bar text
1	[0, 0]	303	#####
2	(0, 50)	1691	##############################
3	[50, 100)	265	#####
4	[100, 200)	274	#####
5	[200, 500)	357	######
6	[500, 1500)	87	##
7	[1500, 10000)	15	

We run the following query:

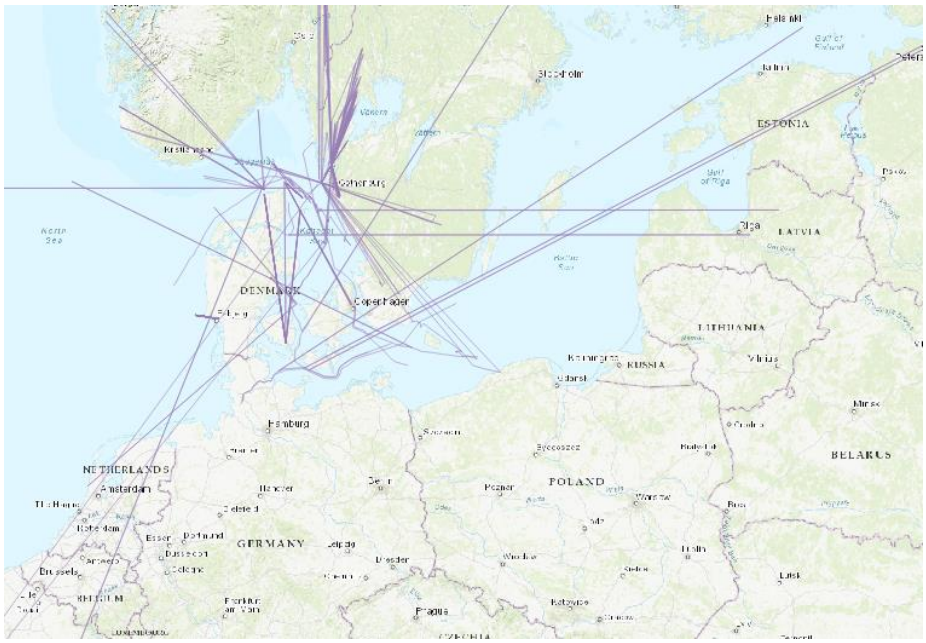
```
WITH
buckets (bucketNo, RangeKM) AS (
        SELECT 1, floatspan '[0, 0]'
 UNION SELECT 2, floatspan '(0, 50)'
 UNION SELECT 3, floatspan '[50, 100)'
 UNION SELECT 4, floatspan '[100, 200)'
 UNION SELECT 5, floatspan '[200, 500)'
 UNION SELECT 6, floatspan '[500, 1500)'
 UNION SELECT 7, floatspan '[1500, 10000)' ),
 histogram AS
        (SELECT bucketNo, RangeKM, count(MMSI) as freq
         FROM buckets LEFT OUTER JOIN
                                Ships ON (length(Trip)/1000) <@ RangeKM
-- The operator "<@" tests if the first argument is contained in the second one.
            GROUP BY bucketNo, RangeKM
            ORDER BY bucketNo, RangeKM )
SELECT bucketNo, RangeKM, freq, repeat('#', ( freq::float / max(freq)
          OVER () * 30 )::int ) AS bar
FROM histogram;
```

4.5. There are trips that are too long, outside the range above. To compute them, we write:

SELECT mmsi, traj

FROM ships

WHERE length(trip) > 1500000

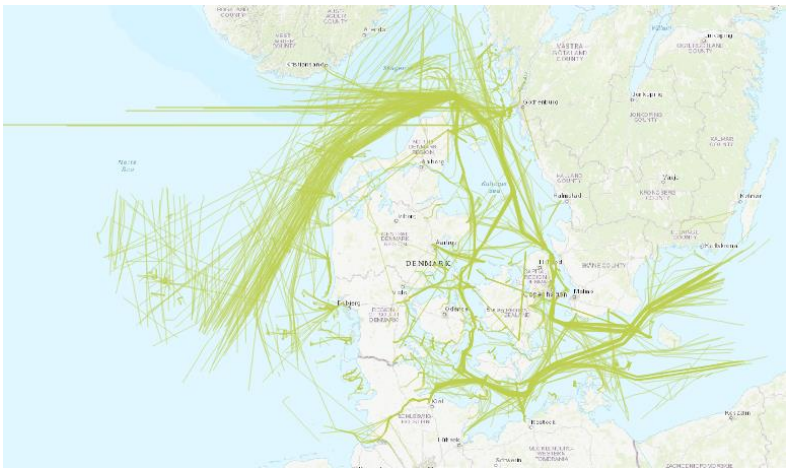


Also trips with length 0 should be excluded.

Run the following SQL. 321 tuples where deleted (either with trips of length 0 or too long trips.

```
DELETE FROM Ships
WHERE length(Trip) = 0 OR length(Trip) >= 1500000;
```

The trips to be considered are visualized as follows:



4.6. We can calculate the *speed* of the ships from the *trip* column and compare the results against the stored SOG value. Calculated speed is measured in m/s whereas SOG is expressed in knots (1 knot is equivalent to 1.852 km/h). We convert both to the same unit: km/h.

Since time interval durations differ from each other, we use the speed time-weighted average for comparing them. The mobilityDB function twavg computes the time-weighted average.

Run the following query

```
SELECT ABS(twavg(SOG) * 1.852 - twavg(speed(Trip))* 3.6 ) SpeedDifference

FROM Ships

ORDER BY SpeedDifference DESC;
```

You should obtain:

Null

Null

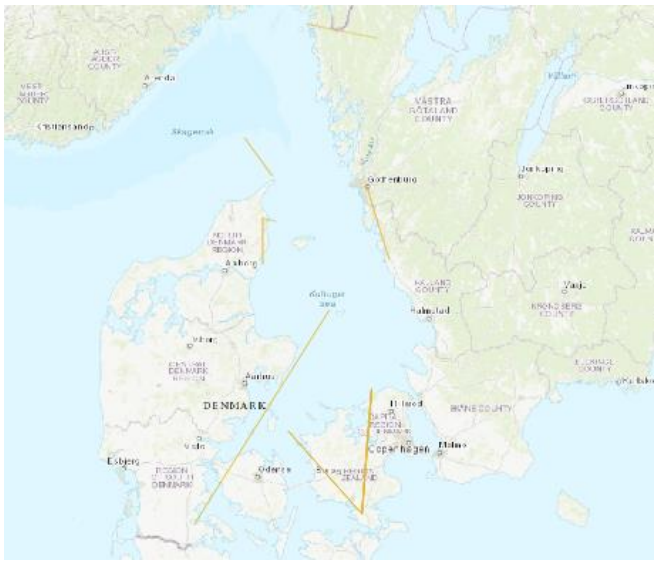
…

107.86110006787943

83.01385241476066 (tuple 35)

…..

You will note that the first 34 tuples show a difference or null or more than 100 km/h. We show these trajectories:



They do not seem to be real trajectories. Thus, delete them and check that they were deleted:

Answer

DELETE

FROM ships

WHERE

ABS(twavg(SOG) * 1.852 - twavg(speed(Trip))* 3.6) IS NULL

OR

ABS(twavg(SOG) * 1.852 - twavg(speed(Trip))* 3.6) > 100

4.7. Now we analyse trips between the ports of Rodby and Puttgarden.

Check that port of Rodby UTM coordinates are 652129 6059729

https://geohack.toolforge.org/geohack.php?pagename=R%C3%B8dbyhavn&params=54_39_43_N_11_21_31_E_type:city_region:DK-85

We can build a rectangle very close to the entrance of the harbour with:

ST_MakeEnvelope(651135, 6058230, 651422, 6058548).

Analogously, the UTM coordinates of the port of **Puttgarden** are **643543 6041415**.

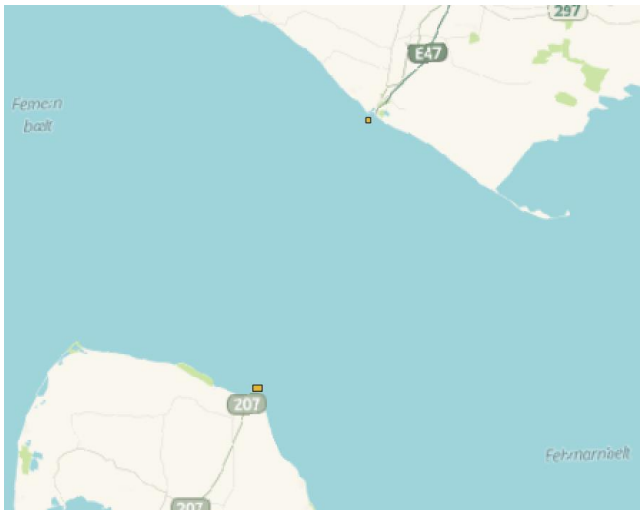
https://geohack.toolforge.org/geohack.php?pagename=Puttgarden&params=54_30_N_11_13_E_region:DE_type:city

We can build a rectangle very close to the entrance of the port:

ST_MakeEnvelope(644339, 6042108, 644896, 6042487)

The PostGIS statement next, corresponds to the definition of both rectangles:

```
SELECT  -- Rodby harbour
1 as id, ST_MakeEnvelope(651135, 6058230, 651422, 6058548, 25832) as g
UNION ALL
SELECT  -- Puttgarden port
2 as id, ST_MakeEnvelope(644339, 6042108, 644896, 6042487, 25832) as g
```



We want to analyse the trajectories that intersect both places above.

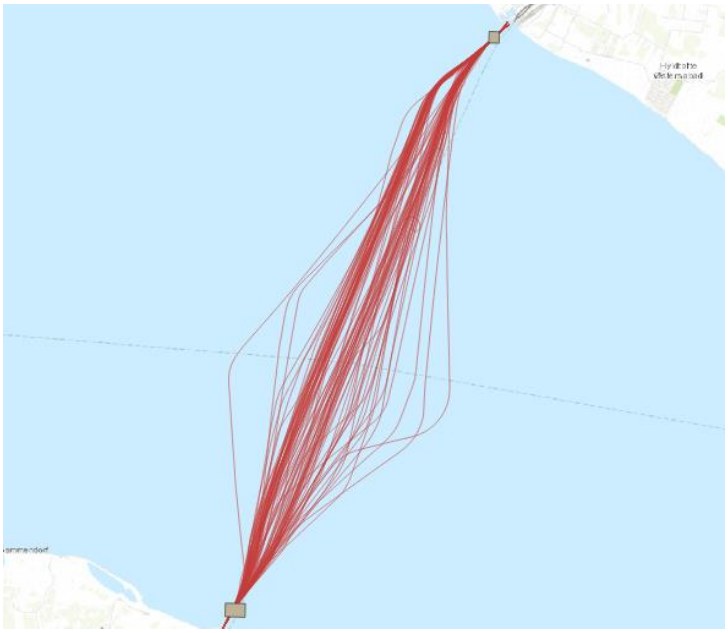
```
WITH Ports(Rodby, Puttgarden) AS
```

```
(SELECT ST_MakeEnvelope(651135, 6058230, 651422, 6058548, 25832), ST_MakeEnvelope(644339, 6042108, 644896, 6042487, 25832) )

SELECT S.*, Rodby, Puttgarden

FROM Ports P, Ships S

WHERE eintersects(S.Trip, P.Rodby) AND eintersects(S.Trip, P.Puttgarden)
```



Notice that the MobilityDB eintersects function checks whether a temporal point has ever intersected a geometry. We obtain **four ships** that continuously go from one port to the other. The figure above shows the trajectories of these four ships.

Spatio-temporal queries can be improved if an appropriate spatio-temporal index has been created. Build a GiST index over trajectories and rerun the query:

```
CREATE INDEX Ships_Trip_Idx ON Ships USING GiST(Trip);
```

4.8. The previous result set contains four vessels and their trips. Calculate how many times each of these ships repeated a one-way trip.

We can ask, for each trip, how many times the vessel touches both ports. For this, MobilityDB has two functions: **atGeometry** and **numSequences**. The former restricts the temporal point to the parts where it is inside the given geometry and returns **sequence set values**. The latter returns the number of sequences. For example, we can see that MMSI = 219000431 has a sequence set containing 32 sequences:

{[0101000020E8640000C33ED38D36E123410D0300008D1C5741@2018-04-01 04:16:42.423054-03,…],

…,

[…,
0101000020E86400003E866CAAC3E02341FA667A63791C5741@2018-04-01 20:01:49-03,
0101000020E8640000D2EA9DA408E123418C3C2415811C5741@2018-04-01 20:01:59-03,
0101000020E8640000F9B6FFFF3BE123415176C7A0861C5741@2018-04-01 20:02:08.000808-03]

}

Below, we display the last three points within the last sequence:

In red we show the zone around the port of Rodby. In green, we can see three points of the same sequence. The ship then moves to another zone that has no intersection with the red zone. Later, it enters the zone around the other port, although we are no displaying this.

Thus, we have as **many sequence**s as times the ship intersects one of those zones. In this case 16 sequences correspond to the port of Rodby and 16 sequences to the port of Puttgarden.

We can check how many sequences correspond to Rodby or Puttgarden running the following query

```
WITH Ports(g) AS (

SELECT  ST_MakeEnvelope(651135, 6058230, 651422, 6058548, 25832)
```

```
UNION
SELECT ST_MakeEnvelope(644339, 6042108, 644896, 6042487, 25832))
SELECT MMSI, array_length(sequences(atGeometry(S.Trip,g)),1),
sequences(atGeometry(S.Trip, g))
FROM Ports P, Ships S
WHERE eintersects(S.Trip, g)
ORDER BY MMSI
```

We show only a portion of the result set:

7	219000429	24	{"[0101000020E8640000A578631303DF23417DF4FF7F3D1C5741@2018-04-01 01:04:03.32978
8	219000429	24	{"[0101000020E864000047078522AC2341E40E00C0DD0C5741@2018-04-01 00:03:11.69105
9	219000431	16	{"[0101000020E86400009073574337AC2341650900C0DD0C5741@2018-04-01 05:00:00.05574
10	219000431	16	{"[0101000020E8640000C33ED38D36E123410D0300008D1C5741@2018-04-01 04:16:42.42305
11	219004263	4	{"[0101000020E8640000DA8BDD383FE02341A4DFFF7F3D1C5741@2018-04-01 07:08:59.4935
12	219016144	1	{"[0101000020E864000097FACC31F5DF23413EFEFF7F3D1C5741@2018-04-01 20:38:45.83438

The previous query does not check that the same trajectory intersects both ports. For example, the ship with MMSI 219000431 has intersected both zones, but the MMSI 219004263 had not.

The correct solution must check both intersections and can be expressed as follows (note the use of cross product instead of union):

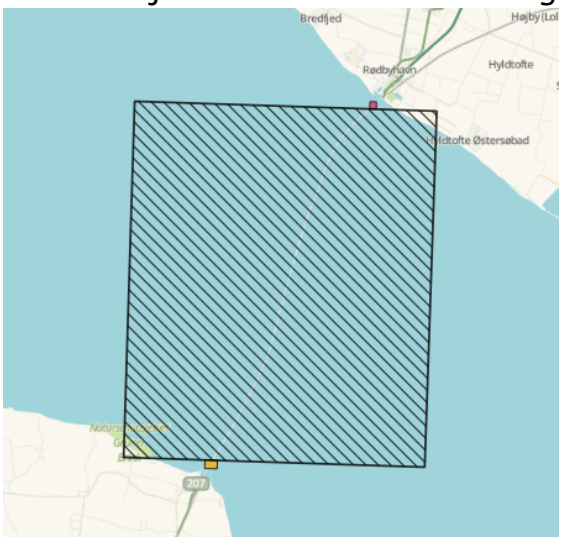
```
WITH Ports(Rodby, Puttgarden) AS (
SELECT ST_MakeEnvelope(651135, 6058230, 651422, 6058548, 25832),
ST_MakeEnvelope(644339, 6042108, 644896, 6042487, 25832) )
SELECT MMSI,
(numSequences(atGeometry(S.Trip, P.Rodby)) +
numSequences(atGeometry(S.Trip, P.Puttgarden)))/2.0 AS NumTrips
FROM Ports P, Ships S
WHERE eintersects(S.Trip, P.Rodby) AND eintersects(S.Trip,
P.Puttgarden)
```

We obtain

mmsi integer	numtrips numeric
211188000	24.0000000000000000
211190000	25.0000000000000000
219000429	24.0000000000000000
219000431	16.0000000000000000

4.9. The zone between both ports is frequently visited by lots of vessels. This zone could be dangerous if two vessels become at less than 300m from each other at some point in time.

Taking into account the rectangles at the entrance of the ports, we can restrict the analysis of trajectories within a rectangle between them



This rectangle can be built by the expression ST_MakeEnvelope(640730, 6058230, 654100, 6042487, 25832).

Restrict the analysis of trajectories exclusively to this zone.

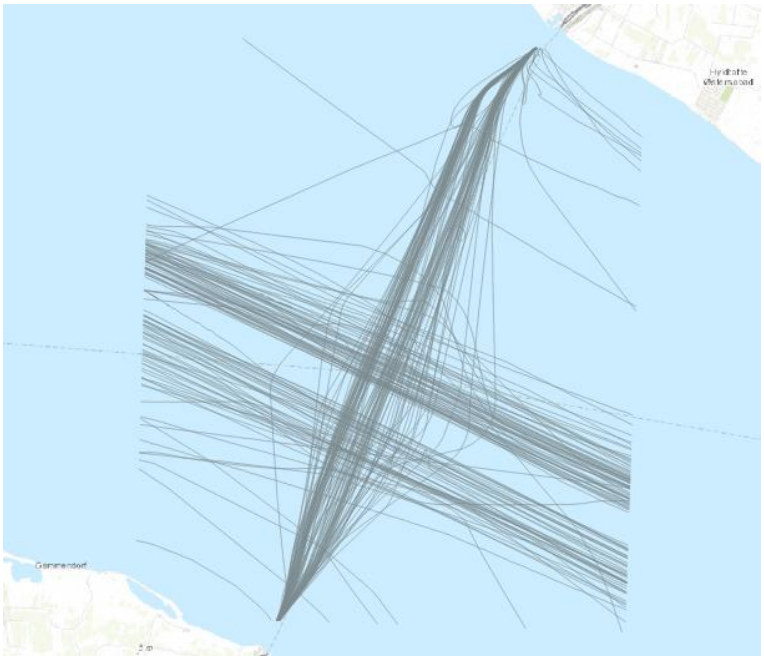
```
SELECT MMSI, atGeometry(S.Trip, belt) AS Trip,
        trajectory(atGeometry(S.Trip, belt)) AS Traj
```

```
FROM Ships S, ST_MakeEnvelope(640730, 6058230, 654100,
6042487, 25832) as belt
WHERE eintersects(S.Trip, belt)
```

Or, alternatively

```
WITH B(Belt) AS (SELECT ST_MakeEnvelope(640730, 6058230,
654100, 6042487, 25832) )
SELECT MMSI, atGeometry(S.Trip, belt) AS Trip,
                trajectory(atGeometry(S.trip, belt)) AS Traj
FROM Ships S, b as belt
WHERE eintersects(S.Trip, belt)
```

We obtain



4.10. We now compute the trajectories that have been at less than 300 meters from each other.

In the visualization, **we want to highlight the minimum distance between ships at some point in time** (probably, ships had been close to each other many times during the trajectory). To show this situation, we compute and draw the shortest line between ships.

```
WITH

B(Belt) AS (SELECT ST_MakeEnvelope(640730, 6058230, 654100,
6042487, 25832) ),

BeltShips AS

(

SELECT MMSI, atGeometry(S.Trip, B.Belt) AS Trip,
trajectory(atGeometry(S.Trip, B.Belt)) AS Traj

FROM Ships S, B WHERE eintersects(S.Trip, B.Belt)

)

SELECT S1.MMSI || '..' || S2.MMSI as both,  S1.MMSI, S2.MMSI, S1.Traj,
S2.Traj, shortestLine(S1.trip, S2.trip) Approach

FROM BeltShips S1, BeltShips S2

WHERE S1.MMSI > S2.MMSI AND edwithin(S1.trip, S2.trip, 300)
```

The method "edwithin" is other "ever relationship", i.e., the generalization of the st_dwithin for temporal points. It returns "true" if the ships have ever been at 300m from each other at some point in time.

Exercise 5

Summarize the most important **MobilityDB functions** we have used. Explain briefly their objective and enumerate all the item exercises where we have used them.

Function	Goal	Used in items…
speed(tpoint): tfloat_seqset		
length(tpoint): float		
eintersects({geo,tpoint},{geo,tpoint}): Boolean		
edwithin({geo,tpoint},{geo,tpoint},float): Boolean		
atGeometry(tgeompoint,geometry): tgeompoint		
shortestLine({geo,tpoint},{geo,tpoint}): geo		
twAvg(tnumber): float		
sequences(ttype_seqset): ttype_seq[] or in a more general form: sequences({ttype_seq,ttype_seqset}): ttype_seq[]		
numSequences(ttype_seqset): integer or in a more general way numSequences({ttype_seq,ttype_seqset}): integer		